

The Impact of DNSSEC on the Internet Landscape

Von der Fakultät für Ingenieurwissenschaften,
Abteilung Informatik und Angewandte Kognitionswissenschaft
der Universität Duisburg-Essen

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

genehmigte Dissertation

von
Matthäus Wander
aus
Lubin (Lüben)

Gutachter:
Prof. Dr.-Ing. Torben Weis
Prof. Dr.-Ing. Felix Freiling

Tag der mündlichen Prüfung: 19. Juni 2015

Abstract

In this dissertation we investigate the security deficiencies of the Domain Name System (DNS) and assess the impact of the DNSSEC security extensions. DNS spoofing attacks divert an application to the wrong server, but are also used routinely for blocking access to websites. We provide evidence for systematic DNS spoofing in China and Iran with measurement-based analyses, which allow us to examine the DNS spoofing filters from vantage points outside of the affected networks. Third-parties in other countries can be affected inadvertently by spoofing-based domain filtering, which could be averted with DNSSEC.

The security goals of DNSSEC are data integrity and authenticity. A point solution called NSEC3 adds a privacy assertion to DNSSEC, which is supposed to prevent disclosure of the domain namespace as a whole. We present GPU-based attacks on the NSEC3 privacy assertion, which allow efficient recovery of the namespace contents.

We demonstrate with active measurements that DNSSEC has found wide adoption after initial hesitation. At server-side, there are more than five million domains signed with DNSSEC. A portion of them is insecure due to insufficient cryptographic key lengths or broken due to maintenance failures. At client-side, we have observed a worldwide increase of DNSSEC validation over the last three years, though not necessarily on the last mile.

Deployment of DNSSEC validation on end hosts is impaired by intermediate caching components, which degrade the availability of DNSSEC. However, intermediate caches contribute to the performance and scalability of the Domain Name System, as we show with trace-driven simulations. We suggest that validating end hosts utilize intermediate caches by default but fall back to autonomous name resolution in case of DNSSEC failures.

Zusammenfassung

In dieser Dissertation werden die Sicherheitsdefizite des Domain Name Systems (DNS) untersucht und die Auswirkungen der DNSSEC-Sicherheitserweiterungen bewertet. DNS-Spoofing hat den Zweck eine Anwendung zum falschen Server umzuleiten, wird aber auch regelmäßig eingesetzt, um den Zugang zu Websites zu sperren. Durch messbasierte Analysen wird in dieser Arbeit die systematische Durchführung von DNS-Spoofing-Angriffen in China und im Iran belegt, wobei sich die Messpunkte außerhalb der von den Sperrfiltern betroffenen Netzwerke befinden. Es wird gezeigt, dass Dritte in anderen Ländern durch die Spoofing-basierten Sperrfilter unbeabsichtigt beeinträchtigt werden können, was mit DNSSEC verhindert werden kann.

Die Sicherheitsziele von DNSSEC sind Datenintegrität und Authentizität. Die NSEC3-Erweiterung sichert zudem die Privatheit des Domainnamensraums, damit die Inhalte eines DNSSEC-Servers nicht in Gänze ausgelesen werden können. In dieser Arbeit werden GPU-basierte Angriffsmethoden auf die von NSEC3 zugesicherte Privatheit vorgestellt, die eine effiziente Wiederherstellung des Domainnamensraums ermöglichen.

Ferner wird mit aktiven Messmethoden die Verbreitung von DNSSEC untersucht, die nach anfänglicher Zurückhaltung deutlich zugenommen hat. Auf der Serverseite gibt es mehr als fünf Millionen mit DNSSEC signierte Domainnamen. Ein Teil davon ist aufgrund von unzureichenden kryptographischen Schlüssellängen unsicher, ein weiterer Teil zudem aufgrund von Wartungsfehlern nicht mit DNSSEC erreichbar. Auf der Clientseite ist der Anteil der DNSSEC-Validierung in den letzten drei Jahren weltweit gestiegen. Allerdings ist hierbei offen, ob die Validierung nahe bei den Endgeräten stattfindet, um unvertraute Kommunikationspfade vollständig abzusichern.

Der Einsatz von DNSSEC-Validierung auf Endgeräten wird durch zwischengeschaltete DNS-Cache-Komponenten erschwert, da hierdurch die Verfügbarkeit von DNSSEC beeinträchtigt wird. Allerdings tragen zwischengeschaltete Caches zur Performance und Skalierbarkeit des Domain Name Systems bei, wie in dieser Arbeit mit messbasierten Simulationen gezeigt wird. Daher sollten Endgeräte standardmäßig die vorhandene DNS-Infrastruktur nutzen, bei Validierungsfehlern jedoch selbständig die DNSSEC-Zielserver anfragen, um im Cache gespeicherte, fehlerhafte DNS-Antworten zu umgehen.

Contents

| | |
|--|-------------|
| List of Figures | vii |
| List of Tables | xi |
| List of Acronyms | xiii |
| | |
| I Fundamentals | 1 |
| | |
| 1 Introduction | 2 |
| 1.1 Outline | 3 |
| 1.2 Contributions | 4 |
| | |
| 2 Domain Name System | 7 |
| 2.1 Namespace | 7 |
| 2.1.1 Record Types | 8 |
| 2.1.2 Lookup | 9 |
| 2.1.3 Wildcards | 10 |
| 2.2 Hierarchical Delegation | 10 |
| 2.3 System Architecture | 11 |
| 2.4 Server Redundancy | 13 |
| 2.5 Caching | 14 |
| 2.6 Network Protocol | 15 |
| 2.7 Root and Top-Level Domains | 16 |
| 2.7.1 Priming | 16 |
| 2.7.2 Root Zone Management | 17 |
| 2.7.3 TLD Management | 19 |
| 2.7.4 Alternative Roots | 20 |
| 2.8 Delegation Interdependency | 20 |

| | | |
|-----------|--|-----------|
| 3 | DNSSEC | 23 |
| 3.1 | Design Goals | 23 |
| 3.2 | Concept | 24 |
| 3.3 | Network Protocol | 25 |
| 3.3.1 | Signature | 26 |
| 3.3.2 | Public Key | 27 |
| 3.3.3 | Secure Delegation | 28 |
| 3.4 | Algorithms | 29 |
| 3.5 | Key Management | 30 |
| 3.5.1 | Key Schemes | 30 |
| 3.5.2 | Key Rollover | 30 |
| 3.5.3 | Update of Authentication Chain | 31 |
| 3.5.4 | Root Key Management | 31 |
| 3.6 | Public-Key Infrastructure | 33 |
| 3.6.1 | Trust Model | 33 |
| 3.6.2 | DNS-based Authentication of Named Entities | 34 |
| 3.7 | Authenticated Denial of Existence | 35 |
| 3.7.1 | NSEC | 35 |
| 3.7.2 | Zone Enumeration | 36 |
| 3.7.3 | NSEC3 | 37 |
| 3.7.4 | Opt-Out | 39 |
| 3.7.5 | Wildcards | 40 |
| 3.7.6 | Costs | 41 |
| 3.7.7 | Minimally Covering Records | 42 |
| | | |
| II | Security Analysis | 43 |
| | | |
| 4 | Attacker Model | 44 |
| | | |
| 5 | Attack Methods | 46 |
| 5.1 | Spoofing Attack | 46 |
| 5.2 | Spoofing Attack with Multiple Queries | 48 |
| 5.2.1 | Birthday Attack | 48 |
| 5.2.2 | Courier Attack | 49 |
| 5.2.3 | Evaluation | 50 |
| 5.3 | Kaminsky Attack | 51 |
| 5.4 | DNS Injection | 52 |
| 5.5 | Capabilities of Resolver Operators | 54 |

| | | |
|----------|--|-----------|
| 5.6 | Capabilities of Authority Operators | 58 |
| 5.7 | System Time | 59 |
| 5.8 | Breaking DNSSEC Keys | 60 |
| 6 | Measurement Study on DNS Injection | 63 |
| 6.1 | Probing for DNS Injectors | 63 |
| 6.1.1 | Method | 64 |
| 6.1.2 | Implementation | 65 |
| 6.1.3 | Measurement Result | 66 |
| 6.2 | Blacklist Testing | 71 |
| 6.2.1 | Method | 71 |
| 6.2.2 | Implementation | 71 |
| 6.2.3 | Measurement Result | 72 |
| 6.3 | Obtaining Bogus Addresses | 74 |
| 6.3.1 | Method | 74 |
| 6.3.2 | Implementation | 76 |
| 6.3.3 | Measurement Result | 76 |
| 6.4 | Impact on Resolvers | 77 |
| 6.4.1 | Method | 79 |
| 6.4.2 | Implementation | 80 |
| 6.4.3 | Measurement Result | 80 |
| 6.5 | Case Study: Unconditionally Affected Open Resolver | 85 |
| 6.6 | Case Study: DNS Injection in Hong Kong | 88 |
| 6.6.1 | Method | 89 |
| 6.6.2 | Measurement Result | 89 |
| 6.7 | Related Work | 90 |
| 6.8 | Summary and Conclusion | 93 |
| 7 | Attacking the NSEC3 Privacy Goal | 96 |
| 7.1 | Hash Crawling | 96 |
| 7.2 | Hash Breaking | 98 |
| 7.2.1 | Brute-Force Attack | 99 |
| 7.2.2 | Dictionary Attack | 99 |
| 7.2.3 | Markov Attack | 100 |
| 7.3 | Implementation | 100 |
| 7.4 | Evaluation | 102 |
| 7.4.1 | Hash Crawling | 103 |
| 7.4.2 | Hash Breaking | 104 |
| 7.5 | Discussion | 107 |

| | | |
|------------|---|------------|
| 7.6 | Related Work | 108 |
| 7.7 | Summary and Conclusion | 109 |
| III | Adoption of DNSSEC | 110 |
| 8 | Server-Side Adoption | 111 |
| 8.1 | Top-Level Domains | 111 |
| 8.1.1 | Public Keys | 112 |
| 8.1.2 | Authenticated Denial of Existence | 114 |
| 8.1.3 | Zone Enumeration | 116 |
| 8.1.4 | NSEC3 Hash Breaking | 117 |
| 8.2 | Registered Domains | 118 |
| 8.2.1 | Algorithms | 119 |
| 8.2.2 | Message Size | 121 |
| 8.2.3 | Validation Result | 122 |
| 8.3 | Summary and Conclusion | 124 |
| 9 | Client-Side Adoption | 125 |
| 9.1 | Method | 125 |
| 9.1.1 | Scripted Test | 126 |
| 9.1.2 | Hidden Test | 126 |
| 9.1.3 | Accuracy | 127 |
| 9.2 | Implementation | 128 |
| 9.3 | Analysis | 128 |
| 9.3.1 | Data Cleaning | 129 |
| 9.3.2 | Results | 130 |
| 9.4 | Related Work | 136 |
| 9.5 | Summary and Conclusions | 137 |
| 10 | Intermediate Caches | 138 |
| 10.1 | Problem Statement | 138 |
| 10.2 | Method | 139 |
| 10.3 | Data Collection | 141 |
| 10.4 | Implementation | 142 |
| 10.4.1 | Reassembly of DNS Messages | 143 |
| 10.4.2 | DNS Parser and Filter | 144 |
| 10.4.3 | Namespace Replication | 145 |
| 10.4.4 | Simulation | 146 |
| 10.4.5 | Limitations | 147 |

| | |
|---------------------------------------|------------|
| 10.5 Result Analysis | 148 |
| 10.6 Related Work | 152 |
| 10.7 Summary and Conclusion | 153 |
| 11 Conclusions and Outlook | 155 |
| 11.1 Conclusions | 155 |
| 11.2 Outlook | 157 |
| Bibliography | 158 |

List of Figures

| | | |
|------|---|----|
| 2.1 | Example domain namespace. Each node is represented by a label. The root label is empty by definition. | 8 |
| 2.2 | Example domain namespace cut into non-overlapping zones. | 10 |
| 2.3 | Stub resolver. | 12 |
| 2.4 | Recursive name server. | 12 |
| 2.5 | Authoritative name server. | 13 |
| 2.6 | Example resolver setup in residential scenario. | 13 |
| 2.7 | Example authoritative name server setup. | 14 |
| 2.8 | DNS message format. | 15 |
| 2.9 | Actors involved in root zone management. | 18 |
| 2.10 | Example lookup for wikipedia.net with an empty resolver cache. | 21 |
| 3.1 | DNSSEC provides end-to-end security and protects the path between validator and signer independent of any untrusted components in between. | 25 |
| 3.2 | Actors involved in root key management. | 31 |
| 3.3 | Trusted authorities as seen from the domain owner of verteiltesysteme.net | 33 |
| 3.4 | Comparison of DNSSEC and X.509 trust models. | 34 |
| 3.5 | NSEC composes an ordered chain of names, whereas NSEC3 composes an ordered chain of hash values of names. | 35 |
| 3.6 | SHA-1 input for the initial (upper row) and additional (lower row) NSEC3 iterations. | 39 |
| 3.7 | NSEC/NSEC3 proves that a matching wildcard does not exist and that a closer match does not exist either. | 40 |
| 4.1 | On-path attackers are listening on the wire or air. Off-path attackers do not. | 44 |
| 5.1 | Comparison of spoofing attack models with $m = 2^{16}$ different transaction IDs. | 50 |
| 5.2 | Comparison of spoofing attack models with $m = 2^{30}$ different transaction IDs and source port numbers. Probability of regular spoofing is barely above zero. | 51 |
| 5.3 | Injection of spoofed DNS response. | 52 |

| | | |
|------|--|----|
| 5.4 | Typical responses when querying an open resolver in China or Iran for a blacklisted domain name. | 53 |
| 5.5 | NXDOMAIN hijacking by an Internet service provider, December 2012. . . | 55 |
| 5.6 | Graffiti in Kadıköy, Istanbul, advising to use Google Public DNS to bypass Turkish Internet filters. [114] | 56 |
| 5.7 | Validating resolver forwards queries to non-validating resolver. | 56 |
| 5.8 | DNSKEY record format for RSA public keys, specified in RFC 3110 [2]. The length of the exponent is given by a length field, which is either 1 or 3 bytes long, indicated by the first octet. The length of the modulus can be derived from the lengths of the exponent and the overall record data. | 61 |
| 6.1 | Measurement for finding injecting networks. | 64 |
| 6.2 | Software architecture of probe measurement: data flow (solid lines) and wait dependencies (dashed lines) are shown. | 65 |
| 6.3 | Web filter notices by Bulgarian, Colombian, Indonesian, Singaporean and Turkish ISPs (top to bottom), implemented with DNS alteration on ISP resolver (<i>not</i> DNS injection). | 67 |
| 6.4 | Uniform distribution of the nine bogus IP addresses in spoofed responses from Chinese networks. | 69 |
| 6.5 | Injected responses for facebook.com over time. | 70 |
| 6.6 | Correct and spoofed responses for facebook.com by latency. | 71 |
| 6.7 | Software architecture of blacklist measurement: data flow (solid lines) and wait dependencies (dashed lines) are shown. | 72 |
| 6.8 | Number of required responses for given thresholds. | 75 |
| 6.9 | Number of bogus addresses per blacklisted domain name. | 77 |
| 6.10 | Test for DNS injection on the path between an open resolver and an authoritative name server of a top-level domain. The destination address can be chosen freely with the use of the King technique. This would not be possible with an ordinary DNS query because the destination address is otherwise chosen by the open resolver and unknown to the query sender. | 78 |
| 6.11 | Domain delegations set up (dashed lines) and DNS messages sent (solid lines) for impact measurement. | 79 |
| 6.12 | Decreasing resolver availability. | 81 |
| 6.13 | Locations of affected resolvers. | 82 |
| 6.14 | Most frequently affected authoritative name servers. | 83 |
| 6.15 | Visualization of BGP routing data created with BGPlay from RIPE NCC [106], August 2013. | 84 |

| | | |
|------|--|-----|
| 6.16 | Two unsuitable models how DNS injection affects the resolver “TW-OR”: 1) within the autonomous system (red line), 2) forwarding out and back into the autonomous system (green lines). | 86 |
| 6.17 | Time for resolving domain names in experiment 1. | 86 |
| 6.18 | Time for resolving domain names in experiment 2. | 87 |
| 6.19 | Possible model how DNS injection affects the resolver “TW-OR”. | 87 |
| 6.20 | A validating recursive name server does not supersede DNSSEC validation on the end host. | 95 |
| 7.1 | Four cases for a gap to be cut by a newly retrieved NSEC3 range. | 97 |
| 7.2 | System architecture of NSEC3 attack. | 101 |
| 7.3 | Hash breaking program flow on CPU (left column) and GPU (right column). | 101 |
| 7.4 | Time needed for hashing 2.5 billion names (less time is better). | 103 |
| 7.5 | Hashing attempts to find a new NSEC3 record in <code>com</code> . This is a calculated average based on the actual, recorded gaps. | 104 |
| 7.6 | Comparison of attack methods. | 105 |
| 7.7 | Effectiveness of n-grams as insertion words. N-grams are sorted by their frequency in the input dictionary (most frequent first). | 106 |
| 7.8 | Names found over time. | 106 |
| 7.9 | Performance comparison CPU vs. GPU. | 108 |
| 8.1 | TLDs over time (April 2013 – February 2015). | 112 |
| 8.2 | Frequency of key rollovers in top-level domains (April 2013 to February 2015). The rightmost data point means that keys were not rolled over. | 113 |
| 8.3 | Frequency of salt changes in top-level domains (April 2013 to February 2015). | 115 |
| 8.4 | Size distribution of DNSKEY responses. | 121 |
| 9.1 | Hidden test: HTTP and DNS queries seen at our servers. | 126 |
| 9.2 | Validation ratio of $\emptyset=5.5k$ trials per week (May 2012 to March 2015). | 131 |
| 9.3 | Top 5 geolocations of participating clients (May 2012 to March 2015). | 131 |
| 9.4 | Visualization of validation ratio per country. October 2014 to March 2015. | 132 |
| 9.5 | Number of resolver IP addresses seen with negative (a) and positive (b) trials. | 135 |
| 10.1 | Lock-in to bogus data from upstream resolver. | 139 |
| 10.2 | The network traces comprise internal traffic between clients and R , and external traffic between R and authoritative name servers. | 142 |
| 10.3 | Software architecture of caching simulation. | 142 |
| 10.4 | Layered reassembly of DNS messages from pcap files. | 143 |
| 10.5 | Partly overlapping TCP segments. | 144 |
| 10.6 | DNS messages over TCP streams are not necessarily aligned to TCP segments. | 144 |

| | | |
|-------|---|-----|
| 10.7 | A resolver traverses through the domain name space along the solid blue lines. It will send a network query for each traversed node, if the response is not in cache. The resolver follows delegations (dashed green lines) and spawns new lookups when encountering out-of-bailiwick server names without glue record. | 146 |
| 10.8 | Client lookups are delayed by 60 seconds in the simulation. | 147 |
| 10.9 | Distribution of query names and types as log-log plot. | 148 |
| 10.10 | Internal and external queries, grouped per 10-minute bucket. | 149 |
| 10.11 | Lookup times with and without intermediate caching. | 151 |
| 10.12 | Changes of lookup times compared to baseline $C1_{s,c}$ | 151 |

List of Tables

| | | |
|------|--|-----|
| 2.1 | IP address changes of root name servers (1997–2014). | 17 |
| 2.2 | Operators of root name servers and number of anycast sites (August 2014). | 18 |
| 3.1 | Max iteration count subject to key length. | 41 |
| 6.1 | Domain names indicating occurrence of DNS injection. | 66 |
| 6.2 | Answer IP addresses for facebook.com | 68 |
| 6.3 | Granularity of blacklist. | 74 |
| 6.4 | IPv4 addresses seen in bogus responses from Chinese DNS injection. The domain count indicates for how many different domain names (out of 404 names) the IPv4 address had been returned. | 78 |
| 6.5 | Most frequently affected networks. | 82 |
| 6.6 | Number of timeouts while waiting for responses for facebook.com (Experiment 1) and www.minghui.org (Experiment 2). | 89 |
| 8.1 | Bit lengths of RSA moduli used by top-level domains (February 2015). | 112 |
| 8.2 | Public RSA exponents used by top-level domains (February 2015). | 114 |
| 8.3 | TLDs with most secure delegations of DNSSEC domains (DS record sets). | 116 |
| 8.4 | Names found from 7.49M NSEC3 hash values after three weeks of computation. | 118 |
| 8.5 | TLDs with most reversed NSEC3 hash values. | 118 |
| 8.6 | Cryptosystems most frequently used for DNSSEC signing. | 119 |
| 8.7 | Bit lengths of RSA moduli. | 120 |
| 8.8 | Public RSA exponents. | 120 |
| 8.9 | Bit lengths of DSA groups. | 120 |
| 8.10 | DNSSEC validation result of 3.4M securely delegated domains. | 123 |
| 9.1 | Incomplete trials removed after deduplication. | 129 |
| 9.2 | Validation ratio per country (\pm 95% CI). May 2012 to March 2015. | 133 |
| 9.3 | Overview of ten validating networks. October 2014 to March 2015. | 134 |

| | | |
|------|---|-----|
| 10.1 | Results of 24.8M simulated lookups with various caching models. | 150 |
|------|---|-----|

List of Acronyms

| | |
|----------------|--|
| A | Address (record type) |
| AAAA | IPv6 address (record type) |
| ACK | Acknowledgement (TCP header flag) |
| AD | Authenticated Data (header flag) |
| AMD | Advanced Micro Devices |
| ANY | Any (query type) |
| API | Application Programming Interface |
| APNIC | Asia-Pacific Network Information Centre |
| ARPANET | Advanced Research Projects Agency Network |
| AS | Autonomous System |
| ASCII | American Standard Code for Information Interchange |
| BGP | Border Gateway Protocol |
| BIND | Berkeley Internet Name Domain |
| CA | Certificate Authority |
| CD | Checking Disabled (header flag) |
| CNAME | Canonical Name (record type) |
| CNNIC | China Internet Network Information Center |
| CPU | Central Processing Unit |
| DANE | DNS-Based Authentication of Named Entities |

| | |
|---------------|--|
| DARPA | Defense Advanced Research Projects Agency |
| DFN | Deutsches Forschungsnetz |
| DHCP | Dynamic Host Configuration Protocol |
| DNS | Domain Name System |
| DNSKEY | DNS Public Key (record type) |
| DNSSEC | Domain Name System Security Extensions |
| DO | DNSSEC OK (header flag) |
| DS | Delegation Signer (record type) |
| DSA | Digital Signature Algorithm |
| ECC | Elliptic Curve Cryptosystem |
| ECDSA | Elliptic Curve Digital Signature Algorithm |
| EDNS | Extension Mechanisms for DNS |
| FIFO | First-In First-Out |
| FIN | Finish (TCP header flag) |
| FQDN | Fully Qualified Domain Name |
| GCC | GNU Compiler Collection |
| GET | GET (HTTP method) |
| GFW | Great Firewall of China |
| GGNFS | GGNFS (software) |
| GNFS | General Number Field Sieve |
| GOST | Gosudarstvennyy Standart |
| GPU | Graphics Processing Unit |
| HKCSL | Hong Kong Communication Services Limited |
| HSM | Hardware Security Module |
| HTML | Hypertext Markup Language |

| | |
|---------------|--|
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| IANA | Internet Assigned Numbers Authority |
| ICANN | Internet Corporation for Assigned Names and Numbers |
| ID | Identifier |
| IDN | Internationalized Domain Name |
| IETF | Internet Engineering Task Force |
| IN | Internet (namespace class) |
| IP | Internet Protocol |
| IPv4 | Internet Protocol version 4 |
| IPv6 | Internet Protocol version 6 |
| ISO | International Organization for Standardization |
| ISP | Internet Service Provider |
| KSK | Key Signing Key |
| LAN | Local Area Network |
| MAC | Media Access Control |
| MD5 | Message Digest Algorithm 5 |
| MITM | Man in the Middle |
| MX | Mail Exchange (record type) |
| NIC | Network Information Center |
| NIST | National Institute of Standards and Technology |
| NODATA | No data (answer type) |
| NS | Name Server (record type) |
| NSEC | Next Secure (record type) |
| NTIA | National Telecommunications and Information Administration |

| | |
|-----------------|--|
| NTP | Network Time Protocol |
| NULL | Null (record type) |
| NVIDIA | Nvidia Corporation |
| NXDOMAIN | Non-Existent Domain (error code) |
| ORSN | Open Root Server Network |
| PCCW | Pacific Century CyberWorks |
| PKI | Public Key Infrastructure |
| PPP | Point-to-Point Protocol |
| RD | Recursion Desired (header flags) |
| REFUSED | Refused (error code) |
| RFC | Request for Comments |
| RIPE | Réseaux IP Européens |
| RIPE NCC | RIPE Network Coordination Centre |
| RR | Resource Record |
| RRSIG | Resource Record Signature (record type) |
| RSA | Rivest-Shamir-Adleman (cryptosystem) |
| RST | Reset (TCP header flag) |
| RTT | Round-Trip Time |
| SEK | Secure Entry Key |
| SEP | Secure Entry Point |
| SERVFAIL | Server Failure (error code) |
| SHA | Secure Hash Algorithm |
| SIDN | Stichting Internet Domeinregistratie Nederland |
| S/MIME | Secure/Multipurpose Internet Mail Extensions |
| SOA | Start of Authority (record type) |

| | |
|-------------|-------------------------------|
| SPF | Sender Policy Framework |
| SRV | Service (record type) |
| SSH | Secure Shell |
| SSL | Secure Sockets Layer |
| SYN | Synchronize (TCP header flag) |
| TCP | Transmission Control Protocol |
| TLD | Top-Level Domain |
| TLS | Transport Layer Security |
| TLSA | TLSA (record type) |
| TSIG | Transaction Signature |
| TTL | Time to Live |
| TXT | Text (record type) |
| UDP | User Datagram Protocol |
| URL | Uniform Resource Locator |
| U.S. | United States |
| VPN | Virtual Private Network |
| WAN | Wide Area Network |
| ZK | Zone Key |
| ZSK | Zone Signing Key |

Part I

Fundamentals

CHAPTER 1

Introduction

The Domain Name System (DNS) is a naming service that maps domain names to IP addresses and other resources. Most Internet applications depend on DNS for looking up host names and destination IP addresses, e.g. web, email or VoIP. When domain name lookups fail, applications will not be able to establish connectivity. A reliable operation of the DNS is thus essential for the health of the Internet.

The original DNS specification did not provide security measures to protect from forged domain name responses. An attacker can craft spoofed DNS responses to divert connection attempts to another host than the intended one. By tampering with domain names, an attacker can for example mount a phishing attack, hijack emails or steal credentials. DNS operators use unilateral measures within the limits of the existing protocol to partially protect from attacks, e.g. use access control or increase entropy in DNS messages to make DNS spoofing more difficult.

DNSSEC has been specified as a security extension to the Domain Name System, which uses cryptographic techniques to ensure data integrity and authenticity. DNSSEC protects from DNS spoofing attacks, provided that cryptographic primitives do not break and trusted DNS components are not compromised. This closes the attack vector of DNS tampering and asserts secure domain name lookups. The trust model of DNSSEC is based on the hierarchical domain namespace. Cryptographic keys are distributed in-band and DNSSEC does not depend on another public-key infrastructure.

There are no security guarantees beyond domain name lookups; applications have to use techniques like TLS or SSH for the protection of application payload. However, applications can utilize DNSSEC for bootstrapping security data, e.g. authentication tokens or public-key certificates. An approach for binding TLS certificates to DNSSEC has been specified as DNS-based Authentication of Named Entities (DANE).

1.1 Outline

The goal of this work is to understand how DNSSEC changes the Domain Name System and potentially influences the Internet. This dissertation is organized as follows:

Chapter 2: Domain Name System

We define the basics of the Domain Name System, describe the system architecture and network protocol. The hierarchical structure of the namespace implies a centralized organization at the root. We sketch how and by whom the root is managed and how the root interacts with top-level domains.

Chapter 3: DNSSEC

The concept of DNSSEC is to add digital signatures and public keys into the Domain Name System. Public keys are authenticated by creating a hierarchical authentication chain. We describe the protocol extensions of DNSSEC, the management of cryptographic keys and the trust model that follows from a hierarchical delegation. DNSSEC uses two alternative methods for secure denial of non-existing names: NSEC and NSEC3. The latter introduces a privacy goal into DNSSEC to prevent bulk retrieval of the domain namespace contents, but usually incurs higher costs than NSEC.

Chapter 4: Attacker Model

The attacker model that we use for the subsequent security analysis of DNSSEC consists of off-path, on-path and in-path attackers.

Chapter 5: Attack Methods

It follows an analysis of attacks on the integrity and authenticity of domain responses, including spoofing attacks and tampering by intermediate components. Besides DNSSEC, we discuss protective measures that add unpredictable randomness to DNS messages. We clarify that a spoofing attack against multiple queries—often referred to as DNS birthday attack—is not accurately represented by the birthday problem.

Chapter 6: Measurement Study on DNS Injection

DNS injection is a spoofing attack that is carried out routinely for blocking access to websites. We characterize networks that employ DNS injection with measurements from outside of the affected networks and assess the potential impact of DNS injection on unrelated third-parties. Part of this work has been published in the open access journal *IEEE Access* [125] in 2014.

Chapter 7: Attacking the NSEC3 Privacy Goal

We develop GPU-based hash breaking methods for attacks on the privacy goal of NSEC3 and evaluate their effectiveness by applying them on the `com` top-level domain. This work was presented at the *IEEE International Symposium on Network Computing and Applications* (NCA) in 2014, where it was awarded as best student paper [127]. The *nsec3breaker* tool was first introduced at the *DNS-OARC Workshop* [126] in May 2013.

Chapter 8: Server-Side Adoption

The NSEC3 attacks are applied on all top-level domains to determine the server-side adoption of DNSSEC. We survey the cryptographic algorithms and key lengths in practical use and check whether the domains validate successfully.

Chapter 9: Client-Side Adoption

We use a web-based method for measuring the client-side adoption of DNSSEC. The occurrence of DNSSEC validation varies geographically, but has increased worldwide according to our three-year measurement. The measurement method and an earlier analysis were presented at the *DNS-OARC Workshop* [124] in October 2012 and the *Passive and Active Measurement Conference* (PAM) [128] in 2013.

Chapter 10: Intermediate Caches

A system architecture of multiple levels of DNS caches managed by different authorities can be detrimental to the availability of DNSSEC-secured name resolution. We study the effectiveness of intermediate caching with trace-driven simulations. Our results indicate that cache sharing is beneficial for the lookup latency and network load to a moderate degree.

1.2 Contributions

This dissertation comprises the following major contributions and findings.

Measurement Study on DNS Injection

- We discovered that DNS injection is used for blocking social media websites in Iran in such a way that it can be observed from other countries. The blocking was temporarily suspended in mid 2013 after the election of Hassan Rouhani as President of Iran, but reinstated in October 2013.
- An algorithm for efficiently obtaining IP addresses used in spoofed DNS responses, which can be used for detecting Chinese DNS injection.

- An impact assessment of Chinese DNS injection on third-parties in other countries. DNS injection can interfere when third-parties are routed to a destination server in China, e.g. to an anycast instance of a distributed name server. We found sporadic evidence but no systematic interference when neither source nor destination are located in China.
- In a case study, we did not find evidence for Hong Kong-based networks to be affected by Chinese DNS injection when communicating with foreign destinations.

Attacking the NSEC3 Privacy Goal

- GPU-based methods for efficient NSEC3 zone enumeration.
- Practical evaluation of NSEC3 attacks in a case study with the `com` top-level domain, which revealed 64% of DNSSEC names in 5 days. The dictionary attack was the most efficient method and found 62% of names in 14 hours.

Server-Side Adoption

- A survey of the DNSSEC usage at top-level domains, including algorithms, key lengths, key rollover intervals, NSEC and NSEC3 parameters.
- A survey of the DNSSEC usage at second-level and other registered domains. We found 5.1M domains that use DNSSEC, of which almost all are signed with RSA. The distribution of signed domains varies significantly between top-level domains (most below `nl`, `br` and `cz`) due to incentives offered by top-level domain operators. Out of a subset of 3.4M DNSSEC-signed domains, 21k domains (0.6%) failed to validate.

Client-Side Adoption

- A web-based method for measuring adoption of DNSSEC validation.
- An analysis after applying the measurement method for three years. The median validation ratio per country rose from 1% in 2012 to about 20% in 2015.

Intermediate Caches

- A method for determining the effectiveness of DNS caching on intermediate, shared caches. Unlike prior work, our method takes the costs of follow-up lookups into account (resolution of out-of-bailiwick server names and target names of CNAME aliases).

- A trace-driven simulation of an intermediate cache placed upstream of 10k client caches. The intermediate cache improves the cache hit ratio from 36% to 62%, reduces the number of external queries per cache miss from 2.04 to 1.32 and reduces the average lookup latency of the 75th percentile from 42 ms to 14 ms.

CHAPTER 2

Domain Name System

The Domain Name System (DNS) is a distributed database, which maps domain names to resources. The primary purpose is to resolve domain names to IP addresses, though the DNS is not limited to this use case. In this chapter, we describe the fundamentals of the Domain Name System, including the namespace, the system architecture and the network protocol.

2.1 Namespace

The domain namespace is structured as a tree, in which *labels* represent nodes of the tree (Figure 2.1). A *domain name* is a structured name, consisting of one or more labels that are separated by a “.” (dot character). A sequence of labels represents a consecutive list of a node, its parent, the parent of its parent and so on. The significance of labels descends from right to left, i.e. labels of child nodes are appended to the left of the domain name (`child.node.parent`). By definition, the root of the tree is represented with an empty label and the root domain name is written as a single dot “.”. The domains on the first level below root are called *top-level domains* (TLDs). Accordingly, domains directly below top-level domains are called *second-level domains* etc.

Domain names are unique if written in complete form with all parent labels up to the root, e.g. `www.example.org.` (including the trailing dot). Such a complete name is called a *fully qualified domain name* (FQDN). Some definitions, e.g. RFC 1983 [88], do not include the trailing dot and it is in fact common to omit the trailing dot in applications like web or email. Besides, it is common to use incomplete or relative domain names in implementations, e.g. enter `host1` and let the operating system add the local domain `example.org`. Relative domain names and domain names without trailing dot are subject to interpretation and thus ambiguous: the domain name `mail` could for example imply the host `mail.example.org.` as well as the top-level domain `mail`. An FQDN with trailing

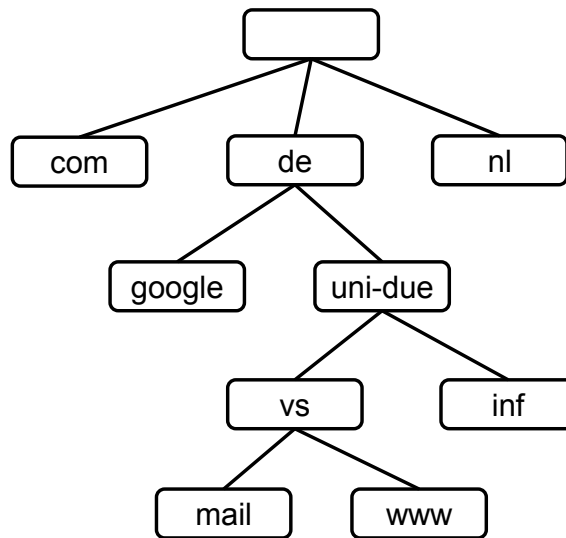


Figure 2.1: Example domain namespace. Each node is represented by a label. The root label is empty by definition.

dot can be used in applications to avoid ambiguity. However, the trailing dot is not always handled correctly in practice. For example, as of July 2014 the Microsoft.com web server dismisses HTTP requests for `http://www.microsoft.com./` with an “Invalid Hostname” error, although `www.microsoft.com.` is a valid FQDN.

Data sets in the DNS are called *resource records*. A resource record is a 5-tuple comprising:

- Owner name: the domain name of the resource record.
- Class: identifier used to distinguish different namespaces within the DNS. The only class in common use is “IN” (Internet).
- Type: identifier used to distinguish the type of data.
- Time-to-live (TTL): time in seconds for how long this resource record can be cached.
- Data: the actual data. Format depends on class and type.

An example resource record in the typical presentation format is:

| Name | TTL | Class | Type | Data |
|------------------|-------|-------|------|-----------|
| www.example.org. | 86400 | IN | A | 192.0.2.1 |

2.1.1 Record Types

Compared to general-purpose database models, DNS is limited in the type of data it carries. Record types in the DNS are tailored to specific services. For example, records of type A

and AAAA are used for IP address resolution: A for 32-bit IPv4 addresses, AAAA for 128-bit IPv6 addresses. An MX (mail exchange) record is used for email services to look up the responsible mail server of a domain. The SRV (service) record provides a more general way to locate the host for a service. The functionality of an SRV record is comparable to an MX record except that domain names are prepended with a service name and a protocol name, e.g. `_ldap._tcp.example.org`. Applications can rely on SRV records for service lookups without specification of a new DNS record type and without resolver or name server adaptation.

The list of record types is maintained by the Internet Assigned Numbers Authority (IANA). New types are specified by the Internet Engineering Task Force (IETF) in Requests for Comments (RFC) documents. Although up to 65,536 types are possible, less than 100 have been specified so far [65]. Administrators can define private types with arbitrary data format but these are limited to local use as there is no in-band signalling of the data format. Another possible approach for transporting custom data is to resort to the NULL or the TXT record type. The NULL record carries arbitrary data of any size within the given DNS message size restriction (cf. Section 2.6). There is no presentation format defined and the payload of the NULL record is opaque to DNS components. The NULL record is not in common use; being marked as experimental [95], the support in practice is unclear. In contrast, the TXT record type is standardized and widely used for carrying printable ASCII characters. The presentation format of ASCII text is self-evident, though the inner semantics of the text are opaque to DNS components. An example use case of the TXT record is the Sender Policy Framework (SPF) [72], which encodes host addresses of authorized mail senders in TXT records. Google uses the TXT record for domain ownership verification in the proprietary Google Webmaster Tools service¹.

2.1.2 Lookup

Resource records are looked up by the question tuple (name, class, type). Multiple resource records can be associated with the same name or with the same question tuple. The number of resource records per question tuple is limited by practical constraints, e.g. DNS message sizes. For example, multiple TXT records can exist for different applications but all records under one question tuple are returned as a whole. Thus, the domain name lookup is a function that maps the question tuple to the set of associated resource records:

$$(\text{name}, \text{class}, \text{type}) \mapsto \{(\text{name}, \text{class}, \text{type}, \text{TTL}, \text{data})\}. \quad (2.1)$$

¹The service shows information about the visibility of websites in Google Search to webmasters. In order to access the information, the webmaster must verify site ownership with a token in an HTML file or a TXT record.

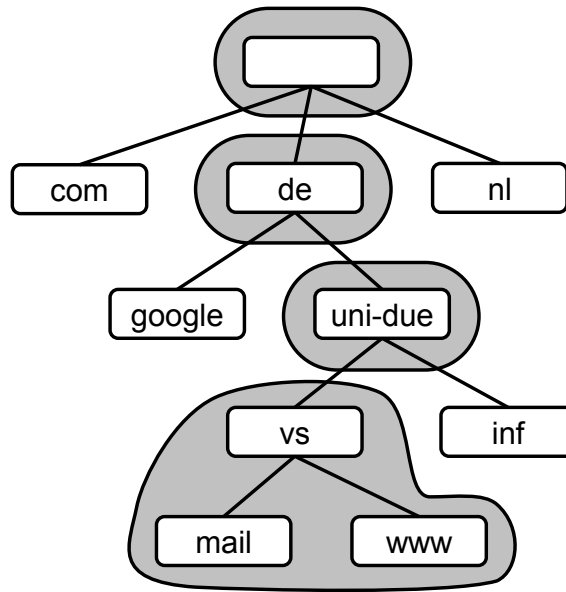


Figure 2.2: Example domain namespace cut into non-overlapping zones.

2.1.3 Wildcards

Wildcards are special resource records that serve as catch-all for non-existing names. An example use case is to provide a web service on arbitrary subdomains without prior DNS setup of each subdomain. Wildcards are indicated with a single asterisk label `*` in server configuration files. When a wildcard matches during lookup, a new resource record will be synthesized and returned with the owner name copied from the question tuple. Wildcards can be set up in conjunction with regular resource records, in which case the regular resource records have precedence over the catch-all wildcard.

2.2 Hierarchical Delegation

The domain namespace is cut into non-overlapping partitions called *zones* (Figure 2.2). A zone is a collection of resource records under a particular domain, e.g. `example.org`. The zone name is called *zone apex*, as it is the highest node of the subnamespace covered by the zone. The purpose of a zone is to ease the management of DNS data; for example, redundant name servers are configured to synchronize data per-zone instead of per-record.

Resource records can exist at the apex of a zone or at any child level, e.g. `a.example.org` or `b.a.example.org`. Alternatively, a zone can delegate subdomains to other name servers. Delegations are set up with NS resource records. The following example shows the delegation of `example.org` within the `org` zone:

```
example.org. 86400 IN NS a.iana-servers.net.  
example.org. 86400 IN NS b.iana-servers.net.
```

Delegations and other resource records are mutually exclusive. When `example.org` has been delegated, the `org` zone cannot additionally serve `www.example.org`. The record has to be placed into the `example.org` zone. An exception are *glue records*, which interlink zones by providing IP addresses of name servers. The necessity for glue records is illustrated by the following delegation:

```
iana-servers.net. 172800 IN NS a.iana-servers.net.  
iana-servers.net. 172800 IN NS b.iana-servers.net.  
iana-servers.net. 172800 IN NS c.iana-servers.net.
```

The delegation references itself because the the domain names of the name servers are within the delegated subzone. The IP addresses cannot be resolved without further information. Thus, the zone delegation is supplemented with the following glue records:

```
a.iana-servers.net. 172800 IN A      199.43.132.53  
a.iana-servers.net. 172800 IN AAAA   2001:500:8c::53  
b.iana-servers.net. 172800 IN A      199.43.133.53  
b.iana-servers.net. 172800 IN AAAA   2001:500:8d::53  
c.iana-servers.net. 172800 IN A      199.43.134.53  
c.iana-servers.net. 172800 IN AAAA   2001:500:8e::53
```

Glue is used only as a hint which server to ask next—it is never returned as the result of a domain name lookup. A lookup for (`a.iana-servers.net`, IN, A) is sent to one of the servers listed above, even though the record has been provided as glue.

2.3 System Architecture

The Domain Name System consists of *resolvers* and *name servers*. A resolver is a software module, capable of sending DNS messages over the network to name servers. There are two types of resolvers: *stub resolvers* with limited capabilities and *recursive resolvers* with full functionality for domain name lookups. Stub resolvers are simple components, traditionally built into the standard library of the operating system. Figure 2.3 shows the components a stub resolver interacts with. Applications access the service of a stub resolver via an API, e.g. the `gethostbyname()` and `gethostbyaddr()` functions. Some stub resolver implementations cache responses in memory (e.g. Windows) whereas others do not (e.g. GNU libc). Stub resolvers send all DNS messages to a name server configured manually or received automatically via the Dynamic Host Configuration Protocol (DHCP) or the Point-to-Point Protocol (PPP). A stub resolver is not capable of locating resource records in the domain

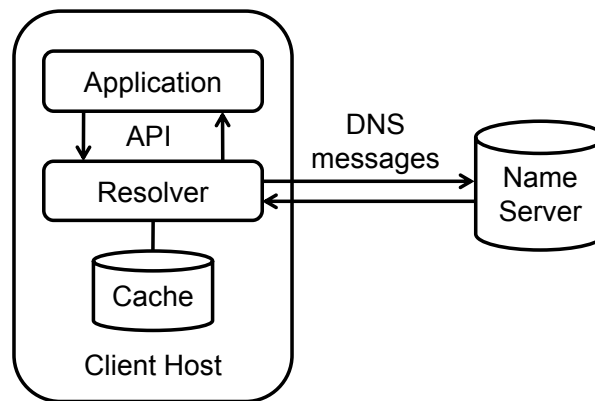


Figure 2.3: Stub resolver.

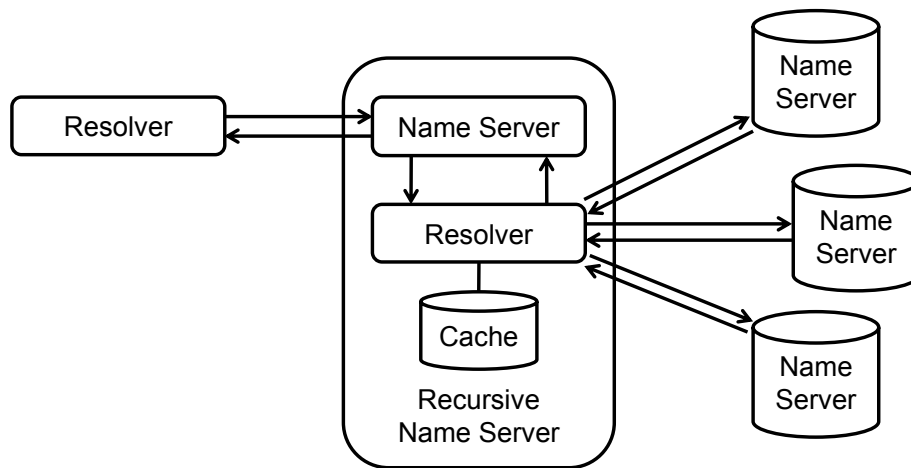


Figure 2.4: Recursive name server.

namespace as this requires the more complex recursive lookup functionality. The name server queried by a stub resolver thus has to provide the service of a recursive resolver.

A recursive resolver is capable of locating the requested question tuple by iterating through the responses from *authoritative name servers*. Figure 2.4 and Figure 2.5 show how recursive resolvers and authoritative name servers interact. An authoritative name server is serving one or more DNS zones as an authority, i.e. it is responsible for serving part of the domain namespace and not just caching it. Authoritative name servers provide the DNS zones they have been configured for; the authority does not resolve requests for delegated parts of the namespace but instead responds with a *referral* to name servers authoritative for that sub-namespace. A recursive resolver follows these referrals and sends queries to name servers until it finds the resource record with the requested information. Recursive resolver implementations almost certainly use a cache as it is essential for an efficient recursive lookup service.

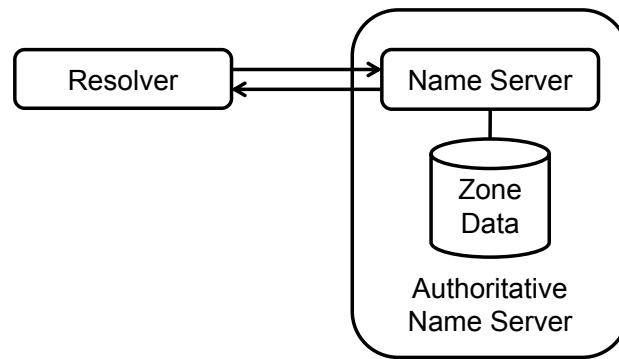


Figure 2.5: Authoritative name server.

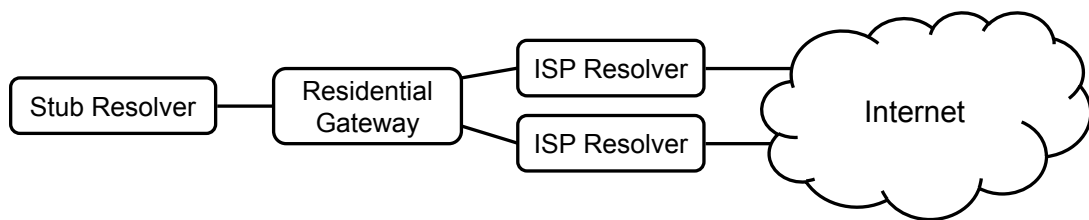


Figure 2.6: Example resolver setup in residential scenario.

In the literature, the term *recursive resolver* is often used synonymously for a name server that provides recursive service, albeit the term describes only the resolver software. A more accurate term is *recursive name server*. An *open resolver* is a recursive name server that is publicly accessible, whether with or without knowledge of the resolver operator. The term *name server* without qualifier is often used in the meaning of an authoritative name server. In terms of the generic client/server model, stub resolvers are clients, recursive name servers are proxy servers and authoritative name servers are servers.

2.4 Server Redundancy

The architecture of the DNS allows the use of redundant name servers for the purpose of load balancing and for failover in case of server failure. For example, stub resolvers can be configured with multiple recursive name servers. When one server fails to respond, the stub resolver will retry the query with the next recursive name server. Though this introduces latency and could be experienced by the user as a slow network, it increases the robustness of the DNS service.

Queries can be forwarded through multiple name servers, either because a name server lacks a recursive resolver implementation or has been configured to use another recursive name server for operational reasons. Figure 2.6 shows a typical scenario for residential users: the stub resolver is configured via DHCP to use a name server on the residential gateway.

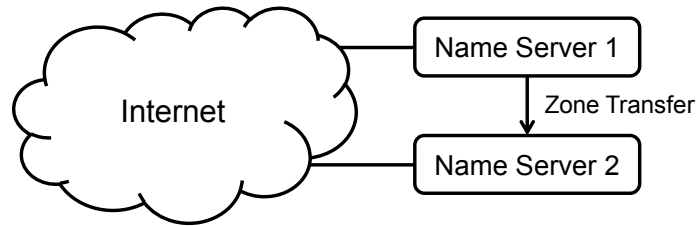


Figure 2.7: Example authoritative name server setup.

This name server provides lightweight services like name resolution of LAN hosts but does not have a recursive resolver.² It is configured via e.g. the Point-to-Point Protocol (PPP) used for the WAN connection and forwards DNS lookup requests to recursive name servers at the Internet service provider (ISP) premises.

Figure 2.7 shows a typical scenario for authoritative name servers: two redundant name servers are configured to serve the same zone. Changes to the zone data are implemented on the primary name server and are automatically synchronized with the secondary server. Operators with large query volumes may use more redundant name servers to distribute the load further. The maximum number of servers authoritative for a zone is limited by the DNS message size, as each name server address must be listed in a referral response. However, there are techniques to distribute the server load beyond this limitation:

1. A load balancer can be used to distribute queries locally to different physical hosts, even though this logical entity operates under one public address to the outside network.
2. Anycast can be used to assign the same public address to multiple name servers at different geographical locations [53]. With routing tables generated by the Border Gateway Protocol (BGP), DNS messages will be forwarded to the topologically closest anycast instance.

2.5 Caching

As mentioned in Section 2.1, each resource record contains a time-to-live (TTL) value. The TTL is chosen by the zone administrator and mandates the maximum time for how long a record can be kept in resolver cache. After a record has expired, the resolver must remove the stale record from cache. The TTL is given as relative timestamp; when returning a record from a cache, the TTL value is decreased in order to reflect the time elapsed since insertion into the cache. Though resolvers are expected to adhere the TTL values, there is no guarantee about it. Resolvers may purge entries when the cache fills up or lose the

²Dnsmasq is a popular implementation of a name server without recursive resolver, often used with Linux-based residential gateways.

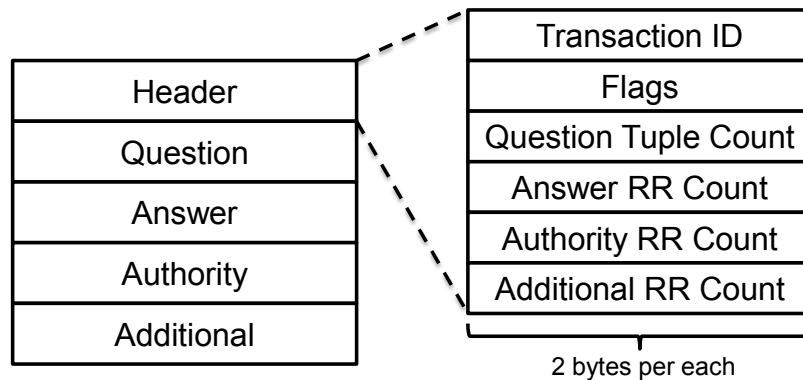


Figure 2.8: DNS message format.

content of the cache at service restart or host reboot. Thus, a zone administrator changing a record must be aware that both versions of the record are in circulation for a transition period, which is as long as the TTL of the old record. There is no mechanism for the zone administrator to force a refresh of cached records. Zone administrators can use a TTL as short as a few seconds or zero to disable caching. However, some resolvers override TTL values below a certain minimum or above a certain maximum [113].

Negative responses, i.e. “name not found” errors (NXDOMAIN), are allowed to be cached as well. Their TTL is derived from the SOA resource record, which is included in each negative answer [9]. Results of unsuccessful name lookups, e.g. due to unreachable servers or other server failures, are allowed to be cached for up to 5 minutes [9].

2.6 Network Protocol

DNS uses a stateless network protocol on top of UDP and TCP. Resolvers send DNS queries from any source port to name servers, which listen on port UDP/53 and TCP/53. Name servers are supposed to respond always, including in case of severe failures. A query/response pair is called *transaction*, not to be confused with database transactions. Queries and responses differ in semantics but use the same message syntax. The message format consists of a 12 bytes header and four variable-sized sections (Figure 2.8). The number of resource records per section is indicated in the header and unused sections may be empty. The overall message size depends on the number of records, the length of the domain names, the record type and data. As an example for a plain DNS transaction without extensions, the query (and response) for (`www.example.org`, IN, A) is 33 bytes (and 185 bytes) long without UDP or IP header.

UDP is the primarily used transport protocol. TCP is used as fallback, when UDP is not suitable due to datagram size limitations. DNS messages sent over UDP must fit into one UDP datagram. IP fragmentation may be used if available, but a DNS message

may not span more than one UDP datagram. Originally, DNS messages over UDP were limited to a maximum size of 512 bytes (plus UDP/IP headers) [95]. This was chosen due to the IPv4 specification [100], which requires hosts to accept packets of up to 576 bytes (512 bytes payload plus 64 bytes header) and makes no assurance about larger packets. The DNS message size limitation was increased later by the *Extension Mechanisms for DNS* (EDNS) [37]. EDNS is an optional and backward compatible DNS extension. With EDNS, the maximum message size is 4096 bytes, though such large UDP datagrams may be fragmented on IP layer. With EDNS and without IP fragmentation, useful message sizes are 1220 to 1440 bytes. Larger DNS messages are likely to be dropped due to size limitations on data link layer.

If a DNS message does not fit into a UDP datagram with the given size limitations, the sender will flag the message as truncated. This applies to responses only as queries practically never exceed the maximum size. A resolver receiving a truncated response will retry the query over a TCP connection. DNS over TCP uses the same message format as over UDP, except that each DNS message is prepended with its size, which can be up to 65535 bytes. TCP abstracts from the underlying packet-switched network, hence a DNS message may span multiple TCP segments of varying size. Compared with UDP, DNS over TCP costs extra round-trips and resources for connection handling. Persistent connections and pipelining—techniques used to compensate for such costs—are consistent with the specification [21, 95] but not explicitly required. Therefore, practical support is unclear. The latency and resource costs of TCP-based DNS are one of the reasons why UDP-based DNS is favored by operators. Another reason is that, while DNS over UDP is supported ubiquitously, some networks and hosts fail to support DNS over TCP correctly. Measurements by Geoff Huston [61] suggest that 2.6% of user hosts rely on resolvers that are incapable of DNS over TCP.

2.7 Root and Top-Level Domains

During name lookup, a resolver iterates through delegations until it finds the name server authoritative for the requested question tuple. The prerequisite for this automatic resolution is the knowledge of the IP addresses of the *root name servers* as starting point. The root name servers are authoritative name servers for the root zone, delegating top-level domains to the corresponding TLD name servers.

2.7.1 Priming

The list of root name servers is managed and published by the Internet Assigned Numbers Authority (IANA). Software vendors ship the current IANA list of root servers with their recursive resolvers. Recursive resolvers use an in-band mechanism called *priming* to update

| Date | Server | Change |
|-------------------|------------------|----------------------|
| June 2, 2014 | B | Added IPv6 address |
| March 26, 2014 | C | Added IPv6 address |
| January 3, 2013 | D | Changed IPv4 address |
| June 10, 2011 | D | Added IPv6 address |
| June 17, 2010 | I | Added IPv6 address |
| December 2008 | L | Added IPv6 address |
| February 2008 | A, F, H, J, K, M | Added IPv6 addresses |
| November 1, 2007 | L | Changed IPv4 address |
| January 29, 2004 | B | Changed IPv4 address |
| November 2002 | J | Changed IPv4 address |
| February 28, 1997 | L, M | Added root servers |
| January 22, 1997 | J, K | Added root servers |

Table 2.1: IP address changes of root name servers (1997–2014).

the root server list to the most current version [75]. Priming queries are sent upon startup and at regular intervals for (., IN, NS) to a randomly selected root name server. If the destination server is unreachable, the resolver will retry with another server until the query yields a response. The priming response contains the current list of 13 root name servers (named with letters from A to M) and their IP addresses as glue. The possible size of the priming response determines the maximum number of root servers. As explained in Section 2.6, legacy systems are bound to 512 bytes DNS messages. 512 bytes suffice for a priming response with 13 IPv4 addresses and 2 IPv6 addresses. EDNS is necessary to accommodate the full list with all IPv4 and IPv6 addresses.

Priming is an adequate bootstrapping mechanism if the IP address list remains stable for extended periods of time. Table 2.1 shows all changes to the root zone since 1997. Historically, root name servers changed their IPv4 address at most once to introduce anycast addressing [84], and introduced an IPv6 address. Thus, priming works even for hosts that were offline for several years.

2.7.2 Root Zone Management

IANA is the role name of the authority that is responsible for the root zone management and other functions unrelated to the DNS. The IANA functions are performed by the Internet Corporation for Assigned Names and Numbers (ICANN), which is a Californian nonprofit organization. ICANN’s legitimacy for performing the IANA functions is based on two pillars: First, ICANN organizes itself to be recognized and accepted by public and private stakeholders from an international scope. ICANN seeks to reach a consensus between the interests within the Internet community and to ensure a stable and secure operation of the Internet’s unique identifiers systems, including the DNS. Second, ICANN

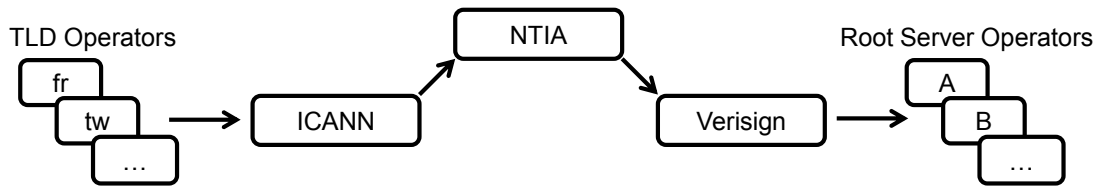


Figure 2.9: Actors involved in root zone management.

| Server | Operator | Anycast Sites |
|--------|-----------------------------|---------------|
| A | Verisign | 5 |
| B | ISI/USC | 1 |
| C | Cogent Communications | 8 |
| D | University of Maryland | 12 |
| E | NASA Ames Research Center | 12 |
| F | Internet Systems Consortium | 55 |
| G | U.S. DOD NIC | 6 |
| H | U.S. Army Research Lab | 2 |
| I | Netnod | 41 |
| J | Verisign | 73 |
| K | RIPE NCC | 17 |
| L | ICANN | 145 |
| M | WIDE Project | 7 |

Table 2.2: Operators of root name servers and number of anycast sites (August 2014).³

maintains an IANA functions contract with the U.S. government, who historically oversees the IANA functions. During the development of ARPANET, Internet pioneer Jon Postel performed the IANA functions under a contract between the Defense Advanced Research Projects Agency (DARPA) and the Information Sciences Institute (ISI) at the University of Southern California (USC). With the transition of the research-only ARPANET to the commercial Internet, the U.S. government passed oversight from DARPA to the National Telecommunications and Information Administration (NTIA), which is a regulatory body for telecommunications under the United States Department of Commerce. The NTIA has awarded the IANA functions contract to the ICANN regularly since 2000.

Figure 2.9 shows the actors involved in the management of the root zone. The IANA functions operator (role performed by ICANN) receives requests from TLD operators to change TLD delegations in the root zone. The ICANN vets the requests for formal and technical correctness and forwards them to the root zone administrator (role performed by NTIA) for authorization. The NTIA sends approved change requests to the root zone maintainer (role performed by Verisign). Verisign is a U.S.-based company selling security products and operating vital DNS infrastructure, including two root name servers. The

³<http://www.root-servers.org/>

updated root zone is then distributed by Verisign to all root server operators shown in Table 2.2.

As the Internet and the DNS originated in the United States, the operators of ten out of the thirteen root name servers are headquartered in the United States for historical reasons. Three root name servers (E, G, H) are operated by U.S. governmental or military agencies. The operators of I-root, K-root and M-root are headquartered in Sweden, the Netherlands and Japan, respectively. However, these locations are only decisive for the legal jurisdiction under which the operator acts. As shown in Table 2.2, most root servers are distributed via anycast to several sites, totaling to several hundred physical servers located on every continent except Antarctica. Anycast has proven in practice to be an effective mechanism for load distribution and to increase resilience, allowing a stable operation of the root zone despite the rather short list of 13 logical root servers.

2.7.3 TLD Management

The top-level domains are grouped into two classes: 1) *generic top-level domains* are designated for a generic topic, e.g. `com` for commercial corporations; 2) *country-code top-level domains* are designated for a regional subdivision of the domain name space, e.g. `fr` for France.

Generic TLDs are subject to a stronger regulation by ICANN than country-code TLDs; ICANN policies govern for example the process of domain registration or how to solve trademark disputes. In order to foster competition, ICANN mandates the separation of generic TLD services into registry and registrar functions. The *registry* is a TLD database, consisting of registered domains and domain owner information. The registry operator maintains the database and is responsible to serve the corresponding DNS zone on authoritative name servers. The *registrar* offers domain registration service to *registrants*, who can be corporations or individuals. After accreditation by ICANN, registrars offer domain registrations under generic TLDs that they cooperate with. Registrars compete against each other and are free to set their own prices or bundle domain registrations with extra services like web hosting. Registry operators charge fixed fees from registrars for each registered domain, bound by contract with ICANN to ensure fairness among all registrars.

In contrast, country-code TLDs can have their own policies. Some country-code TLD operators have written agreements with ICANN, but in general ICANN has no contractual framework to enforce policies for country-code TLDs. Country-code TLDs are two-character domains derived from ISO 3166 [45]. Postel wrote in 1994 that “IANA is not in the business of deciding what is and what is not a country.” [102] By relying on ISO 3166, ICANN delegates disputed choices whether a region is to be classified as country to an external organization. For example, IANA declined in 1997 to delegate a country-code TLD to Palestine due to the lack of an ISO 3166 code and instead delegated the second-level domain

`palestine.int`. After the code “PS“ had been assigned by ISO in October 1999, IANA delegated `ps` to a Palestine ministry in March 2000 [67].

Historically, the list of TLDs was set to the ISO 3166 country-code TLDs and to the following seven generic TLDs: `com`, `edu`, `gov`, `int`, `mil`, `net`, `org`. Despite Postel’s estimation in 1994 of being “extremely unlikely that any other TLDs will be created” [102], ICANN introduced 15 additional generic TLDs in 2000, 2004 and 2011 and began delegation of more than thousand new generic TLDs in 2013. Furthermore, ICANN has delegated internationalized top-level domain names (IDN) in addition to existing country-code TLDs since 2010, e.g. 新加坡 (Chinese for ‘Singapore’; ASCII compatible encoding: `xn--yfro4i67o`) to SGNIC, the operator of `sg`.

2.7.4 Alternative Roots

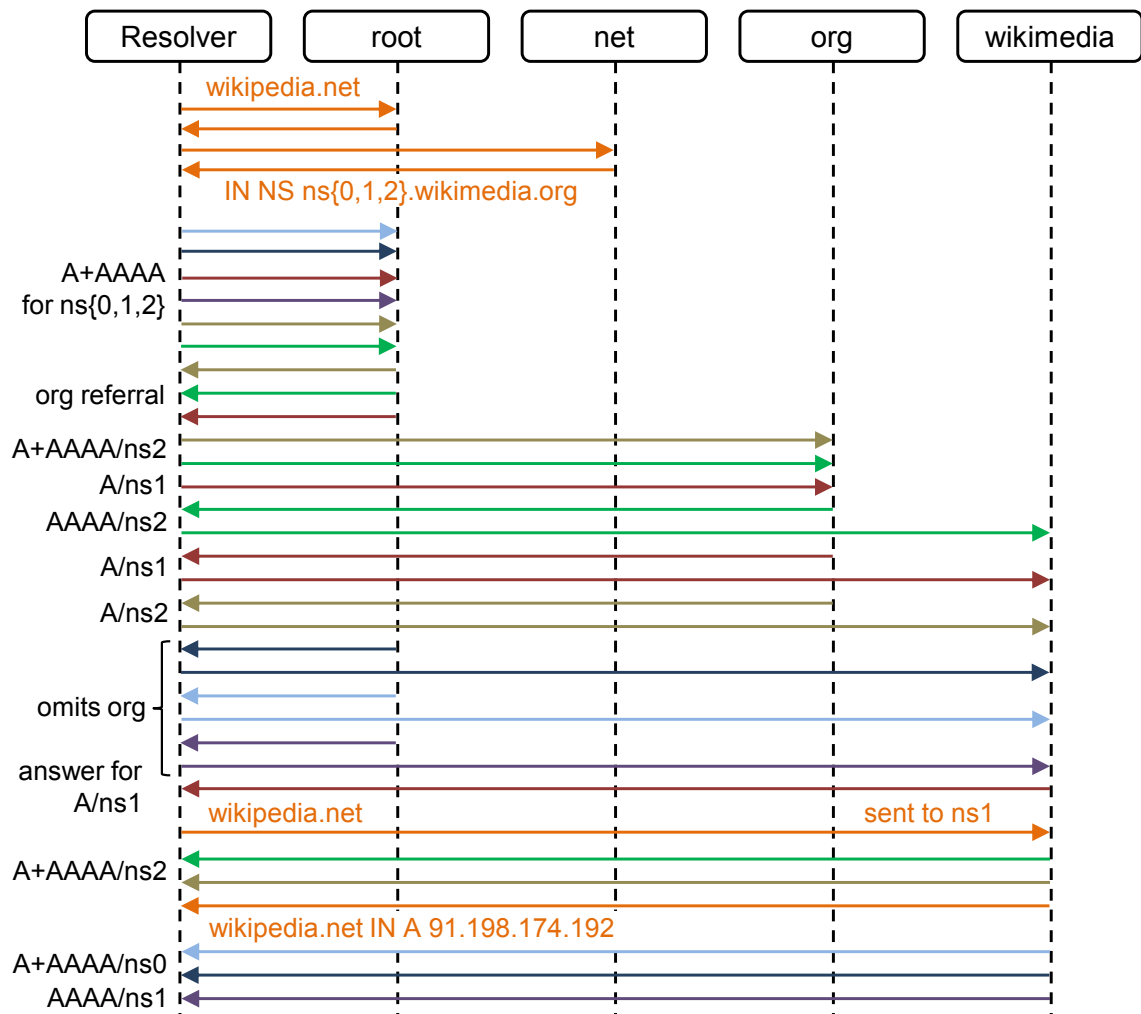
By design, there is exactly one root zone in the DNS. However, another set of root servers can be configured on recursive resolvers to use an alternative DNS namespace. An alternative DNS namespace does not need to be fully disjoint with the ICANN-coordinated namespace. For example, OpenNIC⁴ copies the IANA root zone and adds its own set of top-level domains. OpenNIC’s own TLDs can be accessed only by resolvers that point to the OpenNIC root servers; users of the ICANN-coordinated namespace are not able to resolve OpenNIC’s own TLDs. The Open Root Server Network⁵ (ORSN) is an alternative DNS root that copies the IANA root zone without changes. The motivation of the ORSN operators is to provide a political counterpart to the ICANN-coordinated root server network. Thus, though the root servers are different, they point their delegations to the same TLD name servers as in the IANA root zone. A resolver will query the ORSN servers but then will follow the delegation chain to the same TLD servers as if it had queried the ICANN-coordinated root servers.

2.8 Delegation Interdependency

A domain d is *in-bailiwick* of a domain z when d is a subdomain of z , i.e. d ends with z as right-most suffix. For example, `ns1.example.net` is in-bailiwick of `example.net` but `verteiltesysteme.net` is out-of-bailiwick of `example.net`. Ideally, all subdomain delegations are implemented as in-bailiwick delegations with the server IP addresses provided as glue records (cf. Section 2.2). In such an ideal case, a resolver with an empty cache iterates through the domain name space downwards with each query, e.g. send query to root, then `net`, then `verteiltesysteme.net`. However, out-of-bailiwick delegations are common in practice, as shown in the following example:

⁴<http://www.opennicproject.org/>

⁵<http://www.orsn.org/>

Figure 2.10: Example lookup for `wikipedia.net` with an empty resolver cache.

```
wikipedia.net.      172800  IN      NS      ns0.wikimedia.org.
wikipedia.net.      172800  IN      NS      ns1.wikimedia.org.
wikipedia.net.      172800  IN      NS      ns2.wikimedia.org.
```

`wikipedia.net` is delegated to `ns0, ns1 and ns2.wikimedia.org`. The IP addresses of the name servers cannot be provided as glue records because they are out-of-bailiwick, which means that the server of `wikipedia.net` has no authority to return records for `wikimedia.org`. Instead, the resolver has to spawn new lookup processes for the domain names `ns0, ns1 and ns2.wikimedia.org` while looking up `wikipedia.net`. A real world example is shown in Figure 2.10, observed with a BIND9 resolver with an empty cache. The resolver has spawned six lookups for A and AAAA records of `ns0, ns1 and ns2.wikimedia.org`. In total, 18 DNS transactions were sent. Only 3 DNS transactions would have been necessary with an in-bailiwick delegation of `wikipedia.net`. The use of

out-of-bailiwick delegations increases lookup times, causes additional DNS traffic and creates interdependencies between domains; when `org` fails, the lookup of `wikipedia.net` will fail, too.

CHAPTER 3

DNSSEC

The Domain Name System Security Extensions (DNSSEC) is a cryptographic security protocol suite, which is specified in several RFC documents as extensions to the Domain Name System. Originally specified by the IETF in RFC 2065 [3] in 1997, the security extensions were referring to a set of various security techniques for DNS integrity and authenticity. Today, the term DNSSEC is usually used in singular form and has a narrower meaning, which does not include techniques for transaction-level authentication [12] like TSIG or SIG(0). The foundation of DNSSEC in its current version is laid out in RFCs 4033–4035 [12–14], which have been published in 2005.

3.1 Design Goals

DNSSEC was designed to provide the following security goals [12]:

- Data integrity: resource records transferred with DNSSEC are guaranteed to be unaltered in transit.
- Data origin authenticity: resource records transferred with DNSSEC are guaranteed to originate from the maintainer of the corresponding DNS zone.

Confidentiality in any form was explicitly out of scope [12]. This includes the confidentiality of DNS messages, which are transferred in cleartext without encryption, and the DNS database, which is meant to be publicly accessible by anyone. There are later attempts to curtail this strict openness, e.g. the NSEC3 extension (Section 3.7.3) adds privacy to DNSSEC zones on authoritative name servers. As of 2015, IETF contributors discuss approaches like query name minimization [25] to improve the privacy of DNS and DNSSEC users.

One of the major requirements of DNSSEC was backward compatibility with legacy DNS [17]. DNSSEC should reuse the same network protocol, namespace and infrastructure [3] and add extensions only where needed. This allows for co-existence of secure and insecure domains within the same system and also allows for gradual deployment of DNSSEC. Furthermore, DNSSEC should support all functionality that is supported by legacy DNS, for example wildcards. Another major requirement was the possibility to store private keys offline, remote from the publicly accessible name server [3].

3.2 Concept

DNSSEC introduces digital signatures to the Domain Name System. Resource records are signed with an asymmetric cryptosystem, and the signatures are transferred along with the records to prove their integrity and authenticity. In order to support offline storage of private keys, signatures are pre-generated based on the contents of the DNS zone (*offline signing*). DNS messages are not signed; instead, the resource records inside of DNS messages are signed. The signing process runs periodically on a host in a secure, private network or on a host without network access. The zone administrator then copies the resulting signatures to public name servers, which serve the DNS zones without access to the private key. Offline signing is a designated but not mandatory DNSSEC feature. Server operators who do not need offline signing can store the private key on the authoritative name server and are free to generate signatures on-the-fly depending on the incoming queries.

Each DNSSEC zone uses one or more key pairs to sign the contents of the zone. DNSSEC uses a *chain of trust* or *authentication chain* approach for secure and automatic in-band distribution of public keys. The public key of a DNSSEC zone is signed and authenticated by its parent zone. The parent server returns a delegation to the subzone and a signed fingerprint of the public key of the subzone. With DNSSEC, the delegation of a subnamespace implies the delegation of trust for that subnamespace.

The act of checking the correctness of DNSSEC-signed resource records is called *validation*. It consists of checking basic conditions like matching names and time constraints, verifying the signature and authenticating the public key via its authentication chain. When following the authentication chain upwards, the chain ends in a public key that must be well-known and trusted by the DNSSEC validator. This public key serves as *trust anchor*. Usually, validators use the public key of the root domain as trust anchor to allow for a continuous authentication chain from the root key to any subzone. However, the authentication chain does not necessarily need to end in the root; validators can be set up with multiple trust anchors for e.g. top-level or second-level domains. If a trust anchor is part of the chain, the chain will be authenticated.

The security guarantees sustain an end-to-end communication, i.e. resolvers are able

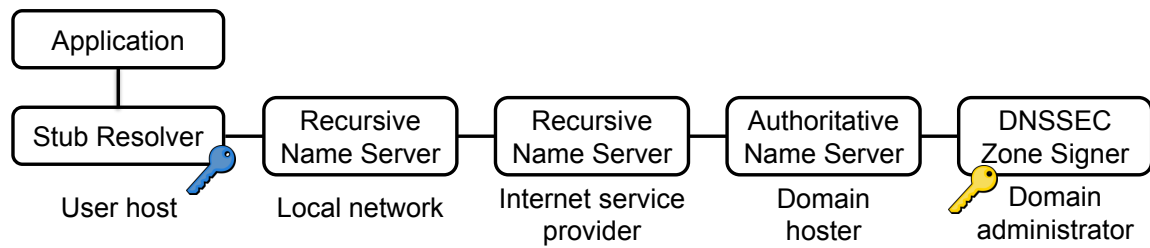


Figure 3.1: DNSSEC provides end-to-end security and protects the path between validator and signer independent of any untrusted components in between.

to authenticate data even if forwarded through untrusted resolvers or name servers (Figure 3.1). DNSSEC guarantees integrity for resource records instead of DNS messages, the latter being transferred hop-by-hop between DNS components. Similarly, DNSSEC authenticates zone data and not name servers.

3.3 Network Protocol

DNSSEC uses the same message format as legacy DNS, but adds new header flags, record types and semantics for DNSSEC-capable actors. The *Extension Mechanisms for DNS* (EDNS, cf. Section 2.6) are mandatory with DNSSEC in order to support DNS messages longer than 512 bytes. This is due to the considerable increase of DNS message sizes when transferring signatures and public keys. DNSSEC-capable name servers must support EDNS messages with a size of at least 1220 bytes [13]. The following header flags have been added with DNSSEC:

- **Checking Disabled (CD)**: tells a recursive name servers to omit validation in this DNS transaction. The recursive name server will return the response independent of whether validation would have succeeded or not. This flag is used 1) to disable upstream validation in case the query sender performs validation on its own anyway (discussed in Section 5.5) and 2) for debugging purposes.
- **Authenticated Data (AD)**: tells a resolver whether the response has been validated by the recursive name server. As the header flags are not secured by DNSSEC, trusting the AD flag is only useful to the resolver when both, the responding name server and the communication channel are trusted. A clear AD flag may indicate that the zone is not signed or that validation is disabled on the recursive name server.
- **DNSSEC OK (DO)**: tells a name server that the sender is able to parse DNSSEC responses. A DNSSEC-capable name server will include DNSSEC signatures in its responses.

The DO flag is a backward compatible method to signal DNSSEC support. Resolvers without DNSSEC support will not set the DO flag and servers without DNSSEC support will ignore it. Thus, legacy implementations will receive DNS messages without signatures. DNSSEC-capable resolvers will receive DNSSEC signatures without having to wait a round-trip time for an additional transaction.

3.3.1 Signature

Signatures are transferred as RRSIG resource records in DNSSEC responses together with the resource records to be authenticated. As described in Section 2.1, a question tuple addresses a set of resource records with an arbitrary number of elements. An RRSIG record authenticates the whole record set of a tuple, i.e. one signature covers all resource records with identical name and type. RRSIG records consist of the following components:

- **Type Covered:** the type of the records that are signed by this RRSIG record.
- **Algorithm:** identifier of cryptosystem used for signing and verification (cf. Section 3.4).
- **Labels:** the number of labels in the original owner name. This field is used to identify resource records which have been synthesized from wildcards (cf. Section 3.7.5).
- **Original TTL:** as DNSSEC caches are still supposed to decrease the TTL value of cached records, the original TTL value is included in the RRSIG record for checking whether the returned TTL is in a valid range.
- **Signature Expiration and Signature Inception:** validity period of the covered record given as absolute time (cf. Section 5.7).
- **Key Tag:** numerical identifier of public key that is used for verification of this signature. The key tag aids the validator to select the appropriate public key efficiently when multiple public keys exist.
- **Signer's Name:** owner name of the DNSKEY record that contains the public key for verification of this signature.
- **Signature:** the actual signature generated by the cryptosystem.

The following example shows an A resource record together with its RRSIG record. The signature has been created with 1024-bit RSA and is transmitted over the network in binary format, but typically presented as Base64 encoding [14].

```

www.example.org.  86400 IN A      93.184.216.119
www.example.org.  86400 IN RRSIG  A 8 3 86400 20140915153249 20140908142533
                               14998 example.org. UjXfp6CbwcCjejsz70s8eq
                               SRDaUE1D0wRB6wefLfqikwInqgCa2/3PnKb05IIQ5
                               iH2SJ0034aZSrEgPK6xW/PK8Q+2GZT79Zoqcqq73c
                               yHhGPSqeHyCbFtLGphigEC857dcAkmHcVPFeQbrhp
                               QQTLUYelKLOsNXelHcdfgGrSQs=

```

3.3.2 Public Key

The corresponding public key can be looked up as DNSKEY record with the information from the RRSIG record. DNSKEY records consist of the following components:

- Flags: define special meaning for public keys, e.g. when a key has been revoked and will be deleted soon.
- Protocol: value is always 3.
- Algorithm: identifier of cryptosystem used for signing and verification (cf. Section 3.4).
- Public Key: the actual public key generated by the cryptosystem. The inner format of the public key differs depending on the algorithm identifier (we show an RSA example in Section 5.8).

A query for the signer's name (`example.org`, IN, DNSKEY) yields three different public keys, which are in use for the `example.org` zone. One of the keys is used to verify the zone signatures and the other two keys are used to establish an authentication chain (cf. Section 3.5). The appropriate public key is selected via the key tag seen in the RRSIG record. The key tag is calculated over the public key in wire format with an algorithm similar to the ones' complement sum that is used e.g. for calculation of IPv4 header checksums. The result is a 16-bit number, which is not necessarily unique; in case of different public keys with identical key tag the validator will have to attempt signature verification with each matching key. However, in most practical cases the key tags differ and allow for efficient selection of the appropriate public key for validation. The following example shows the 1024-bit RSA key with key tag 14998 used to generate the above signature:

```

example.org.  3600 IN DNSKEY  256 3 8 AwEAAcicTQCE7JzF0sn50uWo6ITDJ0j3J+Jrt
                               1H++BMdguFNt5c2JI95UE4k+2Gmsxh7U2q1Y+
                               beOQ+s17x9VARL4HKv+N00GBAfCW14vD6G1mA
                               oWh5wuPlvFLCu/WNjDzTeMM/6+tYIo/Kz8xFu
                               RKcD4JIirv8HAUiW0q09fDSBaknL

```

The integrity and authenticity of the DNSKEY record is proven with a signature over the DNSKEY record set, either self-signed or signed by another trusted key for that zone.

3.3.3 Secure Delegation

For a continuous authentication chain, the parent zone must delegate trust to one of the keys of the child zone. This is accomplished by adding a fingerprint of the trusted child key to the parent zone. Fingerprints are cryptographic hash values, also called message digests, and are transferred as DS (delegation signer) resource records. A delegation that includes a DS record is called a *secure delegation* and indicates that the child zone is signed. DS records consist of the following components:

- Key Tag: identifier of public key that the trust is delegated to.
- Algorithm: identifier of cryptosystem used for signing and verification (cf. Section 3.4).
- Digest Type: identifier of algorithm used to compute the fingerprint.
- Digest: the actual fingerprint. Unlike DNSKEY and RRSIG records, the presentation format is hexadecimal, not Base64.

Note that the key tag and the fingerprint are both hash values but with different purposes. The key tag is 16 bits, computed with a fast checksum algorithm and is not collision resistant. It is used to select the appropriate key from a list of public keys to avoid unnecessary trial and error signature verification. The fingerprint is 160 bits or longer (depending on algorithm), computed with a cryptographic hash algorithm, which is not as fast as checksumming but which provides pre-image and collision resistance. It is used to securely bind a DS record with a DNSKEY record to prevent adversaries from replacing the DNSKEY record unnoticed.

A secure delegation may be comprised of multiple DS records when 1) trust is delegated to multiple coequal keys, 2) different fingerprinting algorithms are used to support implementation diversity, or 3) both. The following example shows a secure delegation to `example.org` with SHA-1 and SHA-256 fingerprints as returned by the `org` name servers. The DS record is placed in the `org` zone and is signed by one of the `org` keys. One RRSIG record authenticates both DS records because they share the same domain name, class and record type.

```
example.org. 86400 IN NS      a.iana-servers.net.
example.org. 86400 IN NS      b.iana-servers.net.
example.org. 86400 IN DS      31589 8 1 7B8370002875DDA781390A8E586C3149384
                                7D9BC
example.org. 86400 IN DS      31589 8 2 3FDC4C11FA3AD3535EA8C1CE3EAF7BFA5CA
                                9AE8A834D98FEE10085CFAEB625AA
example.org. 86400 IN RRSIG   DS 7 2 86400 20140930155830 20140909145830
                                33287 org. AThTEBC/JYbEJcCi1rg6P2w2CDJbr6eXNI
```

```
MBMvjcyNgGqmJRV6h0MggYvj1K42ohE9ms0LJRgpnjuX3
MBbeYkHrJbJpzwfjMy+ffFvKXXG3Lq4xtH+OK6uKatZjY
vr9dNT+ymwyZ5H5wqGgLX1W0Wxj1k7oQV4Cn8y/Q+71XX
TY=
```

A delegation without DS record is an *insecure delegation*. With an insecure delegation, validating resolvers will degrade gracefully to unsigned DNS and continue name resolution without validation. Even if the subzone is actually signed, the lack of a DS record indicates an incomplete authentication chain and therefore DNSSEC validation is not possible. In order to prevent downgrade attacks, the absence of a DS record must be authenticated. Otherwise, a man-in-the-middle attacker could remove the DS record from a secure delegation and thus disable DNSSEC validation. DNSSEC uses a specific technique to prove the non-existence of a DS record, which we explain in Section 3.7.

Note that while DS records are signed in a secure delegation, the NS records are not. Glue records are not signed either. This is due to the design choice of signing authoritative data only, where NS and glue records are not considered authoritative in a delegation.

3.4 Algorithms

The cryptosystem used for signing and verification consists of a digital signature scheme and a cryptographic hash function. The hash function compresses the resource record set to a fixed-length fingerprint, which is then signed. This fingerprinting of record sets is an integral part of the signing and verification process. It is unrelated to the fingerprints of public keys in DS records, thus different algorithms can be used.

With DNSSEC, the zone administrator chooses from multiple specified algorithms and—depending on the algorithm—chooses the key size. The list of currently specified signing algorithms includes RSA, DSA, ECC-GOST and ECDSA. Most algorithms use SHA-1 or one of the SHA-2 variants as hash function. RSA with MD5 is specified but shall no longer be used due to known weaknesses of the MD5 hash function [107]. ECC-GOST is an elliptic curve signing scheme standardized as Russian state standard (*gosudarstvennyy standart*) GOST R 34.10-2001, which uses 256-bit keys and the GOST R 34.11-94 hash function [40]. Implementations do not need to support all algorithms. Support for RSA with SHA-1 is mandatory; support for RSA with other hash functions and ECDSA are recommended; other algorithms are optional [107]. Validating resolvers treat DNSSEC zones that are signed with an unsupported algorithm as if they were insecure, i.e. not signed. Downgrade attacks are not possible because the authenticated DS record securely indicates which algorithm the DNSSEC zone is supposed to use.

The fingerprinting algorithm used in DS records is exchangeable as well. Support for SHA-1 and SHA-256 are mandatory [52]; GOST R 34.11-94 and SHA-384 are optional.

3.5 Key Management

The designated—though not mandatory—scheme is to use two different key pairs for each DNSSEC zone: the *zone signing key* (ZSK) signs the zone data, i.e. is used to create RRSIG records for all resource records. The *key signing key* (KSK) signs the ZSK, i.e. is used to create the RRSIG record for the DNSKEY record set only.

3.5.1 Key Schemes

The idea of separating KSK and ZSK is to replace the keys at different intervals. A short-lived ZSK may justify a reduced cryptographic strength with shorter signatures and thus shorter DNS messages, as well as reducing the computation time for signature generation and verification. In this scheme, the KSK is longer and is replaced less frequently than the ZSK to avoid the operational overhead of communicating the updated fingerprint with the parent.

Operators can compose their own key management scheme with an arbitrary number of keys. The number of DNSSEC keys for a zone is limited only by practical constraints, as each active public key must be in the set of DNSKEY records. If the set is signed by a key that is part of an authentication chain up to a trust anchor, then validating resolvers will trust the whole DNSKEY record set for that zone. It is, however, not possible to divide the DNSKEY records of a zone to different DNSKEY sets, or to retrieve a single DNSKEY record from a name server; either the whole DNSKEY set is transferred or the DNS response is truncated.

3.5.2 Key Rollover

The process of replacing an old key with a new one is called *key rollover*. Due to caching, keys cannot be replaced at an instant of time and instead must be phased out. This applies to KSKs as well as to ZSKs. Replacing the ZSK with a ZSK_{new} requires to sign the zone with ZSK_{new} and to sign ZSK_{new} with the KSK. Replacing the KSK with a KSK_{new} requires to sign the ZSK with KSK_{new} and to update the DS record in the parent zone. During the transition, both the old and new key are part of the DNSKEY set.

It is up to the operator of a DNSSEC zone when to schedule a key rollover, e.g. periodically without specific cause or when a higher cryptographic strength is desired. Keys have no validity period in DNSSEC and thus do not expire. However, signatures expire at given points in time and have to be renewed regularly. This includes the signatures of DS or DNSKEY records in an authentication chain. Keys remain valid as long as their authentication chain is actively renewed. When an operator replaces a key in the authentication chain, the previous key becomes invalid once the previous DS and DNSKEY signatures have expired.

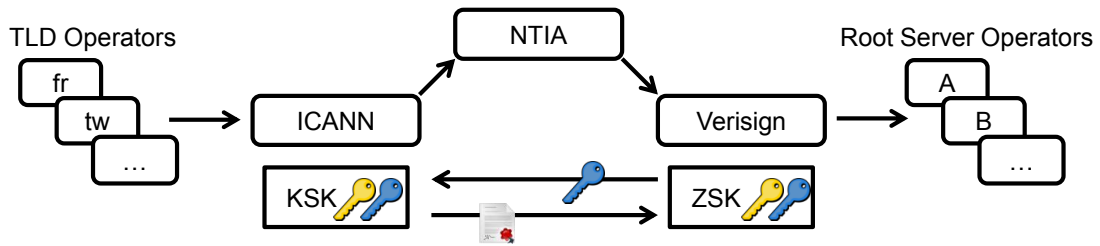


Figure 3.2: Actors involved in root key management.

3.5.3 Update of Authentication Chain

DNSSEC-capable resolvers use a configurable list of trust anchors, which typically includes the root KSK but may also include other KSKs. Should any of these KSKs be rolled over, the resolvers will have to update their trust anchor list. An update mechanism is specified in RFC 5011 [119] to automate this process. Resolvers that support RFC 5011 periodically refresh the DNSKEY set for each locally configured trust anchor. If a key rollover is in place, the resolver will learn the new key, which is signed with the established key during transition. The automatic update mechanism requires resolvers to refresh trust anchors regularly, at least every 15 days. The mechanism is thus suitable for always-on scenarios but not for end-user devices, which are sporadically turned off for longer periods.

Replacing the KSK requires to update the DS record and thus involves interaction of two different authorities. RFC 5011 is not applicable for updating DS records in secure delegations. Originally, DNSSEC did not define a standardized update mechanism. Registrars usually offer a web interface for manual update of the secure delegation. IETF contributors proposed an automatic mechanism for updating secure delegations in RFC 7344 [78]. The idea is to make the parent probe the child authority regularly for any anticipated public key change. With the update mechanisms in RFC 5011 and RFC 7344, key rollovers can be performed without manual interaction.

3.5.4 Root Key Management

The root zone is signed with a regular KSK/ZSK scheme. The key ownership is shown in Figure 3.2 (cf. the root zone management in Section 2.7.2). The root KSK is owned by ICANN and the root ZSK is owned by Verisign. The NTIA oversees the root key operations but is not directly involved in key ownership.

Root KSK

The root KSK is an 2048-bit RSA key and has been created on June 16, 2010. Though a key rollover was intended to be performed within 5 years [86], it is still in planning stage and has not been scheduled as of October 2014. Once a KSK rollover is conducted, validators will be

able to use the RFC 5011 mechanism to update the root KSK securely. In addition, ICANN has specified an out-of-band root KSK publication channel via HTTP/HTTPS. ICANN signs the root KSK with two long-term bootstrapping keys: 1) an S/MIME signature is generated with a 2048-bit RSA key, which will expire in 2029, and 2) an OpenPGP signature is generated with a 1024-bit DSA key, which has no expiry date. The HTTP/HTTPS channel can be used for bootstrapping DNSSEC devices that are not always-on, e.g. broadband routers stored in shelves or end-user devices turned off for longer periods.

ICANN organizes a quarterly KSK ceremony where the root ZSK is signed with the root KSK. The KSK ceremonies are semi-public events, which ICANN publishes as video stream and which are accessible for invited external witnesses. ICANN publishes the procedures and software used in the KSK ceremonies for community review [66]. The KSK ceremonies are held alternately at two U.S.-based colocation centers near Los Angeles and Washington, D.C.¹ The root KSK is stored logically on hardware security modules (HSM), which are stored physically in a safe at each location. The data to be signed is transferred to the HSM with a diskless laptop computer, isolated by air gap from any computer network. There is a total of four HSMs with identical configuration to account for hardware failures. In order to activate an HSM, 3 out of a set of 7 smartcards must be inserted. The smartcards are stored in tamper-evident plastic bags in small locked boxes next to the HSMs. The boxes are unlocked with physical keys; these 14 physical keys (7 for each location) are handed over to selected community representatives, who bring the physical keys to the KSK ceremonies. If one of the physical keys is lost or if a community representative is absent without notice, ICANN will drill the lock out to recover the smartcard. The community representatives witness the execution of the KSK ceremony but are not empowered to take the root KSK with them. All copies of the root KSK and the smartcards required to access the KSK are supposed to remain at the two U.S. sites.

Apart from the smartcards for normal HSM operation, there are also backup smartcards to initialize a factory-default HSM with the root KSK. The backups are encrypted with another set of 7 smartcards, out of which 5 are necessary to decrypt the private root KSK. These 7 smartcards are distributed in tamper-evident bags to community representatives who are supposed to store them at safe locations under their control, e.g. a safe deposit box in a bank. Again, the role of the community representatives is not to keep the root KSK but the key for accessing the backup—the actual root KSK remains at ICANN facilities.

Root ZSK

The root ZSK is a 1024-bit RSA key and is replaced quarterly. The root zone data is signed with the root ZSK daily at Verisign facilities. The root ZSK is stored on an HSM attached to Verisign's production network. The signing procedure requires the participation

¹Equinix LA3 in El Segundo, California, and Terremark NAP in Culpeper, Virginia.

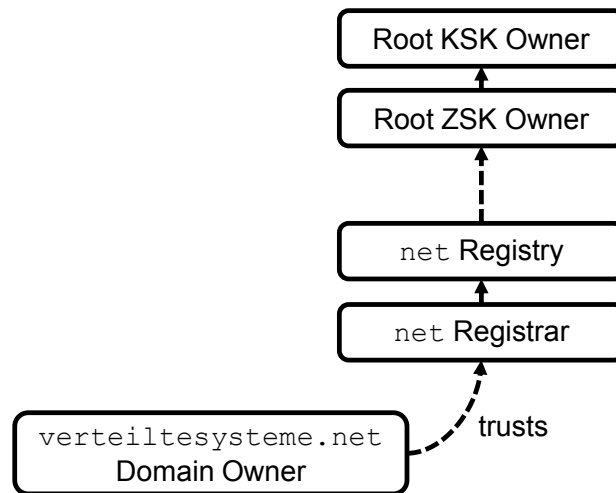


Figure 3.3: Trusted authorities as seen from the domain owner of `verteiltesysteme.net`.

of 2 Verisign staff members or 1 staff member interacting with an automated process. Verisign has published a practice statement on the root ZSK [99] but does not document the ZSK usage as publicly as ICANN does for the root KSK. The secure handling of the root ZSK is thus not transparent to the public community, though the root ZSK is as powerful as the root KSK and allows to sign any DNSSEC domain or delegation.

3.6 Public-Key Infrastructure

DNSSEC adds a public-key infrastructure (PKI) to the Domain Name System. The PKI is used internally for secure DNS responses but DNSSEC can also be leveraged as secure storage for public keys or certificate fingerprints. This way DNSSEC serves as secure bootstrapping method of key material for external applications, e.g. for HTTPS connections or SSH host fingerprints.

3.6.1 Trust Model

DNSSEC uses a hierarchical trust model, mapped onto the hierarchical name space. The root authority has full power over the root domain, i.e. over the whole namespace. Trust for specific top-level domains is delegated by the root authority to the respective TLD authorities, which delegate trust further down. From the point of view of a domain owner, each parent authority up to the root must be trusted, as each of them has full power over the namespace of the domain owner (Figure 3.3). However, trust of child authorities is limited to their respective subdomain, e.g. `org` cannot create signatures for the `net` TLD. In case the registry/registrar/registrant model applies for domain registrations (cf. Section 2.7.3), the domain owner must trust both, the registry for setting up a correct secure delegation

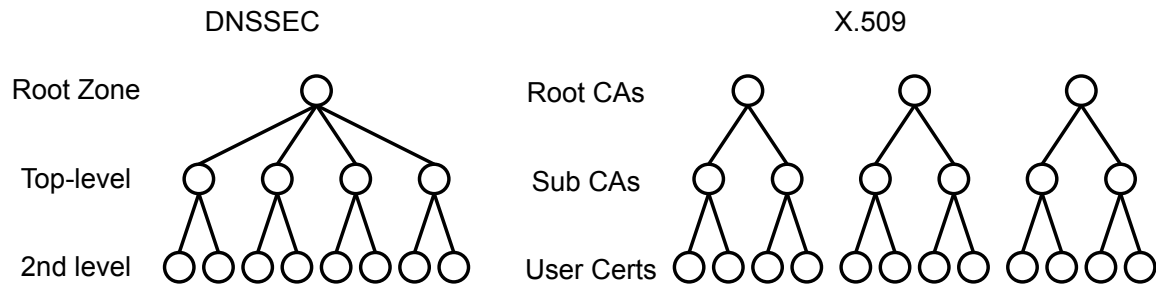


Figure 3.4: Comparison of DNSSEC and X.509 trust models.

and the registrar for submitting the correct public key fingerprint to the registry.

The hierarchical trust model of DNSSEC is fundamentally different from the X.509 trust model (Figure 3.4). X.509 is the prevalent PKI that is used for HTTPS connections in web browsers and for many other TLS certificate-based applications. With X.509, there are several coequal root certificate authorities (CAs). The number of root CAs is unlimited, but the client must store the certificate of each root CA which it trusts. In practice, several dozen root CA certificates are shipped with the operating system and applications, chosen by the software vendors. Root CAs delegate trust to sub-CAs, which are usually within the administrative boundaries of the root CA or one of its sub-organizations. Each CA certificate has full power and can issue client certificates for any domain name or IP address. The hierarchy of X.509 thus does not restrict the influence of certificate authorities. Having unrestricted CA certificates is detrimental when e.g. a national NIC is meant to issue certificates only for a country-code TLD or when a user installs an enterprise CA certificate to authorize only the company VPN or servers. As CA and sub-CA certificates are not restricted to the domains and IP addresses of the authority, the user must fully trust the CA and the CA must employ potentially higher security measures to protect adequately from certificate theft or misuse. The X.509 name constraints extension [32] was supposed to restrict CA certificates to specific domains. However, implementation support is scarce after organizational issues during standardization [30], thus name constraints will be ignored by many applications.

3.6.2 DNS-based Authentication of Named Entities

The DNS-based Authentication of Named Entities (DANE) is a set of related DNSSEC extensions, which allow to bind certificates or public keys to domain names. One of the DANE proposals specifies the association of TLS certificates with domain names. When attempting to connect to a TLS server, a DANE-capable TLS client (e.g. web browser) will look up the TLSA resource record under a specific domain name constructed from the server name, port number and transport protocol. The TLSA record pinpoints the TLS server to a specific certificate or indicates that the server certificate must be issued by a

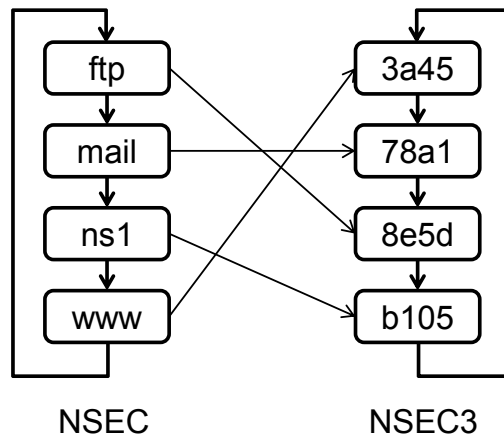


Figure 3.5: NSEC composes an ordered chain of names, whereas NSEC3 composes an ordered chain of hash values of names.

specific certificate authority. TLSA records are either comprised of the full TLS certificate or just a fingerprint of the TLS certificate. The functional principle of a TLSA record is comparable to a DS record, except that DS authorizes a DNSSEC public key, whereas TLSA authorizes a TLS certificate for an external application. Depending on how it is used, DANE TLSA either supplements X.509 certificates with constraints for increased security, or substitutes X.509 entirely with the DNSSEC PKI. Other use cases proposed for DANE are the association of S/MIME certificates or OpenPGP public keys to domains.

3.7 Authenticated Denial of Existence

Authenticated denial of existence is a technique used by DNSSEC to prove the non-existence of a resource record. Securely denying the existence of a record is required whenever the name server returns a negative response. This is the case 1) when the queried domain name does not exist, or 2) when a resource record with the requested type does not exist for an otherwise existing domain name. Negative responses need to be authenticated to prevent denial of service attacks on actually existing domain names. There are a few other cases in which the server needs to deny the existence of a resource record, e.g. when a DS record is absent from a delegation because the subzone is not signed. If the absence of a DS record was not authenticated, an attacker could disable DNSSEC validation for a delegated subzone.

3.7.1 NSEC

Given that DNSSEC signs resource records and not DNS messages, the absence of a resource record cannot be authenticated directly. DNSSEC introduces a new record type

called NSEC to deny the existence of resource records indirectly. In order to cope with offline signing (Section 3.2), NSEC records are in general not synthesized from DNS queries. Instead, NSEC records must be created a priori based on the contents of the DNS zone. This is achieved by creating an ordered chain of all existing domain names (left column of Figure 3.5). NSEC proves the non-existence of a name x by returning an enclosing NSEC range (r_1, r_2) . It holds that $r_1 < x < r_2$ under a definite canonical ordering of names. An NSEC record proves the existence of r_1 and r_2 , and indirectly the non-existence of any names in between them in a canonical order. r_1 is represented as owner name (domain name) of the NSEC record, and the data field of the NSEC record consists of the following components:

- Next Domain Name: r_2 .
- Type Bit Maps: list of record types that exist for r_1 .

A name server returning a negative response needs to look up the pre-generated NSEC record and its signature in the DNS zone and return it to the client. The NSEC record is signed and its correctness can be validated by DNSSEC just as for existing domain names. The following example shows an NSEC record which proves the existence of names `abc` and `foo` and thus disproves the existence of names in between like `bar` or `def`:

```
abc. 3600 IN NSEC foo. NS DS RRSIG NSEC
```

The NSEC record indicates that there are resource records for the types NS, DS, RRSIG and NSEC under the domain name `abc`. The existence of other record types is hence disproved for `abc`.

3.7.2 Zone Enumeration

The chain of NSEC records enables the enumeration of zone data: clients can retrieve a list of all existing names of a DNS zone. This can be achieved by querying for the NSEC record type of the next domain name r_2 repeatedly until the whole NSEC chain has been retrieved. Even if the server refused to respond to NSEC queries, the client could query the server for an increment of the next domain name in canonical order, e.g. (`foo0`, IN, A) instead of (`foo`, IN, NSEC) to retrieve the next NSEC record. With knowledge of all NSEC records, the client can query the server for each existing record type of each existing domain name to retrieve a copy of the DNS zone.

Although the Domain Name System is a publicly accessible database, several major operators are reluctant to publish their DNS zones as a whole. In particular, TLD operators argue that they are obligated to hide the zone data for legal or policy reasons. Nominet, the operator of `uk`, explained in 2004 that denying public access to the `uk` zone file is in the best interest of their local community, and that some kind of technical enforcement is needed

to protect their database right granted under European Union law [36,122]. DENIC, the operator of `de`, explained in 2004 that public access to the `de` zone file would be in conflict with Germany's Federal Data Protection Act [108,110].

3.7.3 NSEC3

The later introduced NSEC3 record type attempts to obstruct zone enumeration by using hash values of names in the non-existence proof [81]. In addition to providing an authenticated denial of existence, NSEC3 adds the privacy goal of not disclosing existing names. NSEC3 proves the non-existence of a name x by returning an enclosing NSEC3 range $(h(r_1), h(r_2))$. h is a hash function and it applies $h(r_1) < h(x) < h(r_2)$. As shown in the right column of Figure 3.5, an NSEC3 chain is ordered by the resulting hash values which does not necessarily equal the order of domain names. The NSEC3 hash function h is defined [81] as

$$h(x, s, 0) = f(x\|s) \quad (3.1)$$

$$h(x, s, i) = f(h(x, s, i-1)\|s), \text{ for } i > 0 \quad (3.2)$$

where x is the input domain name, s an arbitrary salt value, i the number of additional hash iterations, f an underlying hash function and $\|$ the concatenation operator. Similar to NSEC, $h(r_1)$ is represented as owner name (domain name) and $h(r_2)$ is represented in the data field of the NSEC3 record, which is comprised of the following components:

- Hash Algorithm: identifier of underlying hash function f .
- Flags: indicates whether the opt-out feature is used (Section 3.7.4).
- Iterations: i .
- Salt Length: length of the salt.
- Salt: s in hexadecimal format.
- Hash Length: length of the next hashed domain name.
- Next Hashed Domain Name: $h(r_2)$.
- Type Bit Maps: list of record types that exist for r_1 .

The hash values are represented in Base32hex [69], which is a variant of Base32 with a modified alphabet with characters from 0 to 9 and A to V. The following example shows an NSEC3 record in the `ch` top-level domain with 2 iterations and 4 bytes salt D46AE672:

```
E250EFE3M744DASNQHA71LR53TUA5P0K.ch. 3600 IN NSEC3 1 0 2 D46AE672
E252TNL2N9T2FADHHK850MRFKU2NAT16 NS
```

With the parameters from the NSEC3 record, a validating resolver can compute the hash value $h(x)$ of the query name and check whether $h(r_1) < h(x) < h(r_2)$. The purpose of the iterations is to increase the hashing workload for attackers who attempt to reverse hash values to cleartext names. However, it also increases the workload for name servers and resolvers. Note that i is defined as *additional* iterations, i.e. $i = 0$ implies one hash operation and $i = 10$ implies 11 hash operations.

The purpose of the salt is to prevent recurring attacks with pre-computed lists of hash values, e.g. rainbow tables. Whenever the zone administrator changes the salt, all NSEC3 hash values will change and pre-computed tables will become obsolete. From the point of view of the resolver, each NSEC3 record can use a different salt because each NSEC3 record is self-contained and can be validated on its own. From the point of view of the authoritative name server, the server needs to know the hashing parameters for computation of $h(x)$ before the matching NSEC3 record is looked up. Thus, the same salt value and iteration count are used per zone, which are stored in a dedicated NSEC3PARAM record. As noted by Bau and Mitchell [19], the salt does not slow down a one-time hash computation attack because the same salt value is used for all NSEC3 records of a chain. This is a different use of salt than for example in password databases, in which each password is hashed with a different salt to increase the necessary computing time for attackers.

The domain name to be hashed is given as fully qualified domain name in wire format (binary representation) and is thus unique. Therefore the name `www` will result in a different hash value below `example.com` than below `example.org`, even when both zones use the same salt value and iteration count.

Underlying Hash Function

The security of NSEC3 is based on the preimage resistance of the underlying hash function f . NSEC3 uses an algorithm identifier to allow for future use of other hash functions, but currently the only specified function is SHA-1. So far, there are no known preimage attacks on SHA-1. Stevens estimates that SHA-1 collisions can be computed with 2^{61} SHA-1 operations [117], which is well below the 2^{80} required operations with a naive birthday attack (cf. Section 5.2.1). However, collision attacks do not compromise the security goals of NSEC3. Collisions during DNSSEC signing will be detected by the signer, which can restart the signing process with a different salt. Collisions during runtime will be detected by the server, which will return a server error to the client. Runtime collisions can be triggered only with specially-crafted query names. Resolving such a name will lead to a validation failure at the resolver, which is harmless because such a crafted name is most likely not in use anyway and thus superfluous to the client.

The SHA-1 hash function compresses input data of any length $< 2^{61}$ bytes to a 20 bytes hash value [42]. Before hashing, the input data is appended with an 8 bytes data length field

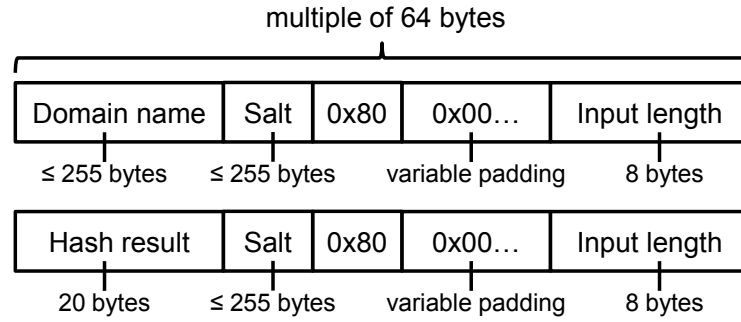


Figure 3.6: SHA-1 input for the initial (upper row) and additional (lower row) NSEC3 iterations.

and 1 or more padding bytes to align the input data to a multiple of 64 bytes (Figure 3.6). The data is then processed in blocks of 64 bytes, each additional block increasing the hashing workload linearly. The number of SHA-1 blocks to be computed is given as

$$b = \left\lceil \frac{\text{len}(n) + \text{len}(s) + 9}{64} \right\rceil + i \cdot \left\lceil \frac{20 + \text{len}(s) + 9}{64} \right\rceil \quad (3.3)$$

where the function len returns the length of a variable in bytes.

If the length of the domain name and salt are ≤ 55 bytes in wire format, they will fit into one SHA-1 block during the initial iteration. If the salt is ≤ 35 bytes, each additional iteration will also fit into one block. This applies to the majority of domain names in common use, hence $b = 1 + i$. An attacker attempting to exhaust CPU time on the server can send queries for non-existent names with the maximum length of 255 bytes [94] but cannot control the salt, hence $b = 5 + i$.

3.7.4 Opt-Out

In the usual DNSSEC setup, there must be one NSEC or NSEC3 record for each existing domain name. This is a considerable overhead for large zones, e.g. for major top-level domains. Having NSEC or NSEC3 records for each domain name is of little benefit when the domain name consists solely of an insecure delegation, which is not signed anyway (cf. Section 3.3.3). For example, in March 2015 `com` served 116 million second-level domains but just 430k of them were signed.²

NSEC3 specifies an opt-out feature, which allows zone administrators to omit NSEC3 records for insecure delegations. With opt-out, NSEC3 records must be created only for DNSSEC-signed domain names, i.e. secure delegations or authoritative zone data. Thus, the zone administrator of `com` saves resources for signing and serving more than 115 million resource records. Opt-out has been proposed for NSEC records [15] but not specified as

²According to numbers on verisigninc.com and verisignlabs.com.

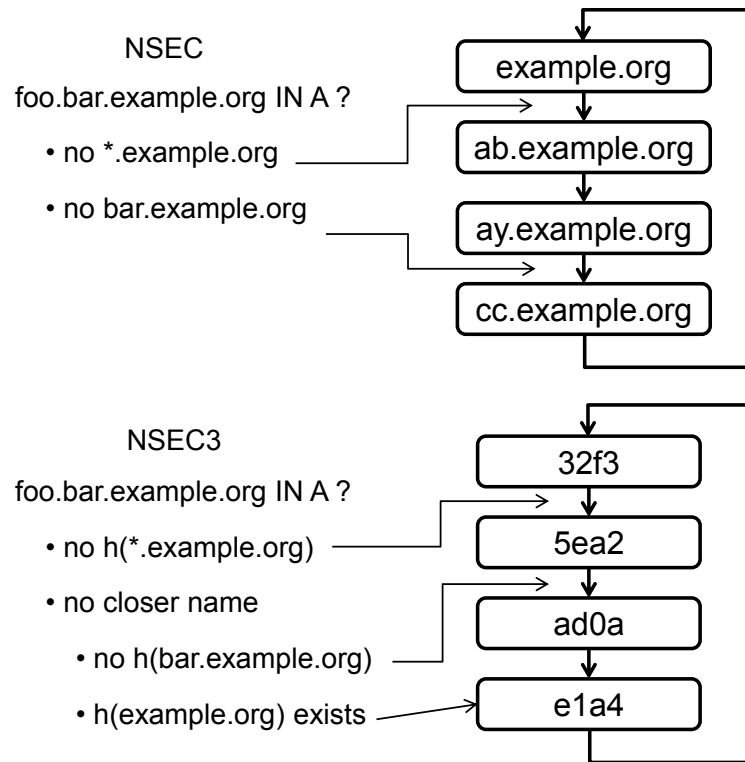


Figure 3.7: NSEC/NSEC3 proves that a matching wildcard does not exist and that a closer match does not exist either.

it changes the denial of existence semantics slightly and is not compatible to the original NSEC specification. It is an optional feature of NSEC3 and can be used at the discretion of the zone administrator.

3.7.5 Wildcards

In legacy DNS, the support of wildcards (Section 2.1.3) is straightforward: when a wildcard matches, the authoritative name server returns a synthesized resource record instead of a negative response. The support for wildcards with DNSSEC requires the use of authenticated denial of existence. In a DNSSEC wildcard response, the signature is pre-generated over a special name. For example, a query for `foo.bar.example.org` may be caught by a wildcard `*.example.org`. An RRSIG record signed over the name `*.example.org` proves that the wildcard response is authentic. In addition, the wildcard response contains a proof that no closer match exists, i.e. there is no wildcard `*.bar.example.org` and there is no record for `foo.bar.example.org`.

When wildcards are not used in a zone, the authoritative name server must securely deny the existence of a wildcard in addition of denying the question tuple. A negative response thus contains 1) a proof that a wildcard does not exist and 2) a proof that a

| Key length (bit) | Max i |
|------------------|---------|
| 1024 | 150 |
| 2048 | 500 |
| 4096 | 2500 |

Table 3.1: Max iteration count subject to key length.

closer match (closer wildcard or exact query name) does not exist. With NSEC, these two proofs usually require two separate NSEC records (e.g. non-existence of `*.example.org` and non-existence of `bar.example.org`), plus RRSIG records for each. With NSEC3, up to three NSEC3 records are required, plus RRSIG records for each. This is shown in an example in Figure 3.7. With NSEC, the resolver can infer the non-existence of `foo.bar.example.org` by the proof of the non-existence of `bar.example.org`. This is not possible with NSEC3 because the hashing masquerades the hierarchical depth of names. Instead, the server proves that the *closest encloser* `h(example.org)` exists and that the closer `h(bar.example.org)` does not exist. Note that the general support of wildcards in DNSSEC requires larger negative answers with two NSEC records and three NSEC3 records, even if a zone administrator does not use wildcards at all.

3.7.6 Costs

NSEC and NSEC3 have different costs, which can be quantified as CPU time, network message size and zone size. NSEC3 requires the authoritative name server to hash the non-existent query name to find the matching NSEC3 record. Although hashing is in general very fast compared to asymmetric signing operations³, it is a significant overhead for name servers with pre-generated NSEC3 signatures [109]. The actual overhead depends on the number of negative responses in the name server traffic and the iteration count. Resolvers hash the query name to validate the non-existence proof but here the hashing overhead is minor compared to the much higher cost for signature verification. RFC 5155 [81] limits the iteration count the zone administrator can choose in order to keep the hashing and signature verification costs reasonable for validators. The limitation has been chosen based on the computation time of RSA operations (Table 3.1). RSA uses quite long keys; the elliptic curve ciphers ECDSA or ECC-GOST use key sizes < 1024 bits and are thus bound to $i = 150$.

Regarding message size, NSEC3 responses are larger than NSEC responses in most cases. This is because 1) NSEC3 hash values are longer than cleartext domain names on average, 2) an NSEC3 record needs more space because it uses additional fields like iteration count

³OpenSSL performance on one Intel X5650 core: 3.3M SHA-1 hashing operations per second (1 block input, no additional iterations), 3k RSA 1024-bit signatures per second and 53k RSA verifications per second.

and salt, and 3) NSEC3 responses usually contain three records (cf. Section 3.7.5) whereas NSEC responses contain two records. Typical negative response sizes⁴ are 655–665 bytes with NSEC but 967–1000 bytes with NSEC3. The message overhead has a significant impact when exceeding network packet limitations because handling fragmentation or truncation multiplies the latency and resource effort of a DNS transaction. The zone size on disk and in memory of authoritative name servers increases also due to reasons No. 1 and 2 as stated above.

Schaeffer compared the maximum server capacity with NSEC and NSEC3 in a laboratory setup [109]. The maximum number of negative responses sent by the NSD name server decreased from 47k responses/s with NSEC to 25k responses/s with NSEC3 and $i = 1$ (19k with $i = 15$). NSEC3 is more expensive than NSEC in terms of quantifiable resources. The server operator thus needs to consider the NSEC3 overhead when provisioning the authoritative name servers to ensure a decent quality of service.

3.7.7 Minimally Covering Records

Instead of returning pre-generated NSEC records, a name server can generate minimally covering NSEC records on-the-fly [131]. For a non-existent query name x , the authoritative name server generates a predecessor and successor name in canonical order with the function

$$(x_{-1}, x_{+1}) = \epsilon(x) \tag{3.4}$$

and returns an on-the-fly signed NSEC record covering $x_{-1} < x < x_{+1}$. The principle is also compatible with NSEC3, in which case the NSEC3 record covers $h(x) - 1 < h(x) < h(x) + 1$. Minimally covering records prevent zone enumeration entirely. It remains the possibility of probing for existing names by sending queries for each candidate name to the server, but this naive method is practically infeasible for zone enumeration.

On-the-fly generation of NSEC or NSEC3 records requires online signing with public-key cryptography, which is several orders of magnitude slower than SHA-1 hashing. It opens an attack vector for CPU exhaustion attacks and furthermore requires access to the private key on all authoritative name servers [131]. Minimally covering records are thus only suitable for non-critical servers with low traffic volumes.

⁴DNSSEC responses for “`_.us`”, “`_.se`”, “`_.de`” and “`_.com`”, as of October 2014.

Part II

Security Analysis

CHAPTER 4

Attacker Model

In this chapter, we classify types of attackers based on their capabilities and how they are involved in the name resolution of a target host and user. We use the terminology of on-path and off-path attackers, as shown in Figure 4.1.

An *off-path attacker* is not directly involved in the domain name lookup of his attack target. The off-path attacker is neither queried during a legitimate name resolution nor able to see the DNS messages exchanged during that name resolution. However, the attacker has means to initiate name resolution on the target host. Examples include the target running a publicly accessible open resolver or triggering name resolution via the target's application logic, e.g. pointing the user to a web page with a crafted image URL. The attacker can send IP packets to the target host, though the host may be protected by a stateful firewall, e.g. due to the use of Network Address Translation. Off-path attackers are capable of performing IP spoofing, i.e. send DNS messages with an arbitrary IP source address claiming to originate from a legitimate name server. Although many carriers use ingress filtering to drop spoofed IP packets near their origin [44], networks without restriction of IP spoofing are still widely available. When there is a race condition between a spoofed DNS message and a legitimate one, an off-path attacker can attempt a denial of service attack to delay or suppress legitimate DNS messages.

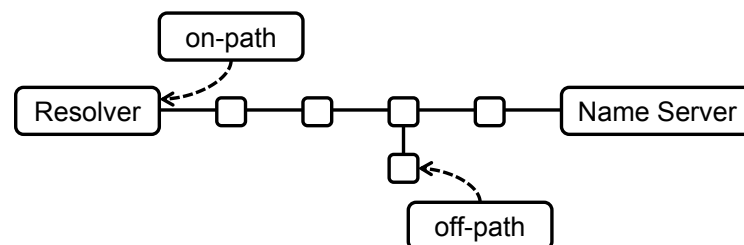


Figure 4.1: On-path attackers are listening on the wire or air. Off-path attackers do not.

An *on-path attacker* has the capabilities of the off-path attacker, and has the additional ability of monitoring the DNS messages of the target's name resolution. This may comprise the whole network path or just the portion in a specific vantage point. On-path attackers are capable of injecting new messages into the communication path and can perform IP spoofing.

Depending on whether an attacker can alter DNS messages in-flight, Duan et al. [41] introduced the distinction between on-path attackers and *in-path attackers*. With this terminology, an on-path attacker uses for example a network tap for passive monitoring of DNS and other network traffic. Legitimate DNS messages cannot be altered or dropped, other than with the denial of service capabilities that the off-path attacker already has. An in-path attacker is the equivalent of a man-in-the-middle attacker, who has the ability to alter or remove transmitted messages. An in-path attacker uses for example a middlebox that inspects each DNS message before forwarding it.

CHAPTER 5

Attack Methods

We now analyze different types of attacks on the Domain Name System and discuss which attacker classes are able to conduct them. The objective of most of the attacks discussed here is to return false resource records, e.g. in order to divert the user application to the wrong server. DNSSEC is meant to thwart attacks against data integrity and authenticity.

Attacks that do not affect the security goals of DNSSEC are not in our scope. This includes DNS amplification attacks [77, 123], which are abusing the Domain Name System to attack the availability of another system, but not the DNS itself. Similarly, we do not discuss the (lack of) privacy of DNSSEC queries and instead refer to other work [54]. However, we do consider the privacy of NSEC3 responses because this was the explicit design goal of NSEC3.

5.1 Spoofing Attack

The standard textbook example for a DNS spoofing attack is a UDP-based DNS response with a forged IP source address that looks like it originates from the queried name server. DNS spoofing is trivially possible for on-path attackers: the attacker inspects the contents of the DNS query and sends a bogus DNS response, which matches the query name and the other parameters. Even if the on-path attacker cannot suppress the legitimate DNS response from the authentic name server, the spoofed DNS response will arrive earlier because the attacker is topologically closer to the user host than the authentic sender is. As resolver implementations ignore subsequent DNS responses, the attack is almost guaranteed to succeed.

DNS spoofing works also for off-path attackers but the attacker will have to infer or guess query parameters in order to forge a matching DNS response. This includes the time at which the query has been sent, the question tuple and the destination name server address that the query has been sent to. Furthermore, the attacker will have to guess the

transaction ID and UDP source port. Resolvers choose these parameters randomly with the purpose of adding entropy to DNS messages and thus to increase the resilience against off-path attackers. In this context, entropy is the amount of unpredictable randomness. The size of both parameters is 16 bits each; the transaction ID can be chosen freely, whereas about 14–16 bits of the UDP source port number can be chosen freely (implementation-specific, some port numbers are reserved). An off-path attacker attempting to guess these numbers will have a probability of

$$\frac{1}{2^{16+14}} \approx 9.3132 \cdot 10^{-10} \quad (5.1)$$

to succeed with a single spoofed DNS response. The attacker can attempt to send a batch of DNS responses with different guesses to increase the success probability. The time window for sending spoofed DNS responses is from when the user host has sent the DNS query until the authentic DNS response arrives (or until the response has timed out). In general, the success probability of batch spoofing is

$$\frac{T_p \cdot \Delta t}{2^S} \quad (5.2)$$

where T_p is the packet transmission rate between attacker and user host, Δt is the time window for sending spoofed responses and S is the amount of entropy in bit.

Herzberg and Shulman [56] suggested a spoofing attack, which exploits IPv4 fragmentation and reduces the amount of entropy that an attacker has to guess. With proper IP fragmentation and timing, it suffices to guess the 16-bit ID field in the IPv4 header of a fragmented DNS message instead of the UDP source port and transaction ID. The fragmentation attack is more difficult to conduct than regular spoofing because additional conditions concerning fragmentation and message order must be met, but has been successfully demonstrated under laboratory conditions [58].

Adding Entropy

Early DNS implementations were using sequential transaction IDs or predictable random numbers. The source port was reused to conserve networking resources, e.g. socket memory. Without these entropy sources, DNS spoofing is almost as easy for off-path attackers as for on-path attackers. Later implementations chose these parameters randomly to protect from off-path DNS spoofing.

Another method for adding entropy is randomly mixed case in query names. Name servers ignore character case in domain names but are supposed to respond with the same character casing as in the query [94]. This can be used to add n bit entropy to the DNS query, where n is the number of characters a to z in the query name [35].

These measures have in common that they do not change the DNS message format, and hence increase resilience against off-path spoofing with unilateral deployment. Furthermore, they do not protect at all against on-path attackers. Adding entropy is thus a quick fix against off-path attackers until DNSSEC or another cryptographic integrity protection become widely available.

DNSSEC

DNSSEC protects from plain DNS spoofing, both from on-path and off-path attackers. An attacker must either guess a valid signature or break the private key of the zone administrator. Guessing a signature of 256 or more bits is practically infeasible, even in high-bandwidth networks. The effort for breaking the private key depends on the cryptosystem used and the key length—a case study is given in Section 5.8.

5.2 Spoofing Attack with Multiple Queries

The above probability model applies when the spoofed responses are matched against one query. The success probability increases vastly when the resolver sends out multiple queries concurrently with different transaction IDs and port numbers but with the same question tuple. Such an attack is known in the literature as birthday attack, and has been applied on DNS for example in [55, 60, 118, 120]. In this section, we suggest another name because the birthday problem is not the appropriate mathematical model for this type of DNS attack.

5.2.1 Birthday Attack

The birthday attack is the exploitation of an underlying birthday problem (also known as birthday paradox) as attack vector. The birthday problem can be figuratively described as following: given 23 people in a room, what is the probability that at least one pair of two people share the same birthday? Years are disregarded and a uniform birthday distribution is assumed. The answer is more than 50%, which appears counterintuitive. The probability of one matching pair in a set of $n = 23$ random elements is significantly higher than the probability of one specific element matching one of n random elements. To calculate the probability, we first sum the number of non-matching combinations: there are $m = 365$ choices for the first element (any value because we have no pair yet), $m - 1$ choices for the second element (all but the first chosen value), $m - 2$ choices for the third element (all but the first two chosen values), \dots , and $m - n + 1$ choices for the n -th element. Divided by the number of all possible combinations m^n , the probability of a successful match is

$$1 - \frac{m \cdot (m - 1) \cdot (m - 2) \dots (m - n + 1)}{m^n} \quad (5.3)$$

$$= 1 - \frac{m!/(m-n)!}{m^n} = 1 - \frac{m!}{m^n \cdot (m-n)!}. \quad (5.4)$$

Thus, in the above example the probability is

$$1 - \frac{365!}{365^{23} \cdot (365 - 23)!} \approx 0.5073. \quad (5.5)$$

The birthday attack can be applied for finding hash collisions in $2^{b/2}$ operations on any hash function, where b is the bitlength of the hash value [92] (e.g. 2^{80} for NSEC3/SHA-1), as opposed to 2^{b-1} operations for finding preimages.

5.2.2 Courier Attack

The birthday problem does not apply to DNS spoofing with multiple queries because we do not match a set of n random elements against each other. We match r randomly spoofed responses against q random queries. Even if $q = r$, the model differs. Imagine the following metaphor: In a town with 365 houses, 23 couriers are sent to random houses and 23 villains are waiting in random houses to rob anyone. What is the probability that at least one courier will be robbed?

With $m = 365$ houses, we compute the combinations to select $r = 23$ villain houses without matching any of the $q = 23$ courier houses, i.e. hitting the complementary $m - q$ houses, as

$$\binom{m-q}{r} \quad (5.6)$$

and divide this by the number of all possible house choices

$$\binom{m}{r}. \quad (5.7)$$

The complementary probability to match a villain house with at least one courier is thus

$$1 - \frac{\binom{m-q}{r}}{\binom{m}{r}}. \quad (5.8)$$

The model has been first introduced by Dagon et al. [34] and is similar to the hypergeometric distribution, except that we are looking for one or more matches, whereas the hypergeometric distribution models a fixed number of matches. The result of the above example is

$$1 - \frac{\binom{365-23}{23}}{\binom{365}{23}} \approx 0.7868 \quad (5.9)$$

and thus significantly more than in the birthday attack. We refer to this type of attack as courier attack.

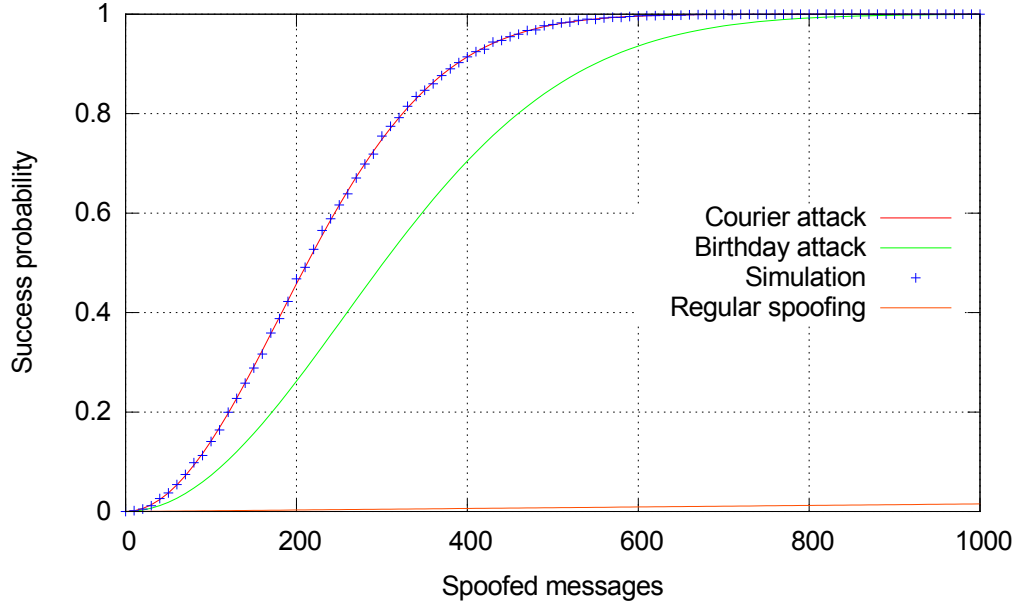


Figure 5.1: Comparison of spoofing attack models with $m = 2^{16}$ different transaction IDs.

5.2.3 Evaluation

We evaluate the models of the courier attack and the birthday attack by comparing an example scenario with simulated data. In the simulation, we randomly select q queries without replacement out of a set of m elements. After all queries have been selected, we reset the set to its initial state. Then, we randomly select r responses without replacement out of the same set. If there is any match between queries and responses, the spoofing attack will be successful. The average result of 10,000 simulation repetitions is the simulated success probability. The y-axis in Figure 5.1 shows the success probability subject to the number of spoofed messages on the x-axis. We use $m = 2^{16}$ in this scenario and $q = r$, i.e. same number of queries and responses. The courier attack accurately models the simulated results; the model of the birthday attack yields significantly lower probabilities. Figure 5.2 shows the results for $m = 2^{30}$.

For reference, both figures also show the success probability of regular spoofing with r responses against one query. Compared with the courier attack, the success probability of regular spoofing is extremely low for a large m . With $m = 2^{30}$, the probability to match one query with 10,000 random responses is

$$\frac{10\,000}{2^{30}} \approx 9.3132 \cdot 10^{-6} \quad (5.10)$$

but the probability to match one of 10,000 random queries with one of 10,000 random

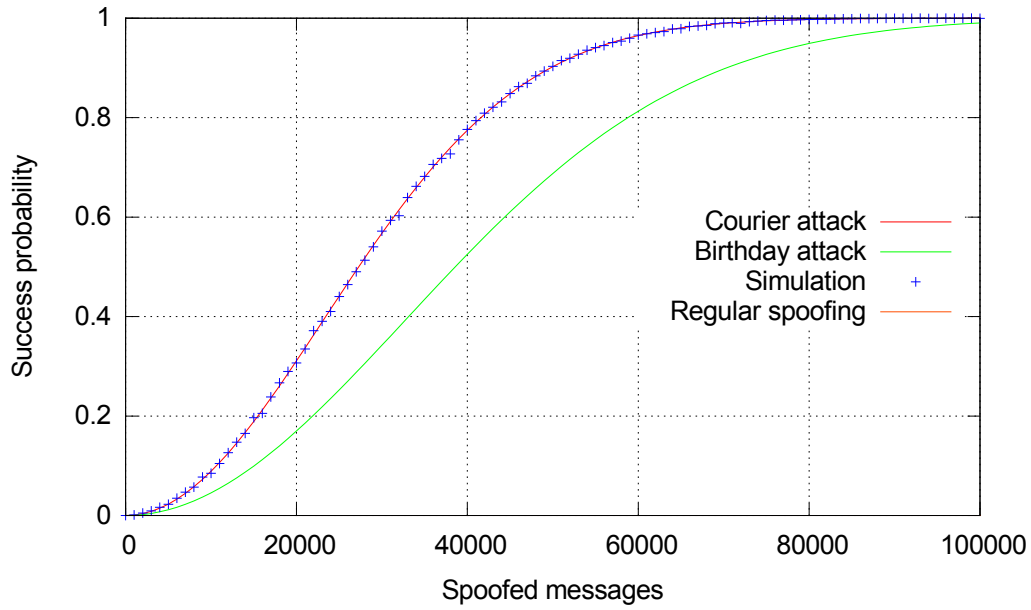


Figure 5.2: Comparison of spoofing attack models with $m = 2^{30}$ different transaction IDs and source port numbers. Probability of regular spoofing is barely above zero.

responses is

$$1 - \frac{\binom{2^{30} - 10\,000}{10\,000}}{\binom{2^{30}}{10\,000}} \approx 0.0889. \quad (5.11)$$

To eliminate this weakness, resolvers should withhold duplicate queries for the same question tuple to the same name server. If properly implemented, the resolver will be immune against the courier attack.

5.3 Kaminsky Attack

With the standard DNS spoofing attack, the attacker has a limited time window Δt for sending spoofed responses. When the attack has not been successful and the resolver has received an authentic response, future lookups for the question tuple will be served from cache. The attacker will have to wait until the cached record expires before being able to repeat the attack attempt. Typical TTL values are in the range of minutes or hours, which diminishes the attack surface from a temporal point of view.

Dan Kaminsky demonstrated in 2008 that an off-path attacker can repeat the attack without having to wait for cache expiry [71]. Instead of triggering queries for the domain name to be spoofed (e.g. `www.example.org`), the attacker triggers queries for non-existent domain names in the same zone, e.g. `xxx001.example.org`. The spoofed response contains a fake delegation to `www.example.org` and a fake glue record pointing to the attacker's name

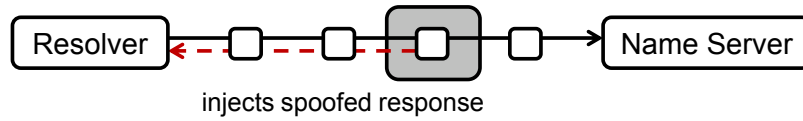


Figure 5.3: Injection of spoofed DNS response.

server. If the spoofing attack is successful, the user’s resolver will cache the fake delegation and future lookups of `www.example.org` will be directed to the attacker’s malicious name server. The result will be practically the same as if `www.example.org` had been spoofed directly: the attacker can divert `www.example.org` to any IP address. If the spoofing attack is not successful, the attacker can repeat the attack with another non-existent domain name, e.g. `xxx002.example.org`. As the attacker is practically always able to find a query name that the user host has not looked up before, the time window Δt for sending spoofed responses can be extended at will.

The same protective measures that apply to plain DNS spoofing apply to the Kaminsky attack as well. In fact, Kaminsky’s discovery prompted wide deployment of source port randomization and contributed to the deployment of DNSSEC.

5.4 DNS Injection

DNS injection is a censorship¹ method for blocking access to websites. It is basically an on-path or in-path spoofing attack, carried out by Internet service providers on a large scale to implement governmental regulations. With DNS injection, the network carrier monitors all DNS queries passing the network and uses deep packet inspection to extract the query name (Figure 5.3). If the query name matches a blacklist, the carrier injects a spoofed response. The spoofed response either indicates a name resolution error or contains a resource record that diverts the user application to the wrong server or to an unreachable destination.

DNS injection is an effective blocking mechanism. Unlike blacklisting on recursive name servers (Section 5.5), which affects only the users of that name server, DNS injection affects all users of a network. This includes users who run their own recursive resolver and users who rely on third-party resolvers like Google Public DNS. The carrier can place the monitoring and packet injection device in a central network location to consolidate administration; it is not necessary to deploy DNS injection physically near the users for full network coverage. However, a careless setup can leak spoofed DNS responses out of the carrier network

¹Censorship is the act of controlling and suppressing speech and expression. The boundaries of what constitutes free speech are limited by what is judged as not deserving protection, which may include for example harassment, slander, pornography or graphic violence. The Domain Name System serves technical parameters like service locators or IP addresses but is rarely used to transport speech or expression by itself. The manipulation of DNS data thus becomes censorship only in the context of suppressing speech, e.g. web or email content. Without this context, we abstain from referring to any DNS manipulation as censorship and instead refer to it as DNS alteration, filtering or blocking.

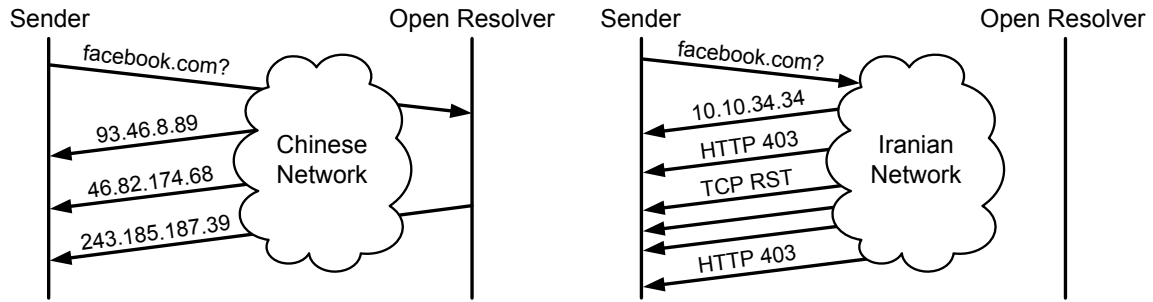


Figure 5.4: Typical responses when querying an open resolver in China or Iran for a black-listed domain name.

and affect uninvolved third-parties whose traffic is routed through that network. Mauricio Vergara Ereche [43] brought this issue to the attention of the DNS community in 2010 when queries sent from Chile to the I-root name server resulted in bogus responses from a Chinese network. We attempt to quantify this threat with measurements in Chapter 6.

DNS Injection in China

The Internet filter of the People’s Republic of China, colloquially known as *Great Firewall of China* (GFW), returns bogus DNS responses for the purpose of blocking websites since at least 2002 [137]. DNS spoofing is used in conjunction with other filtering methods, e.g. inspection of HTTP traffic [31]. Lowe et al. discovered in 2007 that DNS spoofing in China occurs on router-level, i.e. spoofed responses originate from intermediate hops on the path to the actual IP destination [87].

Figure 5.4 shows a DNS query sent from a foreign network into a Chinese network. DNS queries are neither altered nor taken off the network en route to its destination, which typically results in multiple responses. It is not necessary for a successful spoofing attack to suppress DNS messages. As the client host is topologically closer to the injecting device than to the destination name server, injected responses have a head start over genuine ones.

The query matching ignores the record type and class and hence answers always with an A resource record, including e.g. to SOA or TXT queries. DNS over TCP does not seem to be affected at all.

DNS Injection in Iran

Another nation-wide installation of DNS injection is being used in Iran, as we found with our measurements in Chapter 6. Iranian DNS injection is set up as an in-path attack and drops unwanted DNS queries in addition to the injection of spoofed DNS responses. Characteristic for Iranian DNS injection is the use of the bogus answer IP address 10.10.34.34 and spoofing of TCP packets. The right side of Figure 5.4 shows how a UDP-based DNS query triggers

a bogus UDP-based DNS response, two TCP segments with HTTP error 403 (*‘forbidden’*) and FIN/ACK bits set and three TCP RST packets. Note that we did not send any TCP traffic to the Iranian open resolver, thus the TCP sequence and acknowledgement numbers do not match any of our connections.

Different from Chinese DNS injection, the Iranian filter implementation strictly expects certain query flags. Queries with e.g. the *authenticated data* (AD) bit set and the *recursion desired* (RD) bit clear will pass through the DNS filter without triggering a spoofed response.

DNSSEC

DNSSEC protects from DNS injection, provided that the authentic DNS messages are not dropped by the carrier. A validating DNSSEC resolver will reject spoofed responses and accept the subsequent authentic ones. If the carrier drops the authentic DNS responses, then the DNS injection attack will succeed despite DNSSEC validation. The attacker cannot redirect the user to the wrong server but the blocked domain name will not be resolved anyway. Although DNSSEC does not help users in filtered networks against in-path DNS alteration, it remediates the effects of DNS injection on uninvolved third-parties. Validating resolvers retransmit the query to other name servers if one of them returns bogus DNS responses. Given that network and geographical locations of redundant name servers are often dispersed, a DNSSEC validator thus has the chance of finding an unspoiled path to a name server. The latency of the name lookup increases, but the resolver is more resilient against DNS injection in third-party networks.

5.5 Capabilities of Resolver Operators

Name server operators have essentially the same capabilities as in-path attackers. Operators of recursive name servers can alter or omit resource records before passing them to the query sender. Without DNSSEC or another end-to-end integrity protection, the user must trust their recursive name servers to not interfere with name lookups. If queries are forwarded through a chain of multiple recursive name servers, each of them must be fully trusted.

Apart from the possibility of malicious spoofing, there are some notable examples how resolver operators alter domain name resolution. One practice common among some ISPs is the hijacking of negative responses: instead of passing an NXDOMAIN error code to the query sender, the recursive name server replaces the DNS error message with an A resource record pointing to an ISP web server. Web-browsing users will be redirected to a web page indicating an error and showing advertisements or links to advertisements (see example in Figure 5.5). Other services than web will attempt to connect to the IP address on a closed port and fail. The purpose of NXDOMAIN hijacking is to monetize domain typos. In this sense, the practice is comparable to typosquatting (registering slightly misspelled domain

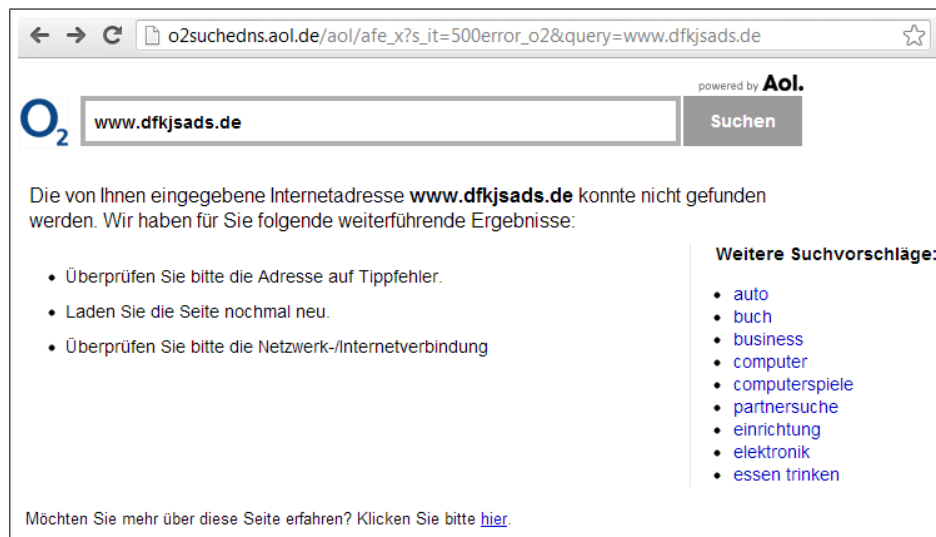


Figure 5.5: NXDOMAIN hijacking by an Internet service provider, December 2012.

names).

Another common practice is web filtering. OpenDNS for example operates recursive name servers with optional paid services for filtering of adult websites or filtering of phishing and malware websites. Blacklisted domain names resolve to an OpenDNS information web page. This type of web filtering is controlled by the user; another type is controlled by the state respectively by the ISP. ISPs in several countries implement state-mandated Internet filtering via blacklists on recursive name servers (cf. Section 6.1.3). Filtering on recursive name servers can be bypassed easily by switching to a name server without filtering. This happened extensively when the Turkish government instructed ISPs to block Twitter and Youtube in March 2014. Turkish users quickly spread the advice to use the Google Public DNS name server addresses 8.8.8.8 and 8.8.4.4 (Figure 5.6), which were not affected by Turkish filters. When the blocking turned out to be ineffective, Turkish ISPs set up BGP hijacking to divert queries for Google Public DNS to their own name servers [8]. The forged IP routes were not advertised to outside networks and thus did not affect Internet users in other countries.

DNSSEC

DNSSEC protects from tampering on recursive name servers if set up properly. To protect from Internet censorship, it is evident that DNSSEC validation must be performed elsewhere than on the filtered resolver. The validating resolver can be located on the user host or on a trusted name server, which is reachable via a secure communication channel, e.g. a private local network or VPN. Although DNSSEC detects integrity failures, a filtering recursive name server can nevertheless block domain names by returning bogus responses. A filtering



Figure 5.6: Graffiti in Kadıköy, Istanbul, advising to use Google Public DNS to bypass Turkish Internet filters. [114]

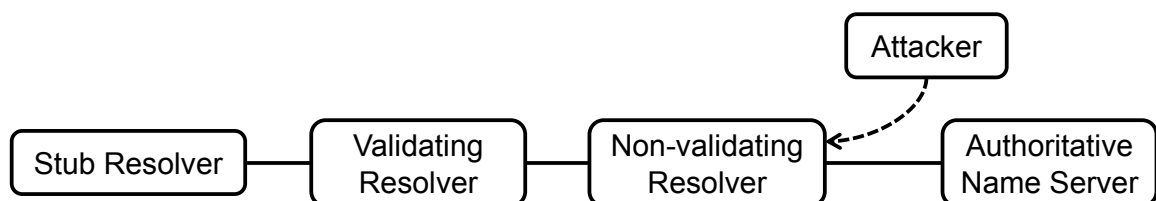


Figure 5.7: Validating resolver forwards queries to non-validating resolver.

resolver corresponds to an in-path attacker in such a way that an untrusted name server can cause denial of service despite DNSSEC validation. In fact, even a recursive name server without malicious intentions decreases the overall resilience. Consider the scenario in Figure 5.7, in which a validating resolver forwards queries to a trustable but non-validating resolver. An attacker might inject a bogus response to the non-validating resolver, which passes it to the validator. Although the validator rejects the bogus resource records and retransmits the query, the non-validating resolver will return the same bogus response from its cache. There is no built-in signalling mechanism to purge the bogus cache entry or to request validation on a case-by-case basis. Note that the “checking disabled” (CD) flag in DNSSEC (cf. Section 3.3) is not an inverse “validation requested” flag: CD=0 indicates that validation is permitted, but not required.

There are two ways to recover from the above scenario: 1) the non-validating resolver enables validation, or 2) the validating resolver does not forward queries to the non-validating resolver. Concerning option 1, it is a desired objective of DNSSEC to support validation on as many resolvers as possible. Yet, the DNSSEC specification in RFC 4035 [13] and the normative clarifications in RFC 6840 [130] recommend that a validating resolver should always set CD=1, i.e. request to skip validation on upstream resolvers. The rationale for this behavior is that validators may have dissenting parameters and policies, e.g. system time, trust anchors or handling of corner cases, which may lead to disparate validation results. In case of a validation failure, the upstream validator would pass a SERVFAIL failure, which is an unverifiable dead-end to the downstream validator. By setting CD=1, the first validator nearest to the user can retrieve the raw resource records and perform validation on its own, independent of potential validation failures on upstream resolvers. While this increases resilience in some scenarios, it is harmful in others, where bogus data is served from a non-validated cache. Apart from malicious attackers attempting DNS spoofing, such a situation can occur simply due to operational carelessness, e.g. stale zone data with outdated signatures on a secondary authoritative name server. Neither enabling nor disabling DNSSEC validation on upstream resolvers provides comprehensive protection and resilience against bogus data. This leads to option 2: validators retrieve resource records directly from authoritative name servers, omitting any upstream resolvers. We discuss this approach in Chapter 10 and evaluate the costs with a trace-driven simulation.

In order to set up intended DNS alteration in conjunction with DNSSEC, e.g. to support OpenDNS-like filtering of malware web sites, the validating resolvers must be configured manually with another trust anchor. The private key of the trust anchor is in possession of the filtering resolver, used to synthesize altered, but signed responses. Although this approach works in principle, it may cause interoperability problems as DNSSEC was not designed to support such a use case. In fact, OpenDNS uses a different approach: DNS requests are tunneled through a cryptographic protocol called DNSCrypt between stub

resolver and recursive name servers. DNSSEC validation runs on the recursive name servers only, not on the end hosts. Therefore, the name servers can alter DNS responses after DNSSEC validation has succeeded. DNSCrypt is not a standardized protocol and thus requires the installation of the client software on the end host. In either case, custom DNSSEC trust anchor or secure tunneling of DNS messages, the user trusts the resolver, which is capable of altering any DNS response.

5.6 Capabilities of Authority Operators

Operators of authoritative name servers can serve zone data other than intended by the zone administrator and have the same capabilities as in-path attackers. In legacy DNS, each authoritative name server has full control over the zone that they have been delegated. The zone administrator must trust the operators of authoritative name servers and their Internet service providers. This includes secondary name servers, e.g. when relying on third-party operators for load distribution. Resolvers have no possibility other than accepting the data returned by authoritative name servers.

Zone administrators furthermore depend on all parent zones up to the root and their authoritative name servers. If one of them tampered with a delegation, they could take away control over a domain or return bogus responses.

DNSSEC

When DNSSEC is used with offline signing, authoritative name servers cannot forge signed responses without access to the private part of the KSK or ZSK. The role of an authoritative name server shifts with DNSSEC from a zone authority to a proxy between the validating resolver and the zone key holder (cf. Figure 3.1). The attacker capabilities are comparable to intermediate recursive name servers: denial of service is possible, but not DNS alteration. The zone administrator can for example rely on external secondary name servers, which cannot alter the zone contents they serve without being noticed by the validator.

The hierarchical trust model of DNSSEC (cf. Section 3.6.1) still requires the zone administrator to trust all parent zones, like in legacy DNS. There is no security mechanism to guarantee a specific zone cut, i.e. that a secure delegation is not changed. Parent zones can withdraw secure delegations at any time by simply not returning a specific referral response, even if the signatures are still valid and have not expired. Capabilities of the parent zone include 1) changing the secure delegation to an insecure delegation (with signed NSEC/NSEC3 records), 2) exchanging the signed DS record to divert the authentication chain to any other KSK, and 3) changing the name servers and IP addresses in the delegation.

5.7 System Time

In legacy DNS, the only time dependency are the caching times of responses and resource records. An incorrectly set system time is without effect because the TTL values are relative timestamps. Clock drifts or shifts have minor effects; records may expire earlier or may remain longer in cache than specified by the zone administrator. Early expiration is harmless as it may also occur due to other reasons (e.g. cache purge after system restart), whereas later expiration is undesired but harmful only when several conditions match (e.g. exceptionally long clock shifting, no cache purge in the meantime and a major change of a previously accessed domain occurs). Changing the system time, intentionally or unintentionally, is usually without consequences in legacy DNS.

DNSSEC

DNSSEC introduces absolute timestamps in signatures. Thereby, signers and validators are required to keep their system clock synchronized. The choice of the validity period is a trade-off between security and effort for signature updates. Once a signature has been published, attackers can replay the signed response until the end of the expiration time. There is no revocation of a signed response, even if the zone administrator replaces the resource record set and its signature on the authoritative name servers. Zone administrators should keep track of the expiration times of obsolete resource records, e.g. reserve no longer used IP address blocks after server relocation until the signatures have expired. Typically chosen validity periods are in the range of days or weeks.

The validity period is given as absolute inception time and absolute expiration time. Minor clock deviations in the range of minutes are harmless as long as zone administrators apply safety margins on signature times. In particular, the signature inception time should be pre-dated to tolerate clock skew and the signatures should be refreshed way before expiration time. Safety margins can be applied on either side, signer or validator. For example, the “Unbound” DNSSEC resolver by default tolerates a clock skew of up to 24 hours to cushion the effect of wrong timezone settings [79].

One of the implications of DNSSEC deployment is that the system time must be kept in sync with a secure method. The Network Time Protocol (NTP) is the state-of-the-art clock synchronization method. NTP provides mitigation against implausible times and offers an optional cryptographic authentication against malicious NTP spoofing. Anecdotal evidence suggests that authentication is seldom used; e.g. the NTP Pool Project² does not offer a generic authentication method (though individual servers from the pool may do) and Debian 7 ships with NTP working out of the box but without authentication. Furthermore, NTP and DNSSEC depend on each other: in order to synchronize with a public NTP pool,

²<http://www.pool.ntp.org/>

the NTP client resolves a domain name like `pool.ntp.org`. If the system time deviates widely from real time, e.g. when the clock has been reset to factory defaults, DNSSEC resolution will fail. Even if the pool domain name is deliberately kept unsigned, resolution will fail due to expired or not yet valid root and TLD signatures. An attacker being able to manipulate the system time of a target host thus has the capability to replay old DNSSEC responses or to deny the name resolution service. As a working name resolution is a necessity for most Internet services like web, mail or VoIP, this attack results essentially in an Internet cutoff.

5.8 Breaking DNSSEC Keys

DNSSEC is designed to depend on the security of the cryptographic primitives it uses. In the following case study, we attempt to break a 512-bit RSA DNSSEC key³. 512-bit RSA is known to be insecure and RSA numbers up to a length of 768 bits have been broken publicly [73]. Yet, there is a significant number of 512-bit RSA keys in circulation as we show in Section 8.2.1. As of September 2014, the domain `iamevil.net` is signed with a KSK/ZSK scheme, each being a 512-bit RSA key. In order to break the KSK, we first need to extract the RSA modulus from the DNSKEY record set, use integer factorization to decompose the modulus into its two primes p and q , reconstruct the private key, and then use the private key to sign bogus zone data. The authoritative name servers of `iamevil.net` return the following DNSKEY set:

```
iamevil.net. 3600 IN DNSKEY 257 3 7 AwEAAcgW9MXxec1/aAsfyCKNkmn4Epdq66rr
xVkJVMYkxHTPmWRxHsS6XaH1PbWXP54VDhNDS2gv7FNn
/odWyab79zc=
iamevil.net. 3600 IN DNSKEY 256 3 7 AwEAAeJ/KW0sRjwfk3Ae0A51THkYwRF/e00s
1X/ffVfU6CAi00LSnETwxdvx+dtxqLd0LUFqYESZPBRx
6PDGrtYsifs=
```

The KSK is indicated by flags field value 257 (least significant bit is set). The public RSA key is encoded as Base64 text and its inner format is shown in Figure 5.8. The exponent is $e = 65\,537$ and the modulus n is a decimal number with 155 digits. The general number field sieve (GNFS) is the fastest factorization method currently known for arbitrary integers of this size [24]. In this study, we use the GNFS implementations of the open-source tools Msieve⁴ and GGNFS. For best performance, each of the three GNFS stages can be computed with the faster implementation of both tools [49]:

1. Selection of polynomial (Msieve).

³With a domain used for testing purposes only and with consent of the domain owner.

⁴Built with GCC 4.3 because versions 4.4 to 4.7 lead to occasional assertion errors.

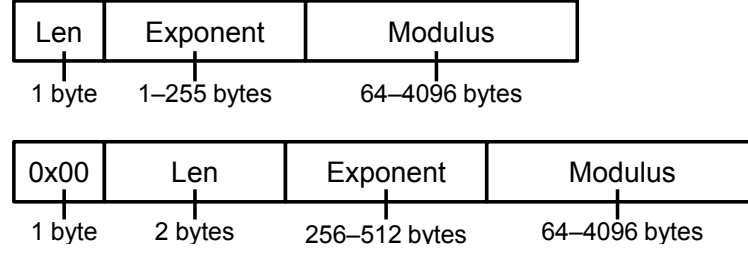


Figure 5.8: DNSKEY record format for RSA public keys, specified in RFC 3110 [2]. The length of the exponent is given by a length field, which is either 1 or 3 bytes long, indicated by the first octet. The length of the modulus can be derived from the lengths of the exponent and the overall record data.

2. Finding relations with lattice sieve (GGNFS).
3. Combining relations (Msieve).

Stage 1 finished within 12 days of computation on host with a 2.67 GHz x86-64 CPU to find a suitable polynomial for the sieving stage. The Msieve implementation of the polynomial selection uses one CPU core, but there also exists a multi-threaded GPU implementation written in CUDA, for which the author reported a performance improvement of 50 to 100 times compared to the CPU implementation. As stages 2 and 3 support multi-core computation, we switched to another host with a 32-core 2.2 GHz x86-64 CPU⁵. The multi-core support of the toolset was fairly basic at that time, leaving CPU cores at idle for several minutes at synchronization points. Nevertheless, stage 2 finished after slightly less real time than stage 1, though the lattice sieving is the most CPU-intensive stage. The final stage 3 took another 16 hours and yielded the prime factors p and q , 77 and 78 digits long. Overall, the integer factorization finished within 24 days of computation. The next step is to recover

$$\phi(n) = (p - 1)(q - 1) \quad (5.12)$$

$$d \equiv e^{-1} \pmod{\phi(n)}. \quad (5.13)$$

We use the extended Euclidean algorithm to obtain

$$g, s, t = \text{egcd}(e, \phi(n)), \quad (5.14)$$

where the greatest common divisor is $g = 1$ and s is the multiplicative inverse of $e \pmod{\phi(n)}$, so that

$$d \equiv s \pmod{\phi(n)}. \quad (5.15)$$

The private key is (d, n) . The DNSSEC signer *dnssec-signzone* from BIND 9 uses the Chinese remainder theorem to speed up signing operations, and thus requires a verbose

⁵Host kindly provided by Arno Wacker and Nils Kopal, University of Kassel.

representation of the private key [68] with additional components, which can be derived from e , p and q :

$$d_P \equiv e^{-1} \pmod{p-1} \quad (5.16)$$

$$d_Q \equiv e^{-1} \pmod{q-1} \quad (5.17)$$

$$q_{\text{inv}} \equiv q^{-1} \pmod{p} \quad (5.18)$$

The recovered private components of the KSK can be used to forge arbitrary resource records with valid signatures for the domain under attack. The following example shows a signature created with the recovered KSK, which validates successfully with BIND 9:

```
www.iamevil.net. 3600 IN A      6.6.6.6
www.iamevil.net. 3600 IN RRSIG A 7 3 3600 20150912091036 (
                               20140911091036 57319 iamevil.net.
                               W8ZVZ4p8DsSP4Zp4T8L0UtrG0ea7FK9B7KC+
                               gRQXInVPzb32lBUFOUWlyTAWgaM3PCjUK/5P
                               7v3PeCW08QRmVQ== )
```

At this point, the attacker has the same capabilities as in legacy DNS, e.g. return spoofed responses with an on-path spoofing attack. With DANE (Section 3.6.2), attacks beyond name resolution become possible, e.g. man-in-the-middle attacks on DANE-secured TLS connections.

The 512-bit RSA KSK was broken after about 3 weeks of computation with open source tools running on one CPU. The attack is possible for larger key sizes in principle, but the costs rise steeply up to the point of being practically infeasible. Shamir and Tromer suggested in 2003 that 1024-bit RSA keys can be broken within one year by a hypothetical device with a manufacturing cost of US\$ 10M [115]. Kleinjung et al. factored a 768-bit RSA modulus in 2009 [73] and estimated that similar academic efforts could factor 1024-bit RSA by the year 2020 [26]. General consensus is that 1024-bit RSA is not secure anymore and 2048-bit RSA is expected to be secure for the next couple of years. NIST deprecated the use of 1024-bit RSA in 2013 and recommends to use ≥ 2048 bits, with replacement of the private key after 1 to 3 years [18]. BSI recommends 2000-bit RSA with a predicted conformance until the year 2021 [29].

CHAPTER 6

Measurement Study on DNS Injection

DNS injection is a technique for blocking domain names with on-path DNS spoofing (cf. Section 5.4). China is known to use DNS injection for Internet filtering in such a way that spoofed responses may leak unintentionally into foreign networks. DNS injection constitutes a DNS spoofing attack that is being performed in practice and which may affect the Internet population worldwide. In this chapter, we study the visibility and characteristics of practical deployments, measured from vantage points outside of the poisoned networks. This study is not limited to Chinese DNS injection and in fact, we discovered another variant of DNS injection used in Iran. We utilize various measurement methods with different purposes:

- Section 6.1: probe the Internet to find affected networks.
- Section 6.2: probe affected networks to determine the domain blacklist.
- Section 6.3: probe affected networks to obtain the list of bogus addresses in responses, which can be used to detect spoofed responses.
- Section 6.4: probe open resolvers to assess impact on third-parties.
- Section 6.5: investigate an affected open resolver in Taiwan (case study).
- Section 6.6: assess impact on Hong Kong networks (case study).

6.1 Probing for DNS Injectors

The purpose of this measurement is to find networks that employ DNS injection. We are specifically interested in globally visible spoofed responses, i.e. bogus responses which have crossed network boundaries.

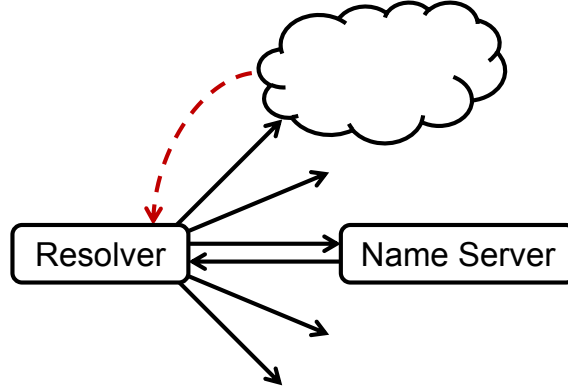


Figure 6.1: Measurement for finding injecting networks.

6.1.1 Method

The measurement method is to send DNS queries to a large amount of destinations and to determine whether the responses have been tampered with (Figure 6.1). The destination hosts do not need to run a name server and even do not need to be online. As DNS injection works on router-level, we will receive a response if the DNS query is routed into or through a network that spoofs bogus DNS responses. Without DNS injection, we do not expect a DNS response. However, there is also a chance that the DNS query reaches a responsive name server, e.g. an open resolver. We thus need to identify whether a response is genuine or spoofed.

The measurement consists of two parts: first, a sparse probing to identify filtered domain names, and second, a dense probing to analyze the extent of DNS injection. The rationale of this approach is to avoid unnecessary network load. For the first part, we compose a candidate list of websites from a range of categories that could be censored, including file sharing, information freedom, human rights, online gambling, sexual content, controversial religious or political content, graphic violence and social media. We send DNS requests for each candidate domain name into each publicly announced BGP IPv4 prefixes.

For the second part, we send DNS requests to every IPv4 /24 subnet for domain names that have shown evidence of DNS injection in the first part. We omit the following address spaces that are not routed globally: 0/8, 10/8, 127/8, 172.16/12, 192.168/16 and 224/3. To spread the load per destination network over time, we iterate through the IPv4 address space with an offset (1.0.0.99, 2.0.0.99, 3.0.0.99, etc.). We refrain from probing all IPv4 addresses because we do not expect substantially different results than from probing one IPv4 address from each /24 subnet.

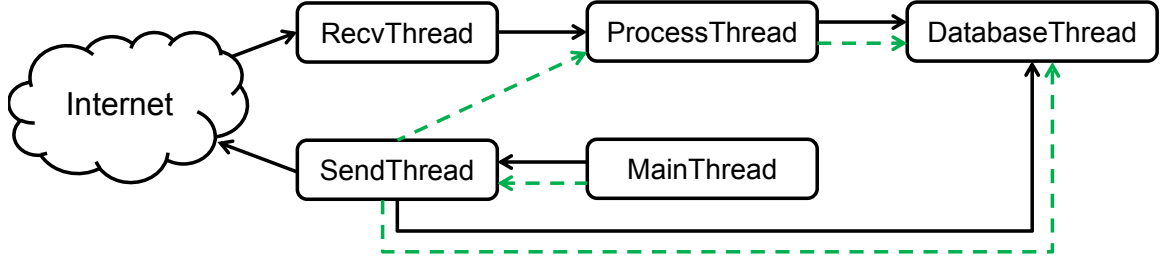


Figure 6.2: Software architecture of probe measurement: data flow (solid lines) and wait dependencies (dashed lines) are shown.

6.1.2 Implementation

The measurement software is implemented as a multi-threaded Python program. The default Python interpreter does not execute threads in parallel due to the global interpreter lock [20], however we use threading mainly as programming model for functional decomposition and not for parallel execution. The software architecture is shown in Figure 6.2 and consists of five threads, which communicate over synchronized FIFO queues:

- MainThread: iterates through destination IP addresses
- SendThread: sends DNS queries over socket
- RecvThread: receives DNS responses from socket
- ProcessThread: parses DNS responses and extracts relevant information
- DatabaseThread: writes sent and received DNS messages to disk

The solid black lines in Figure 6.2 represent the data flow between threads and the dashed green lines represent wait dependencies. For example, the SendThread pauses when the processing queue of the ProcessThread contains > 10 entries. This throttles the rate of outgoing DNS queries, which in turn throttles the rate of incoming DNS responses until the backlog has been parsed and written to disk.

Apart from throttling data flow, there is no shared state between the sender and receiver components, i.e. there is no retry on failure and incoming responses are not matched against pending queries. We use exactly one UDP socket for sending and receiving the DNS messages to and from all destinations. The receive buffer size of the socket is raised to 10 MB to ensure that it does not fill up.¹

We disable stateful packet inspection on our firewall for frictionless large-scale measurements.² Stateful packet filters can interfere with packet transport to plenty of different

¹Socket option `SO_RCVBUF`; Linux furthermore requires to raise the kernel parameter `net.core.rmem_max`.

²Linux Netfilter allows to disable stateful inspection per-host and per-service with the `NOTRACK` target.

| Domain name | China | Iran |
|--------------------|-------|------|
| drive.google.com | X | |
| facebook.com | X | X |
| sites.google.com | X | |
| twitter.com | X | X |
| www.minghui.org | X | |
| www.torproject.org | X | |
| www.xxx.com | X | |

Table 6.1: Domain names indicating occurrence of DNS injection.

destinations; build-up and teardown of the state table causes unnecessary CPU costs and bears the risk of packet loss due to table overflow. Another benefit of disabling stateful inspection is that broken packets can be observed that would have been filtered otherwise, e.g. responses from wrong source port or IP address.

A remaining problem with measurements “in the dark” is how to distinguish between no response 1) due to the absence of DNS injection or 2) due to packet loss caused by congestion in our campus network. We observed during development that the faster the measurement program ran, the less responses arrived. We tackle this problem by limiting the sending rate to ≤ 1000 queries per second and monitoring our network health with DNS pings to name servers.

6.1.3 Measurement Result

Part 1: Identification of Filtered Domain Names

The first part of the measurement ran in July 2013 with an educated guess of 47 candidate domain names. For each name, we sent a DNS query from our vantage point in AS680 (DFN, Germany) to 422,228 IPv4 addresses, each in a different publicly announced BGP prefix.³ This lead to 682,640 responses which comprised 9958 distinct answer IP addresses. After manually removing genuine, unaltered responses, we investigated the most frequent remaining addresses which appeared bogus. Part of them originate from filtering open resolvers which our probe queries hit by coincidence. Thus as a byproduct of our measurements, we confirm systematic filtering on ISP resolvers being used in Bulgaria, Colombia, Indonesia, Singapore and Turkey. In these cases, the bogus answer IP address points to an ISP webserver with a notice that the website is blocked (Figure 6.3). The remaining addresses are an evidence of DNS injection; a geolocation analysis⁴ suggests China and Iran as origin (Table 6.1). This is further analyzed in the second part of the measurement.


³Measurement host kindly provided by Arno Wacker and Nils Kopal, University of Kassel. BGP routing table from APNIC, provided by Philip Smith at <http://thyme.apnic.net/>.

⁴Using GeoLite from MaxMind [91].

Вие попаднахте на тази страница, защото достъпът до сайта е ограничен във връзка с изпълнение на Разпореждания от 17.06.2013г., 28.06.2013г., 15.07.2013г., 31.07.2013г., 06.08.2013г., 04.09.2013г., 24.09.2013г., 03.10.2013г., 23.10.2013г., 01.11.2013г., 27.11.2013г., 05.12.2013г., 21.12.2013г., 02.01.2014г., 06.02.2014г., 13.02.2014г., 26.02.2014г., 07.03.2014г., 04.04.2014г., 25.04.2014г., 17.06.2014г., 25.06.2014г., 04.07.2014г., 29.07.2014г., 11.08.2014г., 18.08.2014г., 24.09.2014г., 03.10.2014г., 23.10.2014г., 05.11.2014г., 28.11.2014г. и 05.12.2014г. на Софийски районен съд.


¡Atención !


De acuerdo con lo previsto en la [Ley 679 del 3 de agosto de 2001](#) expedida por el Congreso de la República y el [decreto 1524 del 24 de julio de 2004](#) expedidos con el fin de proteger a los menores de edad, Telefónica Telecom se permite informar que la página a la que intenta acceder **esta restringida**.





AVAILABLE SPACE

+62 21 79187250 | sales@metranet.co.id





Situs terlarang tidak dapat diakses melalui jaringan ini karena terindikasi mengandung salah satu unsur Pornografi, Judi, Phising, SARA atau PROXY. Jika anda merasa situs ini tidak termasuk ke dalam kategori diatas, silahkan menghubungi aduankonten [at] depkominfo [dot] go [dot] id.

The website you are attempting to access is unavailable as it contravenes the Broadcasting (Class Licence) Notification issued by the Media Development Authority of Singapore.

View more information on MDA regulations.

Continue to StarHub.com >

Bu internet sitesi (sites.google.com) hakkında **Burdur Sulh Ceza Mahkemesi**'nin 17/11/2009 tarih ve 2009/629 sayılı kararına istinaden **Telekomünikasyon İletişim Başkanlığı** tarafından **KORUMA TEDBİRİ** uygulanmaktadır.

(The PROTECTION MEASURE has been taken for this website (sites.google.com) according to Decision Nr. 2009/629 dated 17/11/2009 of "Burdur Sulh Ceza Mahkemesi" has been implemented by "Telekomünikasyon İletişim Başkanlığı".)

Figure 6.3: Web filter notices by Bulgarian, Colombian, Indonesian, Singaporean and Turkish ISPs (top to bottom), implemented with DNS alteration on ISP resolver (not DNS injection).

| Count | Answer IP address | Country | AS# | AS Name |
|---------|-------------------|---------|-------|------------------|
| 11,418 | 10.10.34.34 | – | – | – |
| 97,936 | 173.252.110.27 | US | 32934 | Facebook |
| 202,688 | 46.82.174.68 | DE | 3320 | Deutsche Telekom |
| 202,930 | 93.46.8.89 | IT | 12874 | Fastweb |
| 203,132 | 203.98.7.65 | NZ | 4768 | TelstraClear |
| 203,165 | 78.16.49.15 | IE | 2110 | BT Ireland |
| 203,257 | 8.7.198.45 | US | 3356 | Level 3 |
| 203,751 | 59.24.3.173 | KR | 4766 | Korea Telecom |
| 203,756 | 37.61.54.158 | AZ | 28787 | Baktelekom |
| 203,795 | 243.185.187.39 | – | – | – |
| 204,163 | 159.106.121.75 | US | – | US DoD NIC |

Table 6.2: Answer IP addresses for `facebook.com`.

Part 2: Extent of DNS Injection

To analyze the extent of DNS injection, we sent DNS queries from AS24961 (myLoc, Germany) to 14.4M IPv4 /24 subnets. It follows an analysis of measurement results for `facebook.com` from February 2014. 14,479,104 DNS queries were sent and 1,960,297 responses received, out of which 99.0% were free from errors and included an A resource record with an IPv4 address.

At the time of this analysis, the authoritative name servers return one public IPv4 address for `facebook.com` with no indication of DNS-based load balancing. However, the responses in our measurement data contain 284 distinct answer IPv4 addresses. 273 of them occurred only a few times (in total 559 times) and are of no further interest. The remaining eleven answer IPv4 addresses are shown in Table 6.2. The only authentic answer IP address can be easily identified as it belongs to AS32934 (Facebook). Note that the number of bogus responses is larger than genuine ones due to the mechanics of DNS injection: to receive a genuine response the probe query needs to hit an open resolver whereas spoofed responses are always injected, even when the destination host is offline.

There is a remarkably large occurrence of nine answer IP addresses with an almost equal distribution of around 200k responses. A geolocation analysis reveals that 99.9% of the queried IP addresses returning one of these bogus answers are located in mainland China. This indicates that Chinese DNS injection filters return a random address for `facebook.com` from a set of nine fixed IPv4 addresses. The network owners of the bogus addresses do not seem to be related to the network owners who send the spoofed DNS responses. None of the nine addresses host a publicly reachable webserver, and two addresses (159.106.121.75 and 243.185.187.39) are not even routable in the default-free zone⁵. Figure 6.4 shows the

⁵The default-free zone is the set of all routers that do not have a default route.

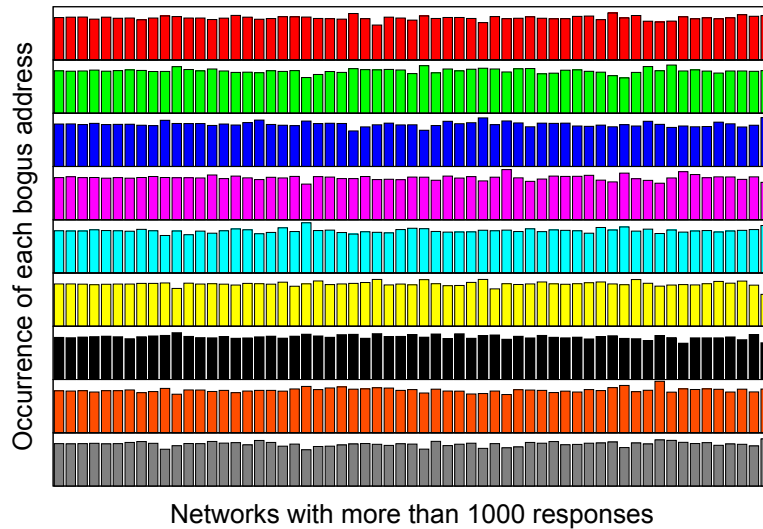


Figure 6.4: Uniform distribution of the nine bogus IP addresses in spoofed responses from Chinese networks.

distribution of the nine bogus addresses, grouped by large AS networks for which we received more than 1000 spoofed responses. The uniform distribution indicates that the selection of the bogus answer address in the DNS response does not depend on the destination address that the query has been sent to; i.e. the addresses are selected randomly from a fixed set.

We asked a network operator who happens to own one of the nine bogus IP addresses whether there is any suspicious network traffic visible. They explained in August 2013 that the IP address is unused and traffic of about 150 packets per second is dropped at the network borders. This is significantly more than what is expected from Internet background radiation [133] and may be an indication that hosts affected by Chinese DNS injection attempt to connect to the IP address without success. A geolocation analysis of a short (not representative) packet trace shows packets originating from 51 countries and regions, including the U.S., Hong Kong, Pakistan and Taiwan.

Another remarkable answer IP address is 10.10.34.34, a private address as per RFC 1918 [105], i.e. the address is not routed globally. 99.9% of the responders of this answer IP address are located in Iran. According to Anderson [6], the address 10.10.34.34 is used by Iranian ISPs to display a webpage with filter notice. We executed the probing measurement at different dates and observed a significant drop of spoofed Iranian responses for a certain period while the Chinese results remained relatively stable. Figure 6.5 shows the number of queried autonomous systems for which we received spoofed responses. The Guardian reported on July 2, 2013 that the newly elected President of Iran Hassan Rouhani spoke out to loosen filtering of social media like Facebook [38]. Our measurements reflect that the DNS filter for `facebook.com` has been lifted by Iranian networks shortly after that. However, a few weeks later the DNS injection filter was active again. On October 7 the

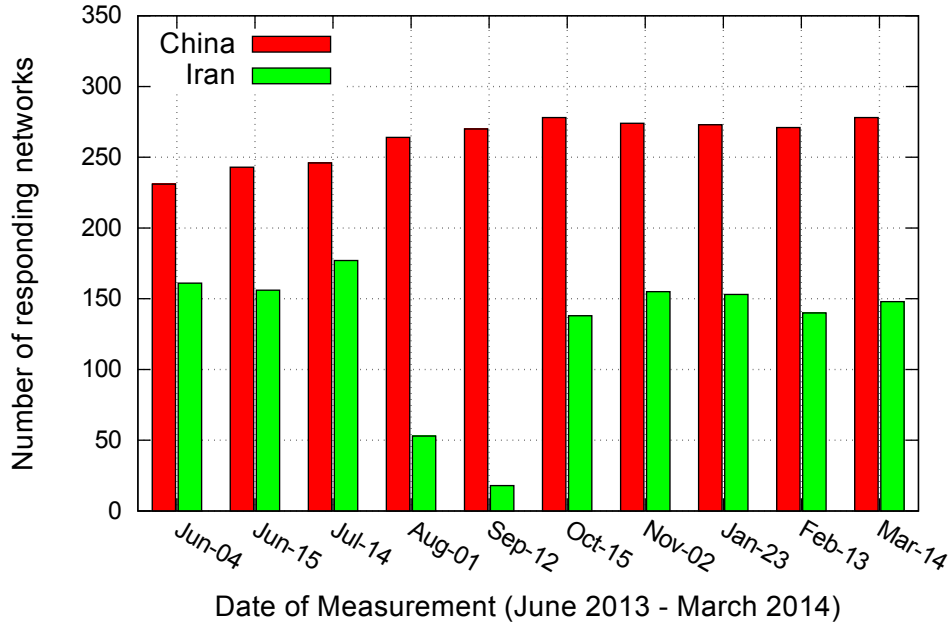


Figure 6.5: Injected responses for `facebook.com` over time.

Iranian Ministry of Communications and Information Technology announced that removing filters for social media is under consideration and the public will be notified once a decision has been made [93]. As of March 2014, the filter was still in place for `facebook.com` according to our measurements.

A large fraction of the 1.9M responses are duplicate responses which claim to originate from 1,087,945 distinct IPv4 addresses. This is common for Chinese DNS injection from which we typically received two spoofed responses. Figure 6.6 shows the round-trip times of correct and spoofed responses (y-scale is logarithmic and cumulative). The black line shows the latency of correct responses. Correct responses originate from open resolvers, which are found worldwide. The graph increases (log-)linearly because there is no particular geographical cluster and thus no cluster of latencies to be observed from our vantage point. The other graphs show the latency of the first response (or second, or third response from the same address) with bogus address. The latency of spoofed responses increases step-wise, indicating that clusters of responses originate from the same network or geographical location within a typical latency band.

Probing other domains like `twitter.com` (filtered in China and Iran), `www.minghui.org` and `www.strongvpn.com` (filtered in China) yields results that are similar to `facebook.com`. The temporary decrease of spoofed Iranian responses could be observed for `twitter.com` as well. From Chinese DNS filters, we received up to 300k spoofed responses with invalid UDP checksum for some domain names. Apparently the DNS injection system suffers from bugs and partly generates invalid datagrams. Considering these faulty responses in the overall

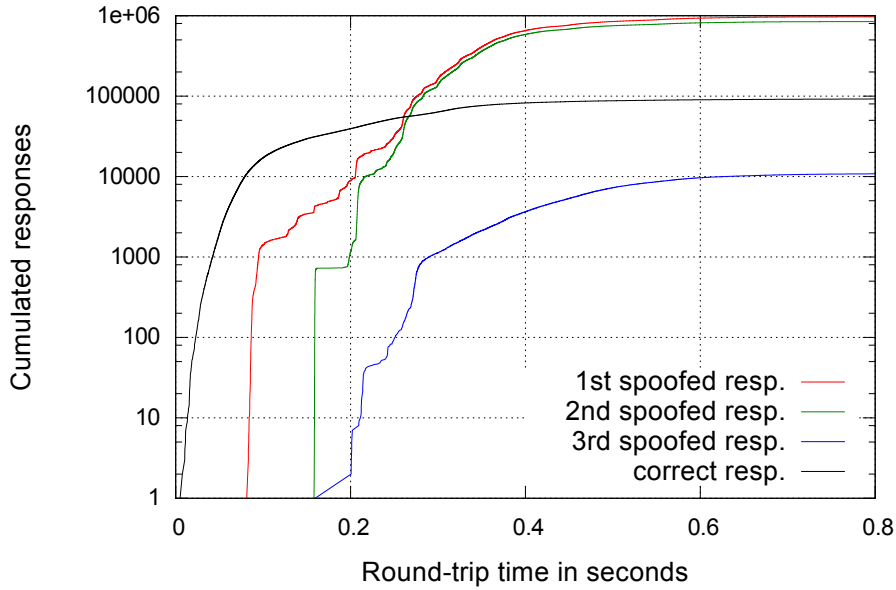


Figure 6.6: Correct and spoofed responses for `facebook.com` by latency.

statistics, each filtered domain name caused the same amount of spoofed responses. There was no evidence for inconsistent domain blacklists among affected Chinese networks. The set of bogus addresses returned varies depending on the domain name, which is discussed further in Section 6.3.

6.2 Blacklist Testing

The purpose of this measurement is to determine the blacklist and granularity of Chinese and Iranian DNS filters.

6.2.1 Method

The method consists of sending queries for various candidate domain names into Chinese and Iranian networks known to spoof DNS responses. Unlike the probe measurement in Section 6.1, this method allows for efficient testing of thousands of domain names because the scope is limited to known DNS injectors. For each domain name `$NAME`, we test five variants with a prefix or a suffix, as shown in Table 6.3. `$RANDOM` is a fixed alphanumeric string, which we have verified to be not blacklisted.

6.2.2 Implementation

The software architecture is based on the probe measurement program (Section 6.1.2) but has been adapted for stateful measurements. The `ControlThread` holds a state object for

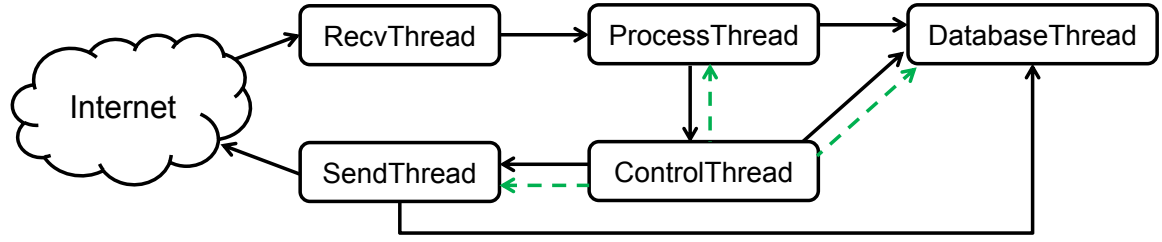


Figure 6.7: Software architecture of blacklist measurement: data flow (solid lines) and wait dependencies (dashed lines) are shown.

each ongoing series of experiments for one domain name and decides which query will be sent next. Incoming responses are fed back into the ControlThread and matched to running experiments with transaction ID and IP address.

To minimize the effect of packet loss, each experiment runs 10 times with different IP addresses. For each queried destination address, we automatically check that it responds to a domain name known to be blacklisted and that it does *not* respond to a domain name known to be *not* blacklisted. This ensures that the destination is affected by DNS injection but does not run an open resolver. We negate the presence of an open resolver only if 6 queries for well-known domain names have timed out consecutively. Hence, the remaining responses do not originate from an open resolver with a high probability and instead are an evidence of DNS injection.

When sending a query, the ControlThread puts the state object of each domain name under test into a double-ended queue. The queue is ordered by waiting time of pending experiments. The waiting time is the time window for receiving responses; the purpose of the waiting time is to ensure 1) that all responses are matched to their corresponding experiment (not just the first response), and 2) that there is an idle time between two experiments to avoid network congestion. The main loop of ControlThread (shown as Algorithm 1 on page 73) processes incoming responses and fetches the oldest state object from the queue. If the waiting time has not been reached yet, the ControlThread will either enqueue the experiments of the next domain name or, if the queue is full, sleep until the waiting time expires. Pending responses time out after > 20 seconds.

6.2.3 Measurement Result

We ran the blacklist measurement in March 2014 separate for Chinese and Iranian networks. The IP address list of DNS injectors originates from the probe measurement in Section 6.1. To get a candidate list, we extracted domain names from the Alexa list of the 1,000,000 most visited websites [5]. The list contains actually less than 1M domain names because some sites are distinguished by URL path but use the same domain name. Combined with a few names from other sources this led to 999,935 unique candidate domain names.

Algorithm 1 Main loop of ControlThread.

```

1: repeat
2:   if is_empty(queue) then
3:     state ← new                                ▷ Create state object for next domain name
4:     send_next_experiment(state)
5:     queue ← queue + state                        ▷ Append state to end of queue
6:   end if
7:   wait_for(DatabaseThread, SendThread, ProcessThread)
8:   process_incoming_responses()
9:   state ← pop_first(queue)
10:  if waiting_time_reached(state) then
11:    if experiments_finished(state) then
12:      state ← delete
13:    else
14:      send_next_experiment(state)
15:      queue ← queue + state
16:    end if
17:  else                                          ▷ Waiting time not yet reached
18:    queue ← state + queue                      ▷ Insert state at head of queue
19:    wait_for(DatabaseThread, SendThread)
20:    if is_full(queue) then
21:      sleep(1)                                ▷ Wait 1 second for response or timeout
22:    else
23:      state ← new
24:      send_next_experiment(state)
25:      queue ← queue + state
26:    end if
27:  end if
28: until all input domain names have been processed

```

| Query name | China | Iran |
|-----------------------|-------|------|
| 1. \$NAME | 383 | 14 |
| 2. www .\$NAME | 384 | 14 |
| 3. \$RANDOM.\$NAME | 368 | 1 |
| 4. \$RANDOM\$NAME | 167 | 1 |
| 5. \$NAME.\$RANDOM | 146 | 1 |
| Total | 404 | 14 |

Table 6.3: Granularity of blacklist.

The measurement resulted in 1024 domains for China and 14 domains for Iran that are blacklisted in at least one of the five requested domain name variants. Some of these blocked domains are the result of overblocking, where e.g. `x1.appspot.com` and `x2.appspot.com` are blocked because `$RANDOM.appspot.com` is blacklisted. After manual testing of the common suffixes, we reduced the results to 404 domain names blacklisted in China and 14 domain names blacklisted in Iran (Table 6.3). The results for China indicate that blocked domain names are in most cases also blocked when prepended with `www`. or a random subdomain. 21 domain names were blocked only if prepended with `www`. but not without it, e.g. `www.nytimes.com`. Furthermore, about half of the names are blocked if random prefixes are prepended without a separating dot, which results e.g. in case of `facebook.com` in an overblocking of `iamnotonfacebook.com`. The Iranian DNS blacklist is less extensive, in terms of total blacklisted names and the use of prefix or suffix wildcards.

6.3 Obtaining Bogus Addresses

As shown in Section 6.1, the DNS injection filters deployed return DNS responses with distinct IP addresses in the A resource record. With the list of bogus IP addresses it is easy to detect DNS responses spoofed by DNS injection. Iranian networks return always the same bogus IP address but Chinese networks return an address randomly chosen from a predefined set of bogus IP addresses. In this section, we present an efficient method to obtain the set of bogus IP addresses for one domain name.

6.3.1 Method

The idea of this method is to send DNS queries repeatedly to known DNS injectors until all bogus IP addresses have been collected. From r spoofed responses, we can collect n distinct bogus IP addresses. The challenge is to estimate when we have obtained the complete set of bogus addresses and stop sending further queries. Assuming that the whole set consisted

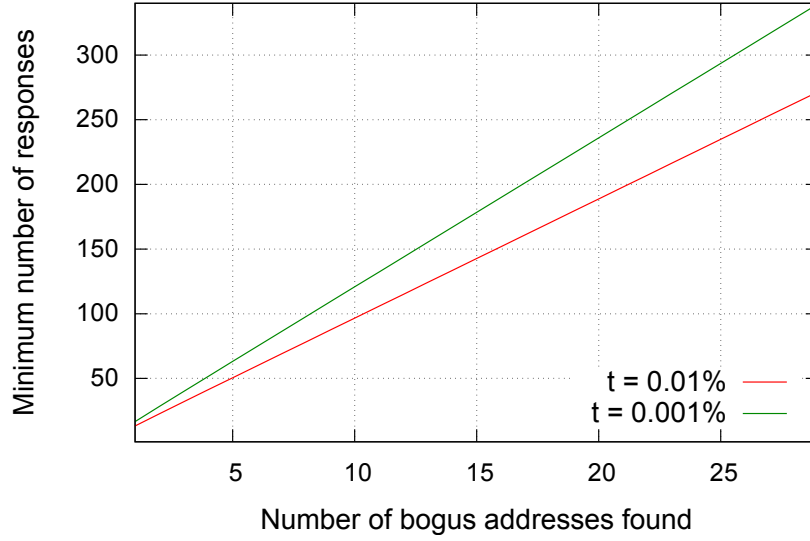


Figure 6.8: Number of required responses for given thresholds.

of $n + 1$ IP addresses, the probability p to miss one of the IP addresses in one response is

$$p = 1 - \frac{1}{n + 1} \quad (6.1)$$

and p^r in r responses, under the assumption that all n bogus addresses occur equally likely. This is a lower bound because if the whole set consisted of $n + 2$ or more IP addresses, then the probability to miss one of them would be even lower. We continue sending DNS queries until the probability to have missed a bogus IP address is below a predefined threshold t . The minimum amount of responses r required to reach the threshold t is thus:

$$r = \log_p(t) \quad (6.2)$$

We stop sending further queries once r responses have been received because the set of obtained IP addresses is complete at that point with a probability of $1 - t$.

Analysis

The minimum number of required responses r increases with each additional bogus IP address discovered. In Figure 6.8 two examples are shown for probability thresholds $t = 0.01\%$ and $t = 0.001\%$. The y-axis shows the minimum number of responses r for a given number of known bogus addresses n on the x-axis.

The underlying assumption in the calculation is a uniform distribution of bogus addresses in spoofed responses. As shown in Table 6.2, this is true in case of `facebook.com` but the distribution is unknown for other blocked domain names. We now discuss the

impact when the underlying assumption is not met for an example with a heavily askew distribution. We determined the actual distribution of bogus addresses for `twitter.com` with the large-scale probing method from Section 6.1. The probing yielded a cluster of eight equally distributed addresses, each occurring on average 76,495 times (± 185), and one address occurring 1,011,034 times. To calculate the worst case probability to miss one of these IP addresses with the above algorithm, we assume to have obtained eight bogus addresses in total and missed the least frequent one with 76,306 occurrences. For an anticipated $t = 0.01\%$ and with $n = 8$, the algorithm will send queries until $r = 79$ spoofed responses have been received. However, t is only valid for uniform address distributions. The actual probability to miss the least frequent address in this example is:

$$t' = \left(1 - \frac{76,306}{1,622,998}\right)^{79} \approx 2.23\% \quad (6.3)$$

While the actual failure probability is significantly larger than $t = 0.01\%$, it is an acceptable probability given that only 79 DNS messages needed to be sent. Determining the address distribution for blacklisted domain names is a high cost operation requiring significantly more than 79 DNS messages per domain name. As we have observed different address distributions for `twitter.com` in earlier measurements, it is also a volatile information. It is thus a reasonable heuristic to assume a uniform address distribution and to apply a safety margin to the threshold probability t to account for deviations.

6.3.2 Implementation

The implementation is an adaptation of the blacklist measurement software (Section 6.2.2). We omit a detailed description here because the conceptual differences to the blacklist implementation are minor.

Given a threshold t , queries are sent for each domain name under test until the calculated minimum number of responses r has been received. To disperse the network load, each query is sent to a different known DNS injector. If a query times out e.g. due to random packet loss, it is resent to another IP address. As with the blacklist test, we test for and ignore open resolvers.

6.3.3 Measurement Result

We applied the above measurement method with $t = 0.01\%$ to 404 domain names blocked in China. The list of known DNS injectors originates from the probing measurement in Section 6.1.3 and the list of domain names from the blacklist measurement in Section 6.2.

The number of bogus IP addresses per domain name ranges from 1 address after 14 responses to 19 addresses after 180 responses. As shown in Figure 6.9, most domain names

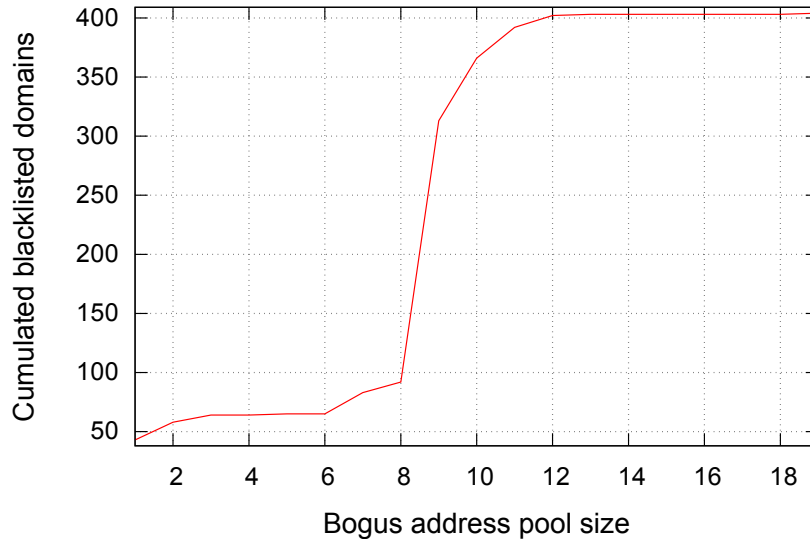


Figure 6.9: Number of bogus addresses per blacklisted domain name.

return a set of 9 bogus addresses. While a few bogus addresses are returned exclusively for one domain name, most are used for several blocked names. The most commonly returned addresses are the nine bogus addresses that are also used for `facebook.com` (Table 6.4, compare with Table 6.2). For example, `59.24.3.173` was returned for 257 domains. For one domain name the DNS filter returns a CNAME record (alias name) instead of an IP address, whereas the CNAME target does not seem to be blocked. In total, we obtained a set of 33 distinct IPv4 addresses from spoofed responses.

These IP addresses can be used for opportunistic detection of spoofed responses from DNS injection. An example usage scenario is a web browser extension checking resolved addresses against the list of well-known bogus addresses. With this method, it is possible to find further names of the domain blacklist if the names are returning one of the known bogus addresses.

6.4 Impact on Resolvers

In this section we analyze the impact of DNS injecting networks on unrelated resolvers from other networks. The basic idea is to query open resolvers worldwide for blacklisted domain names and check if the response is poisoned by DNS injection. Our measurement method uses the King technique [51] to send DNS queries between the open resolver and an arbitrary destination address (Figure 6.10). By using all authoritative name servers of a domain as destination addresses, we can test all paths between a resolver and all name servers of a domain name under test.

| Domain count | IPv4 address | Domain count | IPv4 address |
|--------------|-----------------|--------------|----------------|
| 1 | 123.50.49.171 | 91 | 23.89.5.60 |
| 1 | 209.85.229.138 | 91 | 249.129.46.48 |
| 1 | 74.125.127.102 | 91 | 253.157.14.165 |
| 1 | 74.125.155.102 | 91 | 49.2.123.56 |
| 1 | 74.125.39.102 | 91 | 54.76.135.1 |
| 1 | 74.125.39.113 | 91 | 77.4.7.92 |
| 2 | 211.5.133.18 | 120 | 189.163.17.5 |
| 29 | 202.106.1.2 | 225 | 203.98.7.65 |
| 29 | 202.181.7.85 | 225 | 46.82.174.68 |
| 29 | 203.161.230.171 | 228 | 243.185.187.39 |
| 29 | 209.145.54.50 | 228 | 8.7.198.45 |
| 29 | 216.234.179.13 | 228 | 93.46.8.89 |
| 29 | 64.33.88.161 | 229 | 159.106.121.75 |
| 30 | 4.36.66.178 | 230 | 78.16.49.15 |
| 91 | 118.5.49.6 | 249 | 37.61.54.158 |
| 91 | 188.5.4.96 | 257 | 59.24.3.173 |
| 91 | 197.4.4.12 | | |

Table 6.4: IPv4 addresses seen in bogus responses from Chinese DNS injection. The domain count indicates for how many different domain names (out of 404 names) the IPv4 address had been returned.

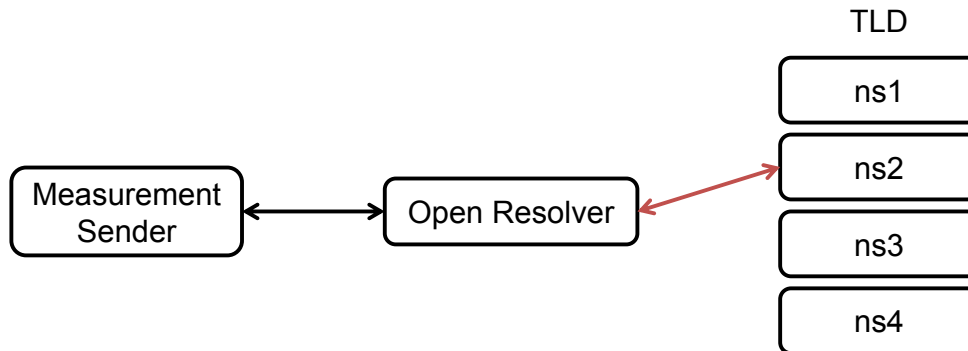


Figure 6.10: Test for DNS injection on the path between an open resolver and an authoritative name server of a top-level domain. The destination address can be chosen freely with the use of the King technique. This would not be possible with an ordinary DNS query because the destination address is otherwise chosen by the open resolver and unknown to the query sender.

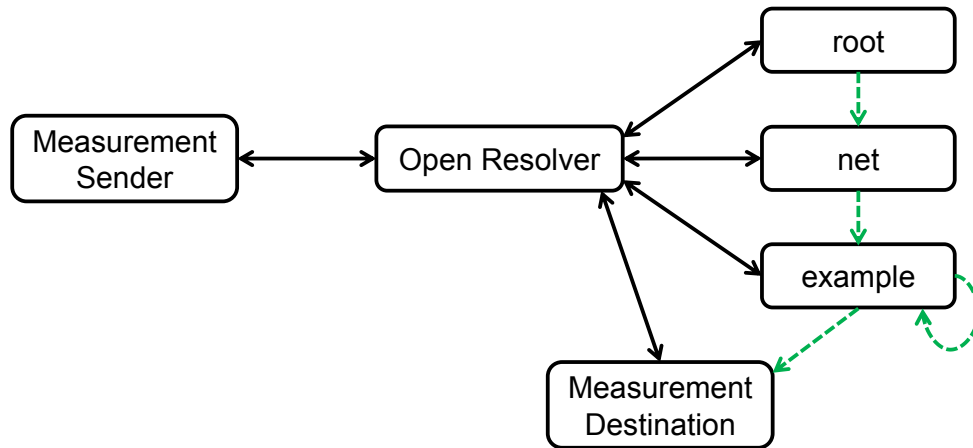


Figure 6.11: Domain delegations set up (dashed lines) and DNS messages sent (solid lines) for impact measurement.

6.4.1 Method

The objective is to send a DNS query between a resolver and an IPv4 destination address, e.g. 192.0.2.1. We set up a domain name—here referred to as `example.net`—with delegations in the form of:

```

experiment1.example.net. IN NS ns.experiment1.example.net.
ns.experiment1.example.net. IN A 192.0.2.1
ns.experiment1.example.net. IN AAAA 2001:DB8::DE1E:6A7E

```

When a resolver attempts to look up the name `domain.xy.experiment1.example.net`, it will follow the delegation chain as indicated in Figure 6.11. The solid black lines represent DNS messages sent over the network and the dashed green line represent delegations referring to a subzone on another name server. The name server authoritative for `example.net` refers the resolver to 192.0.2.1, to which the resolver will subsequently send the query for `domain.xy.experiment1.example.net`. As the destination is not configured as authoritative name server for the queried name, we expect a REFUSED error response from the name server to the resolver, which in turn gives a SERVFAIL error back to us. If `domain.xy` or any other part of the query name is filtered by DNS injection, then the resolver will receive one or more spoofed responses and return an A record with a bogus IP address to us. Hence, a negative result (no DNS spoofing) is indicated with SERVFAIL and a positive result (spoofed response) is indicated with an A record. By setting up several delegations (experiment 1, 2, ...), we can test the network paths between the resolver and arbitrary measurement destinations for occurrence of DNS injection. The measurement method requires `domain.xy` to be blacklisted with any random suffix. As evidenced by Table 6.3, this prerequisite applies to some but not all domain names blacklisted by DNS injection.

As shown above, each delegation also contains an AAAA record with an IPv6 address. This is not the IPv6 address of the measurement destination but instead is the address of our name server, pointing to itself. The reason for this seemingly useless delegation is the behavior of IPv6-capable resolvers: when the IPv4 measurement destination returns an error, the resolver attempts to look up the AAAA record of `ns.experiment1.example.net` to find a working IPv6 name server. This causes additional AAAA queries, which the resolver sends to the root, `net` and our name servers in order to resolve `ns.experiment1.example.net`. By putting a cacheable glue AAAA record into the delegation, an IPv6-capable resolver will send one additional IPv6 query to our name server, but none to the root and `net` name servers, reducing the overall network load.

6.4.2 Implementation

Again, we use the same measurement framework that has been used for the blacklist measurement (Section 6.2.2). Each open resolver is tested with all experiments, i.e. all root and top-level domain servers. Unanswered queries are sent again up to a total of five attempts. The timeout value also serves as idle time between two unanswered queries in order to cope with effects like congestion or momentary network outage at the destination. Therefore, the timeout and idle time increases from 30 seconds (first attempt) to 150 seconds (fifth attempt). Should an open resolver time out five times consecutively, we do not expect any further responses and abort the series of experiments for this open resolver.

6.4.3 Measurement Result

We applied the measurement method in July 2013 between 255k open resolvers and 1144 authoritative name servers to analyze the impact of Chinese DNS injection on foreign third-parties. We chose `www.minghui.org` (i.e. instead of placeholder `domain.xy` in the above examples) as blacklisted domain name because this name was filtered with any random suffix in our blacklist measurement (Section 6.2). The list of authoritative name servers comprises all 1144 servers that were authoritative for the DNS root zone or for any of 314 TLDs at the time of the measurement. We omit 11 name servers from this analysis that were authoritative for the three Chinese TLDs `cn`, `xn--fiqs8s`, `xn--fiqz9s` because we are interested in unwanted effects on third-parties—hosts located in China are naturally affected by DNS injection.

We selected the 255k open resolvers randomly from a list of around 25M open resolvers⁶. We actually attempted the measurement with 997,021 open resolvers, however, 709,446 open resolvers timed out repeatedly during the measurement. This was to be expected: open resolvers are often connected via dynamic links with ephemeral availability.

⁶List kindly provided by Jared Mauch at <http://www.OpenResolverProject.org/>.

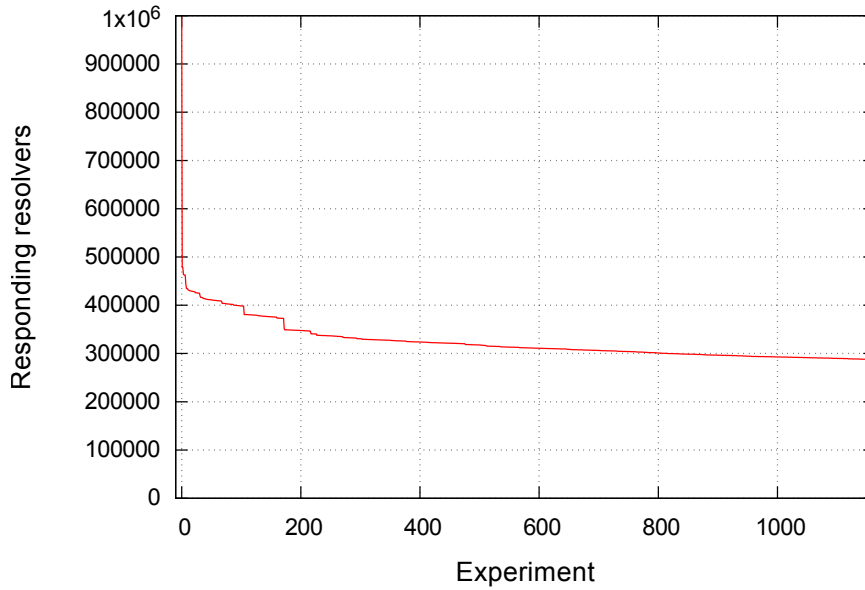


Figure 6.12: Decreasing resolver availability.

Most open resolvers had already been offline when the measurement started. This can be seen in Figure 6.12, which shows that most resolvers failed to respond with the first experiment. Another portion stopped responding during one of the later experiments. The graph decreases smoothly, except for some irregular drops which are a side effect of unreliable resolver implementations. TLD servers have usually a high availability but individual servers of smaller TLDs can show occasional downtimes. The common resolver behavior is to return SERVFAIL when the authoritative name servers have timed out. Some open resolvers, however, in this case do not return any response to the query sender and thus time out themselves. As the downtimes of the TLD servers vary over time, the timeouting open resolvers add up at different experiment numbers. Besides timeouts, 17,138 open resolvers did respond but failed to resolve two well-known domain names correctly. We omit these resolvers from analysis because their measurement results are unreliable. Out of 270,437 open resolvers with complete and usable measurement results, 15,435 were located in mainland China and as expected all of them were affected by DNS injection. We consider an open resolver as affected if it returns a spoofed response in at least one experiment. For further analysis, we consider the remaining 255,002 open resolvers outside of China, which should not be affected by a foreign Internet filter.

The tested open resolvers were located worldwide in 188 countries or regions. 15,225 resolvers (6.0%) from 79 countries were affected by Chinese DNS injection. The most frequently affected locations are shown in Figure 6.13. In absolute numbers (a), most affected resolvers were located in Romania (6836), Italy (3673), Iran (1146), Panama (701) and Indonesia (692). The absolute numbers are however biased due to uneven geographical

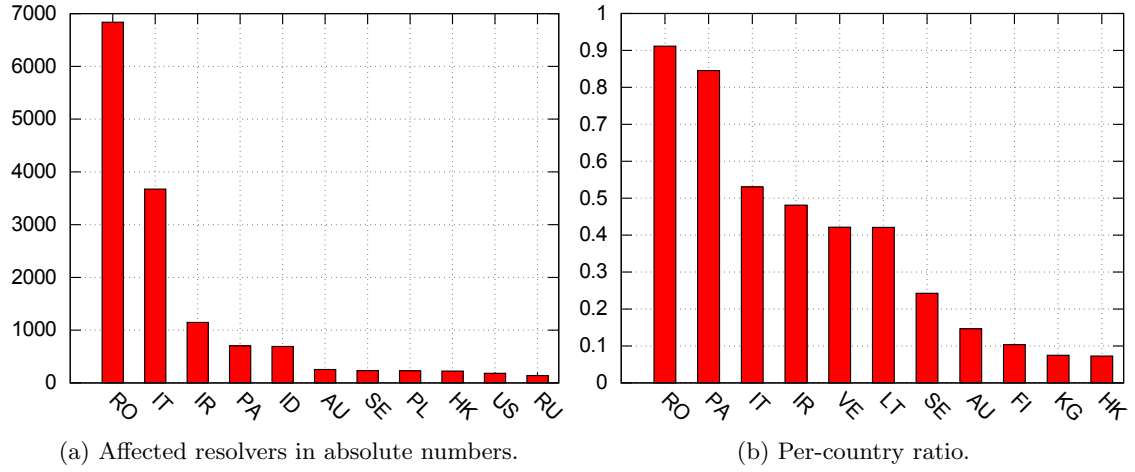


Figure 6.13: Locations of affected resolvers.

| Affected | Country | AS# | AS Name |
|----------|---------|-------|----------------|
| 6826 | RO | 9050 | Romtelecom |
| 3455 | IT | 3269 | Telecom Italia |
| 701 | PA | 11556 | C&W Panama |
| 263 | IR | 12880 | ITC Iran |
| 231 | SE | 3301 | TeliaSonera |
| 166 | IR | 48159 | TIC Iran |

Table 6.5: Most frequently affected networks.

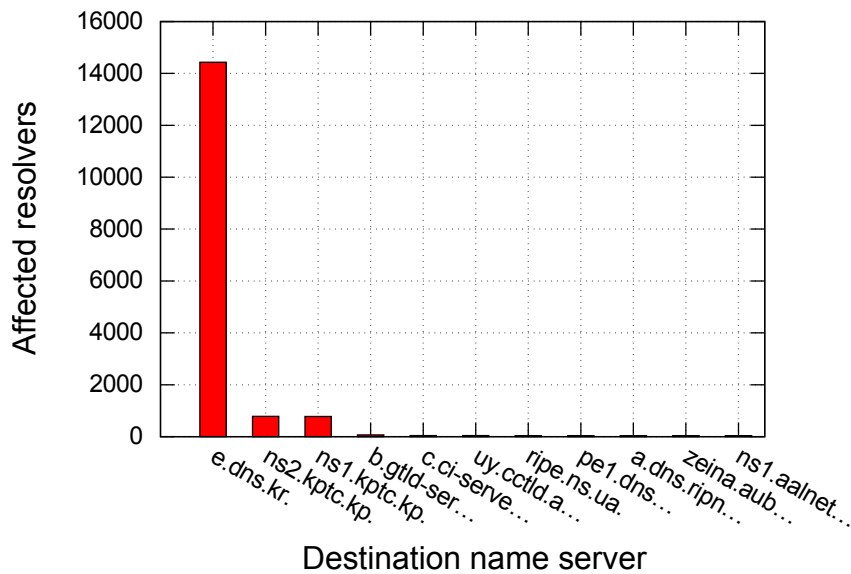


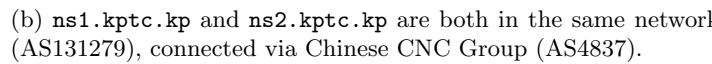
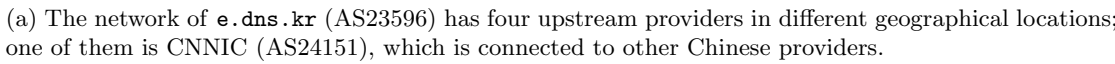
Figure 6.14: Most frequently affected authoritative name servers.

distribution of the tested resolvers. The per-country ratios of affected resolvers (b) are 91% for Romania, 85% for Panama, 53% for Italy and 48% for Iran (omitting regions with < 100 resolvers in the result set). Indonesia is affected only to a minor degree of 6%. Table 6.5 shows the most frequently affected networks, grouped by AS number. For most countries, the positive hits are essentially caused by one national network. This means a national network routed traffic to one of the measurement destinations through China whereas other networks—even those in geographical proximity—chose other routes.

The majority of spoofed responses were injected on behalf of one particular measurement destination. This can be seen in Figure 6.14: 14,431 resolvers (5.7%) received spoofed responses when sending queries to `e.dns.kr`. There are six authoritative name servers for the South Korean TLDs `kr` and `xn--3e0b707e`, but `e.dns.kr` is the only one which was affected by Chinese DNS injection. According to the operator KRNIC⁷, `e.dns.kr` is using anycast addressing (AS23596) with locations in Daejeon (South Korea), Beijing (China), São Paulo (Brazil) and Seoul (South Korea). This is also reflected in public BGP data from RIPE NCC (Figure 6.15a). Given the absence of injection for the other five name servers which are all hosted outside of China, this suggests that DNS injection for `kr` occurs only on paths to the anycast instance in Beijing.

DNS injection does not occur on all routes through mainland China, which can be seen in the results for the North Korean TLD `kp`. The two name servers for `kp` are both located in the same network (AS131279, Star JV). Public BGP data (Figure 6.15b) suggests that this network uses one Chinese upstream provider (AS4837, CNC Group) and no anycast

⁷KISA KRNIC: <http://krnic.or.kr/jsp/eng/dns/nameServer.jsp>



routing. In earlier studies AS4837 has been shown as censored network [10, 135]. Despite being routed through a Chinese network, only a minor portion of 794 (0.3%) resolvers were affected by DNS injection.

Apart from the three South Korean and North Korean name servers, the remaining 1141 measurement destinations did not show significant traces of DNS injection. There were around 30 resolvers affected per each name server but exemplary analysis shows that these resolvers are suffering from injection regardless of the measurement destination. These resolvers are located in random networks, suggesting that this is not a network issue but rather a host-specific issue. We investigate one of these resolvers in the following section.

6.5 Case Study: Unconditionally Affected Open Resolver

The purpose of the following case study is to understand why a small subset of resolvers seems to be affected unconditionally by Chinese DNS injection. We do so by sending queries to one resolver, hereafter called “TW-OR”, and attempt to find out whether TW-OR forwards queries to a destination in China. TW-OR was an open resolver located in Taiwan in AS3462 (Chunghwa Telecom). In Section 6.4, we collected results from 8613 Taiwanese open resolvers in AS3462, but only 5 of them (0.05%) were affected by DNS injection. Hence, AS3462 was not affected by a general routing issue. Nevertheless, TW-OR received spoofed responses in 1142 out of 1144 experiments, which suggests that all DNS queries of TW-OR were routed through China.

In the first experiment, we test the hypothesis that TW-OR forwards all DNS queries to a resolver in China. We sent 100 queries for a domain under our control to TW-OR and observe the incoming resolver IP addresses at our authoritative name server. Each query was for a slightly different subdomain name to ensure that the response was not cached. TW-OR does forward queries to another resolver, but despite our expectation of seeing IP addresses from Chinese networks, all 100 queries originated from IP addresses in TW-OR’s network (AS3462).

In the second experiment, we sent 100 queries to our domain prefixed with a blacklisted domain name, i.e. `www.minghui.org.1.verteilt.esysteme.net`. Again, the responses were not served from cache. In 79 cases TW-OR returned a spoofed response and we did not see any query at our authoritative name server. This suggests that TW-OR does indeed forward queries to a resolver in China over a poisoned path. In 21 cases TW-OR retrieved the correct response from our authoritative name server. Figure 6.16 shows two possible models: 1) DNS injection occurred within AS3462 between TW-OR and the recursive resolver that interacted with our authoritative name server (red line), 2) TW-OR forwarded DNS queries to a resolver in China, which in turn forwarded DNS queries back to AS3462 (green lines). Based on operational experience, both models are unsuitable: 1) TW-OR

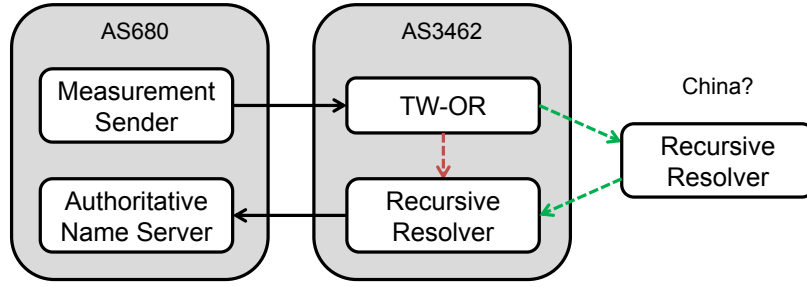


Figure 6.16: Two unsuitable models how DNS injection affects the resolver “TW-OR”: 1) within the autonomous system (red line), 2) forwarding out and back into the autonomous system (green lines).

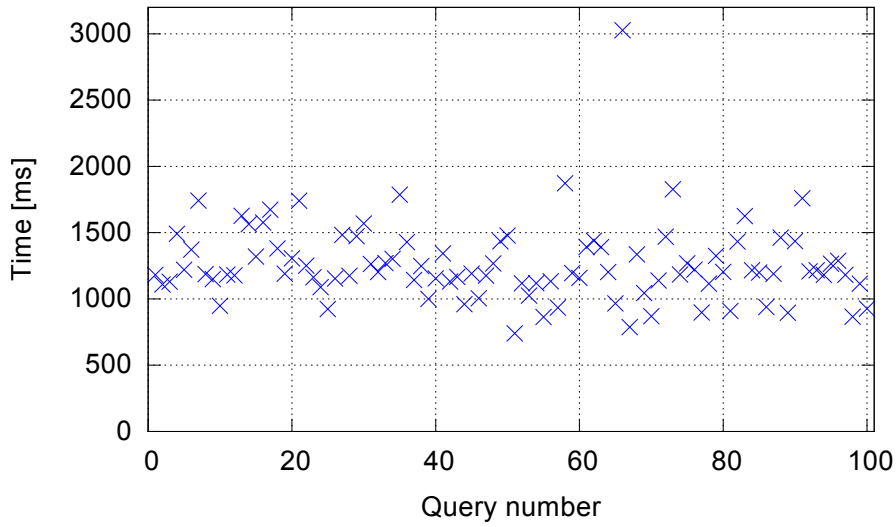


Figure 6.17: Time for resolving domain names in experiment 1.

would not route internal traffic through an external AS, 2) when forwarding DNS queries to an external resolver, this resolver would not forward queries to another resolver in TW-OR’s networks.

We now analyze the latencies of the above DNS transactions and then suggest a third model. The round-trip time (RTT) between our network and TW-OR was about 330 ms. A name resolution of our domain should take $2 \times \text{RTT}$: our network to TW-OR, TW-OR to authoritative name server, and back. The DNS transactions in the first experiment took in most cases significantly more than $2 \times \text{RTT}$, as shown in Figure 6.17. In the second experiment, spoofed responses took about $1 \times \text{RTT}$ and genuine responses about $2 \times \text{RTT}$, see Figure 6.18. These results could have been caused by the model shown in Figure 6.19: TW-OR forwarded DNS queries to a presumed recursive resolver in China. The Chinese resolver did not resolve the domain, because we did not see any query at our authoritative name server. Instead, we believe that the Chinese resolver issued an error to TW-OR with varying

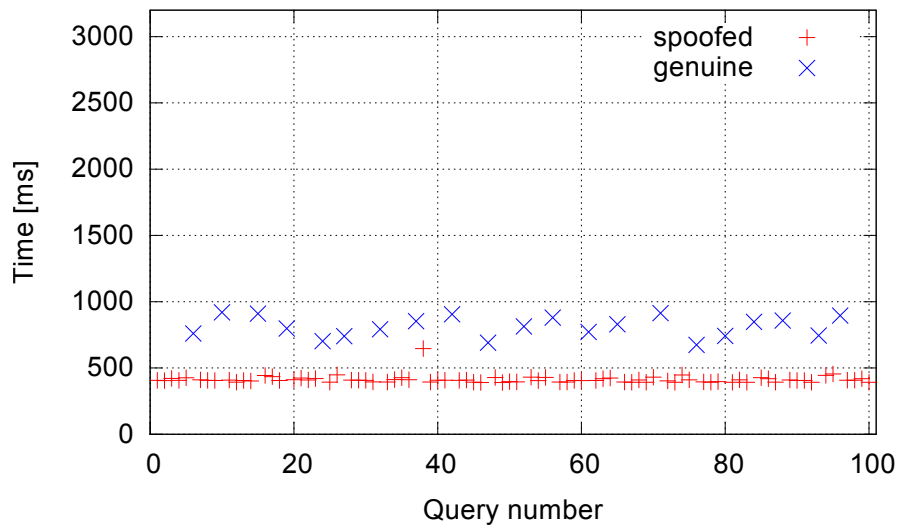


Figure 6.18: Time for resolving domain names in experiment 2.

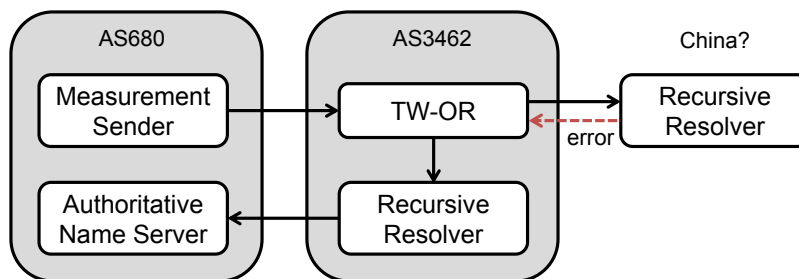


Figure 6.19: Possible model how DNS injection affects the resolver “TW-OR”.

delay. In experiment 1 TW-OR received the error and fell back to the resolver in AS3462, which resolved the domain name. In experiment 2 TW-OR received a spoofed response and returned it to us. This explains why the latency of spoofed responses was much lower than of genuine responses: spoofed responses did not require two full round-trips between our network and TW-OR. About one fifth of responses in experiment 2 were not poisoned and the latency of these responses was lower on average than in experiment 1. A possible explanation is that TW-OR in about one fifth of cases did not forward the DNS query to the presumed Chinese resolver, but chose directly to query the AS3462 resolvers. This would explain why the latencies in experiment 2 did not scatter as much as in experiment 1: TW-OR did not wait for the response of the presumed Chinese resolver. The discrepancy in the ratio of spoofed responses (81%) to the impact measurement in Section 6.4 (99.8%) lies in different measurement methods: here, TW-OR collected a successful response from our name server with a one-fifth chance but in the impact measurement TW-OR got an error with a one-fifth chance and then probably retried over the poisoned path. This shows that the impact measurement is fairly reliable in triggering spoofed responses when a poisoned path exists.

We do not have certainty whether the above model applies due to the lack of additional data. Yet, this is a probable model and explains why some open resolvers suffer from DNS injection while others in the same network do not. An open question is why TW-OR has been set up at all to forward DNS queries to China. This may have been the result of a malware similar to “DNSChanger” or an erroneous network configuration.

6.6 Case Study: DNS Injection in Hong Kong

We would have expected that there is a cluster of foreign third-parties affected by DNS injection in Southeast Asia. Yet, in the impact measurement in Section 6.4 most positive hits were not in geographical proximity to China. In the following case study, we investigate whether two large Hong Kong-based network carriers are affected by DNS injection. The purpose is to find out whether a measurement from inside the networks yields a different result than the impact measurement in Section 6.4.

Hong Kong is a Special Administrative Region of People’s Republic of China. Due to its history as a former British colony and China’s constitutional practice of “one country, two systems”, Hong Kong adopts its own constitutional basic law. The Hong Kong Basic Law guarantees freedom of speech [97] and is not subject to the Internet filtering practice of mainland China.

| PCCW | | | CSL | | |
|---------|------------|----|---------|------------|---|
| Attempt | Experiment | | Attempt | Experiment | |
| | 1 | 2 | | 1 | 2 |
| 1. | 65 | 24 | 1. | 48 | 8 |
| 2. | 36 | 3 | 2. | 34 | 0 |
| 3. | 32 | 1 | 3. | 32 | 0 |
| 4. | 32 | 0 | 4. | 32 | 0 |
| 5. | 32 | 0 | 5. | 32 | 0 |

Table 6.6: Number of timeouts while waiting for responses for `facebook.com` (Experiment 1) and `www.minghui.org` (Experiment 2).

6.6.1 Method

We send DNS queries for `facebook.com` (experiment 1) and `www.minghui.org` (experiment 2) to various measurement destinations. If the path between our vantage point in Hong Kong and the measurement destination is poisoned, then we will receive a spoofed response. As we are using name servers as measurement destinations, we expect a DNS response in any case, even though it can be an error message. In case the response is missing, pending queries are resent after > 30 seconds for up to a total of five attempts per server.

6.6.2 Measurement Result

Part 1: Root and Top-Level Domains

In our first measurement, we used 1174 IPv4 root and top-level domain servers as of November 2013 as measurement destinations. We performed the measurement with two carriers: 1) AS4515 (PCCW), accessed over a hotel Wi-Fi, 2) AS38819 (HKCSL), accessed over 3G mobile broadband. For each carrier, the measurement took 10 minutes and processed around 2500 DNS transactions. Although some servers timed out repeatedly, the radio links did not have any detrimental effects on the accuracy of the results. This can be seen in Table 6.6: the number of timeouts converges with both carriers to 32 in experiment 1, implying that 32 name servers do not respond for reasons beyond our measurement method. Those servers were omitted and hence the number of timeouts converges to 0 in experiment 2.

We identify spoofed messages via the IP address in the answer, as we have obtained the set of bogus IP addresses earlier with the measurements in Section 6.1 and Section 6.3. For both carriers, the only bogus responses were spoofed on behalf of name servers that were authoritative for one of the Chinese TLDs `cn`, `xn--fiqs8s` or `xn--fiqz9s`. There was no indication of DNS injection for any other measurement destination.

Part 2: Second-Level Domains

For the second measurement, we used the authoritative name servers of the Alexa list of 1 million most popular websites [5]. Retrieving and resolving the name servers of each domain name yielded a list of 256,251 distinct IPv4 addresses. We abstained from running such a broad measurement with CSL to avoid overstress of the 3G mobile network. With PCCW, the measurement took 40 minutes and finished for all but 17,123 servers (6.7%) that had timed out repeatedly. We received DNS responses from 4133 server addresses (1.6%) that indicated Chinese DNS spoofing. An additional 7 addresses indicated Iranian DNS spoofing. Exemplary analysis suggests that a major portion of these addresses were servers in China and Iran and thus naturally affected by DNS injection.

In order to determine whether there is any anomalous result, we repeated the measurement at about the same time in a different network. The control measurement ran in AS680 (DFN, Germany) and took about 20 minutes. 15,212 servers (5.9%) timed out, which is less than in the PCCW measurement. Although this suggests that our Wi-Fi measurement suffers from packet loss, the influence is rather low and does not distort the result all too much. In the control measurement, there are 4154 server addresses (1.6%) affected by DNS spoofing from China and 7 addresses affected from Iran. The numbers of both measurements being close together implies that the Hong Kong-based PCCW is not unusually affected from DNS injection, despite its proximity to DNS injectors in China.

6.7 Related Work

DNS Injection in China

Various aspects of the Chinese GFW have been studied extensively in earlier work. We limit the scope of this literature review to work that discusses DNS injection. In 2007 Lowe et al. [87] probed 1607 open resolvers in China whether their responses differ from resolvers in the U.S. Almost all Chinese resolvers consistently returned bogus responses for a list of 393 apparently filtered domain names. Lowe et al. noticed that DNS spoofing occurs on router-level, and that the spoofed responses contain anomalous ID and TTL values in the IPv4 header. Although the GFW is known to tamper with DNS since at least 2002 [137], to our knowledge this is the earliest report about router-level DNS spoofing. The bogus responses collected by Lowe et al. referred to a set of 21 distinct answer IP addresses, which is a subset of the 33 IP addresses that we collected in our measurement in Section 6.3. Zittrain and Edelman [137] suggested as rationale for diverting names to a few external IP addresses that these IP addresses may be blocked on border routers. As IP addresses of domains change regularly, it is easier to maintain a domain blacklist than to maintain an IP blacklist.

Brown et al. [28] assessed how many networks have seen a route to the Beijing-based anycast instances of the F-, I- or J-root server. They determined that BGP routers in several AS worldwide saw paths to Beijing at least once in 2010, especially in Asia and South America. This indicates the potential for choosing a poisoned route, though it is unclear whether any DNS traffic was actually routed through the GFW.

An anonymous study [10] examined in 2011/2012 the potential of collateral damage that Chinese DNS injection could cause. Our probing of the IPv4 name space in Section 6.1 and the impact measurement in Section 6.4 are reappraisals, inspired by the anonymous authors findings. While the anonymous authors found 388,988 IPv4 /24 subnets that indicated DNS spoofing in November 2011, we attributed 960,078 responding /24 subnets to Chinese DNS injection in February 2014. Although theoretically possible that the affected subnets have more than doubled during that time, it is rather likely that our setup (Section 6.1.2) is less prone to packet loss. This shows the extent of Chinese DNS injection, which is affecting a major portion of the IPv4 address space.

Furthermore, the anonymous authors have shown that the **de** and **kr** TLDs were affected by Chinese DNS injection. The measurement method was to query 43,842 open resolvers outside of China for `domain.xy.RANDOM.tld`, where `domain.xy` is a blacklisted name, `RANDOM` a random string and `tld` the top-level domain under test. The destination name server is expected to respond with an NXDOMAIN error, except when the response is spoofed and refers to a bogus address. With this method, it is not possible to determine which name server of the tested TLD the query is sent to. To increase the likelihood of testing all name servers of a TLD, the test has been repeated 200 times with different random strings, requiring 62,400 queries per resolver to test all 312 TLDs at that time. The method we used in our measurement is not per domain but per name server. As we can control which name server is being queried, 1155 queries per resolver suffice to test 317 TLDs and to identify which of the name servers is affected. A disadvantage of our measurement method is that it elicits REFUSED errors at the destination name server, as opposed to NXDOMAIN errors. Both errors are harmless for the server, but REFUSED errors are rather unusual and thus more likely to be noticed by server administrators. After being notified of abnormal traffic from two server administrators, we refrained from repeating the measurement with the TLD name servers.

In our results, only one TLD name server showed a major occurrence of DNS tampering. Thus, we can confirm that **kr** was still affected in 2013 but only for one name server for which an anycast instance is located in China. In 2012, Peter Koch [74] confirmed that **de** was affected by DNS tampering by utilizing 1762 RIPE Atlas measurement probes: 339 probes (19%) were routed to an anycast instance in China, out of which 218 (12%) were affected by DNS injection. There was no indication of tampering for anycast locations of **de** other than Beijing. In our mid 2013 measurement, the **de** name servers were no longer

affected, including `a.nic.de` which still comprised an anycast instance in Beijing at that time. This suggests that operators of global anycast name server networks can confine the effect with careful routing configuration.

Topology of the Great Firewall of China

Joss Wright [132] suggested that the GFW is not a homogenous filter but manifests regional variance in DNS filtering. Wright’s measurement results show timeouts and erroneous responses, which cannot have been caused by the GFW with today’s knowledge, but also show a subset of the bogus IP addresses that we collected. The authors of the anonymous study mentioned above [10] investigated the network origins of spoofed responses with a Traceroute-like measurement. They identified 3120 IPv4 router addresses in 39 Chinese autonomous systems as source of DNS injection, dominated by the two large ISPs Chinanet and CNC Group.

In an attempt to understand the topology of the GFW, let us consider a study about the injection of TCP RST packets. TCP RST injection is akin to DNS injection, used as part of the GFW to interrupt HTTP connections with unwanted keywords. In 2011, Xu et al. [135] located the placement of filtering devices in China. They identified that most filters overall—but not all—are placed in border networks to filter international HTTP traffic. The large ISP Chinanet pursues a different strategy and places most of its filters in provincial networks.

We would have assumed that the same infrastructure would be used for DNS injection and thus the DNS injectors would exist in the same locations. Yet, according to an anonymous study in 2014 [11] all DNS injectors are located in border networks, topologically close to foreign networks. This implies that DNS injection is not used for filtering domestic traffic and emphasizes its use as a national firewall technique. Furthermore, the anonymous authors explored the internal structure of a DNS injector node by exploiting side-channel information derived from the ID and TTL fields in the IPv4 header. They characterized one node in one location as a network tap, which distributes DNS traffic to about 367 independent processes (running on an unknown number of hosts) for processing and packet spoofing. Load balancing is based on source and destination IPv4 address. This node injects an estimated amount of 2800 spoofed responses/s on average, which varies with a typical diurnal pattern.

DNS Injection in Iran

Given the quantity of literature about Chinese Internet filtering, there are few studies about Iranian Internet filtering. We found in Section 6.1 and Section 6.2 that Iranian DNS injection can be observed from foreign networks. This leads to the question whether Iranian

DNS injection could affect third-parties accidentally, similar to Chinese DNS injection. Collin Anderson [6] describes Iran as largely isolated from the rest of the Internet; the country created a national information network in IPv4 address space 10/8 that is only accessible within the country. This is reflected in our measurements in the use of the address 10.10.34.34 in spoofed responses. Being an unroutable private address allocation for most users worldwide, Iranian users will access a filter web page. International traffic is routed through two autonomous systems (AS12880 and AS6736) with ties to the Iranian government [7]. Topologically speaking, Iran is a network sink for international traffic. It is unlikely that international traffic is transiting through Iran and thus safe to assume that Iranian DNS injection does not affect foreign third-parties at all.

Halderman et al. [16] measured Internet filtering from *inside* of an Iranian network and showed a pattern of spoofed packets similar to our observation in Figure 5.4. The spoofed HTTP 403 responses are sent to the inside client, while the spoofed TCP RST packets are supposed to be sent to an outside server. In our measurements in Section 6.1 and Section 6.2 the client was outside, which may have confused the filter implementation and caused both packet types to be sent to the outside client. Halderman et al. further describe that most blocking is realized with HTTP-based filtering and very few domain names are filtered by DNS tampering. This explains why we found only 14 blocked domain names, although a lot more websites are reported to be blocked in Iran.

6.8 Summary and Conclusion

In this chapter, we studied the effects of Internet filtering by DNS injection from vantage points outside of the censoring networks. After probing a representative set of 14.5M public IPv4 addresses, 960k addresses (6.6%) indicated DNS injection in China and 11k (0.08%) in Iran. Chinese DNS injection can be observed for a much larger part of the public IPv4 address space, simply due to the fact that the number and size of Chinese networks is larger. We did not observe evidence for large-scale DNS injection in other countries from vantage points in AS680, AS24940 and AS24961. This may be due to our choice of domain names selected for probing or because different filtering methods are used, which are not observable outside of the affected networks.

DNS Injection in China

Chinese blacklists have a varying granularity; most names are blocked with any prefix and some names are also blocked with any suffix. This will e.g. in case of filtering `facebook.com` result in an overblocking of `iamnotonfacebook.com`. Bogus answer addresses for blocked domain names are returned randomly from a fixed set of IP addresses. We presented an algorithm for obtaining the current set of bogus addresses efficiently and with a decent

probability of being complete. After application of this algorithm we collected 33 bogus IP addresses, most of them returned for several blocked domain names. There was no evidence for a cooperation between the network operators of the bogus IP addresses and the operators of the GFW.

After testing 255,002 open resolvers outside of China, we determined that 6% are potentially affected by Chinese DNS injection when querying top-level domains outside of China. The majority was only affected on a single path: `e.dns.kr` which is an authoritative name server for the South Korean TLDs `kr` and `xn--3e0b707e`. One of the anycast instances of `e.dns.kr` is hosted in Beijing, China. There was no evidence for DNS traffic with destinations outside of China to be systematically affected by Chinese DNS injection.

DNS Injection in Iran

We found DNS injection to be used in Iran for blocking access to 14 domain names, including `facebook.com`, `plus.google.com`, `twitter.com` and `youtube.com`. During our measurements, we observed a significant drop of spoofed responses from Iranian networks in mid 2013. This coincided with a report from The Guardian in July 2013 about the newly elected President of Iran Hassan Rouhani who spoke out to loosen filtering of social media [38]. However, around September or October 2013 the DNS filters were set up to block social media again. The incident reflects the political uncertainty in Iran about whether social media should be blocked or not.

Detection and Protection

Iranian and Chinese DNS injection can be detected via the IPv4 addresses that are returned in spoofed DNS responses. The Iranian filter returns always 10.10.34.34, while the Chinese filter returns a random address out of a small set of addresses. We presented an efficient method to obtain the set of bogus addresses from the Chinese DNS filter. These addresses can be used for passive opportunistic detection of DNS filtering, e.g. in a DNS resolver or in a web browser. Such an IP address-based detection can be combined with other criteria for detecting spoofed responses, e.g. TTL value in the IPv4 header and round-trip times as suggested by Duan et al. [41].

Once detected, spoofed responses can be dropped by the client-side packet filter or DNS resolver. As Chinese DNS injection does not tamper with genuine DNS messages, the DNS filter can thus be bypassed. This not possible with the Iranian DNS filter, which drops original DNS queries. Bypassing the DNS filter from inside of an Iranian network thus requires to hide the DNS query, e.g. with a VPN or Tor.

DNSSEC can be used to remediate the unwanted effects of DNS injection on unrelated third-parties, if fully deployed on client and server side. It is not enough to rely on a

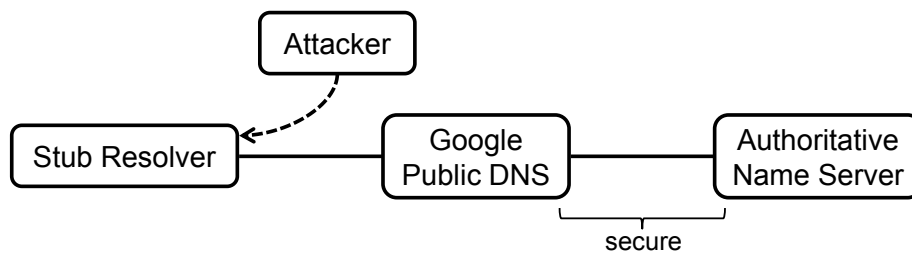


Figure 6.20: A validating recursive name server does not supersede DNSSEC validation on the end host.

DNSSEC validator in a different network, which is illustrated in Figure 6.20. One of the auxiliary experiments in the open resolver measurement in Section 6.4 was to determine whether the resolver rejects domain names with invalid DNSSEC signature. 26,170 (10%) of the open resolvers indicated DNSSEC validation by rejecting a crafted invalid signature. These resolvers could have been immune against DNS injection leaking from third-party networks. However, another experiment indicated that 18,092 (7%) of them relied on the validating Google Public DNS service. It is unclear whether the resolver validates on its own and thus whether the path between the resolver and Google Public DNS is protected. Please refer to Chapter 9 for an in-depth analysis about the adoption of DNSSEC validation.

Although DNSSEC makes it more difficult to implement DNS filtering, the Chinese TLDs `cn`, `xn--fiqs8s` and `xn--fiqz9s` are DNSSEC-signed since November 2013. Apparently the TLD operator CNNIC hopes to benefit from DNSSEC in other ways, e.g. protect from computer crime motivated DNS spoofing. We reviewed the possibilities of running DNSSEC validation in conjunction with DNS alteration in Section 5.5. CNNIC suggested another approach, which is called *weak trust anchor* [82]. When a weak trust anchor has been set up, e.g. for `cn`, a DNSSEC resolver performs opportunistic validation: signed DNSSEC responses are validated, but in case the response is missing the resolver retries with insecure DNS. CNNIC argues that the purpose is to cope with interoperability problems when DNSSEC responses get dropped due to lack of support for long DNS messages. However, weak trust anchors could also be used to implement DNS filtering despite DNSSEC support. The major drawback of weak trust anchors is that downgrade attacks become possible not only by the ISP but also by other attackers. Weak trust anchors are not part of the DNSSEC specification suite.

The DNS community is well-aware of Chinese DNS injection and the adverse effect it can have on third-parties. Care should be taken when name servers are supposed to be hosted in China or Iran. Even if the ISP does not spoof responses, one of the upstream providers might do.

CHAPTER 7

Attacking the NSEC3 Privacy Goal

NSEC3 has been specified to thwart zone enumeration attacks (Section 3.7.3). In addition to secure denial of existence, NSEC3 records shall not disclose existing domain names. In this chapter, we explore attacks against the NSEC3 privacy goal. We describe an algorithm for retrieving NSEC3 hashes (Section 7.1) and three methods for NSEC3 hash breaking: brute-force attack, dictionary attack and Markov attack (Section 7.2). The methods are designed to run on Graphics Processing Units (GPUs) that are used in consumer-grade graphic cards. We chose GPUs as platform because they are considerably more efficient for this type of attack than CPUs are, which is shown in the evaluation in Section 7.4.

7.1 Hash Crawling

The first part of an NSEC3 zone enumeration attack is to retrieve all NSEC3 hash values from the server. DNS clients cannot query directly for NSEC3 records and instead have to query for random non-existing names until all NSEC3 records have been retrieved. Dan Bernstein proposed an algorithm [23] which sends queries to the server only for those names which yield a new NSEC3 record. The approach is sketched as Algorithm 2 and comprises choosing random names and calculating their hash values. If a name is found whose hash value is not covered yet by any of the retrieved NSEC3 ranges, a DNS query will be sent to the server. The server will respond with a non-existent name (NXDOMAIN) error and return an NSEC3 record which can be added to the set of known NSEC3 ranges. The algorithm terminates as soon as all retrieved NSEC3 ranges form a closed circular chain.

The loop in lines 9 to 12 is the computationally expensive part, which we offload to the GPU for parallel computation. For performance reasons, we are keeping track of aggregated gaps between NSEC3 records instead of actual NSEC3 records, i.e. the inverse of the hash space covered by NSEC3 ranges found so far. The check in line 12 is a binary search over the sorted list of gaps in logarithmic time. Another performance optimization is in the

Algorithm 2 NSEC3 hash crawling

```

1: function CRAWL( $z, s, i$ )                                ▷ zone name, salt, iterations
2:   var
3:      $R \subseteq \mathbb{N} \times \mathbb{N}$ 
4:      $x \in \{'-', '.', '0' \dots '9', 'a' \dots 'z'\}^{len}$ 
5:      $y \in \{0 \dots 2^{160} - 1\}$ 
6:   end var
7:    $R \leftarrow \{\}$                                           ▷ Crawled NSEC3 ranges, set of tuples
8:   repeat                                                  ▷ Until all NSEC3 ranges are known
9:     repeat                                              ▷ Until a new range has been found
10:       $x \leftarrow \text{genRandomName}() \parallel z$ 
11:       $y \leftarrow h(x, s, i)$ 
12:      until  $\forall (r_0, r_1) \in R : y < r_0 \vee y > r_1$ 
13:       $R \leftarrow R \cup \text{queryNSEC3}(n)$ 
14:    until  $\forall (r_0, r_1) \in R : \exists a, b \in R : r_0 = a_1 \wedge r_1 = b_0$ 
15:    return  $R$ 
16: end function

```

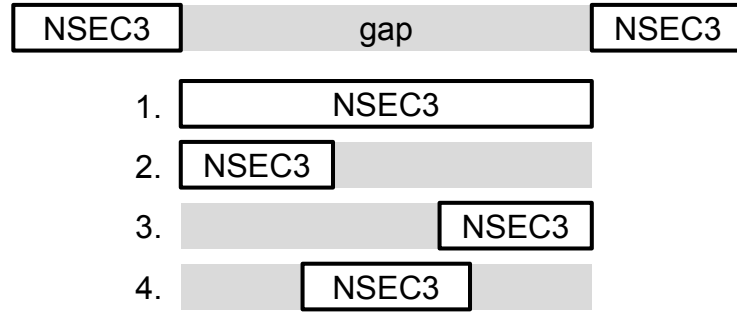


Figure 7.1: Four cases for a gap to be cut by a newly retrieved NSEC3 range.

genRandomName function, which iterates linearly through names starting from a randomly chosen name. Random input names are not necessary because small changes in the input name will result in seemingly random output hashes due to the avalanche effect of the underlying SHA-1 hash function.

The query for a non-existent name results in a new NSEC3 record, which is written to a database and is used to update the sorted list of gaps in memory. The new NSEC3 record closes a gap or cuts a part of it (Figure 7.1). However, we also have to consider that names on the server may have been added, removed or renamed in the meantime. This happens several times a day for large TLDs and obsoletes part of the NSEC3 records crawled so far. If a retrieved NSEC3 record overlaps outside of the boundaries of an expected gap, we will remove the affected, obsolete NSEC3 record from our database and update the gap data structure accordingly. Furthermore, different authoritative name servers may serve different versions of zone data because zone updates are not synchronized precisely. This

can lead to loops with continuous DNS queries and data updates. In case of inconsistent NSEC3 records, we use the SOA serial number from the negative DNS response to identify fresh NSEC3 records and drop old ones.

The worst case complexity of our algorithm is unbound due to unpredictable output hash values. However, we provide averages for a case study in Section 7.4. The network traffic caused by the algorithm is linear to the number of names in a zone. For every NSEC3 record one query is sent, plus an insignificant amount of extra queries for handling inconsistent NSEC3 records after zone updates. An extra query is necessary if the question tuple (name, class, type) matches an existing record, i.e. when `genRandomName` yields an actually existing name. During testing and evaluation, we have been using AAAA as question type and never collided with an existing record.

Once the crawling completed, the attacker has a local copy of all NSEC3 records, which consist of hash values of all domain names used in the zone. The NSEC3 records also comprise the information about which record type exists for each domain name. If capable of reversing the hash values, the attacker will be able to subsequently iterate through the known set of question tuples and query for (name, class, type) with varying name and type. Alternatively, the attacker can query each name for (name, class, ANY) to retrieve all record types of a domain name with one transaction. The result will be a full copy of the zone.

7.2 Hash Breaking

After having crawled the NSEC3 records of a DNS zone, the next part is to find the corresponding cleartext names. We discuss three GPU-based hash breaking methods in this section. All methods can be executed offline, i.e. communication with the name server is not required. Each of the attack methods consists of a performance-critical loop that generates a candidate name, computes the NSEC3 hash and checks whether the resulting hash value matches one of the crawled NSEC3 hashes. The attack methods differ on how they generate a candidate name but the remaining steps are the same.

The crawled NSEC3 hash values are stored as sorted array in GPU memory. The candidate name is hashed with the given salt and number of iterations; the server returns both values with each NSEC3 record. A binary search is used to check whether the resulting hash value is an element of the sorted array. Since random global memory access limits the performance of GPU applications, we also use a bloom filter to detect misses more efficiently. A bloom filter is a probabilistic data structure to efficiently check if an element is part of a set. It is created by using a bit array of length N and setting k bits in it for every element of the set. Since the positions of these k bits are specific to each element, a query has to check k bits in the worst case. If one of the checked bits is zero, the element is definitely

not in the set. On the other hand, if all k bits are set, the element might be in the set. In this case, binary search will be used to get a definite answer. Using this hybrid scheme, one memory access is sufficient in most cases to recognize hash misses. Since the vast majority of all checked candidate names are misses, this optimization provides a speedup of about 300% (varies subject to domain parameters and computing hardware).

Compared with NSEC3 hash crawling, the hash breaking requires significantly more computation time. Distributed computing can be used to spread the workload among several hosts. To support distributed computing, each candidate name in the search space must be addressed by an index number, so that slices of the search space can be distributed to different hosts. This is trivially possible with the brute-force attack and dictionary attack, and there are Markov-based algorithms for efficient enumeration by an index number [96]. Thus, hosts can jump into the middle of the search space without having to enumerate previous candidate names.

7.2.1 Brute-Force Attack

A brute-force attack enumerates all names of a specific length (aaa, aab, aac, ...) and checks whether their hash value is an element of the array of crawled NSEC3 hashes. While this attack is exhaustive and thereby guaranteed to find every present name of the specified length, it is computationally feasible only for names up to a certain length. The characters used in the brute-force attack can be limited to a set of about 37 to 39 characters. Although the DNS protocol supports arbitrary octets in domain names, only few are in practical use. Unicode characters used in internationalized domain names (IDN) are encoded to this limited ASCII character set.

7.2.2 Dictionary Attack

While brute-force attacks are expensive up to the point of being infeasible for long names, dictionary attacks allow for a quick trial of promising candidate names. In a dictionary attack the name generation step boils down to looking up a name in a static list.

Using a dictionary alone to break hashes can only yield as many results as it has entries. We increase the number of candidate names programmatically to find more results. The dictionary wordlist is inflated by inserting strings from an insertion wordlist into each dictionary word. For every pair from both lists we put the insertion word at every position of the dictionary word. The number of total candidate names is

$$|D| + |I| \sum_{w \in D} 1 + \text{len}(w), \quad (7.1)$$

where D is the dictionary, I is the insertion wordlist and len gives us the number of char-

acters of a word. We use the most frequent n-grams of the dictionary as insertion words, because the combination of a word and a common n-gram often yields another promising candidate name. This is demonstrated in the evaluation in Section 7.4.

7.2.3 Markov Attack

Markov chain-based approaches use a language model to estimate how probable a word is [90]. For every character in a word, the occurrence probability is determined based on the previous characters. Multiplying each character probability yields the probability of the whole word. The order of the Markov chain is here defined by the number of previous characters considered in the probability calculation. The language model is generated in a training phase, in which wordlists are used to calculate transition probabilities. In our approach, we use a *first-order Markov chain* to prevent overtraining, i.e. character bigrams. If the order is chosen too high, a Markov attack will be basically a dictionary attack.

Using this language model it is possible to efficiently enumerate all words down to a defined probability threshold [96]. Parameters for this algorithm are the maximal word length and the minimal word probability. The enumerated words are not sorted by their probability, so the maximal word length and the minimal word probability have to be chosen in such a way that it is feasible to process all resulting candidate names in a reasonable time—otherwise highly probable candidate names might not be enumerated in a given time frame. The number of words to be enumerated can be calculated efficiently with given thresholds for word length and word probability. An attacker can thus find appropriate thresholds subject to the computing power in terms of checked words per time and a given time span [90].

We used the names found by the brute-force and dictionary attacks to train the Markov model. That way we do not have to assume that registered names are derived from some natural language. Instead, the language model is based on a subset of actually registered names in the DNS zone.

7.3 Implementation

The NSEC3 crawling and hash breaking is implemented in Python and OpenCL. Figure 7.2 shows the system architecture. The Python code running on the CPU comprises the overall program logic. It sends domain queries to authoritative name servers and invokes the OpenCL *kernels*, which run on the GPU for computationally intensive tasks. Results are saved in a PostgreSQL database.

The program flow of the NSEC3 hash breaking is shown in Figure 7.3. The Python program creates a bloom filter for the set of NSEC3 hash values, and passes the hash set and bloom filter to the GPU. An OpenCL kernel starts corresponding to the type of attack

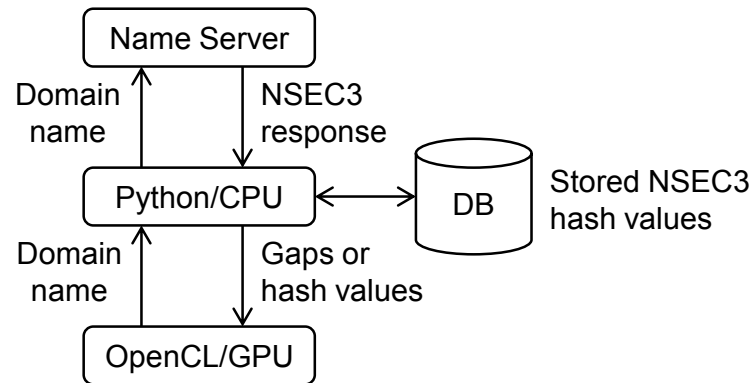


Figure 7.2: System architecture of NSEC3 attack.

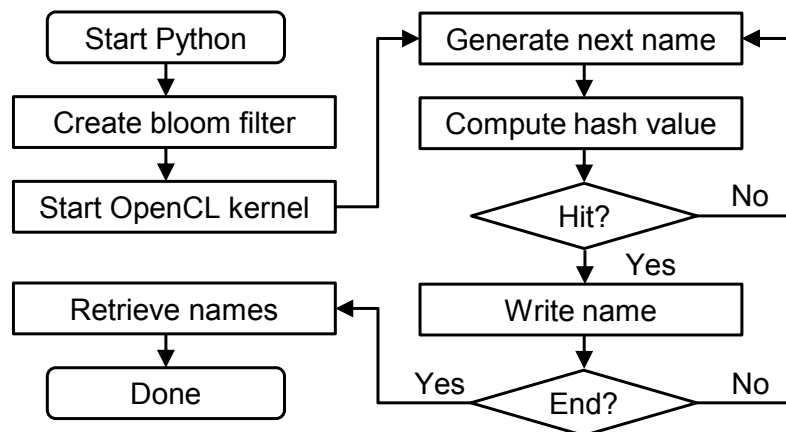


Figure 7.3: Hash breaking program flow on CPU (left column) and GPU (right column).

(i.e. brute-force, dictionary, Markov). The search space is split into chunks, which are processed in parallel by OpenCL *work-items*. Each work-item generates the next candidate name, hashes it and checks whether the candidate hash value matches one of the input hash values. The hit check consists of one to four bloom filter lookups, and a binary search over the ordered input hash set. The parameters of the bloom filter are chosen such that the first lookup will already be a miss in most cases, thus limiting the number of global memory accesses. All work-items are independent of each other except for competing global memory accesses. Work-items hold the program state in private memory except for the input buffers (e.g. around 70 Mbytes in our case study in Section 7.4) because they exceed the size of the private memory (less than 256 kbytes).

The program code of the crawler, the brute-force attack and the dictionary attack are written from scratch. The brute-force implementation uses two different functions for the generation of candidate names. First, a function to map a numerical index to a word by using a number system conversion algorithm:

$$\text{getNameByIndex} : \text{index} \mapsto \text{word} \quad (7.2)$$

This allows GPU workers to efficiently determine their starting candidate name. Second, a dedicated incrementation function to map a candidate name to the next candidate name:

$$\text{getNextName} : \text{word} \mapsto \text{word}_{\text{next}} \quad (7.3)$$

This avoids unnecessary index conversions because in most cases only the rightmost character changes, i.e. is incremented within the character set.

The Markov attack is an OpenCL port of Simon Marechal’s implementation in C for the password cracker *John the Ripper* [90]. To lower the memory requirements of our implementation, we use a reduced set of 37 characters instead of ASCII or arbitrary 8-bit octets. With this optimization it is possible to keep some lookup tables in fast local memory of the GPU and to avoid global memory access to a certain degree.

7.4 Evaluation

The rationale for using OpenCL was to utilize AMD graphic cards which have been reported to work faster than NVIDIA cards with 32-bit integer operations [121] like SHA-1 uses. To test this assertion, we hashed $37^6 = 2.5$ billion names with various iterations on a quad-core Intel 2.67 GHz CPU, an NVIDIA GTX 690 and an AMD HD 7970. Figure 7.4 shows that the HD 7970 needs the least time to finish. For an iteration count of $i = 0$, the CPU computed 12 Mhash/s, the NVIDIA GPU computed 270 Mhash/s and the AMD GPU computed 667 Mhash/s.

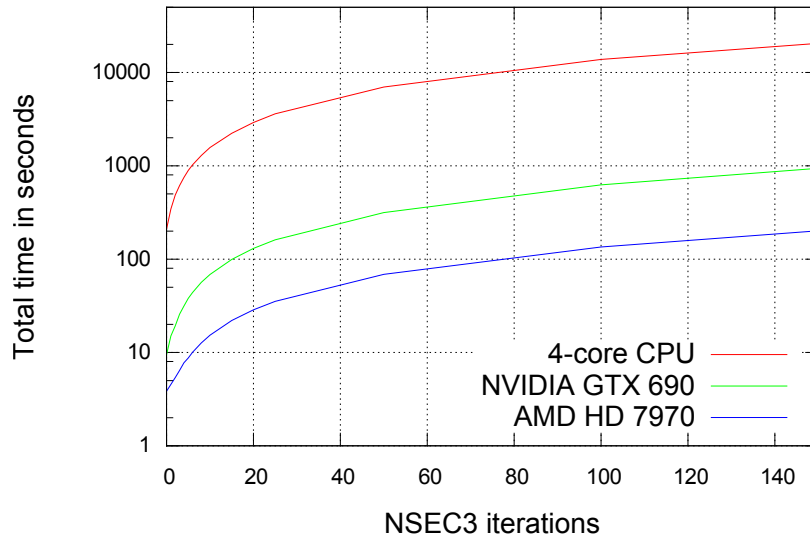


Figure 7.4: Time needed for hashing 2.5 billion names (less time is better).

We now evaluate our proposed attack methods by applying them in a case study on the `com` TLD. We chose `com` as reference because it is the largest and most used TLD in terms of registered domains (> 100 million). `com` is using opt-out and had about 345k DNSSEC-signed domains in May 2014, which implies the same number of NSEC3 hash values in the zone. The number of additional iterations is below average (cf. survey in Section 8.1) with $i = 0$, but we argue in Section 7.5 that the attack is also applicable on higher iteration counts. There is no salt set up for `com`, which is meaningless for this study because the salt does not offer any protection from a one-time attack (Section 3.7.3).

7.4.1 Hash Crawling

Recall that the hash crawling algorithm (Section 7.1) consists of keeping track of gaps and finding NSEC3 records that fill the gaps. The number of hashing attempts needed for a certain chance to find a new NSEC3 record can be derived from the sum of sizes of gaps. For example, when the sum of sizes of gaps constitutes 50% of the overall hash space, there is a 50% chance that the next hashing attempt will cut a gap. With 10% gaps, $\frac{1}{0.1} = 10$ hashing attempts will be needed on average. With each NSEC3 record found, the sum of sizes of gaps strictly decreases and the average number of hashing attempts needed to find a gap strictly increases, except when resolving inconsistencies after zone updates (occurred 5 times).

An implementation detail is that we pass only the 1000 leftmost gaps to the GPU. Working with many gaps increases the computation time of certain implementation parts, e.g. for assembling the gap buffer when passing it to the GPU and for the binary search over

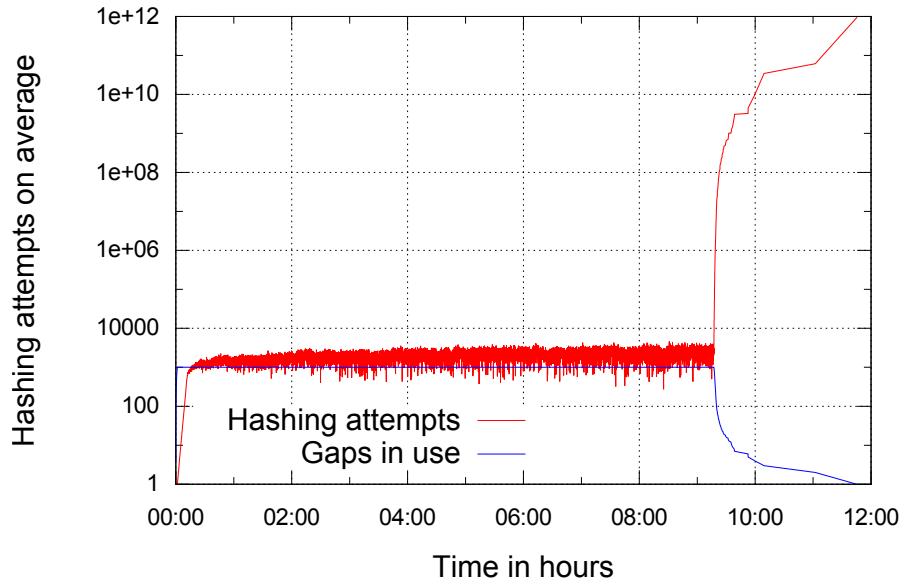


Figure 7.5: Hashing attempts to find a new NSEC3 record in `com`. This is a calculated average based on the actual, recorded gaps.

the list of gaps. However, 1000 gaps suffice to find a new NSEC3 record in an insignificant amount of time, even with CPU computation. This can be seen in Figure 7.5: the number of necessary hashing attempts oscillates around an average of 2059 (± 671) when the number of gaps in use is capped at 1000. Two thousand SHA-1 hashes can be computed in less than 1 ms. The number of hashing attempts needed does not increase significantly until the number of pending gaps falls clearly below 1000. After 12 hours of hash crawling, we retrieved 345,664 NSEC3 hash values from the `com` TLD. This process could be sped up further by sending and processing network queries in parallel. However, this would put additional strain on the `com` servers, so we decided to send queries one by one.

7.4.2 Hash Breaking

The numbers of names found by each of the three attack methods are shown in Figure 7.6. The time has been measured with one AMD HD 7970 GPU. We were using a 37 character set for all attacks ('-', '0'...'9', 'a'...'z'). The underscore character and multi-label names (e.g. `label11.label12.com`) are omitted because they are not used in the `com` zone (and seldom in other TLD zones).

The **brute-force attack** finished in less than 5 minutes for all words with up to 7 characters and found 32,795 names. The brute-force attack for 8 and 9 characters took 1.4 hours and 51 hours. The attack becomes infeasible for longer names on a single GPU because the computing time multiplies with 37 for each additional character. Overall, the brute-force

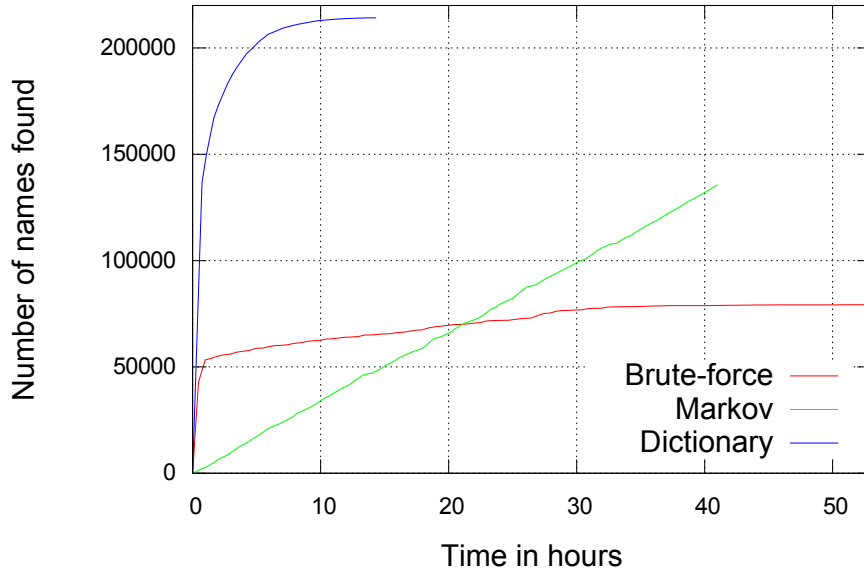


Figure 7.6: Comparison of attack methods.

attack for up to 9 characters checked 1.3×10^{14} candidate names and found 79,243 names (22.92% of NSEC3 hashes).

The success of the **dictionary attack** depends on the quality of the dictionary. We extracted domain labels from the Alexa [5] and Quantcast [33] lists of top 1 million websites and similar lists into a common dictionary. Another data source was the list of reverse domain names from the Carna botnet [27]. Note that the list of reverse domain names (IP address to domain name mappings) does not by far comprise all domain names. The majority of reverse names are structured names that would increase the dictionary size without providing gain, e.g. names like `ip-1-2-3-4.example.net` that differ only in numerical labels. We processed the list of reverse domain names with regular expressions to remove structured names and keep mostly alphabetic words. In total, the dictionary consists of 7.1 million words. Based on this dictionary, we created a list of insertion words consisting of the 200,000 most common n-grams with $n=1$ to $n=15$. This led to 1.7×10^{13} candidate names, out of which 214,181 names were found after 14 hours of computation (61.96% of NSEC3 hashes).

Interestingly, the dictionary attack has found all but 1725 names (2.17%) that the brute-force attack had found. This suggests that the quality of the dictionary is high, at least when used for the `com` TLD. The use of the insertion wordlist is also a very effective mechanism: the number of names found increased from 40,045 with the plain list by a factor of ≈ 5.34 . As shown in Figure 7.7, the most frequent n-grams are also the most effective insertion words. Nevertheless, even the less frequent ones contribute to the number of names found. This suggests that a larger list of insertion n-grams would increase the yield of names found.

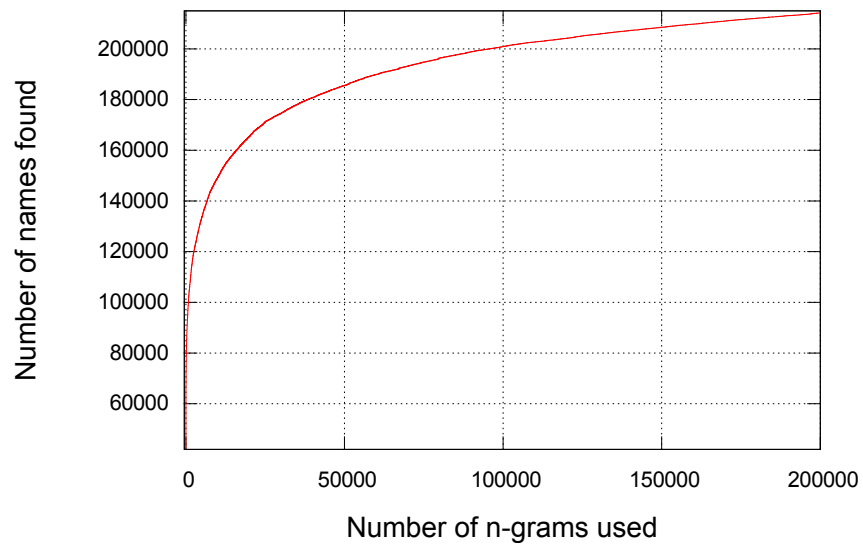


Figure 7.7: Effectiveness of n-grams as insertion words. N-grams are sorted by their frequency in the input dictionary (most frequent first).

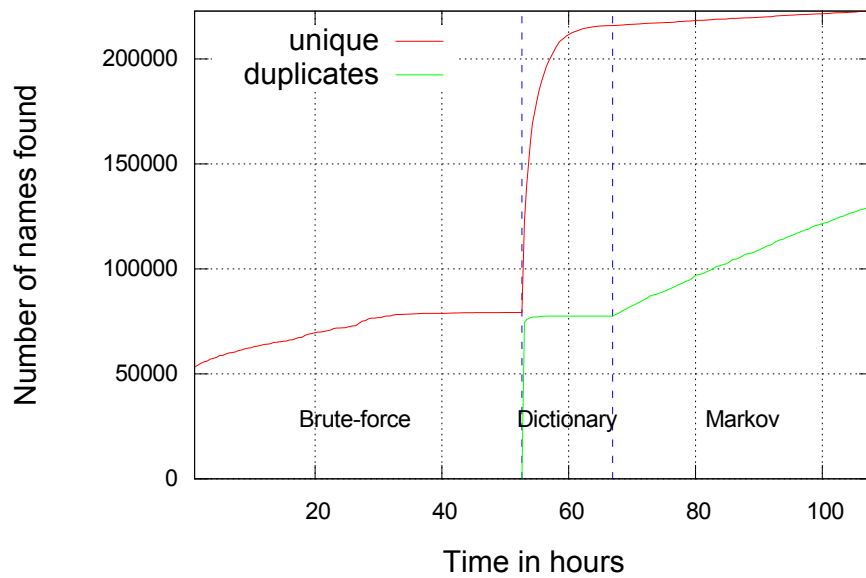


Figure 7.8: Names found over time.

The **Markov attack** is the slowest of the three methods due to sophisticated program logic and extra memory accesses. The brute-force attack computes more than 700 Mhash/s with long-running jobs, the dictionary attack computes 350 Mhash/s and our Markov implementation computes 120 Mhash/s. We trained the language model with the names found so far by the brute-force and dictionary attacks. It took 41 hours to compute 1.3×10^{13} candidate names, which lead to 135,574 names found (39.22% of NSEC3 hashes).

However, these are only the statistics for each attack itself. We have to consider duplicate results found by more than one method. Figure 7.8 shows the number of unique names found, and additionally the number of duplicate results. Overall, 222,749 NSEC3 hashes (64.44%) were broken after applying the three attacks. The dictionary attack was the most efficient method.

7.5 Discussion

In case of `com`, NSEC3 has protected the remaining 35.56% of domain names that had not been found within 5 days of GPU computation. The name server operator pays for this privacy protection with operational costs in terms of additional network traffic and CPU load for hashing. As explained in Section 3.7.6, NSEC3 is more expensive than NSEC. The server-side costs arise continuously for all negative responses during normal operation, not just during zone enumeration attacks. Furthermore, the operator has to scale their server capacity based on the maximum load that the servers are expected to handle. Even with few negative responses in normal operation, the operator must provision resources for potential peaks of NSEC3 responses.

An attacker attempting zone enumeration has to compute a vast amount of NSEC3 hashes to find the correct domain names. The workload is significantly higher on attacker-side than on server-side, but the computation can be performed in parallel batches on cost-efficient consumer-grade hardware. For example, the AMD Radeon HD 7970 that we used cost 360€ in 2013. Using a spare 5 year old desktop machine this setup had an average power consumption of about 279 watts, which would have cost us at home about 8.43€ for 5 days of computation.

The NSEC3 specification [81] acknowledges that NSEC3 is susceptible to dictionary attacks and advises to adjust the iterations to increase the hashing workload for attackers. However, this increases the NSEC3 overhead at about the same ratio, i.e. doubling the workload for attackers doubles the server-side CPU overhead for operators. This is different than e.g. increasing the key size of a cipher, where the workload increases slightly for legitimate users but exponentially for attackers. An attacker with one GPU can compensate for a doubling of iterations with an investment of another GPU (plus electricity). A slight increase of iterations, e.g. from $i = 0$ to $i = 7$, is a minor increase in protection, roughly

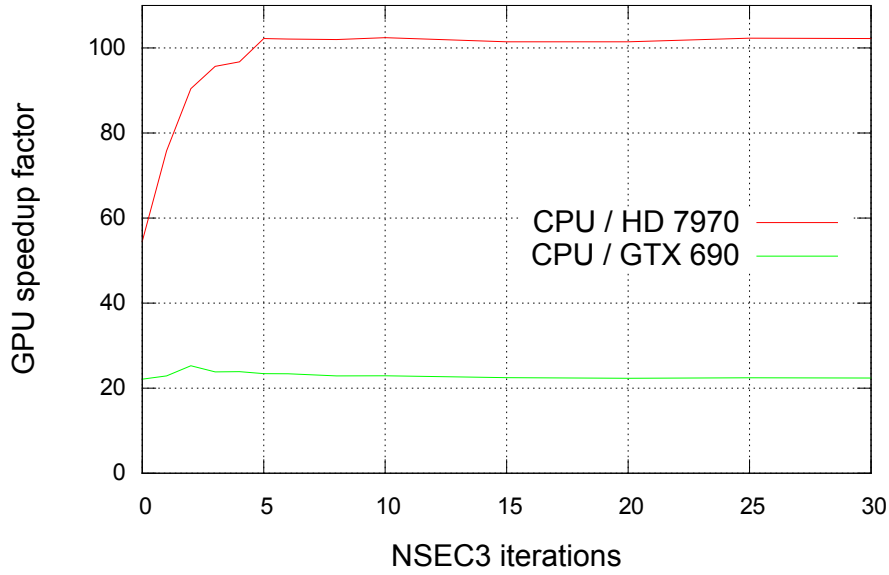


Figure 7.9: Performance comparison CPU vs. GPU.

comparable to switching from 512-bit to 515-bit RSA. A large increase, e.g. from $i = 0$ to $i = 100$, centuples the hashing overhead of NSEC3 for server operators.

Increasing the iterations from low values even improves the relative efficiency of a GPU, which is shown in Figure 7.9. The HD 7970 GPU is about 55 times faster than a quad-core 2.67 GHz CPU with $i = 0$, but 100 times faster with $i \geq 5$. The utilization improves until saturation because the stream processors are performing more parallel hashing work between global memory accesses. Although it depends on individual operational costs and percentage of negative responses, it is thus likely that operators of busy servers pay more for increasing iterations than attackers do.

7.6 Related Work

Dan Bernstein [23] implemented a CPU-based proof of concept of NSEC3 zone enumeration and inspired this work about GPU-based attacks. GPU-based NSEC3 hash attacks are two orders of magnitude faster than CPU-based attacks.

Rafiee et al. [103] proposed IPv6 penetration testing as use case for NSEC3 zone enumeration: where scanning an IPv6 network is not feasible, IPv6 host addresses can be retrieved from second-level or lower levels of domains. Furthermore, Rafiee et al. describe that 60% of domain labels on third-level are 9 characters or shorter (e.g. **www**). Even without a dictionary, these names can be easily broken with a brute-force attack with one GPU.

Goldberg et al. [50] proposed NSEC5, which replaces the SHA-1-based hash function in NSEC3 with an RSA-based keyed hashing scheme. The cryptographic key pair used for

keyed hashing is different than the DNSSEC zone signing key. Hence, NSEC5 prevents zone enumeration without exposing the zone signing key on authoritative name servers. The performance impact on authoritative name servers is comparable to online signing of minimally covering records (see fundamentals in Section 3.7.7), i.e. significantly higher than iterated SHA-1 hashing.

7.7 Summary and Conclusion

We presented an attack analysis on the privacy goal of the NSEC3 authenticated denial of existence method that is used in DNSSEC. After applying the attack method on the `com` TLD, we conclude that GPUs are very efficient for NSEC3 hash breaking: 64.44% of all NSEC3 hashes of the `com` TLD could be reversed after 5 days of GPU computation. Our findings suggest that increasing the hash iterations on the DNSSEC server puts the attacker at an economic advantage. This is due to 1) an improvement of the relative efficiency of a GPU compared with a CPU when increasing iterations, and 2) because hash breaking attacks can be performed offline on consumer-grade hardware but defensive measures run online on server-grade infrastructure.

We recommend zone administrators to use NSEC by default, except when the NSEC3 privacy assertion or the NSEC3 opt-out feature (Section 3.7.4) explicitly justifies the extra server costs. The *crawler* and *nsec3breaker* tools¹ we developed can be used as part of a cost-benefit assessment. The tools allow to assess the effectiveness of an NSEC3 zone enumeration attack with a given set of resources; the effort necessary for reversing hash values corresponds to the privacy benefit that NSEC3 offers to zone administrators.

¹<http://dnssec.vs.uni-due.de/nsec3>

Part III

Adoption of DNSSEC

CHAPTER 8

Server-Side Adoption

In this chapter, we analyze the server-side adoption of DNSSEC. We first survey the use of DNSSEC in top-level domains over a 20-month observation period, including the key size, frequency of key rollovers and parameters used for authenticated denial of existence. Then, we apply the NSEC3 attack from Chapter 7 to get an as complete list of signed second-level domains as possible. We gather DNSSEC parameters from the second-level domains, survey their algorithms and key sizes and quantify broken DNSSEC configurations.

8.1 Top-Level Domains

In order to survey the adoption of DNSSEC for top-level domains, we processed the public IANA root zone file¹ and probed the authoritative name servers of all TLDs for various record types. The observation period was from April 2013 to February 2015 with four probings per day. Timeouts were handled by resending the query to a randomly chosen server of the TLD under test for up to a total of 10 attempts; truncated messages were handled by falling back to TCP.

Figure 8.1 shows the number of TLDs over time. In April 2013, there were 317 TLDs, out of which 105 (33%) had deployed DNSSEC. We define DNSSEC as deployed when there is a secure domain delegation at the parent, i.e. a DS record in the root zone. The number has increased steadily since then, except for 11 test TLDs, which have been removed from the root zone on October 31, 2013. As of February 2015, there are 828 TLDs in the root zone, out of which 647 (78%) have deployed DNSSEC. The vast increase is due to the introduction of new generic TLDs, which began in October 2013 and is still ongoing. 516 new domains originate from the new generic TLD program; 6 more new domains are internationalized domain names (IDN) delegated in addition to country-code TLDs. All new generic TLDs are deployed with DNSSEC because registry operators are bound by contract

¹<http://www.iana.org/domains/root/files>

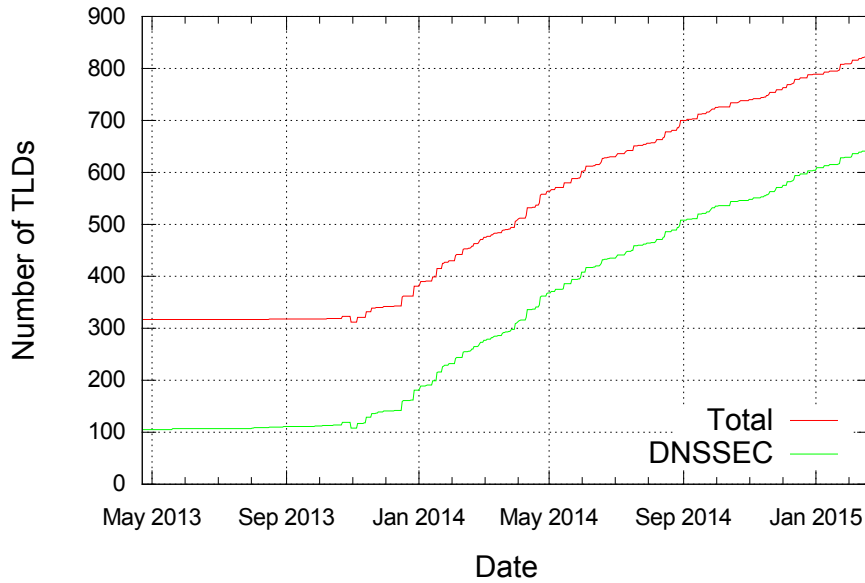


Figure 8.1: TLDs over time (April 2013 – February 2015).

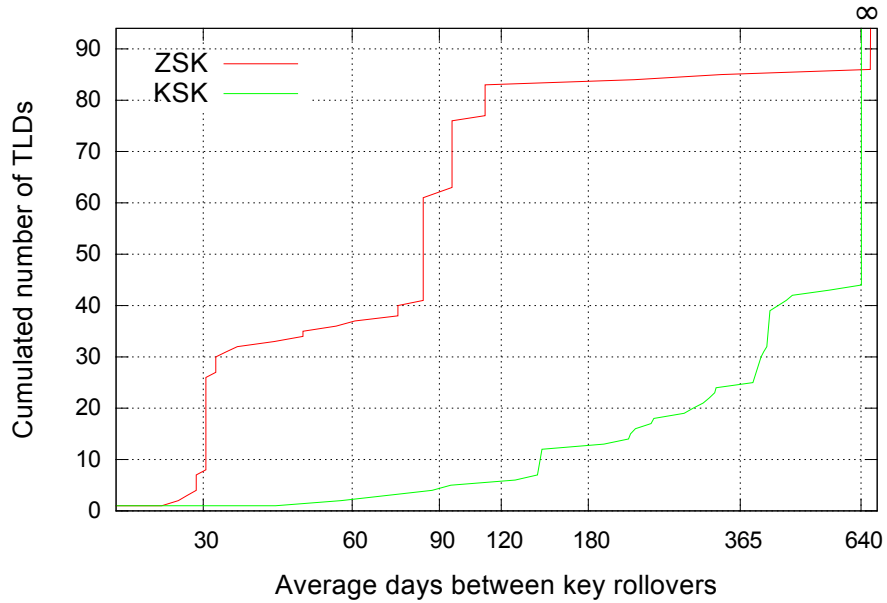
| KSK | | ZSK | |
|--------|-------|--------|-------|
| Length | Count | Length | Count |
| 1024 | 1 | 1024 | 594 |
| 1280 | 4 | 1032 | 1 |
| 2048 | 772 | 1048 | 2 |
| 2056 | 1 | 1152 | 3 |
| 4096 | 5 | 1280 | 225 |
| | | 2048 | 32 |

Table 8.1: Bit lengths of RSA moduli used by top-level domains (February 2015).

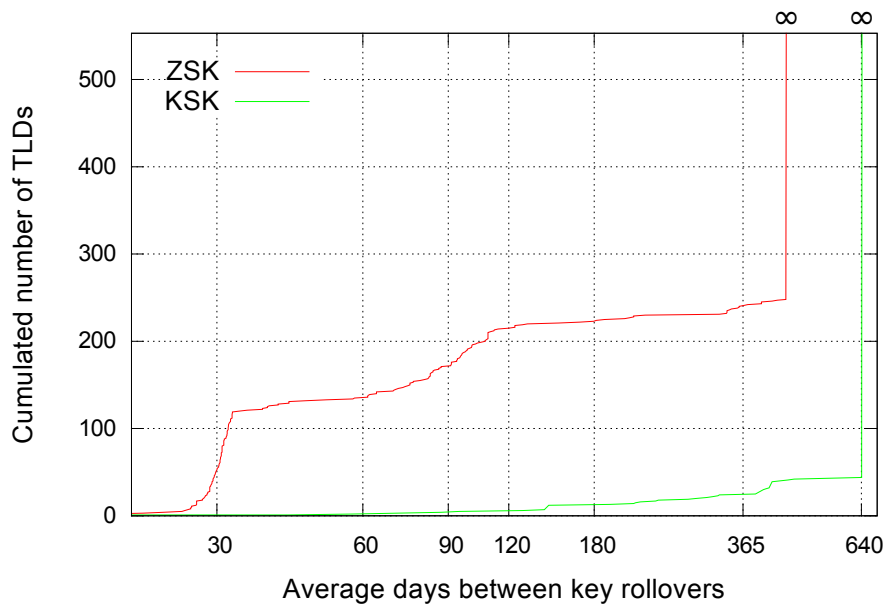
with ICANN [64]. There is a minor discrepancy between signed and securely delegated domains: 8 TLDs serve DNSSEC signatures without a secure delegation. This happens usually during a testing phase before DNSSEC is activated in the root zone.

8.1.1 Public Keys

Of the 647 TLDs that deployed DNSSEC, all domains use a KSK/ZSK scheme with RSA as cryptosystem. Table 8.1 shows the key sizes in use, separately for KSKs and ZSKs. Note that the total number of keys is larger than the number of TLDs due to key rollovers or stand-by keys that are in the zone but not used for signing. The bulk of TLDs use a 2048-bit RSA modulus as KSK. 1024 bits is the most frequently used size for ZSKs. While 2048-bit RSA is considered secure, the use of 1024-bit keys is disputed (Section 5.8). There



(a) TLDs that were signed before April 2013.



(b) TLDs first signed after April 2013.

Figure 8.2: Frequency of key rollovers in top-level domains (April 2013 to February 2015). The rightmost data point means that keys were not rolled over.

| KSK | | ZSK | |
|--------------|-------|--------------|-------|
| Exponent | Count | Exponent | Count |
| 3 | 15 | 3 | 15 |
| 65,537 | 762 | 65,537 | 834 |
| $2^{32} + 1$ | 6 | $2^{32} + 1$ | 8 |

Table 8.2: Public RSA exponents used by top-level domains (February 2015).

is also a major portion of 1280-bit ZSKs. Almost all 1280-bit ZSKs can be attributed to ARI Registry Services, which is an infrastructure provider used by many new generic TLDs.

We now analyze the average key lifetime to determine the time frame for breaking the above keys. Figure 8.2 shows the frequency of ZSK and KSK rollovers. The results are grouped into two different sets: a) TLDs that have been signed for the whole observation period, b) TLDs that have been signed for part of the observation period, in particular newly introduced TLDs. Domains that did not perform a key rollover are represented by the rightmost datapoint. Most TLDs from the first set replaced the ZSK every 30 to 100 days; 43 TLDs also replaced the KSK at at least once during our 20-month observation. 7 TLDs from the first set did not replace their 1024-bit RSA ZSK. Key rollovers are less common in the second set. Of those TLDs that replaced keys, most rollovers happened as well every 30 to 100 days. About half of the TLDs from the second set did not replace the ZSK during our observation. Although part of this result is distorted by newly-created TLDs that existed for a short period, 254 of them had existed for > 100 days, 114 thereof for > 365 days. The long-lived ZSKs are mostly 1280-bit and a few 2048-bit keys. The operators apparently decided to use longer keys, which are replaced less often.

Table 8.2 shows the public RSA exponents. The most common is $e = 65\,537$, which is also the most frequently used exponent in other RSA-based applications [83]. The choice of e affects the performance of RSA signature verification: modular exponentiation can be computed faster with smaller exponents. $e = 3$ may offer an even better verification performance, but is susceptible to signature forgery on broken implementations that do not handle the message padding correctly [98]. Although this is an implementation weakness and $e = 3$ can be implemented securely, $e = 65\,537$ is a more conservative choice. $e = 2^{32} + 1$ is also suitable, albeit less supported in cryptographic implementations according to anecdotal evidence, as it does not fit into a 32-bit integer.

8.1.2 Authenticated Denial of Existence

In February 2015, 107 TLDs were using NSEC and 540 TLDs were using hashed NSEC3 as authenticated denial of existence. We now survey the NSEC3 parameters in use. 17 TLDs were using zero iterations and also an empty salt value, including the well-established `com`,

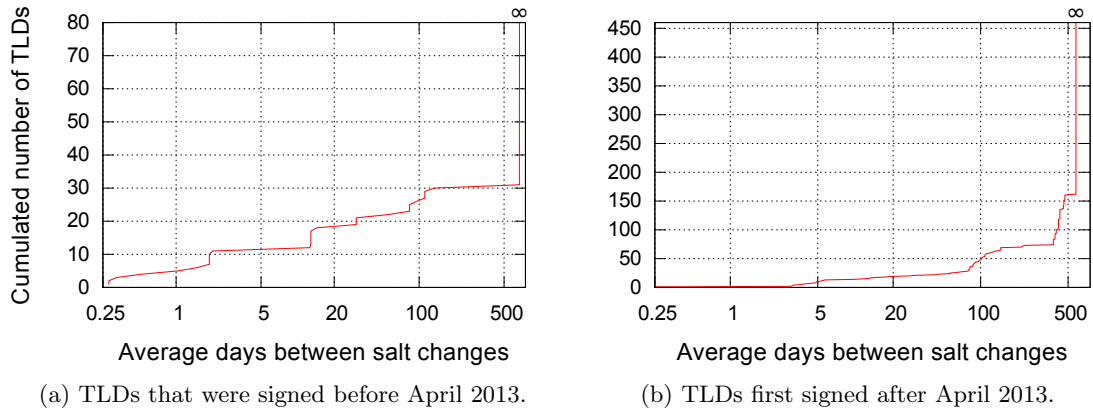


Figure 8.3: Frequency of salt changes in top-level domains (April 2013 to February 2015).

`edu`, `net` and `uk`. Note that zero iterations imply one hashing operation, as explained in Section 3.7.3. Using zero iterations with NSEC3 obstructs zone enumeration to a basic degree while keeping the server-side CPU cost to the essential minimum. It furthermore allows the operator to use the NSEC3 opt-out feature (Section 3.7.4), which is a popular option especially in large zones: 473 TLDs were using opt-out (88%). The most frequent iteration count was $i = 1$, in use by 60% of all TLDs with NSEC3. From a conservative security perspective, choosing one iteration over zero is a reasonable choice: when a preimage attack against SHA-1 is discovered, hashing twice might provide a safety margin to make the attack ineffective. Large iteration values are not common: only 9 TLDs (2%) were using $i \geq 15$ and 3 thereof $i \geq 100$.

The median salt length was 4 bytes, and we observed three common usage patterns for the salt value. First, 36 TLDs (7%) had no salt, i.e. the salt length is 0 bytes. Second, 309 TLDs (57%) had a salt value but did not change it during our 20-month observation. Using a constant salt value is essentially like using no salt. The NSEC3 salt provides only protection against pre-computed rainbow table attacks if changed regularly. Lastly, the remaining 195 TLDs (36%) were using the salt for its intended purpose and changed it at least once during our observation. The average interval between salt changes is shown in Figure 8.3, separately for well-established and newly signed TLDs. If salts are replaced, the typical interval is every 5 to 100 days, which is a reasonable choice. Changing the salt requires to create new signatures for all NSEC3 records. However, when the signing process runs anyway for other reasons, a new random salt can be chosen at no additional cost. This will occur at the latest when the ZSK is rolled over. Five TLDs changed the salt more than once per day, i.e. signed the whole zone several times daily.

| TLD | NSEC(3) | Secure Del. | Address RR | Empty | Other |
|-----------------------------|--------------------------|-------------|------------|---------|-----------|
| nl | NSEC3, opt-out, $i = 5$ | 2,279,702 | 5 | 1 | 1 |
| br | mixed ² | 566,694 | 0 | 0 | 34,625 |
| cz | NSEC3, $i = 10$ | 448,984 | 0 | 0 | 717,267 |
| com | NSEC3, opt-out, $i = 0$ | 426,182 | 0 | 0 | 1 |
| se | NSEC | 349,514 | 9 | 0 | 940,946 |
| eu | NSEC3, opt-out, $i = 1$ | 320,311 | 7 | 1 | 1 |
| fr | NSEC3, opt-out, $i = 1$ | 205,662 | 0 | 6 | 3 |
| no | NSEC3, opt-out, $i = 5$ | 119,759 | 4 | 2 | 2 |
| be | NSEC3, opt-out, $i = 5$ | 92,385 | 0 | 1 | 2 |
| net | NSEC3, opt-out, $i = 0$ | 81,391 | 0 | 0 | 1 |
| org | NSEC3, opt-out, $i = 1$ | 46,382 | 10,737 | 4,976 | 448 |
| ovh | NSEC3, opt-out, $i = 1$ | 29,372 | 0 | 0 | 1 |
| nu | NSEC3, $i = 5$ | 21,126 | 0 | 0 | 235,308 |
| de | NSEC3, opt-out, $i = 15$ | 20,004 | 185,107 | 89,689 | 2 |
| pl | NSEC3, opt-out, $i = 12$ | 18,110 | 7 | 0 | 1 |
| <i>[632 others omitted]</i> | | | | | |
| Total: | | 5,146,705 | 926,279 | 131,610 | 9,272,944 |

Table 8.3: TLDs with most secure delegations of DNSSEC domains (DS record sets).

8.1.3 Zone Enumeration

We attempt to enumerate the TLD zones to determine the number of registered domains that are signed with DNSSEC. We define a registered domain as a domain name that has been registered by a registrant at a TLD registry. First, we use NSEC zone enumeration to retrieve all names from TLDs with NSEC authenticated denial of existence. Second, we use the hash crawling method from Section 7.1 to retrieve hash values from TLDs with NSEC3 hashed authenticated denial of existence. Depending on whether a zone uses the NSEC3 opt-out feature (Section 3.7.4), we get either the hash values of all names (opt-out disabled) or the hash values of signed names (opt-out enabled). Some country-code TLDs restrict or used to restrict domain registration on second-level and instead offered registration of third-level domains under a set of predefined labels. An enumeration of second-level domains would miss those domains. As we are interested in registered domain names under any public suffix, we also enumerate well-known second-level domains like `com.br`, `co.kr`, `com.tw`, `co.uk` and aggregate them under the respective TLD.

The crawling finished after about 3 to 4 days and yielded 7.99M names from NSEC records and 7.49M hash values from NSEC3 records. The types field in the NSEC/NSEC3 records allows us to classify the existing resource records within TLD zones. Table 8.3 shows the top 15 TLDs with most secure delegations, i.e. DS records. The number of DS records

²For example, `br` and `edu.br` use NSEC; `com.br`, `eco.br` and `org.br` use NSEC3 with opt-out and $i = 10$.

gives an accurate measure for registered domains that are using DNSSEC, which totals to 5.1M. Most DNSSEC-signed registered domains can be found under **nl**, **br**, **cz**, **com** and **se**. The accumulation of DNSSEC-signed domains under these TLDs is not a coincidence, because the operators provided an incentive for registrars to adopt DNSSEC [63]. SIDN (**nl**) has offered an 8% discount in registry fees for a two-year period. Registro.br (**br**) has provided free technical support on DNSSEC and requires DNSSEC for high-value domains, e.g. new domains under the **b.br** suffix dedicated to financial institutions. CZ.NIC (**cz**) has offered technical support and financial support of DNSSEC-related marketing campaigns. IIS (**se**) has offered a discount on DNSSEC-signed domains, which has been subsidized by the Swedish government. Although there was no such initiative for **com**, the incentives provided by country-code TLDs also fueled the deployment under **com** and other TLDs: a registrar, which adapted its business processes for DNSSEC support under one TLD, can add support for other TLDs with little extra cost.

Column “Address RR” in Table 8.3 indicates the number of authoritative A or AAAA record sets (or CNAME or MX in a few cases) without a delegation. TLD zones consist mostly of domain delegations, but some TLD operators allow to put registered domain names directly into the TLD zone, e.g. in **de**. These names are DNSSEC-signed by the TLD operator without cooperation of the domain administrator. Thus, signed address records do not reflect domains whose administrators deliberately chose to deploy DNSSEC. Another source for signed address records in a TLD zone are dangling glue records: when a delegation is deleted without also deleting the corresponding glue record, the dangling glue record becomes a signed authoritative address record. This appears to have happened in many cases under **org**, which does not enforce removal of dangling glue records. Besides authoritative data, both **org** and **de** have a significant number of NSEC3 records with empty types field. These indicate an *empty non-terminal*, e.g. when an address record exists for **www.example.de** but not for **example.de**, or when a dangling glue record like **ns1.example.org** remains after the delegation of **example.org** has been removed. Empty non-terminals in a TLD zone are caused by the same reasons as authoritative address records. Other record types indicate either special resource records in a negligible number of cases, or insecure delegations when opt-out is not in use, e.g. under **cz** and **se**.

8.1.4 NSEC3 Hash Breaking

After having crawled all NSEC3 hash values, we use the NSEC3 hash breaking method from Section 7.2 to reveal as many domain names as possible with modest resources. The purpose is to acquire a list of cleartext domain names for further analysis of DNSSEC deployment. As every fully qualified domain name yields a unique hash value, the hash breaking attack must be performed separately for each TLD zone. For an efficient use of resources, we omit zones with ≤ 10 NSEC3 hash values. This gives us a list of 301 top-level

| | Candidate Names | Names Found |
|-------------|----------------------|-------------|
| Brute-Force | 5.2×10^{14} | 1,353,657 |
| Dictionary | 3.2×10^{14} | 3,198,966 |
| Markov | 1.1×10^{13} | 96,817 |
| Total | 8.4×10^{14} | 4,649,424 |

Table 8.4: Names found from 7.49M NSEC3 hash values after three weeks of computation.

| TLD | Names Found | Total | Ratio |
|-----|-------------|-----------|--------|
| nl | 1,446,992 | 2,279,709 | 63.47% |
| cz | 839,858 | 1,166,251 | 72.01% |
| br | 390,582 | 597,492 | 65.37% |
| com | 278,894 | 426,183 | 65.44% |
| eu | 267,247 | 320,320 | 83.43% |
| uk | 227,368 | 276,208 | 82.32% |
| nu | 220,066 | 256,434 | 85.82% |
| fr | 124,446 | 205,671 | 60.51% |
| de | 107,557 | 294,802 | 36.48% |
| no | 84,923 | 119,767 | 70.91% |

Table 8.5: TLDs with most reversed NSEC3 hash values.

and second-level domain zones, on which we apply varying efforts depending on the number of hash values in the zone to be broken. The methods include the brute-force attack for up to 8 characters, the dictionary attack with several dictionaries of different sizes and the Markov attack trained per-domain on intermediate results.

We distribute the computing load to 4 GPUs and 22 CPU cores over a period of three weeks. The CPU cores are contributing about 2% of the total computing power. The GPUs are one AMD HD 7970, one AMD HD 6970 and two dual-GPU NVIDIA GTX 690. One of the GTX 690 is throttled to about $\frac{1}{3}$ to $\frac{1}{4}$ of its capacity due to overheating problems. More than 90% of the total computing power are provided by three GPUs. Table 8.4 shows the number of processed candidate names and the number of cleartext names found after three weeks of computation. The results are broken down per TLD in Table 8.5. This demonstrates the applicability of NSEC3 zone enumeration attacks on other TLDs than `com` (cf. the case study in Section 7.4), including TLDs with larger iteration counts.

8.2 Registered Domains

In the following study, we examine the adoption of DNSSEC for domains that are registered under top-level domains, or under public suffixes marketed by TLD registries (e.g. `com.br`).

| Cryptosystem | SEK | ZK |
|---------------------|-----------|-----------|
| RSA/MD5 | 0 | 0 |
| DSA/SHA-1 | 2,178 | 2,279 |
| RSA/SHA-1 | 1,550,859 | 1,848,283 |
| RSA/SHA-256 | 1,875,294 | 2,785,784 |
| RSA/SHA-512 | 1,220 | 1,158 |
| GOST R 34.10-2001 | 30 | 30 |
| ECDSA P-256/SHA-256 | 27 | 25 |
| ECDSA P-384/SHA-384 | 21 | 17 |
| Total: | 3,429,630 | 4,637,576 |

Table 8.6: Cryptosystems most frequently used for DNSSEC signing.

The list of domains originates from the NSEC and NSEC3 zone enumeration in the previous sections. We retrieved the DNSKEY record set from 3.4M domains and the DS record set from their parent in March 2015.

89% domains appear to use a KSK/ZSK scheme. 348k other domains either returned only one DNSKEY record or have the same Secure Entry Point (SEP) bit value for all DNSKEY records. The SEP flag is a hint for distinguishing between KSK and ZSK. However, the SEP flag does not change the validation process and is not used by all domains for its intended purpose. We thus use a different terminology, which is suitable for domains that do not use the SEP bit or the KSK/ZSK scheme. We use the term *Secure Entry Key* (SEK) for a key in the DNSKEY set that is authenticated by a parent DS record, and *Zone Key* (ZK) for any other key in the DNSKEY set. When the KSK/ZSK scheme is used, an SEK is identical to a KSK and an ZK is identical to a ZSK.

8.2.1 Algorithms

Table 8.6 shows the cryptosystems used for signing, separately for SEK and ZK. The most frequently used cryptosystem is RSA with one of the SHA hash functions. The deprecated RSA/MD5 is not in circulation at all. DSA is used on a minor scale. The elliptic curve cryptosystems GOST and ECDSA are rarely used and we consider their deployment as experimental.

Table 8.7 shows the RSA key lengths in use. The most common combination is 2048-bit RSA for SEK and 1024-bit RSA for ZK. Recall that 1024-bit RSA has not been broken in public yet, but does not provide any security margin against future attacks (Section 5.8). Furthermore, there is a worrying amount of 512-bit RSA keys in circulation, which do not offer a reasonable security benefit over unsigned DNS. The most common public RSA exponent is $e = 65\,537$, as shown in Table 8.8. The few occurrences of $e = 65\,337$ and $e = 65\,535$ are probably a typo but without security implications.

| Length | SEK | ZK |
|--------|-----------|-----------|
| 512 | 8,704 | 14,416 |
| 768 | 4 | 30 |
| 1024 | 724,324 | 4,333,715 |
| 1032 | 0 | 535 |
| 1152 | 19 | 90 |
| 1280 | 878 | 215,944 |
| 1304 | 254 | 102 |
| 1536 | 154,748 | 123 |
| 2048 | 2,454,645 | 64,947 |
| 2560 | 117 | 88 |
| 3072 | 7 | 3 |
| 4096 | 83,602 | 5,142 |
| Other: | 71 | 77 |
| Total: | 3,427,373 | 4,635,221 |

Table 8.7: Bit lengths of RSA moduli.

| Exponent | SEK | ZK |
|--------------|-----------|-----------|
| 3 | 202 | 252 |
| 65,337 | 26 | 60 |
| 65,535 | 124 | 288 |
| 65,537 | 3,421,138 | 4,629,425 |
| $2^{30} + 3$ | 38 | 51 |
| $2^{32} + 1$ | 5,845 | 5,145 |
| Total: | 3,427,373 | 4,635,221 |

Table 8.8: Public RSA exponents.

| Length | SEK | ZK |
|--------|-------|-------|
| 512 | 3 | 5 |
| 768 | 2 | 3 |
| 1024 | 2,173 | 2,271 |
| Total: | 2,178 | 2,279 |

Table 8.9: Bit lengths of DSA groups.

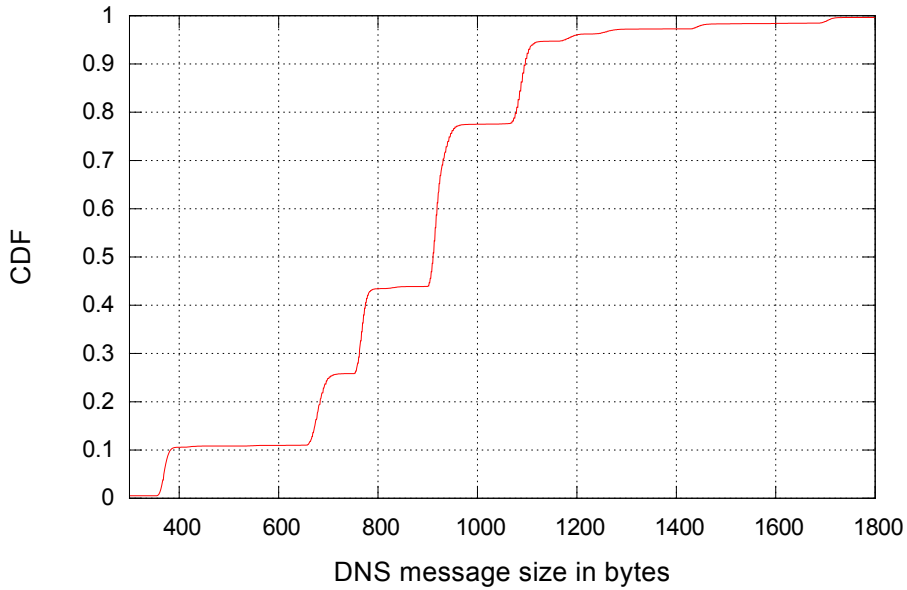


Figure 8.4: Size distribution of DNSKEY responses.

The DSA key lengths are shown in Table 8.9. Algorithms for computing the discrete logarithm in DSA are similar to RSA integer factorization, though require more computational effort [111]. Adrian et al. [4] estimate that computing the discrete logarithm with 768-bit finite groups is within range of academic teams and 1024-bit groups within range of state-level attackers. The maximum DSA key length specified in DNSSEC is 1024 bits [1], which does not provide enough security for future use. Changing the DNSSEC protocol to support larger key sizes would be possible but is not reasonable. A 1024-bit public DSA key is represented with all necessary parameters by 405 octets, whereas a 1024-bit public RSA key with $e = 65537$ is represented by 132 octets. DNS is sensitive to message size and DSA has an inefficient ratio of security level to key size in a DNSKEY record. Although DSA signatures are shorter than e.g. RSA signatures, we consider the size of the public key as more important because DNSKEY record sets are larger than the average address record set. We thus recommend to deprecate the use of DSA in DNSSEC in favor of elliptic curve cryptosystems.

GOST and ECDSA P-256 use 256-bit keys with 64 bytes representation. ECDSA P-384 uses 384-bit keys with 96 bytes representation. The security level of ECDSA P-256 is comparable to 3072-bit RSA [59].

8.2.2 Message Size

Figure 8.4 shows the distribution of DNSKEY response sizes that we have observed. Most responses were 700 to 950 bytes long. 2.2% of all DNSKEY responses were longer than

1440 bytes and thus potentially susceptible for delays due to handling of dropped or truncated messages (cf. Section 2.6). A prompt DNSKEY response is vital because the DNSKEY set is required for validation of any other zone data. When delayed, this will significantly slow down the first resolution of any name in the zone. The DNSKEY response size depends primarily on the number of DNSSEC keys in the zone, the cryptosystems used, key lengths, and the number of signatures over the DNSKEY set. All of these factors can be controlled by the zone administrator:

- **Number of keys.** A separation of keys like in the KSK/ZSK scheme is reasonable only if the keys are stored within different security perimeters, e.g. the ZSK in the production network and the KSK offline. The KSK/ZSK scheme does not improve the system security when both keys are stored on the same host. In fact, using different key bit-lengths degrades the overall security in such a scenario compared to just using one long key. We thus recommend to use one key by default with a bit-length suitable to resist attacks for several years (256-bit elliptic curve or 2048-bit RSA), and to introduce the KSK/ZSK scheme only when explicitly justified.
- **Cryptosystem and key length.** Having different cryptosystems and key lengths in the same zone do not aggregate to a higher security level in DNSSEC, as the weakest link constitutes the overall security.
- **Number of signatures.** The DNSKEY set must be signed by each SEK listed in the parent DS records but not by every zone key in the DNSKEY set.

DNSSEC always returns the whole DNSKEY set and does not allow to retrieve a single DNSKEY selectively. This hampers the introduction of new cryptosystems that are not yet implemented on all existing systems. Zone administrators performing a phase-out from one cryptosystem to another have to cope with the increased size of the DNSKEY set and have to serve both signatures on all responses. This explains why ECDSA is not used three years after specification [59] despite offering better security than RSA with smaller message sizes: using two cryptosystems in parallel increases the operational costs. There is no implicit feedback mechanism for the zone administrator to become aware of client-side ECDSA support and to determine a reasonable point in time for switching cryptosystems. Herzberg and Shulman [57] suggested that cipher suite negotiation could be retrofitted into DNSSEC without breaking legacy systems. This would solve the message size issue but has not been specified.

8.2.3 Validation Result

All domains in this study ought to be signed, as this was indicated by the parent DS record set. We attempt to validate the DNSKEY record set by the trusted SEKs to determine

| Result | Count |
|---------------------------------|-----------|
| No DNSKEY (dangling DS) | 17,751 |
| No trusted DNSKEY (dangling DS) | 1,066 |
| No RRSIG for trusted DNSKEY | 238 |
| Signature expired | 2,138 |
| Signature verify failure | 5 |
| Validation failure | 21,198 |
| Validation success | 3,416,700 |

Table 8.10: DNSSEC validation result of 3.4M securely delegated domains.

whether the registered domains have a valid authentication chain. First, we check whether any DNSKEY matches any of the DS records in terms of key tag, algorithm number and fingerprint. Then, we check whether there is an RRSIG record created with any of the SEK that is authenticated by a parent DS record. We attempt to verify the signature of the RRSIG with the trusted SEKs and compare the validity period of the RRSIG with the time of retrieval of the DNSKEY response. If validation succeeds, all DNSKEY records are authentic and can be used for validation of other signatures of that zone.

5,846 domains did not return a processible DNSKEY response, e.g. timed out repeatedly or returned a badly formatted DNS response. We assume that the server failed to return any DNS response and do not count this as DNSSEC failure. Table 8.10 shows the validation results. For 17,751 domains, the response was well-formed in principle but did not contain any DNSKEY record. This usually indicates a lack of server-side DNSSEC support. 1,066 responses contained a DNSKEY record but none of the keys was authenticated by the parent DS record set. This is typically caused by an improper key rollover: the SEK is replaced in the DNSKEY set but the parent DS record remains unchanged. This failure could be omitted by automating key rollovers and DS updates (cf. Section 3.5.3). In 238 cases there was a trusted SEK but the DNSKEY set was not signed by it. This is either caused by a lack of server-side DNSSEC support, i.e. a legacy server returns the queried DNSKEY set but fails to include the corresponding RRSIG set, or by an integrity failure of the zone data, i.e. missing resource records.

2,138 domains returned an expired response. In a few cases one server returned stale zone data while another server would have given us a valid response, which is a server synchronization failure. However, it was more common that the signatures have not been renewed, which could be avoided with automatic signing. We did not observe signatures ahead of their validity period. The actual signature verification failed in only 5 cases. This shows that the reliability of DNSSEC depends in practice on operational issues and not on the implementation of cryptographic primitives. 21,198 DNSKEY responses failed to validate correctly, i.e. DNSSEC has degraded the availability of 0.6% domains in this study.

8.3 Summary and Conclusion

After NSEC and NSEC3 zone enumeration of all TLDs in March 2015, we found 5.1M domains registered at TLD registries that are delegated with DNSSEC. A negligible number of domains may be missing due to server failures, but altogether this gives us a nearly complete measure of server-side DNSSEC deployment. Most TLDs use NSEC3, and about 60% of NSEC3 hash values were broken after 3 weeks of computation on four GPUs. A survey among 3.4M securely delegated domains shows that almost all ($> 99\%$) use RSA, most (89%) with a KSK/ZSK scheme. 0.6% securely delegated domains have a broken authentication chain on at least one name server, i.e. fail to validate.

We question the broad use of the KSK/ZSK scheme with an RSA 1024-bit ZSK, and instead suggest to use one key by default for second-level domains; in case of RSA, the key should be at least 2048 bits. Adoption of the KSK/ZSK scheme is useful when explicitly justified, e.g. in high-security signing setups where the KSK and ZSK are stored at different locations. ECDSA is a viable option to RSA, offering a security level comparable to 3072-bit RSA and shorter messages. GOST may be an option, too, but is less known outside of Russia and we expect to find less implementation support for GOST than for ECDSA. DSA should not be used in the future for DNSSEC due to insufficient key lengths. Operational problems threaten the reliability of DNSSEC. The most common failures—dangling DS records and expired signatures—can be avoided by automating the DNSSEC signing process.

CHAPTER 9

Client-Side Adoption

After having surveyed the adoption of DNSSEC signing, we now survey the client-side adoption of DNSSEC validation. We apply a web-based measurement method to determine the fraction of web clients protected by DNSSEC validation. The measurement data has been collected over a period of almost three years and has been fed with geolocation data to facilitate a spatio-temporal analysis.

9.1 Method

Different validation measures are possible, e.g. the number of clients protected by validation, the number of resolvers performing validation or the number of responses received by validating resolvers. We chose to count the number of web-based clients because from this measure one can deduce the amount of users protected by DNSSEC.

We set up a signed DNSSEC zone, which is securely delegated with a valid authentication chain. Our test zone uses a KSK/ZSK scheme with 1024-bit RSA keys and a DS record in the `net` top-level domain. We use the following terminology for domain names in our test zone: `sigok` is an A record with a valid signature and `sigfail` is an A record with a placeholder signature, which is syntactically correct but fails to validate. The idea is to determine whether the web browser loads an HTTP resource from the `sigfail` domain name, which is an indication that the name has been resolved without DNSSEC validation. We are using two types of test methods: a scripted test that provides user feedback and a hidden test that can be embedded into other web pages.

Algorithm 3 Scripted test: determine whether the client is DNSSEC-protected.

```

1: success ← load_img(uri = sigfail)                                ▷ broken signature
2: if success then
3:   return 'no DNSSEC validation'
4: else
5:   success ← load_img(uri = sigok)                                ▷ valid signature
6:   if success then
7:     return 'DNSSEC validation enabled'
8:   else
9:     return 'inconclusive'
10:  end if
11: end if

```

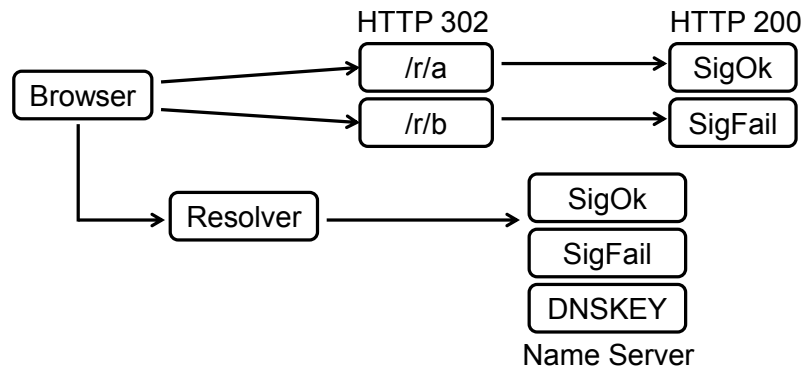


Figure 9.1: Hidden test: HTTP and DNS queries seen at our servers.

9.1.1 Scripted Test

When a web browser accesses our scripted test¹, it runs the short JavaScript program shown as Algorithm 3. Should the browser load the image successfully from the **sigfail** domain, then it is not protected by DNSSEC, as the resolver failed to recognize the invalid signature. A load failure may indicate a validating resolver, or an unrelated error. To rule out error sources like a stalled network connection or generic DNSSEC resolution failure, the script attempts to load an image from the signed **sigok** domain name. If the second image has been loaded, the resolver correctly validates DNSSEC signatures. Should the second image fail to load as well, then the test will be considered as inconclusive. The result is displayed to the user and posted to our web server in background.

9.1.2 Hidden Test

The web-based hidden test uses the following HTML snippet, which can be embedded into existing web pages:

¹<http://dnssec.vs.uni-due.de>

```


```

The two image URLs redirect the web browser to a transparent 1×1 pixel image at `ID-sigok` and `ID-sigfail`. `ID` is a placeholder for a number used to identify the client. Figure 9.1 shows the HTTP and DNS queries that we can observe at our servers. As most clients do not resolve domain names by themselves, the client IP address usually differs from the resolver IP address. The `ID` number allows us to match browser queries and resolver queries together despite different source IP addresses. The rationale behind this method is as follows:

1. By using an HTTP redirect we can embed a static HTML code snippet into existing web pages and track the queries by client `ID`. When including the `ID` directly into the image URLs this would require to dynamically generate the HTML code.
2. The DNS zone is moderately sized when using 16 bit for the `ID`. As we need to deliver valid and invalid signatures, we pre-generate the DNS zone, sign it and then replace the `sigfail` signatures with broken placeholders. This results in an 88 MBytes zone file with 2^{19} resource records (A and RRSIG, NSEC and RRSIG, for both `sigok` and `sigfail`). If we dynamically created and signed the resource records as needed, this would require either a customized name server or an unusual zone layout which might pose a pitfall for some resolvers.
3. By deriving the `ID` number from the client IP address we get a simple stateless mapping which does not change while the same client is visiting multiple web pages and is unlikely to collide with another client at the same time.

DNSSEC validation is enabled when there are HTTP GET requests for the two redirect URLs and the `sigok` image but none for the `sigfail` image. Validation is disabled when there are HTTP GET requests for the redirect URLs and for both images.

9.1.3 Accuracy

For a positive test result, we require the client to load an image from the signed `sigok` domain name. This is meant to catch faults that could spoil the result, e.g. blocking our signed domain name, not automatically loading images, not following cross-domain HTTP redirects or failing to receive EDNS0 messages > 512 bytes. The DNS responses for `sigok` and `sigfail` are 256–258 bytes long, consisting of an A and RRSIG record. When the DNS resolver signals that it can handle sufficiently long EDNS0 messages, our authoritative name server will include additional records (delegation and glue records) up to a message size of 1282 bytes. The DNSKEY response is 709 bytes long. Our expectation is that messages of ≤ 709 bytes are short enough to be transported over UDP without IP fragmentation

or truncation in most networks. Nevertheless, one of the images could fail to load for an unrelated reason, e.g. temporary network fault or user closes web page before it has been loaded completely. Should this happen, then the following faults are possible:

1. None of the images are loaded: does not affect our results.
2. `sigfail` loads and `sigok` does not load: does not affect our results.
3. `sigok` loads and `sigfail` does not load: causes a false positive in our results.

To estimate the ratio of false positives caused by case no. 3, we calculate the number of occurrence of case no. 2. Both cases can occur only with non-validating resolvers and correspond to the same fault pattern. It is thus reasonable to assume that both cases occur with the same frequency. Note that this type of fault cannot cause false negatives.

9.2 Implementation

We are using Apache 2 with standard logfiles and BIND9 with query logging enabled. The servers are running on the same host to ease the analysis setup. The authoritative name server is reachable via IPv4 and IPv6, but serves A records only, so the web server will be accessed via IPv4 only. The HTTP redirect script refers the client to `ID-sigok.verteiltesysteme.net/ok.png?ID`, where ID is a 16-bit hexadecimal number derived from the client IP address. The IPv4 address-to-ID derivation function is SHA-256 with the output truncated to 16 bits. Protocol-relative URLs are used in the HTML snippet that is embedded in the origin website to support both HTTP and HTTPS. A wildcard HTTPS certificate is used to cover the 2^{17} different `sigok` and `sigfail` domain names. To minimize the impact of DNS caching, we use a TTL value of 60 seconds on the `sigok` and `sigfail` resource records. Concerning browser caching, we return *no-cache* headers² in image responses. Furthermore, we perform an extensive data cleaning to remove results without the minimum set of expected responses (Section 9.3.1), which should remove any interference of caching.

The logfiles are 4 GBytes in size after three years of operation. The input data is processed with a sliding window approach: requests from the logfiles are parsed chronologically and grouped by matching ID within a 30 seconds time window.

9.3 Analysis

The following analysis of the hidden test comprises 20M DNS and HTTP requests logged starting from May 2012 to March 2015. Three sources provided web clients for this measurement (sorted descending by amount of traffic):

²“Cache-control: max-age=0, no-cache, must-revalidate”.

| Missing query | Both | sigok | sigfail |
|---------------|---------|--------------|----------------|
| HTTP redirect | 545,317 | 17,252 | 18,501 |
| DNS query | 15,993 | 4,016 | 3,078 |
| HTTP image | 6,677 | 1,742 | - |

Table 9.1: Incomplete trials removed after deduplication.

1. Visitors of websites that kindly included the HTML snippet of our hidden test.
2. Participants of *autosurf* traffic generators (visits are mostly unattended but in end user environment and thus serve our purpose).
3. Visitors of the scripted test automatically participated in the hidden test.

Requests with matching ID and within a 30 second time window are grouped together into one *trial*. This leads to a total of 2.5M trials, which bears further processing to remove implausible or incomplete trials.

9.3.1 Data Cleaning

The data needs to be processed before evaluation to remove noise or incomplete trials. An essential data cleaning step is the removal of duplicate results. Web clients browsing a participating website for a while are grouped into several trials when loading several page views over a couple of minutes. The deduplication period should be long enough to cover the whole browsing session of the user but not longer than the assignment of a dynamic IP address. Dynamic IP addresses cause two problems in conjunction with deduplication:

1. **Overestimation:** the same client may be counted twice with different IP addresses (unlike clients with static IP addresses).
2. **Underestimation:** another client may be filtered when assigned the same IP address.

While both effects could cancel each other out, we cannot assume that they are equally likely. Xie et al. [134] estimated the time interval between two different users on the same dynamic IP address to be >12 hours in 80% of all cases. We thus remove duplicate trials, which contain the same client IP address within a 12-hour period after first appearance. Experiments with different deduplication periods from 2 hours to 7 days show a minor influence on the overall validation ratio ($\pm 0.3\%$ points over a 10-month period). After deduplication, there are 1.46M trials left for further analysis.

Next, we remove trials that lack the minimum required set of requests. A complete trial requires at least both HTTP redirects to **sigok** and **sigfail**, both DNS queries and an HTTP query to the **sigok** image. Incomplete trials can be caused by errors unrelated to

DNSSEC validation and are removed from the data set to avoid distortion of the results. Error sources for incomplete trials are client-side faults, closed browser tabs, noise like web crawlers or caching artefacts. The major portion of caching artefacts is removed by client IP address deduplication, but not all: trials consisting solely of DNS requests are not deduplicated as the web client IP address is unknown in these cases. These erroneous trials will be filtered due to being incomplete. Figure 9.1 shows the amount of incomplete trials that we removed from our data set.

As explained above in Section 9.1.3, incorrectly missing **sigfail** HTTP image queries are causing false positives in our measurement. The equivalent fault pattern of a missing **sigok** HTTP image query occurred in 1,742 trials, which makes an estimate of 0.21% false positives. 2,241 trials (0.27%) are classified as positive, i.e. validation enabled, but we could not find a matching DNSKEY query in our server log. As the DNSKEY set is necessary for DNSSEC validation, the absence of a DNSKEY query indicates a false positive. The number of trials with a missing DNSKEY is comparable to the above estimate of false positives, though slightly higher. We believe this is owed to our zone setup, which does not contain the client ID in DNSKEY queries; instead, we correlate DNSKEY queries with trials by the resolver IP address. This matching fails when the **sigok** DNS query arrives from a different IP network than the DNSKEY query, for example when a validator forwards queries to different resolvers in different networks or when the resolver switches between IPv4 and IPv6. We remove all 2,241 trials without DNSKEY query from our results. Although we might remove a few legitimately positive trials unnecessarily, this assures that false positives do not distort the results. This limitation could be improved in future by including the ID into DNSKEY records.

A negligible amount of trials ($< 0.01\%$) became useless because a hash collision occurred in our IP address to ID mapping, i.e. two different client IP addresses mapped to the same ID at the same time. This demonstrates that a 16-bit ID suffices for this type of time-based measurement.

9.3.2 Results

After data cleaning, there are 841k trials from 557k distinct client IP addresses. We consider a trial as negative if it contains an HTTP image query for **sigfail** or if all DNS queries are sent with DO=0 flag. In contrast, a positive result does not contain any **sigfail** HTTP query and at least one DNS query was sent with EDNS0 and DO=1 flag. The overall validation ratio is 12%, shown as a per-week timeline in Figure 9.2. There is a significant regional variance in the validation ratio, which influences the overall measure depending on the user population of our data sources. Major changes in the regional participation (Figure 9.3) cause changes to the validation graph. It is hence more meaningful to provide results per-country than an overall ratio. Nevertheless, the overall timeline suffices to demonstrate

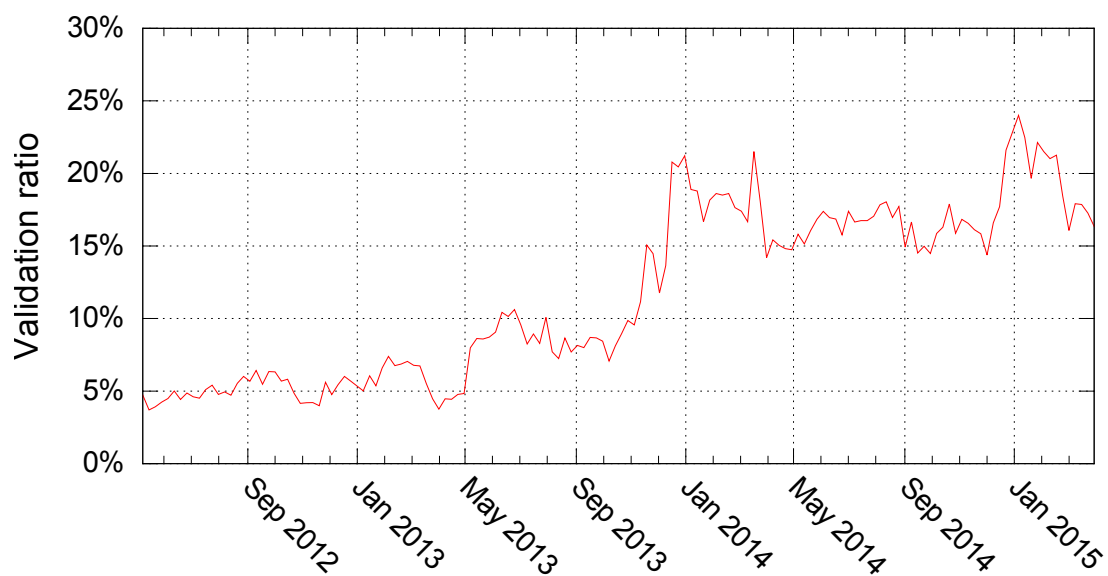


Figure 9.2: Validation ratio of $\varnothing=5.5k$ trials per week (May 2012 to March 2015).

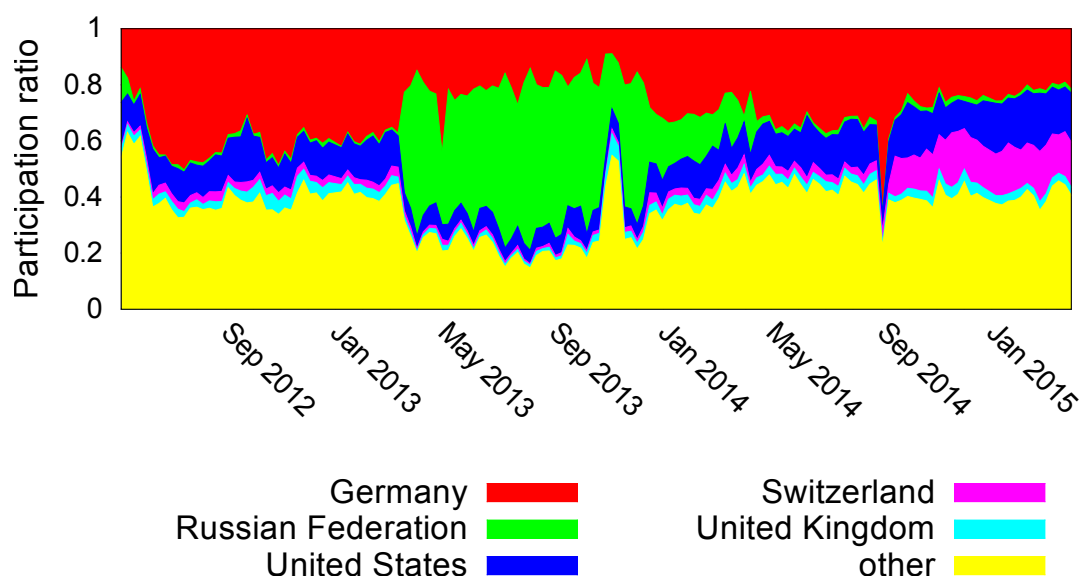


Figure 9.3: Top 5 geolocations of participating clients (May 2012 to March 2015).

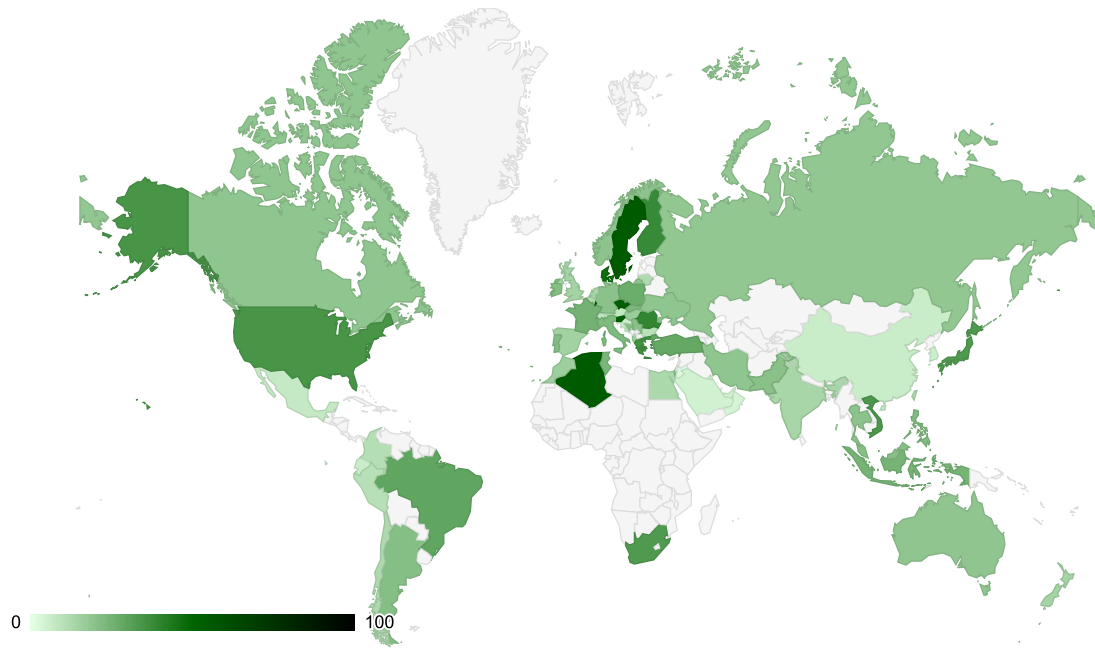


Figure 9.4: Visualization of validation ratio per country. October 2014 to March 2015.

the general tendency of an increasing DNSSEC validation deployment. A major increase occurred in May 2013 when Google Public DNS—the most frequently used public resolver service—turned on DNSSEC validation for all users. We observed another increase at the end of 2013, when several networks apparently turned on DNSSEC validation.

After geolocation and AS number analysis³ of client IP addresses, there are 76 countries and regions in our data set with > 500 trials and 44 countries with > 2000 trials. Table 9.2 shows their validation ratio from May 2012 to March 2015. The validation ratio is the mean of the samples in a given year. The \pm value for each mean represents the 95% confidence interval over the estimated sample variance. Note that the values for 2015 have been collected in a 3-month period and are thus not as stable as the 12-month periods in 2013 or 2014. We see a clear increase of DNSSEC validation for almost all regions. At the beginning of our measurement, there were few countries with major validation deployment, most notably Sweden, the Czech Republic and the United States. The median validation ratio in our data set was 1%. By 2014 and 2015, the median validation ratio rose to 17.7% and 20.7%. Interestingly, the already high validation ratio of the Swedish adopters did not increase since 2012, suggesting that DNSSEC deployment efforts in Sweden have plateaued.

Figure 9.4 visualizes the validation ratios of the most recent 6 months on a world map (for countries with > 100 trials). For the same period, Table 9.3 depicts autonomous systems providing most validating clients in absolute numbers. V is the number of positive trials (validating) and N the number of negative trials (non-validating). We see in column

³Using GeoLite from MaxMind [91].

| # | Country | Trials | From 2012/05 | 2013 | 2014 | To 2015/03 |
|-----|----------------|---------|-----------------|-----------------|-----------------|------------------|
| 1. | Sweden | 7,236 | 56.4% \pm 2.7 | 55.3% \pm 1.5 | 55.9% \pm 2.6 | 58.1% \pm 4.5 |
| 2. | Czech Republic | 5,019 | 30.6% \pm 2.8 | 33.7% \pm 2.0 | 41.4% \pm 2.6 | 52.1% \pm 4.3 |
| 3. | Finland | 2,060 | 13.5% \pm 3.4 | 25.7% \pm 3.1 | 37.3% \pm 3.6 | 45.4% \pm 6.8 |
| 4. | Ukraine | 12,010 | 1.8% \pm 0.6 | 33.9% \pm 1.0 | 21.8% \pm 2.0 | 13.9% \pm 4.4 |
| 5. | United States | 86,546 | 13.5% \pm 0.5 | 19.2% \pm 0.4 | 26.6% \pm 0.5 | 38.0% \pm 0.9 |
| 6. | Belarus | 7,367 | 0.0% \pm 0.0 | 21.9% \pm 1.0 | 16.8% \pm 5.2 | 37.9% \pm 18.0 |
| 7. | Viet Nam | 5,760 | 0.1% \pm 0.1 | 47.9% \pm 3.0 | 43.0% \pm 2.8 | 36.4% \pm 5.8 |
| 8. | Indonesia | 4,818 | 3.0% \pm 0.9 | 25.8% \pm 2.1 | 26.7% \pm 2.4 | 22.4% \pm 3.5 |
| 9. | Brazil | 8,824 | 4.8% \pm 1.1 | 12.8% \pm 1.3 | 25.7% \pm 1.4 | 28.3% \pm 2.6 |
| 10. | Turkey | 5,955 | 0.7% \pm 0.5 | 19.8% \pm 1.6 | 23.7% \pm 1.9 | 30.0% \pm 3.2 |
| 11. | Denmark | 3,120 | 3.1% \pm 1.5 | 4.7% \pm 1.0 | 38.9% \pm 3.5 | 50.0% \pm 6.6 |
| 12. | Iran | 2,604 | 6.8% \pm 2.1 | 16.2% \pm 2.5 | 17.7% \pm 2.6 | 22.2% \pm 4.5 |
| 13. | Romania | 6,077 | 0.3% \pm 0.2 | 3.7% \pm 0.8 | 30.7% \pm 2.2 | 44.8% \pm 4.0 |
| 14. | Greece | 7,703 | 3.3% \pm 0.7 | 6.6% \pm 0.9 | 30.9% \pm 2.1 | 36.0% \pm 3.9 |
| 15. | Colombia | 2,454 | 0.5% \pm 0.6 | 11.6% \pm 2.5 | 21.8% \pm 2.5 | 10.7% \pm 4.4 |
| 16. | Poland | 11,985 | 3.1% \pm 0.7 | 7.5% \pm 0.7 | 25.2% \pm 1.5 | 24.1% \pm 2.5 |
| 17. | Malaysia | 3,236 | 0.5% \pm 0.5 | 17.0% \pm 2.2 | 14.0% \pm 2.2 | 23.3% \pm 4.9 |
| 18. | Ireland | 2,057 | 10.5% \pm 2.7 | 8.6% \pm 1.8 | 18.9% \pm 3.5 | 23.1% \pm 8.7 |
| 19. | Hungary | 2,178 | 9.6% \pm 2.4 | 10.0% \pm 1.8 | 14.7% \pm 3.5 | 16.1% \pm 5.4 |
| 20. | Australia | 6,025 | 1.4% \pm 0.7 | 5.7% \pm 1.0 | 17.7% \pm 1.5 | 20.1% \pm 3.2 |
| 21. | Italy | 9,049 | 4.6% \pm 1.0 | 6.4% \pm 0.7 | 20.0% \pm 1.7 | 23.0% \pm 2.8 |
| 22. | Slovakia | 3,073 | 2.6% \pm 1.1 | 4.3% \pm 1.1 | 23.3% \pm 3.0 | 25.5% \pm 5.1 |
| 23. | Norway | 3,267 | 2.3% \pm 1.6 | 5.9% \pm 1.2 | 18.6% \pm 2.8 | 18.2% \pm 3.1 |
| 24. | Portugal | 2,722 | 2.3% \pm 1.1 | 5.8% \pm 1.6 | 18.6% \pm 2.5 | 14.9% \pm 4.5 |
| 25. | Germany | 213,471 | 3.8% \pm 0.2 | 7.9% \pm 0.2 | 15.7% \pm 0.3 | 19.5% \pm 0.7 |
| 26. | Pakistan | 2,158 | 0.0% \pm 0.0 | 11.0% \pm 2.0 | 13.2% \pm 2.9 | 24.5% \pm 6.2 |
| 27. | China | 12,016 | 0.7% \pm 0.6 | 9.9% \pm 0.8 | 10.9% \pm 0.9 | 5.8% \pm 1.6 |
| 28. | France | 13,700 | 4.3% \pm 0.7 | 4.9% \pm 0.6 | 13.6% \pm 1.1 | 30.6% \pm 2.6 |
| 29. | Argentina | 2,023 | 1.3% \pm 0.9 | 7.9% \pm 1.9 | 17.8% \pm 3.3 | 21.3% \pm 7.3 |
| 30. | Netherlands | 16,134 | 4.2% \pm 0.8 | 6.9% \pm 0.6 | 11.9% \pm 0.9 | 14.3% \pm 1.6 |
| 31. | Switzerland | 34,300 | 5.8% \pm 0.8 | 8.8% \pm 0.7 | 9.2% \pm 0.5 | 8.6% \pm 0.6 |
| 32. | Canada | 11,077 | 1.4% \pm 0.6 | 3.9% \pm 0.5 | 16.3% \pm 1.4 | 21.5% \pm 2.7 |
| 33. | Kazakhstan | 3,225 | 0.0% \pm 0.0 | 8.3% \pm 1.0 | 12.2% \pm 5.3 | 14.3% \pm 15.3 |
| 34. | Spain | 13,553 | 0.4% \pm 0.2 | 6.5% \pm 0.7 | 15.5% \pm 1.1 | 11.8% \pm 1.9 |
| 35. | Russian Fed. | 172,670 | 1.1% \pm 0.4 | 5.6% \pm 0.1 | 37.3% \pm 0.9 | 19.9% \pm 2.7 |
| 36. | United Kingdom | 25,222 | 1.3% \pm 0.3 | 5.2% \pm 0.4 | 14.4% \pm 0.9 | 15.8% \pm 1.7 |
| 37. | India | 18,101 | 0.2% \pm 0.2 | 5.3% \pm 0.6 | 10.4% \pm 0.7 | 14.2% \pm 1.6 |
| 38. | Serbia | 4,870 | 1.1% \pm 0.6 | 3.9% \pm 0.8 | 13.1% \pm 1.9 | 16.2% \pm 3.9 |
| 39. | Belgium | 3,557 | 0.6% \pm 0.5 | 6.0% \pm 1.3 | 12.5% \pm 2.2 | 11.0% \pm 3.9 |
| 40. | Egypt | 2,029 | 0.0% \pm 0.0 | 11.3% \pm 2.9 | 9.0% \pm 2.7 | 11.4% \pm 4.6 |
| 41. | Mexico | 3,763 | 0.4% \pm 0.4 | 4.5% \pm 1.1 | 8.0% \pm 1.6 | 8.1% \pm 2.9 |
| 42. | Austria | 17,409 | 1.1% \pm 0.3 | 5.2% \pm 0.5 | 5.7% \pm 0.6 | 8.2% \pm 1.4 |
| 43. | Israel | 2,258 | 0.0% \pm 0.0 | 4.6% \pm 1.4 | 11.3% \pm 3.1 | 18.9% \pm 7.0 |
| 44. | Bulgaria | 2,093 | 0.0% \pm 0.0 | 3.5% \pm 1.1 | 8.2% \pm 2.9 | 12.8% \pm 6.3 |

Table 9.2: Validation ratio per country (\pm 95% CI). May 2012 to March 2015.

| AS Name | AS# | V | $\frac{V}{V_{total}}$ | $\frac{V}{V+N}$ | client = resolver |
|------------------|---------|-------|-----------------------|-----------------|----------------------|
| Comcast | AS7922 | 2434 | 9.6% | 83.8% | 3.0% |
| Unitymedia | AS20825 | 1716 | 6.8% | 89.3% | 2.0% |
| Amazon | AS16509 | 1010 | 4.0% | 59.6% | 0.0% |
| KabelBW | AS29562 | 795 | 3.1% | 89.1% | 2.4% |
| Finecom | AS15600 | 600 | 2.4% | 79.7% | 0.3% |
| Deutsche Telekom | AS3320 | 582 | 2.3% | 5.9% | 27.0% |
| DFN | AS680 | 481 | 1.9% | 24.5% | 1.9% |
| M-net | AS8767 | 389 | 1.5% | 66.8% | 3.1% |
| Orange Polska | AS5617 | 298 | 1.2% | 86.9% | 0.3% |
| RCS-RDS | AS8708 | 269 | 1.1% | 91.8% | 0.7% |
| 6738 other | | 16734 | 66.1% | 16.1% | 8.6% |

Table 9.3: Overview of ten validating networks. October 2014 to March 2015.

$\frac{V}{V_{total}}$ that although a few networks lead the statistics in our data set, two-thirds of results originate from a long tail of networks. We thus believe that our results are not distorted by a selection bias in the contributing clients to the point of being unusable. $\frac{V}{V+N}$ is the fraction of positive to all results within one AS. While some are fairly high, no AS is fully protected by DNSSEC. Some AS are using DNSSEC validation only to a small degree, as we can see for Deutsche Telekom and DFN. DFN is the German Research Network and does not have a common policy for DNS or DNSSEC validation among its participating institutions. The last column client=resolver shows trials, for which the client IP address equals at least one of the DNS resolvers. This measure is exceptionally high for Deutsche Telekom while the validation ratio is low, suggesting that Deutsche Telekom does not provide a validating DNSSEC infrastructure and instead end users operate their own recursive name servers with DNSSEC validation.

Overall, the number of client hosts relying on themselves for DNS resolution is fairly low: only 32.6k trials (3.9%) comprise a DNS request from the client's IP address. Some clients are relying on multiple resolvers, which includes resolvers in the same and in different autonomous systems. The most frequently seen resolvers in external networks, i.e. where client AS \neq resolver AS, are in AS15169 (Google) AS36692 (OpenDNS), AS44038 (Bluewin) and AS3356 (Level 3), emphasizing that Google Public DNS is the largest public resolver. In the most recent 6 months, 21k out of 125k trials (16.7%) in our data set contain DNS queries originating from Google, either exclusively or in conjunction with other networks. Regarding the 25.3k positive trials only, 9.5k (38%) had been using only Google for DNS resolution, supposedly Google Public DNS. Another 1.7k (7%) had been using Google in conjunction with a resolver in another network. The broad use of Google Public DNS may pose a distorted picture of the client-side protection of DNSSEC, because we do not know whether validation happens in the client's network or only at Google's site. Although

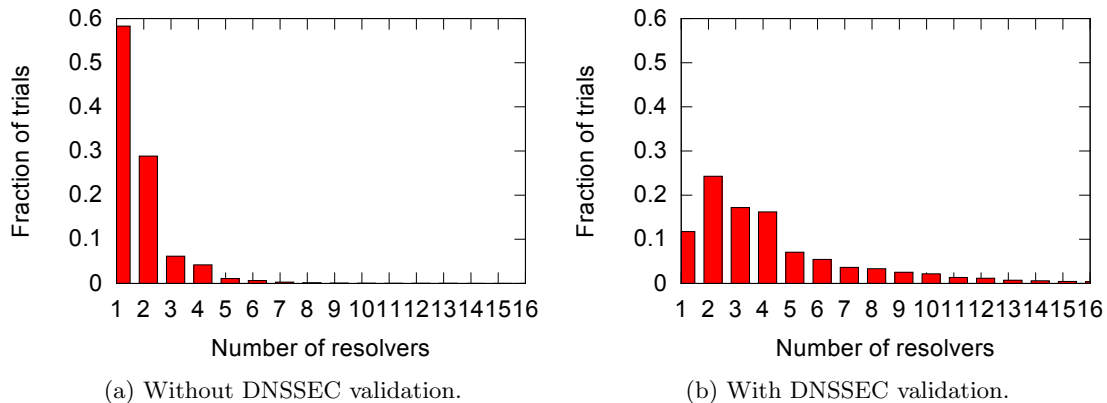


Figure 9.5: Number of resolver IP addresses seen with negative (a) and positive (b) trials.

definitely relying on a validator, the network path between the client and Google Public DNS may be nonetheless vulnerable for DNS alteration (cf. Figure 6.20). This type of scenario appears as positive trial in our results, i.e. client is relying on DNSSEC validation.

Figure 9.5 shows the number of resolvers seen per trial. Trials with a negative result, i.e. no DNSSEC validation, comprise queries from one resolver in 58% of all cases and two resolvers in 29% of all cases. With validation, we see ≥ 2 resolvers in most cases, which follows from the failure handling of the broken `sigfail` signature. First, a validating resolver will retry upon failure in order to possibly retrieve valid data through another resolution path. If the validator is set up to forward the query to other resolvers, this will manifest in DNS queries from different IP addresses at our name server. Second, security-unaware clients do not distinguish between validation failures or DNS failures caused by other reasons. This is due to the lack of signalling of validation failures in DNS responses. When set up with multiple resolvers, the client will rotate through them on validation failure. The security guarantee of DNSSEC will break if any insecure path exists, e.g. untrusted network link or non-validating resolver. Our data suggests that several clients are relying on mixed validating and non-validating resolvers. 57k trials (6.8%) are negative although one of the resolver had retrieved our DNSKEY set, which we consider as upper bound estimate for mixed validation. A lower bound estimate are trials with mixed DNS queries with `DO=0` and `DO=1` flag, which applies to 11k negative trials (1.3%). Mixed validation appears as negative trial in our results, i.e. client is not protected by DNSSEC validation.

9.4 Related Work

Studies about the adoption of DNSSEC validation can be grouped into two classes: 1) passive measurements, and 2) active measurements. Passive measurements are typically collected at authoritative name servers and determine validating resolvers. Active measurements are mostly web-based and determine clients protected by validation *or* validating resolvers.

Passive Measurements

Gudmundsson and Crocker [80] measured the validation ratio in 2010/11 by analyzing network traces from a subset of authoritative name servers for `org`. They applied different criteria and found out that looking for DS queries is more effective in their scenario than looking for DNSKEY queries. The ratio of validating resolvers was 0.8% (the authors state a number of 1.2%, which appears to be a mistake according to their raw numbers in Table V), which accounted for 8–10% observed queries to `org`. The geographical distribution and the number of clients served by these resolvers is unknown.

Fujiwara performed a similar measurement for `jp` over a period of one year [46, 47]. He acquired 2-day network traces from all authoritative name servers for `jp` on selected dates and interpolated interjacent numbers by analyzing partial log files. The number of resolvers querying for DNSKEY rose from 3,000 (0.2%) in March 2011 to 10,000 in February 2012.

Fukuda et al. [48] suggested that counting DS or DNSKEY records overestimates the number of DNSSEC validators, because a major portion of these queries are related to stub resolvers behind non-validating resolvers, e.g. caused by browser plugins for debugging purposes. Less than 50% of potential DNSSEC validators seen in network traces were confirmed with an active measurement as actually validating resolvers. Fukuda et al. proposed to use the ratio of DS queries to all queries of a host to approximate the number of validators at a higher reliability.

Active Measurements

Yu et al. [136] suggested identification of validating resolvers by their pattern of DNS query retries when faced with bogus DNSSEC responses. The method works solely by fingerprinting DNS messages; web-based HTTP interaction is not required. Although counting validating resolvers is a different measure than counting web clients like we do, the validation ratio of 4.8% measured by Yu et al. in 2012 is close to our results from that time.

Lian et al. [85] measured the number of validating clients with an advertising network. They collected 529k results with a JavaScript ad in one week in 2012 or early 2013. The method is similar to ours, but comprised 25 different kinds of deliberately broken DNSSEC domains and also a control domain without DNSSEC at all. Tests that required a fallback

to TCP failed exceptionally often, in particular with Asian Pacific clients, indicating that a significant portion supports DNS over UDP only. The overall validation ratio was $< 3\%$. The authors observed a geographical variation but did not provide results per country.

Huston and Michaelson [62] used an advertising network for measuring validating clients in May 2013. The measurement program was created with Adobe Flash, which runs on many platforms except for mobile devices. 8.3% out of 2.5M test runs indicated DNSSEC validation and 4.3% indicated mixed validating and non-validating resolvers. The population of Google Public DNS users was estimated at 7.2%.

9.5 Summary and Conclusions

We presented a web-based method to determine whether a client is protected by DNSSEC validation. After applying this method in a practical measurement over almost 3 years, we found a major increase of client-side DNSSEC adoption. The occurrence of DNSSEC validation varies geographically. Sweden and the Czech Republic are early adopters of widespread validation, but other countries have closed the gap in the meantime. The median per-country validation ratio rose from 1% in 2012 to about 20% in 2015.

The validating Google Public DNS service is the largest DNS resolver and thus presumably affecting the validation ratios to a large extent. It is plausible—though unknown—that in many cases the network path between the client and Google is unprotected. For a reasonably secure setup, DNSSEC validation should be deployed near the end host. Furthermore, a portion of clients is using mixed validating and non-validating resolvers. This does not provide any security benefit because the client will fall back to the non-validating resolver in case of validation failure.

CHAPTER 10

Intermediate Caches

When forwarding queries through daisy-chained resolvers, each additional component increases the chance of a lookup failure. This is particularly true for DNSSEC, which increases the failure potential due to operational problems or due to its protocol design. The choice of whether to forward DNS queries through a chain of resolvers is a trade-off between availability and performance/scalability: multiple levels of resolver caching potentially improve lookup times and reduce DNS traffic.

In the following study, we evaluate the effectiveness of a large intermediate resolver cache operated at the ISP premises. Our metrics are client-side lookup latencies and server-side network load. We use trace-driven simulations to compare the effectiveness with a hypothetical scenario, in which the clients do not rely on a shared intermediate cache. The goal is to assess whether intermediate caches are necessary for the health of the Domain Name System, or whether validating DNSSEC resolvers should query authoritative name servers directly.

10.1 Problem Statement

As we mentioned in Section 3.1, DNSSEC was designed as backward compatible extension to the Domain Name System. New semantics have been added for name servers and resolvers, but in principle the same infrastructure ought to be reused. The DNS infrastructure was meant to be upgraded incrementally, without changes to the system architecture. In legacy DNS, resolvers are often daisy-chained, forming a structure of multi-level caches (cf. Figure 2.6). For example, the resolver of a host forwards DNS queries to the resolver on the local network gateway, which forwards DNS queries to the resolver of the ISP. In the security analysis in Section 5.5, we identified that daisy-chaining of resolvers degrades the availability of DNSSEC. To avoid being locked in to a validation failure of an upstream resolver, the DNSSEC specification mandates that validating resolvers should signal CD=1

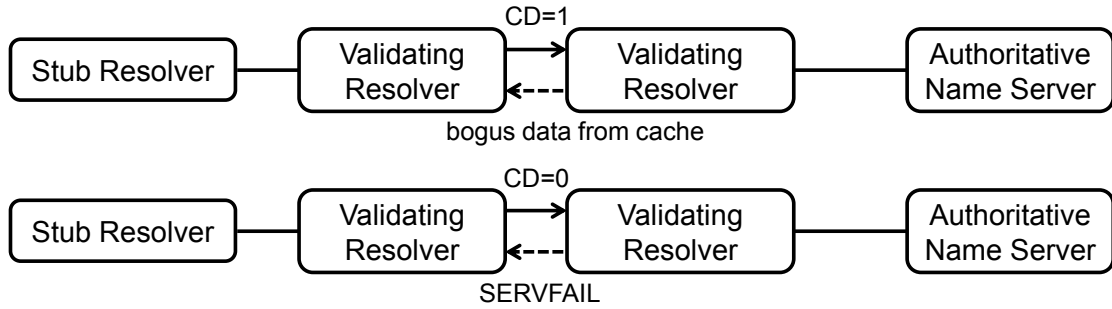


Figure 10.1: Lock-in to bogus data from upstream resolver.

(checking disabled) to upstream resolvers. This allows a downstream validator to retrieve the raw resource records and perform validation according to its local policies and trust anchors. However, it leads to a lock-in to potentially bogus data cached on an upstream resolver. Such an incident can be caused by a third-party spoofing attacker, stale data on an authoritative name server or deliberate DNS alteration by the upstream resolver operator. Figure 10.1 depicts both lock-in effects with $CD=1$ and $CD=0$.

Apart from lock-in to false data, upstream resolvers have been observed to introduce compatibility problems, e.g. due to a lack of EDNS0 support or by stripping off DNSSEC signatures. Home routers are especially prone to this type of failure [22, 39]. We are thus investigating the effects on client latency and authoritative name server load when intermediate caches were not used. In particular, we consider the lookup costs of out-of-bailiwick delegations, as these compose a major portion of DNS traffic, cf. the example in Section 2.8.

10.2 Method

We analyze DNS traffic¹ collected at a recursive name server R , which provides name resolution service for a campus network with 40,000 students and 4,500 staff members. The network traces cover two weeks of anonymized internal DNS traffic (between clients in the campus network and R) and external DNS traffic (between R and authoritative name servers in the Internet). The purpose of the internal traffic is to obtain a real world sample of requested names and types. The purpose of the external traffic is 1) to replicate the relevant subset of the domain namespace, and 2) to record the costs of the DNS transactions, i.e. round-trip time and message size.

Based on the data from the network traces of R , we simulate the resolver behavior shown as Algorithm 4. The algorithm corresponds to the reference algorithm sketched in RFC 1034 Section 5.3.3 [94] and resembles the high-level behavior of any recursive resolver

¹Data kindly provided by Harald Schöler, University of Duisburg-Essen.

Algorithm 4 Recursive resolver name lookup.

```
1: var
2:    $C_a : (\text{qname}, \text{qtype}) \mapsto \text{answer}$  ▷ Answer cache
3:    $C_r : \text{bailiwick} \mapsto \text{referral}$  ▷ Referral cache
4: end var
5:
6: function LOOKUP( $\text{qname}, \text{qtype}$ )
7:   if  $(\text{qname}, \text{qtype}) \in C_a$  then ▷ Answer in cache?
8:     return  $C_a(\text{qname}, \text{qtype})$ 
9:   end if
10:   $\text{sname} \leftarrow \text{FindBestServersToAsk}(\text{qname})$  ▷ Find referral in cache (or use root)
11:  while True do ▷ Abort when answer is found
12:     $\text{response} = \text{SendQueryToServer}(\text{qname}, \text{qtype}, \text{sname})$ 
13:    if  $\text{response.is\_answer}()$  then
14:       $C_a(\text{qname}, \text{qtype}) = \text{response}$  ▷ Put answer in cache
15:      return  $\text{response}$  ▷ Lookup finished
16:    else if  $\text{response.is\_cname}()$  then
17:       $C_a(\text{qname}, \text{qtype}) = \text{response}$ 
18:      return  $\text{Lookup}(\text{response.cname\_target}, \text{qtype})$  ▷ Resolve CNAME target
19:    else if  $\text{response.is\_referral}()$  then
20:       $C_r(\text{response.delegation\_name}) = \text{response}$  ▷ Put referral in cache
21:      for  $\text{delegation} \in \text{response}$  do
22:        if  $\text{delegation.servername}$  is out-of-bailiwick then ▷ No glue
23:           $\text{Lookup}(\text{delegation.servername}, A)$  ▷ Resolve server name
24:        end if
25:      end for
26:    end if
27:  end while
28: end function
29:
30: function FIndBestServersToAsk( $\text{qname}$ )
31:   $\text{bailiwick} \leftarrow \text{qname}$ 
32:  while  $\text{bailiwick} \notin C_r \wedge \text{bailiwick} \neq '.'$  do ▷ Until we find a match
33:     $\text{bailiwick} \leftarrow \text{bailiwick.parent}()$ 
34:  end while
35:  return  $C_r(\text{bailiwick}).\text{servernames}$ 
36: end function
```

implementation. We assume zero costs for cache hits and determine the specific costs of cache misses subject to partial cache contents. When a cache miss occurs, the resolver will first determine the best matching server to which it will send the query (line 10, function defined in lines 30–36). The resolver will then repeat sending queries and iterate through referrals until it finds the answer. When encountering a CNAME alias response, the resolver will initiate a follow-up lookup of the CNAME target name (line 18). Furthermore, the resolver initiates lookups for each out-of-bailiwick delegation to retrieve the IP addresses of authoritative name servers (lines 21–25). We simulate the following models and investigate how the lookup costs vary between them:

- $C1_{s,c}$: the client-side resolvers are served by one intermediate cache, which communicates with authoritative name servers. This is the baseline, which corresponds to the current deployment.
- $Cn_{s,c}$: each client-side resolver has its own cache, omits the intermediate cache and communicates directly with authoritative name servers.
- $C1_c$: like $C1_{s,c}$, but without resolution of out-of-bailiwick server names.
- Cn_c : like $Cn_{s,c}$, but without resolution of out-of-bailiwick server names.
- $C1$: like $C1_c$, but without resolution of CNAME responses.
- Cn : like Cn_c , but without resolution of CNAME responses.

In $C1_c$ and Cn_c , we assume that all delegations are in-bailiwick delegations, which contain glue records and which do not require separate server name lookups. The purpose of these two models is to determine the overhead of out-of-bailiwick server name lookups. In $C1$ and Cn , we furthermore assume that answers are final and that follow-up lookups of CNAME target names are not required. The purpose is to determine the overhead of resolving CNAME aliases.

10.3 Data Collection

The data has been collected at the network interface of the recursive name server R . The setup is shown in Figure 10.2. The network traces have been created with *tcpdump* in pcap format over a period of 16 days in September 2013. The campus network served by R is a heterogeneous network, comprising work stations of faculty and administration, eduroam Wi-Fi, student dorms, associated institutes and schools. The resolvers querying R are either stub resolvers on end hosts or resolvers that serve work groups. R is actually one of four recursive name servers that serve the campus network. We limit our study to R because it is the busiest name server (probably because it is listed first in network configurations) and

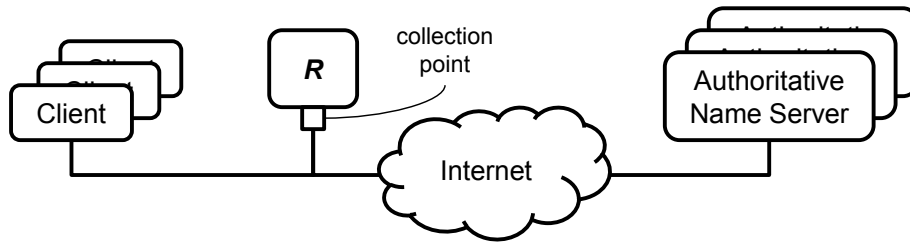


Figure 10.2: The network traces comprise internal traffic between clients and R , and external traffic between R and authoritative name servers.

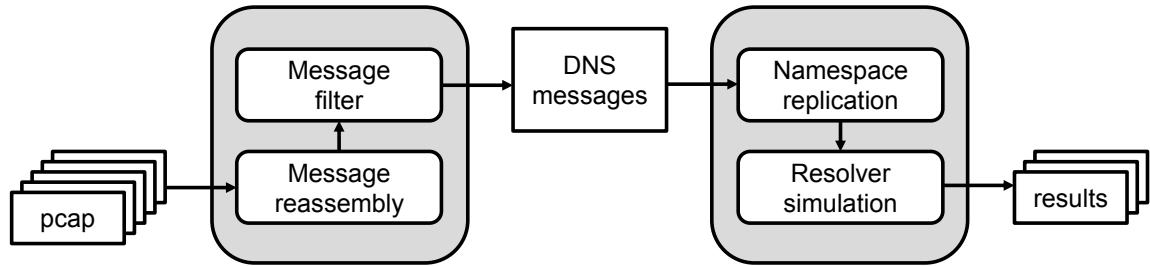


Figure 10.3: Software architecture of caching simulation.

because we do not expect a substantially different DNS traffic profile on the other name servers.

Anonymization

The network traces have been anonymized during data collection in order to remove personally identifiable information of the campus network users. The anonymization method consists in mapping IP addresses and resetting Ethernet MAC addresses. Any IP address that matches one of the two campus /16 IPv4 network prefixes has been mapped to a 24-bit value during data collection. The mapping function is the Keyed-Hash Message Authentication Code (HMAC) as defined in RFC 2104 [76] with a secret key chosen by a system administrator that operates R . The rationale for using a mapping function instead of zeroing IP addresses is that we need to preserve the traffic profile of individual resolvers for simulation of Cn . The rationale for using a keyed-hash function is that we get a random but deterministic mapping, which does not change when the network capture is interrupted and restarted for operational reasons. It is unlikely but possible that two different campus IP addresses map to the same anonymized IP address. In this case, we would treat two different client-side resolvers as one resolver. External IP addresses are kept unchanged.

10.4 Implementation

The implementation is decomposed into two software modules, as shown in Figure 10.3:

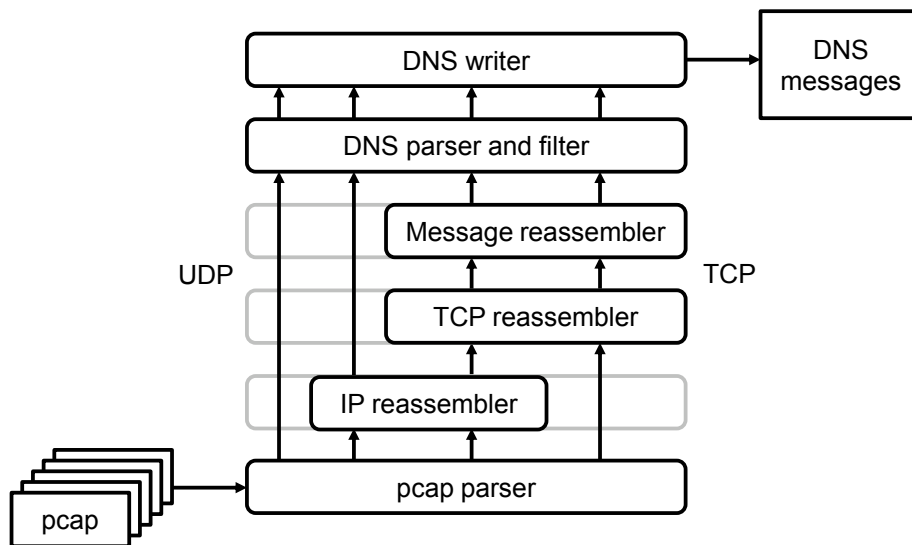


Figure 10.4: Layered reassembly of DNS messages from pcap files.

1. Reassembly of DNS messages from pcap files and filtering of broken or unwanted DNS messages.
2. Replication of domain namespace and simulation of DNS lookups according to the cache models listed in Section 10.2.

Both modules are implemented in Python and use *dnspython*² for DNS message parsing. *dpkt*³ is used for reading pcap files and parsing Ethernet, IPv4, IPv6, TCP and UDP headers.

10.4.1 Reassembly of DNS Messages

The pcap network traces contain Ethernet frames carrying IP packets with TCP segments or UDP datagrams. In order to extract entire DNS messages, we parse the input pcap files with a layered approach shown in Figure 10.4, which reassembles IP fragments and TCP segments and writes DNS messages into one binary output file. The DNS messages are ordered by their appearance in the network trace; fragmented messages appear once they can be reassembled into a complete message.

Each layer is implemented as generator function that pulls input data from the lower layer and that yields the output to the upper layer. The *pcap parser* reads 1 to n pcap files, extracts IP packets and passes the IP payload to the next layer. If fragmented, the pcap parser will extract fragmentation offset and ID value and will pass them to the IP reassembly layer. The *IP reassembler* merges IPv4 or IPv6 data fragments and passes the payload

²<http://www.dnspython.org/>

³<https://code.google.com/p/dpkt/>

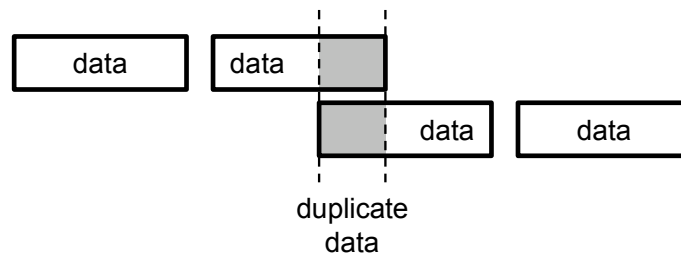


Figure 10.5: Partly overlapping TCP segments.



Figure 10.6: DNS messages over TCP streams are not necessarily aligned to TCP segments.

data once the IP packet has been reassembled entirely. Duplicate IP fragments are ignored. Incomplete IP packets are dropped after 10 minutes according to packet timestamps recorded in the pcap files.

With UDP transport, each datagram contains exactly one entire DNS message, which is passed through to the DNS parser. With TCP, the *TCP reassembler* extracts the TCP payload and passes each data segment to the message reassembler. TCP streams are tracked as unidirectional channels, i.e. each bidirectional TCP connection will be processed as two independent streams. Each TCP stream is initialized by a SYN segment (or SYN/ACK segment) and concluded by a segment with FIN or RST flag set, but only if the sequence numbers match. Duplicate or out-of-order data is handled in accordance with TCP sequence number arithmetic [101]. Retransmitted data does not need to be aligned to the same segment sizes, i.e. partly overlapping segments are handled appropriately (see Figure 10.5). Data segments are passed in correct stream order to the message layer once they are available. Idle streams are torn down after 10 minutes.

The objective of the *message reassembler* is to extract DNS messages from TCP streams. Each DNS message in a TCP stream is prepended with a two-bytes message length field (cf. DNS network protocol in Section 2.6). As neither the length fields nor the DNS messages are necessarily aligned to TCP data segments (Figure 10.6), the message reassembler joins TCP data segments into DNS messages and passes them to the DNS parser. Incomplete message buffers are deleted after 10 minutes idle time.

10.4.2 DNS Parser and Filter

The *DNS parser and filter* attempts to parse DNS messages. Broken messages are ignored, and so are queries and responses for classes other than “IN” or for the query type “ANY”.

Furthermore, we filter all DNS messages for domains served locally in the campus network. This includes domains for which *R* is set up as authoritative name server, and domains which *R* resolves by querying another authoritative name server in the campus network. The list of authoritatively served domain names has been determined by pre-processing the pcap files for authoritative DNS responses (QR=1, AA=1), which originate from any of the campus IP address ranges. After DNS message reassembly and filtering, 169M DNS messages have been extracted from 114 GBytes pcap files and are written into one 24 GBytes binary output file with meta information like timestamp, IP addresses and port numbers.

10.4.3 Namespace Replication

We process external DNS messages between *R* and authoritative name servers to replicate the domain namespace that is needed for the cache behavior simulation. The resulting namespace is time-variant, i.e. changes of delegations, message size or round-trip times are reflected in the replicated namespace view.

Queries are matched to responses with a sliding window approach by their transaction ID, query port number and destination server address. Pending queries time out after 1 minute according to the message timestamps. Matching responses are used in a two-stage process: 1) replicate the domain bailiwicks that name servers are responsible for, and 2) replicate the contents of the domain namespace along with the transaction costs and caching times.

First, we first need to learn the domain bailiwicks of name servers. This information cannot be determined from a self-contained server response because it depends on the delegation returned from the parent domain. We thus use referral responses to gradually build up a delegation graph from root to the leafs. With knowledge of the bailiwick of a name server, we strip off out-of-bailiwicks resource records from responses. This resembles the behavior of recursive resolvers for protecting themselves from out-of-bailiwick cache poisoning attacks [116]. Responses frequently contain out-of-bailiwick records, which are usually not caused by attacks but are ignored anyway. For example, the same name servers that serve `com` also serve the `net` top-level domain. When querying for a domain `example.com` that is delegated to the name servers `a` and `b.iana-servers.net`, the `com` servers will include glue records for the server names under `net`, although they are out-of-bailiwick of `com`.

After the first stage has ensured that all response contents are in-bailiwick, the second stage comprises the replication of the domain namespace. For each bailiwick, we store whether the authoritative name servers respond with a referral or an answer for the requested query name and type. A referral consists of a delegation to other name servers; we store the target bailiwick and TTL value of the delegation, the server names and whether there are in-bailiwick glue records. When encountering an answer, we store the TTL value of the answer records. The actual record data of answers is not of interest for us, except

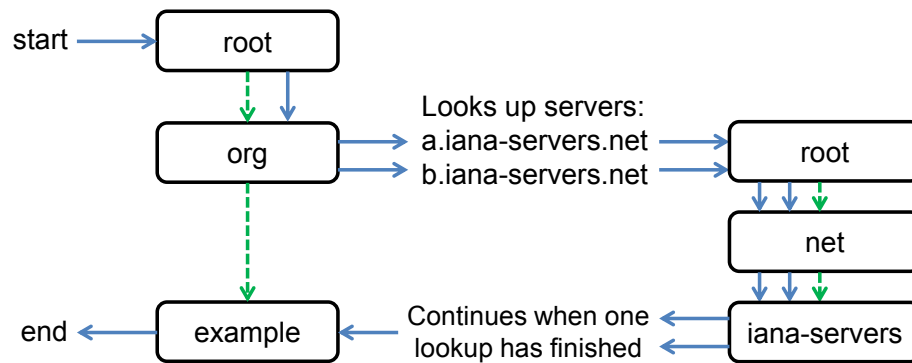


Figure 10.7: A resolver traverses through the domain name space along the solid blue lines. It will send a network query for each traversed node, if the response is not in cache. The resolver follows delegations (dashed green lines) and spawns new lookups when encountering out-of-bailiwick server names without glue record.

for CNAME records, for which we store the target name, which the CNAME alias points to. For each response, we also store the transaction costs in terms of message sizes (query plus response) and round-trip time.

The result of the above process is a data structure comprising the domain namespace, which the simulated resolvers can iterate through. Both stages of the replication process are executed interleaved and the resulting data structure is not fixed. Instead, changes in the domain namespace or variations of round-trip times are reflected by continuously updating the bailiwick mappings and authoritative responses to the latest state as seen in the network traces.

It is worth noting that the namespace builds up slowly at the beginning, because the network traces were recorded while R had an already populated cache. After one or two days most entries, if not all, should have expired and been looked up again, so that our replicated view of the namespace will be near to complete.

10.4.4 Simulation

Based on the internal DNS messages between client-side resolvers and R , we push internal lookup requests into six simulation models as defined in Section 10.2. The $C1$ models work with one shared resolver cache and disregard client IP addresses. The Cn models simulate one resolver cache for each anonymized client IP address.

The resolvers work as shown in Algorithm 4 (page 140), with the traversal through the namespace visualized in Figure 10.7. The resolver iterates through domain bailiwicks (blue lines) by following delegations (dashed green lines). Sending network queries is simulated by looking up the response in the replicated domain namespace. When encountering out-of-bailiwick server names, the resolver spawns new parallel lookups for each server name.

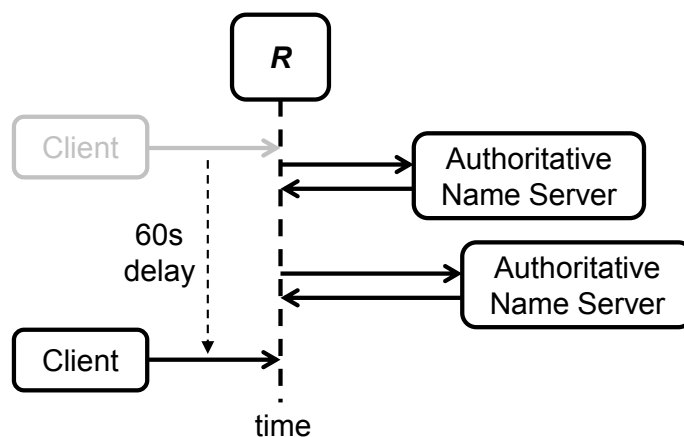


Figure 10.8: Client lookups are delayed by 60 seconds in the simulation.

The network traffic caused by a lookup is the sum of all external DNS transactions. The lookup latency is the sum of round-trip times of external DNS transactions, except when resolving out-of-bailiwick server names, which occurs in parallel and thus does not cumulate. If there is an in-bailiwick delegation with glue, we will assume zero latency (but still count the network traffic for resolving out-of-bailiwick server names). If all server names of a delegation are out-of-bailiwick, we will consider the latency for looking up the first server name (but still count the network traffic for resolving the others). We simulate the caching of answers and referrals, and purge expired entries based on the respective TTL values.

The simulation runs interleaved with the namespace replication, i.e. the resolvers access the current namespace contents at the simulated time. The simulation of internal queries is delayed by 60 seconds as shown in Figure 10.8 in order to incorporate external DNS transactions into the replicated namespace before the resolvers attempt to access them. Furthermore, we omit the first two days of client queries while the replicated namespace is still in the process of being populated and limit our simulation to the remaining 14 days.

10.4.5 Limitations

Although our simulation resembles the actual behavior of recursive resolvers very closely, a few limitations apply.

- For the sake of simplicity, we simulate successful lookups only. We disregard server failures, timeouts and retransmissions, which increase the lookup costs in reality.
- Lookups that result in negative responses are not simulated.
- When measuring the time between queries and responses, we consider the time when the messages were fully available. In case of DNS over TCP, this neglects the time needed to establish the TCP connection before a TCP-based query is sent.

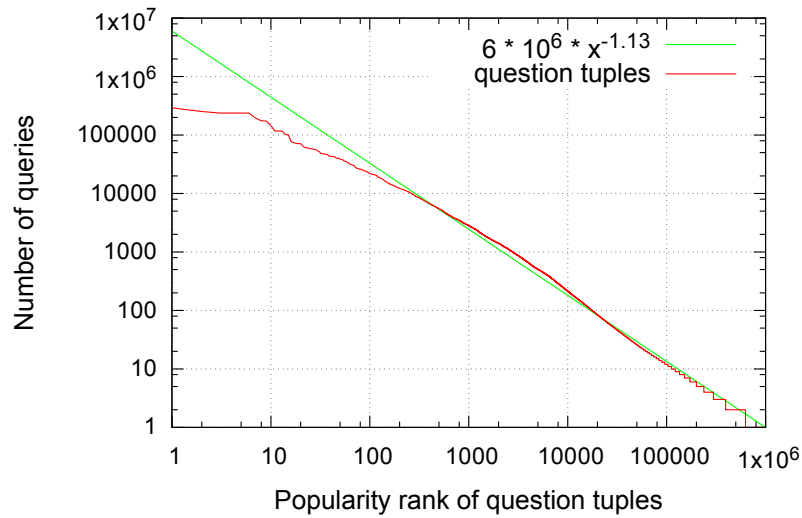


Figure 10.9: Distribution of query names and types as log-log plot.

- The simulation includes client-side A and AAAA lookups. However, the lookup of out-of-bailiwick server delegations includes A records only, because *R* had not been using IPv6 in production at that time. Dual-stack resolvers would look up both A and AAAA records, which increases the lookup costs of out-of-bailiwick delegations.
- *R* is located in a data center, with a low-latency link into the WAN. When pushing recursive resolvers to the edges of the Internet, the last mile would introduce a latency penalty on each external DNS transaction.

Due to the above aspects, our simulation results resemble a lower bound for the actual costs. We consider the time for various aspects of the simulation (time of client queries, time of cache expiry, changes of replicated namespace over time) but process each lookup sequentially. In reality, parallel lookups may influence each other, depending on the inter-arrival times of responses from different name servers.

10.5 Result Analysis

The simulations comprise 24.8M lookups from 10k client-side resolvers over a period of two weeks. The distribution of query names and types is shown in Figure 10.9 as log-log plot. The most frequent questions are for `www.google.com`, a content-delivery network serving `www.apple.com`⁴ and `daisy.ubuntu.com`. The long tail suggests that the power law applies, but the graph flattens for the most frequent question tuples. This may be the result of client-side caching, which could be effective to reduce lookups of the 100 most

⁴Resolved over a chain of CNAME records with different TTL values.

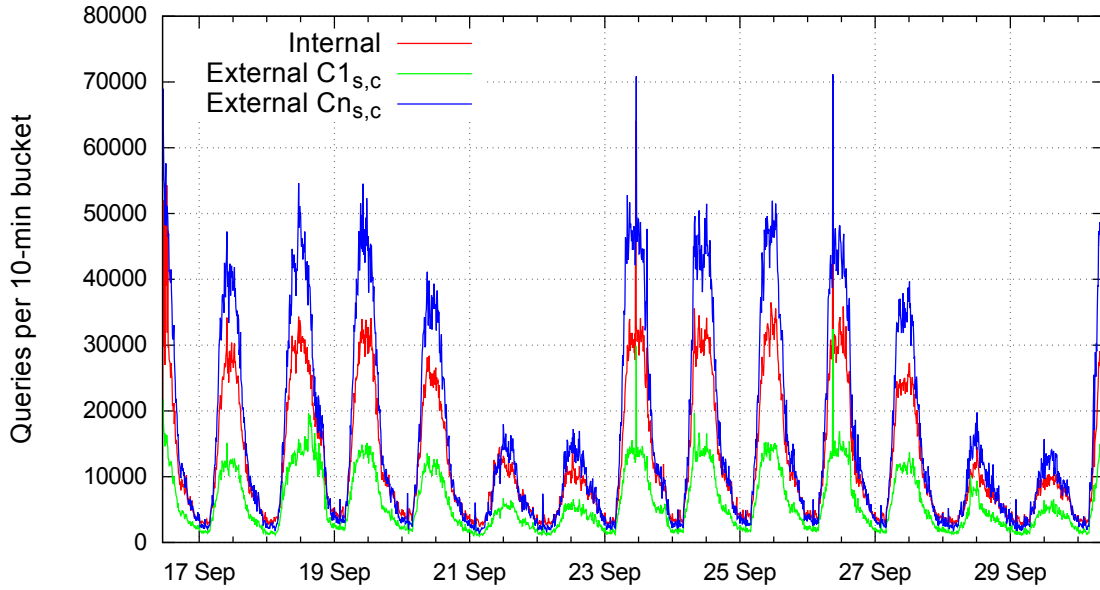


Figure 10.10: Internal and external queries, grouped per 10-minute bucket.

frequently accessed domains. Although the individual domain names are specific to the user population, we find similar domain distributions in other networks and countries [70, 129], independent of the users' language. We thus argue that the observations of our trace-driven simulations apply universally, albeit to a varying extent.

The simulation models are defined in Section 10.2: the $C1$ models simulate one shared resolver cache and the Cn models simulate one cache per n independent resolvers. The $_s$ modifier indicates that the models considers the resolution of out-of-bailiwick server names in delegations. The modifier $_c$ indicates that the models considers follow-up lookups of CNAME aliases. The baseline is $C1_{s,c}$, which corresponds to the actual behavior of R .

Figure 10.10 depicts the simulated queries over time, grouped into 10-minute buckets. The internal queries (red line) are subject to a diurnal cycle, with few activity at night and on weekends. The number of external queries for $C1_{s,c}$ (green line) sent over the Internet is well below the internal queries. For $Cn_{s,c}$ (blue line), there are about 2–3 times as many external queries. The external queries exceed the internal queries most of the time, indicating that it costs on average more than one external query to respond to one internal lookup.

Comparing the costs of $C1_{s,c}$ and $Cn_{s,c}$ in Table 10.1, the cache hit ratio decreased from 62.51% to 36.70% and the external traffic tripled in terms of message size (not counting protocol headers on lower layers). The average number of external queries per lookup rose from 0.49 to 1.31. This measure is smoothed by the amount of cache hits, which do not trigger any external queries. Considering cache misses only, there are 1.32 external queries

| | 1 shared cache | | | 10,278 caches | | |
|-----------------------------|----------------|---------|---------|---------------|---------|---------|
| | $C1_{s,c}$ | $C1_c$ | $C1$ | $Cn_{s,c}$ | Cn_c | Cn |
| Hit ratio | 62.51% | 62.36% | 61.97% | 36.70% | 36.54% | 35.80% |
| Sum of messages | 2.44 GB | 2.33 GB | 2.09 GB | 7.55 GB | 5.67 GB | 4.50 GB |
| Queries/lookup | 0.49 | 0.47 | 0.42 | 1.31 | 1.00 | 0.77 |
| Queries/miss | 1.32 | 1.25 | 1.09 | 2.04 | 1.58 | 1.20 |
| Server lookups/miss | 0.22 | 0 | 0 | 0.73 | 0 | 0 |
| CNAME/miss | 0.29 | 0.29 | 0 | 0.44 | 0.44 | 0 |
| $Q_{0.5}$ time/lookup [ms] | 0 | 0 | 0 | 12 | 12 | 11 |
| $Q_{0.75}$ time/lookup [ms] | 14 | 14 | 14 | 42 | 40 | 31 |
| $Q_{0.9}$ time/lookup [ms] | 110 | 110 | 103 | 173 | 171 | 159 |
| $Q_{0.5}$ time/miss [ms] | 25 | 25 | 21 | 27 | 26 | 20 |
| $Q_{0.75}$ time/miss [ms] | 120 | 119 | 110 | 114 | 110 | 97 |
| $Q_{0.9}$ time/miss [ms] | 193 | 191 | 188 | 206 | 194 | 183 |

Table 10.1: Results of 24.8M simulated lookups with various caching models.

per lookup for $C1_{s,c}$ and 2.04 for $Cn_{s,c}$. This demonstrates that it is not sufficient to compare the cache hit ratios, because the cost of a cache miss varies significantly. The reason becomes evident when looking at the number of follow-up lookups per cache miss: the number of server name lookups (due to out-of-bailiwick delegations without glue) has increased by about 230% and the number of CNAME alias lookups has increased by about 50%.

Figure 10.11 depicts the lookup latencies for $C1_{s,c}$ and $Cn_{s,c}$, covering both, cache hits and misses. 90% of all queries finish within 110 ms with the intermediate cache, but take up to 170 ms without it. The delta of each model to the baseline $C1_{s,c}$ is shown in Figure 10.12a. If all subdomain delegation contained glue ($C1_c$), 3% of all lookups would be faster by ≥ 5 ms (2% faster by ≥ 10 ms). If CNAME aliases were not used either, 12% of all lookups would be faster by ≥ 5 ms (5% faster by ≥ 10 ms).

The latency penalty without an intermediate cache is shown in Figure 10.12b. About 36% of lookups are served from client cache, but cache misses take longer than without an intermediate cache. The median latency overhead is 12 ms for $Cn_{s,c}$, the 75th percentile is 28 ms and the 90th percentile is 63 ms. Part of the overhead can be reduced, as the results for Cn_c and Cn indicate.

We should note that $C1$ and Cn are idealized models that are probably not feasible in practice to this extent. CNAME aliases are often used by content-delivery networks, which deliberately use a low TTL value on the last record set of the CNAME chain for fine-grained load balancing.

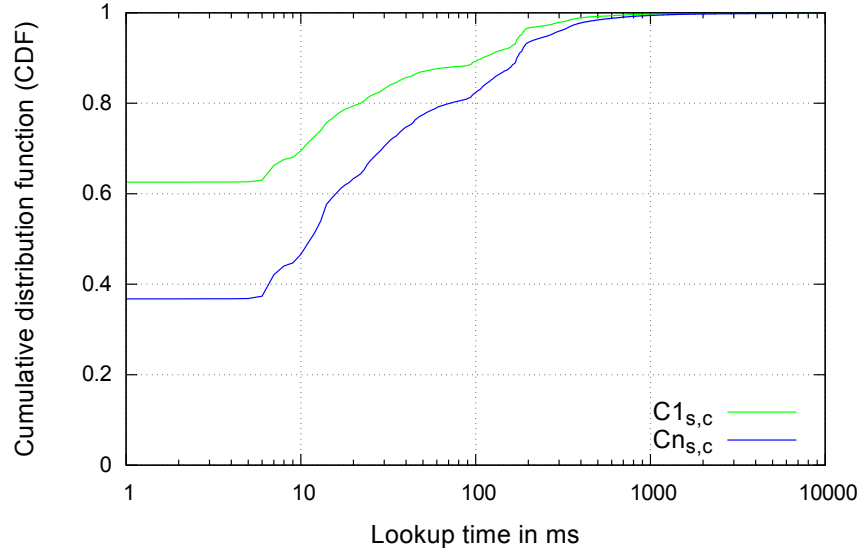


Figure 10.11: Lookup times with and without intermediate caching.

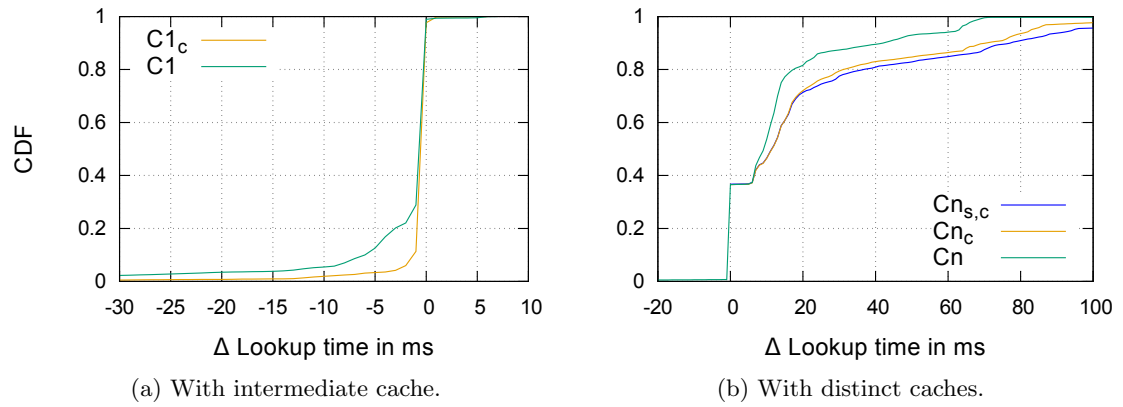


Figure 10.12: Changes of lookup times compared to baseline $C1_{s,c}$.

10.6 Related Work

Prior work falls into two categories: 1) effectiveness of DNS caching, 2) relevance of the bailiwick in domain name delegations.

DNS Caching

Jung et al. [70] studied the effectiveness of DNS caching in 2002 by evaluating external DNS traffic recorded over the WAN link of a campus network. Their log-log plot of most popular domain names closely fits ours in Figure 10.9, emphasizing that this is a naturally occurring pattern. The authors found that referral caching improves the DNS performance significantly, whereas answer caching is of limited value. The lookup latency rises with the number of referrals that a recursive resolver has to traverse. Jung et al. simulated the cache hit ratio for 1216 clients, which was 89% with a shared cache and 71% with distinct caches. Our study follows up on these findings and contributes the actual costs of a cache miss, which vary for shared and distinct caching models. The effectiveness of caching does not only depend on the number of referrals but also whether the referrals contain in-bailiwick glue records. Unlike the number of parent referrals, the choice of whether to use in-bailiwick or out-of-bailiwick server names can usually be controlled by domain administrators.

Bill Manning [89] suggested to share DNS caches between 10 or fewer clients to reduce the attack surface for DNS tampering. In an experiment with 140 clients, he redistributed the clients from one shared cache to 10 caches and observed traffic for 15 minutes after cache flush. The number of external queries increased by about 16%. Although the scope of the experiment is limited, it contributes a bit to the current state of knowledge that cache sharing improves the scalability to a moderate degree.

Zheng Wang [129] analyzed the query name distribution at the authoritative name servers of the `cn` top-level domain. This is the same measure that we have shown in Figure 10.9, but we collected it client-side on a recursive resolver, while Wang collected it server-side on authoritative name servers. Both distributions are nonetheless similar to each other. Models with 1, 10 or 100 clients per cache show that the distributions flattens more with a larger cache for popular domains, but does not change in the long tail. This strengthens our assumption that client-side caching flattens the most popular domains in our Figure 10.9.

Schomp et al. [112] suggested that abandoning shared DNS caches would have a modest impact on TCP-based applications. They compared the DNS lookup times in a network trace with a shared resolver to a simulated scenario with name resolution running on the end hosts. In Figure 3 the authors show that 85% of DNS lookups were delayed by ≤ 50 ms when a shared resolver was not used. The graph also shows that 20% of lookups became faster, which is not possible under normal conditions; the delay between the hosts and

the shared resolver is too short to explain this effect. One possible explanation is that the shared resolver evicted records prematurely from cache due to capacity limitations⁵. Another reason may be an effect of the tool *dig*, which was used to gather the costs of resolving domain names. *dig* in iterative mode (`+trace` option) automatically resolves out-of-bailiwick server names by recursively asking the name server that is configured in the operating system's `resolv.conf`. This feature can be neither turned off nor is it documented in the man page, but it can be observed with *tcpdump*. As the authors took the average of 5 measurements, the times used in the simulation will be partly served from the recursive resolver's cache and thus underestimated. Furthermore, the costs for following CNAME chains will not be considered, because *dig* does not resolve CNAME aliases.

Bailiwick

Ramasubramanian and Sirer [104] studied the transitive trust that follows from relying on authoritative name servers of parent domains. The number of dependencies of a domain increases with the use of out-of-bailiwick server names in delegations. The authors determined in 2005 that a domain depends on 46 name servers on average, and 6.5% of domains depend on more than 200 name servers in their delegation graph. This emphasizes another aspect besides the performance impact: out-of-bailiwick delegations introduce dependencies on name servers managed by third-parties, which threatens the reliability of a domain. When DNSSEC is not used, each additional dependency increases the potential for DNS attacks.

10.7 Summary and Conclusion

We investigated the effectiveness of DNS caching on intermediate resolvers with trace-driven simulations. A shared resolver in front of 10k clients compared to 10k clients with distinct caches reduces lookup times of the 75th percentile by 66% from 42 to 14 ms and reduces network traffic by 68% from 7.55 to 2.44 GBytes. Although our simulation resembles the actual behavior of recursive resolvers very closely, the results form a lower bound because we did not consider interference like server errors and retransmissions. This is the first study about DNS caching effectiveness that considers the costs of looking up server names in out-of-bailiwick delegations. We have shown that if all delegations were in-bailiwick delegations with glue, this would reduce the query count per cache miss and improve the scalability of the Domain Name System. This applies in particular when intermediate resolvers are not used as shared caches. We thus advise domain administrators to set up in-bailiwick delegations with glue records by default, i.e. delegate a domain like `example.net` to a server name like `ns1.example.net`. Out-of-bailiwick delegations increase lookup costs and add

⁵Suggested by Kyle Schomp in private correspondence.

interdependencies between domains. They should be used only if operationally justified, e.g. to ease management.

Intermediate caching is beneficial, but the benefit is not large enough to justify an unconditional enforcement. Our recommendation is to perform DNSSEC validation on the end host and to utilize an intermediate cache if available, but to fall back to autonomous name resolution in case of DNSSEC validation failure.

CHAPTER 11

Conclusions and Outlook

The Domain Name System is an essential Internet service for mapping domain names to resources. In this dissertation we discussed the impact of the DNSSEC security extensions.

11.1 Conclusions

DNS spoofing attacks are common in some parts of the world to block access to domain names and websites. DNS injection, which is a class of DNS spoofing used in China and Iran, can be observed from any vantage point by probing the IPv4 address space. We provided methods for identification of blocked names and opportunistic detection of spoofed responses without DNSSEC. Foreign third-parties that are routed through a poisoned network will be affected by DNS injection, too. We determined that 6% of open resolvers outside of China were affected in 2013 when accessing the `kr` top-level domain because one of the anycast server nodes of `kr` has been hosted in China. When neither the query sender nor the destination server were located in China, we observed sporadic evidence for Chinese DNS injection but not a systematic issue with transit routing. We argue that this will not occur systematically with Iranian DNS injection either, because Iran’s domestic network is largely isolated from the Internet. DNSSEC protects foreign third-parties from inadvertent DNS injection, but is of limited suitability for circumventing domain blocking from inside of an affected network due to cleartext domain names in DNSSEC messages.

Privacy is considered only in the NSEC3 protocol extension, which aims to prevent disclosure of the domain database. We demonstrated efficient GPU-based attacks against the NSEC3 privacy goal, which reveals 64% of DNSSEC names of the `com` top-level in 5 days and works well also for other domains. Operators should not assume that NSEC3 prevents zone enumeration and instead consider it as a slight to moderate slowdown at the price of higher resource costs per negative response. NSEC3 is popular for its opt-out feature, which is not specified for the alternative NSEC method, but this will become less relevant

with an increasing adoption of DNSSEC signing.

We showed that DNSSEC signing has been adopted in practice, even though it forms a minority compared to legacy DNS. A measurement-based survey of all top-level domains yielded 5.1M DNSSEC domains, with RSA as dominant signing algorithm. Out of a subset of 3.4M domains, 0.4% have used 512-bit or 768-bit RSA keys, which is demonstrably insecure, and 92% have used at least one 1024-bit RSA key, which is considered as not sufficiently secure anymore. A separation of keys as in the KSK/ZSK scheme was used by 89% of domains, which is only useful if the keys are stored within different security perimeters. Our recommendation to domain administrators is to use only one signing key in the general case, which should be ≥ 2048 bits long when RSA is used. Elliptic curve cryptosystems with 256-bit keys are a reasonable alternative. With appropriately sized keys it is not the cryptosystem that constitutes the weak link in security but operational issues.

In three decades of operation, the Domain Name System has proven robust as it tolerates configuration errors and component failures to a certain degree. DNSSEC introduces a new layer of complexity, which bears the risk of increasing fragility. Errors in key management or cryptographic data are not tolerated, as this would contradict the integrity and authenticity security goals of DNSSEC. We found 21k DNSSEC-signed domains (0.6%) that failed to validate due to a broken authentication chain. Part of these failures like expired signatures are caused by an improperly conducted manual maintenance, which is tedious and error-prone. Instead, signing and key management should be automated as far as possible in the general case.

Concerning client-side adoption, we have noted a worldwide increase of web clients relying on DNSSEC validation. The adoption varies geographically; out of 44 countries, the median validation ratio per country is about 20%. A significant portion of clients is using the validating Google Public DNS service, which may pose a distorted picture if the communication path between the client and Google is unprotected. Furthermore, an estimated 1.3% to 6.8% of results indicated that the client was using mixed validating and non-validating resolvers. For a reasonable protection, validation should be deployed at the edges of the Internet, either on or close to the end hosts. This would enable applications to utilize DNSSEC for other purposes than IP address resolution, e.g. authentication of TLS certificates with DANE.

The client-side deployment of DNSSEC is impaired by intermediate caches, which degrade the availability of DNSSEC name resolution. We observed that although shared, intermediate caches are useful to improve the performance and scalability of the Domain Name System, there are cases in which a validating resolver must query the authoritative name servers directly to retrieve a valid response. Clients should be provided with the capability to fall back to autonomous name resolution in case the cached path fails.

11.2 Outlook

The trust model of DNSSEC corresponds to the hierarchical namespace, in which the authority over subdomains is delegated to other name servers. This allows for distinct sub-namespaces being managed under different policies by independent organizational entities. However, parent domains must be trustworthy and the authority concentrates in the root of the namespace. The DNSSEC specification offers instruments that could be leveraged to increase the autonomy of domain operators. For example, public keys of top-level domains could be updated automatically via the trust anchor update mechanism [119], and override any potentially disparate data in the root domain. This would allow an autonomous security operation of top-level domains, which does not depend on the trust of the root. It is unclear at this time whether the trust anchor update mechanism works ubiquitously, but this will be tested in practice once the root KSK is replaced.

Concerning the weak NSEC3 privacy assertion, the existing remedies are effective but expensive as they involve on-the-fly signing. An alternative could be to equip authoritative name servers with GPU-based NSEC3 accelerators. This would allow server operators to increase the hash iteration count and to relieve the CPU for other server duties. One of the challenges of this approach is to ensure a low latency of server responses to avoid penalties for legitimate queries.

DNSSEC resolvers do not signal the cause of a validation failure and instead return a generic server failure, which is indistinguishable from network errors. Unlike e.g. TLS certificate failures, there is currently no application-level handling of DNSSEC validation failures. Future work should explore whether it is useful for applications to react to validation failures and whether it is useful to escalate validation failures to the user.

We noted that intermediate resolvers can interfere with DNSSEC name resolution, and that end hosts can tackle this by falling back to autonomous name resolution on a case-by-case basis. However, this will not work in networks in which middleboxes restrict outgoing DNSSEC traffic. Coping with middleboxes that obstruct a clean UDP or TCP path to the authoritative name server remains an open issue for future work.

Bibliography

- [1] D. Eastlake 3rd. DSA KEYS and SIGs in the Domain Name System (DNS). RFC 2536, March 1999.
- [2] D. Eastlake 3rd. RSA/SHA-1 SIGs and RSA KEYS in the Domain Name System (DNS). RFC 3110, May 2001.
- [3] D. Eastlake 3rd and C. Kaufman. Domain Name System Security Extensions. RFC 2065, January 1997.
- [4] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, et al. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. <https://weakdh.org/imperfect-forward-secrecy.pdf>, May 2015.
- [5] Alexa – The Web Information Company. Top 1 000 000 Sites on the web. <http://s3.amazonaws.com/alexastatic/top-1m.csv.zip>.
- [6] Collin Anderson. The Hidden Internet of Iran: Private Address Allocations on a National Network. *CoRR*, abs/1209.6398, 2012.
- [7] Collin Anderson. Dimming the Internet: Detecting Throttling as a Mechanism of Censorship in Iran. *CoRR*, abs/1306.4361, 2013.
- [8] Collin Anderson, Philipp Winter, and Roya. Global Network Interference Detection Over the RIPE Atlas Network. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*, San Diego, CA, August 2014. USENIX Association.
- [9] M. Andrews. Negative Caching of DNS Queries (DNS NCACHE). RFC 2308, March 1998.
- [10] Anonymous. The Collateral Damage of Internet Censorship by DNS Injection. *SIGCOMM Comput. Commun. Rev.*, 42(3):21–27, 2012.
- [11] Anonymous. Towards a Comprehensive Picture of the Great Firewall’s DNS Censorship. In *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*, San Diego, CA, August 2014. USENIX Association.
- [12] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033, March 2005.

-
- [13] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035, March 2005.
 - [14] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034, March 2005.
 - [15] R. Arends, M. Kesters, and D. Blacka. DNS Security (DNSSEC) Opt-In. RFC 4956 (Experimental), July 2007.
 - [16] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. Internet Censorship in Iran: A First Look. In *Free and Open Communications on the Internet*, Washington, DC, USA, 2013. USENIX Association.
 - [17] D. Atkins and R. Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833, August 2004.
 - [18] Elaine B. Barker, William C. Barker, William E. Burr, W. Timothy Polk, and Miles E. Smid. SP 800-57. Recommendation for Key Management, Part 1: General (Revision 3). Technical report, National Institute of Standards & Technology, July 2012.
 - [19] Jason Bau and John C. Mitchell. A Security Evaluation of DNSSEC with NSEC3. *IACR Cryptology ePrint Archive*, 2010:115, 2010.
 - [20] David Beazley. Understanding the Python GIL. In *PyCON Python Conference. Atlanta, Georgia*, 2010.
 - [21] R. Bellis. DNS Transport over TCP - Implementation Requirements. RFC 5966, August 2010.
 - [22] R. Bellis and L. Phifer. DNSSEC Impact on Broadband Routers and Firewalls. *Nominet*, 2008.
 - [23] Daniel J. Bernstein. Breaking DNSSEC, August 2009. Keynote lecture at Workshop on Offensive Technologies (WOOT).
 - [24] Daniel J. Bernstein and Arjen K. Lenstra. A general number field sieve implementation. In Arjen K. Lenstra and Jr. Lenstra, Hendrik W., editors, *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics*, pages 103–126. Springer Berlin Heidelberg, 1993.
 - [25] Stephane Bortzmeyer. DNS query name minimisation to improve privacy. <http://tools.ietf.org/html/draft-bortzmeyer-dns-qname-minimisation>, May 2014. IETF Internet-Draft (work in progress).
 - [26] Joppe W Bos, Marcelo E Kaihara, Thorsten Kleinjung, Arjen K Lenstra, and Peter L Montgomery. On the Security of 1024-bit RSA and 160-bit Elliptic Curve Cryptography. *IACR Cryptology ePrint Archive*, 2009:389, 2009.
 - [27] Carna Botnet. Internet census 2012. <http://internetcensus2012.bitbucket.org/>.
 - [28] Martin A. Brown, Doug Madory, Alin Popescu, and Earl Zmijewski. DNS Tampering and Root Servers. AMS-IX General Meeting, Nov 2010.

- [29] BSI TR-02102. Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Technical report, Bundesamt für Sicherheit in der Informationstechnik, February 2015.
- [30] David Chadwick. How Trust Had a Hole Blown In It. The Case of X.509 Name Constraints. In *5th Annual PKI R&D Workshop*. NIST, 2006.
- [31] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. Ignoring the Great Firewall of China. In *In 6th Workshop on Privacy Enhancing Technologies*, 2006.
- [32] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.
- [33] Quantcast Corp. Top US sites. <http://ak.quantcast.com/quantcast-top-million.zip>.
- [34] David Dagon, Manos Antonakakis, Kevin Day, Xiapu Luo, Christopher P Lee, and Wenke Lee. Recursive DNS Architectures and Vulnerability Implications. In *NDSS*, 2009.
- [35] David Dagon, Manos Antonakakis, Paul Vixie, Tatuya Jinmei, and Wenke Lee. Increased DNS Forgery Resistance Through 0x20-bit Encoding: Security via Leet Queries. In *Proceedings of the 15th ACM Conference on Computer and Communications Security, CCS '08*, pages 211–222, New York, NY, USA, 2008. ACM.
- [36] Jan Daley. Confirming the Nominet position. *namedroppers mailing list*, May 2004.
- [37] J. Damas, M. Graff, and P. Vixie. Extension Mechanisms for DNS (EDNS(0)). RFC 6891, April 2013.
- [38] Saeed Kamali Dehghan. Iran’s president signals softer line on web censorship and Islamic dress code. *The Guardian*, July 2013. <http://www.theguardian.com/world/2013/jul/02/iran-president-hassan-rouhani-progressive-views>.
- [39] Thorsten Dietrich. DNSSEC Support by Home Routers in Germany. In *Proc. 60th Réseaux IP Européens (RIPE) Meeting*, 2010.
- [40] V. Dolmatov, A. Chuprina, and I. Ustinov. Use of GOST Signature Algorithms in DNSKEY and RRSIG Resource Records for DNSSEC. RFC 5933, July 2010.
- [41] Haixin Duan, Nicholas Weaver, Zongxu Zhao, Meng Hu, Jinjin Liang, Jian Jiang, Kang Li, and Vern Paxson. Hold-On: Protecting Against DNS Packet Injection. In *Securing and Trusting Internet Names (SATIN)*, 2012.
- [42] D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, September 2001.
- [43] Mauricio Vergara Ereche. Odd behaviour on one node in I root-server. *dns-operations mailing list*, Mar 2010. <https://lists.dns-oarc.net/pipermail/dns-operations/2010-March/005260.html>.

-
- [44] P. Ferguson and D. Senie. Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing. RFC 2827, May 2000.
 - [45] International Organization for Standardization. *ISO 3166-1:2013 Codes for the representation of names of countries and their subdivisions – Part 1: Country codes*. ISO, Geneva, 3 edition, 2013.
 - [46] Kazunori Fujiwara. DNSSEC validation measurement. DNS-OARC Workshop, San Francisco, CA, USA, March 2011.
 - [47] Kazunori Fujiwara. Number of possible DNSSEC validators seen at JP. IEPG Meeting at IETF 83, Paris, France, March 2012.
 - [48] Kensuke Fukuda, Shinta Sato, and Takeshi Mitamura. A Technique for Counting DNSSEC Validators. In *INFOCOM, 2013 Proceedings IEEE*, pages 80–84. IEEE, 2013.
 - [49] Jeff Gilchrist. Beginners Guide to NFS factoring using GGNFS and MSIEVE, August 2010. http://gilchrist.ca/jeff/factoring/nfs_beginners_guide.html.
 - [50] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, Sachin Vasant, and Asaf Ziv. NSEC5: Provably Preventing DNSSEC Zone Enumeration. In *22th Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2015.
 - [51] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *SIGCOMM Internet Measurement Workshop*, 2002.
 - [52] W. Hardaker. Use of SHA-256 in DNSSEC Delegation Signer (DS) Resource Records (RRs). RFC 4509, May 2006.
 - [53] T. Hardie. Distributing Authoritative Name Servers via Shared Unicast Addresses. RFC 3258, April 2002.
 - [54] Dominik Herrmann, Karl-Peter Fuchs, Jens Lindemann, and Hannes Federrath. EncDNS: A Lightweight Privacy-Preserving Name Resolution Service. In *Computer Security-ESORICS 2014*, pages 37–55. Springer, 2014.
 - [55] Amir Herzberg and Haya Shulman. Unilateral antidotes to DNS poisoning. In *Security and Privacy in Communication Networks*, pages 319–336. Springer, 2012.
 - [56] Amir Herzberg and Haya Shulman. Fragmentation Considered Poisonous. In *Communications and Network Security (CNS), 2013 IEEE Conference on*, pages 224–232. IEEE, 2013.
 - [57] Amir Herzberg and Haya Shulman. Negotiating DNSSEC Algorithms over Legacy Proxies. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis Askoxylakis, editors, *Cryptology and Network Security*, volume 8813 of *Lecture Notes in Computer Science*, pages 111–126. Springer International Publishing, 2014.
 - [58] Tomas Hlavacek. IP fragmentation attack on DNS. RIPE 67 Meeting, Athens, Greece, October 2013.

- [59] P. Hoffman and W.C.A. Wijngaards. Elliptic Curve Digital Signature Algorithm (DSA) for DNSSEC. RFC 6605, April 2012.
- [60] A. Hubert and R. van Mook. Measures for Making DNS More Resilient against Forged Answers. RFC 5452, January 2009.
- [61] Geoff Huston. A Question of Protocol. Talk at DNS-OARC workshop, October 2013.
- [62] Geoff Huston and George Michaelson. Measuring DNSSEC Use. IEPG Meeting at IETF 87, Berlin, Germany.
- [63] ICANN. DNSSEC workshop held at ICANN 45 meeting, October 2012. <http://toronto45.icann.org/node/34375>.
- [64] ICANN. Base Registry Agreement. <http://newgtlds.icann.org/en/applicants/agb/base-agreement-contracting>, November 2013.
- [65] Internet Assigned Numbers Authority. Domain Name System (DNS) Parameters: Resource Record (RR) TYPES. <https://www.iana.org/assignments/dns-parameters/>. Accessed October 2014.
- [66] Internet Assigned Numbers Authority. Root KSK Ceremonies. <https://www.iana.org/dnssec/ceremonies>. Accessed April 2015.
- [67] Internet Assigned Numbers Authority. IANA Report on Request for Delegation of the .ps Top-Level Domain. <https://www.iana.org/reports/2000/ps-report-22mar00.html>, March 2000.
- [68] J. Jonsson and B. Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447, February 2003.
- [69] S. Josefsson. The Base16, Base32, and Base64 Data Encodings. RFC 4648, October 2006.
- [70] Jaeyeon Jung, Emil Sit, Hari Balakrishnan, and Robert Morris. DNS Performance and the Effectiveness of Caching. *Networking, IEEE/ACM Transactions on*, 10(5):589–603, 2002.
- [71] Dan Kaminsky. Black Ops 2008: It’s The End Of The Cache As We Know It. Black Hat USA, August 2008.
- [72] S. Kitterman. Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1. RFC 7208, April 2014.
- [73] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen Lenstra, Emmanuel Thomé, Joppe Bos, Pierrick Gaudry, Alexander Kruppa, Peter Montgomery, Dag Arne Osvik, Herman te Riele, Andrey Timofeev, and Paul Zimmermann. Factorization of a 768-bit RSA modulus. Cryptology ePrint Archive, Report 2010/006, 2010. <http://eprint.iacr.org/>.
- [74] Peter Koch. Using RIPE Atlas: A DENIC Case Study. <https://labs.ripe.net/Members/pk/denic-case-study-using-ripe-atlas>.

-
- [75] Peter Koch and Matt Larson. Initializing a DNS Resolver with Priming Queries. <https://tools.ietf.org/html/draft-ietf-dnsop-resolver-priming>, March 2015. IETF Internet-Draft (work in progress).
 - [76] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, February 1997.
 - [77] Marc Kührer, Thomas Hupperich, Christian Rossow, and Thorsten Holz. Exit from Hell? Reducing the Impact of Amplification DDoS Attacks. In *USENIX Security Symposium*, 2014.
 - [78] W. Kumari, O. Gudmundsson, and G. Barwood. Automating DNSSEC Delegation Trust Maintenance. RFC 7344, September 2014.
 - [79] NLnet Labs. unbound.conf manpage (unbound 1.5.3), 2015. <https://www.unbound.net/documentation/unbound.conf.html>.
 - [80] Ólafur Gudmundsson and Stephen D. Crocker. Observing DNSSEC validation in the wild. In *Securing and Trusting Internet Names (SATIN)*, 2011.
 - [81] B. Laurie, G. Sisson, R. Arends, and D. Blacka. DNS Security (DNSSEC) Hashed Authenticated Denial of Existence. RFC 5155, March 2008.
 - [82] Xiaodong Lee, Haikuo Zhang, Nan Wang, Peng Zuo, Xiali Yan, Ce Luo, and Hongtao Li. Weak Trust Anchor Introduction. <https://tools.ietf.org/html/draft-zhang-dnsop-weak-trust-anchor-00>, May 2014. IETF Internet-Draft (work in progress).
 - [83] Arjen K Lenstra, James P Hughes, Maxime Augier, Joppe W Bos, Thorsten Kleinjung, and Christophe Wachter. Public Keys. In *Advances in Cryptology-CRYPTO 2012*, pages 626–642. Springer, 2012.
 - [84] Matthew Lentz, Dave Levin, Jason Castonguay, Neil Spring, and Bobby Bhattacharjee. D-mystifying the D-root Address Change. In *Proceedings of the 2013 Conference on Internet Measurement Conference, IMC '13*, pages 57–62, New York, NY, USA, 2013. ACM.
 - [85] Wilson Lian, Eric Rescorla, Hovav Shacham, and Stefan Savage. Measuring the Practical Impact of DNSSEC Deployment. In *USENIX Security*, pages 573–588, 2013.
 - [86] F. Ljunggren, T. Okubo, R. Lamb, and J. Schlyter. DNSSEC Practice Statement for the Root Zone KSK Operator, October 2010.
 - [87] Graham Lowe, Patrick Winters, and Michael L. Marcus. The Great DNS Wall of China, 2007.
 - [88] G. Malkin. Internet Users’ Glossary. RFC 1983, August 1996.
 - [89] Bill Manning. Intermediate DNS Caching as an Attack Vector. *Internet Protocol Journal*, 12(2), 2009.

- [90] Simon Marechal. Advances in password cracking. *Journal in Computer Virology*, 4(1):73–81, 2008.
- [91] MaxMind. GeoLite geolocation and autonomous system database. <http://www.maxmind.com>.
- [92] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.
- [93] Ministry of Communications and Information Technology. <https://www.ict.gov.ir/fa/news/8741>, October 2013. Announcement in Persian language.
- [94] P.V. Mockapetris. Domain Names - Concepts and Facilities. RFC 1034, November 1987.
- [95] P.V. Mockapetris. Domain Names - Implementation and Specification. RFC 1035, November 1987.
- [96] Arvind Narayanan and Vitaly Shmatikov. Fast Dictionary Attacks on Passwords Using Time-space Tradeoff. In *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS '05*, pages 364–372, New York, NY, USA, 2005. ACM.
- [97] National People’s Congress (People’s Republic of China). The Basic Law of the Hong Kong Special Administrative Region of the People’s Republic of China, 1997.
- [98] Yutaka Oiwa, Kazukuni Kobara, and Hajime Watanabe. A New Variant for an Attack Against RSA Signature Verification Using Parameter Field. In Javier Lopez, Pierangela Samarati, and JosepL. Ferrer, editors, *Public Key Infrastructure*, volume 4582 of *Lecture Notes in Computer Science*, pages 143–153. Springer Berlin Heidelberg, 2007.
- [99] T. Okubo, F. Ljunggren, R. Lamb, and J. Schlyter. DNSSEC Practice Statement for the Root Zone ZSK Operator, May 2010.
- [100] J. Postel. Internet Protocol. RFC 791, September 1981.
- [101] J. Postel. Transmission Control Protocol. RFC 793, September 1981.
- [102] J. Postel. Domain Name System Structure and Delegation. RFC 1591, March 1994.
- [103] Hosnieh Rafiee, Christoph Mueller, Lukas Niemeier, Jannik Streek, Christoph Sterz, and Christoph Meinel. A Flexible Framework for Detecting IPv6 Vulnerabilities. In *Proceedings of the 6th International Conference on Security of Information and Networks, SIN '13*, pages 196–202, New York, NY, USA, 2013. ACM.
- [104] Venugopalan Ramasubramanian and Emin Gün Sirer. Perils of Transitive Trust in the Domain Name System. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 35–35. USENIX Association, 2005.
- [105] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918, February 1996.

-
- [106] RIPE NCC. BGPlay. <https://stat.ripe.net/special/bgplay>.
- [107] S. Rose. Applicability Statement: DNS Security (DNSSEC) DNSKEY Algorithm Implementation Status. RFC 6944, April 2013.
- [108] Marcos Sanz. DNSSEC and the Zone Enumeration. *European Internet Forum*, October 2004.
- [109] Yuri Schaeffer. NSEC3 Hash Performance, March 2010. NLnet Labs document 2010-002.
- [110] Otto Schily. Bundesdatenschutzgesetz (BDSG) [*Federal Data Protection Act*]. *Bundesgesetzblatt*, page 66, January 2003.
- [111] Oliver Schirokauer, Damian Weber, and Thomas Denny. Discrete Logarithms: The Effectiveness of the Index Calculus Method. In Henri Cohen, editor, *Algorithmic Number Theory*, volume 1122 of *Lecture Notes in Computer Science*, pages 337–361. Springer Berlin Heidelberg, 1996.
- [112] Kyle Schomp, Mark Allman, and Michael Rabinovich. DNS Resolvers Considered Harmful. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 16. ACM, 2014.
- [113] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. On Measuring the Client-Side DNS Infrastructure. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 77–90. ACM, 2013.
- [114] Kaan Sezyum. Image posted on twitter. <https://twitter.com/kaansezyum/status/446940008843206657>, March 2014.
- [115] Adi Shamir and Eran Tromer. On the Cost of Factoring RSA-1024. *RSA CryptoBytes*, 6:10–19, 2003.
- [116] Sooel Son and Vitaly Shmatikov. The Hitchhiker’s Guide to DNS Cache Poisoning. In *Security and Privacy in Communication Networks*, pages 466–483. Springer, 2010.
- [117] Marc Stevens. New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis. In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 245–261. Springer Berlin Heidelberg, 2013.
- [118] Joe Stewart. DNS Cache Poisoning–The Next Generation, 2003.
- [119] Mike St.Johns. Automated Updates of DNS Security (DNSSEC) Trust Anchors. RFC 5011, September 2007.
- [120] Anna Szmit, Mariusz Tomaszewski, and Maciej Szmit. Domain Name Servers’s Pseudo-Random Number Generators and DNS Cache Poisoning Attack. *Polish Journal of Environmental Studies*, 15(4c):1–6, 2006.

- [121] Michael Bedford Taylor. Bitcoin and the Age of Bespoke Silicon. In *Proceedings of the 2013 International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, CASES '13, pages 16:1–16:10, Piscataway, NJ, USA, 2013. IEEE Press.
- [122] The European Parliament and the Council of the European Union. Directive No. 96/9/EC of 11 March 1996 on the legal protection of databases. *Official Journal of the European Communities*, (L77):20–28, 1996.
- [123] Roland van Rijswijk-Deij, Anna Sperotto, and Aiko Pras. DNSSEC and its Potential for DDoS Attacks: A Comprehensive Measurement Study. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 449–460. ACM, 2014.
- [124] Matthäus Wander. Measuring Occurrence of DNSSEC Validation. DNS-OARC Workshop, Toronto, Canada, October 2012.
- [125] Matthäus Wander, Christopher Boelmann, Lorenz Schwittmann, and Torben Weis. Measurement of Globally Visible DNS Injection. *Access, IEEE*, 2:526–536, 2014.
- [126] Matthäus Wander and Lorenz Schwittmann. GPU-based NSEC3 Hash Breaking. DNS-OARC Workshop, Dublin, Ireland, May 2013.
- [127] Matthäus Wander, Lorenz Schwittmann, Christopher Boelmann, and Torben Weis. GPU-Based NSEC3 Hash Breaking. In *Network Computing and Applications (NCA), 2014 IEEE 13th International Symposium on*, pages 137–144. IEEE, 2014.
- [128] Matthäus Wander and Torben Weis. Measuring Occurrence of DNSSEC Validation. In *Passive and Active Measurement*, volume 7799 of *Lecture Notes in Computer Science*, pages 125–134. Springer Berlin Heidelberg, 2013.
- [129] Zheng Wang. Analysis of DNS Cache Effects on Query Distribution. *The Scientific World Journal*, 2013.
- [130] S. Weiler and D. Blacka. Clarifications and Implementation Notes for DNS Security (DNSSEC). RFC 6840, February 2013.
- [131] S. Weiler and J. Ihren. Minimally Covering NSEC Records and DNSSEC On-line Signing. RFC 4470, April 2006.
- [132] Joss Wright. Regional Variation in Chinese Internet Filtering. *Information, Communication & Society*, 17(1):121–141, 2014.
- [133] Eric Wustrow, Manish Karir, Michael Bailey, Farnam Jahanian, and Geoff Huston. Internet Background Radiation Revisited. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, IMC '10, pages 62–74, New York, NY, USA, 2010. ACM.
- [134] Yinglian Xie, Fang Yu, Kannan Achan, Eliot Gillum, Moises Goldszmidt, and Ted Wobber. How dynamic are IP addresses? In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 301–312, New York, NY, USA, 2007. ACM.

- [135] Xueyang Xu, Z.Morley Mao, and J.Alex Halderman. Internet Censorship in China: Where Does the Filtering Occur? In Neil Spring and GeorgeF. Riley, editors, *Passive and Active Measurement*, volume 6579 of *Lecture Notes in Computer Science*, pages 133–142. Springer Berlin Heidelberg, 2011.
- [136] Yingdi Yu, Duane Wessels, Matt Larson, and Lixia Zhang. Check-Repeat: A New Method of Measuring DNSSEC Validating Resolvers. In *INFOCOM, 2013 Proceedings IEEE*, pages 3147–3152. IEEE, 2013.
- [137] J. Zittrain and B. Edelman. Internet Filtering in China. *Internet Computing, IEEE*, 7(2):70–77, 2003.