Domain Validation++ For MitM-Resilient PKI

Markus Brandt
Fraunhofer SIT
TU Darmstadt

Tianxiang Dai
Fraunhofer SIT

Amit Klein
Fraunhofer SIT

Haya Shulman
Fraunhofer SIT
TU Darmstadt

Michael Waidner
Fraunhofer SIT
TU Darmstadt

ABSTRACT

The security of Internet-based applications fundamentally relies on the trustworthiness of Certificate Authorities (CAs). We practically demonstrate for the first time that even a weak *off-path* attacker can effectively subvert the trustworthiness of popular commercially used CAs. Our attack targets CAs which use Domain Validation (DV) for authenticating domain ownership; collectively these CAs control 99% of the certificates market. The attack utilises DNS Cache poisoning and tricks the CA into issuing fraudulent certificates for domains the attacker does not legitimately own – namely certificates binding the attacker's public key to a victim domain.

We discuss short and long term defences, but argue that they fall short of securing DV. To mitigate the threats we propose Domain Validation++ (DV++). DV++ replaces the need in cryptography through assumptions in distributed systems. While retaining the benefits of DV (automation, efficiency and low costs) DV++ is secure even against Man-in-the-Middle (MitM) attackers. Deployment of DV++ is simple and does not require changing the existing infrastructure nor systems of the CAs. We demonstrate security of DV++ under realistic assumptions and provide open source access to DV++ implementation.

CCS CONCEPTS

• **Security and privacy → Security protocols**; *Public key encryption*; *Distributed systems security*;

KEYWORDS

PKI security, DNS cache poisoning, Certificates, CA attacks

ACM Reference Format:
Markus Brandt, Tianxiang Dai, Amit Klein, Haya Shulman, and Michael Waidner. 2018. Domain Validation++ For MitM-Resilient PKI. In *2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18), October 15–19, 2018, Toronto, ON, Canada.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3243734.3243790

1 INTRODUCTION

Stability and security of web ecosystem rely on Certificate Authorities (CAs) to ensure that services are trusted and communication to them is secure. CAs vouch for trustworthiness of a service by issuing a digital certificate that binds a domain name to a public key of the service. Upon receiving the request for a certificate for domain, say `vict.im`, a CA validates that the server issuing the request owns and controls the domain for which it requests the certificate. After a successful validation of the ownership of the domain, CA issues the certificate. The certificate contains, among others, the public key of the requesting server and the requested domain. Domain name within the certificate is a key element on which trust can be built. The certificate is signed by the private key of the CA. The server then uses this certificate to prove its identity to clients in the Internet. The clients use the server's public key in the certificate to establish a secure (encrypted and authenticated) connection to the server.

Browsers have hundreds of registered CAs, and a valid certificate for any domain signed by any of these trusted CAs is accepted by the browsers. Hence, correctly verifying ownership of the domain during the certificate issuance is critical to ensure the security of the clients and services. There are a number of approaches that CAs can use to establish ownership of domains: Domain Validation (DV), Organisation Validation (OV) and Extended Validation (EV). DV provides a number of techniques (e.g., using Email or Domain Name System (DNS)) that allow to prove in an automated way that the applicant owns a given domain name. The idea behind DV is that only the owner of the domain can receive the communication sent to the services in that domain and can respond to them. EV and OV are meant to ensure more stringent certification and are carried out with some human interaction.

Although EV and OV provide a higher assurance of ownership, they are cumbersome and inefficient since they require manual processes for establishing the identity of the applicant, e.g., communication with the customer that requests the certificate, phone calls to the company, additional documents, such as personal identification card, or impose checks against official government sources. In addition, manual verification during certificates generation is more lengthy and results in high costs, e.g., certificates' prices can exceed 1000 USD. In contrast, the automation offered by DV enabled to reduce the certificates' prices while improving efficiency of the certificates' issuance process and ultimately increasing the market share of DV supporting CAs to 99%.

In this work we explore the security of the DV procedure used by the CAs to establish ownership of domains. We identify vulnerabilities and show that even a weak off-path attackers can trick the

DV process and issue certificates for domains they do not own. We evaluate the attack against CAs and show that vulnerabilities exist in popular and large CAs. Our attack exploits vulnerabilities in DNS, which allow us to inject incorrect mappings into the caches of DNS resolution platforms of CAs. These mappings map the target domain to attacker's controlled IP addresses. As a result, the CAs perform the DV process against the hosts that are controlled by the attacker and not against the real owner of the domain. We discuss obstacles and challenges and show how to overcome them and launch a successful off-path attack during the certificate issuance. Our results demonstrate that *Public Key Infrastructure (PKI)*[1] *which is meant to provide security against strong Man-in-the-Middle (MitM) attackers, is relying on a weak building block that can be circumvented by an off-path attacker.*

We discuss short term patches but show that they do not mitigate the vulnerability. A cryptographic defence for DNS, DNSSEC [RFC4033-RFC4035], would prevent the attacks, but it would take long until the domains are protected with DNSSEC and DNS resolvers perform validation (we explain this in Section 3.6.2). We build upon the ideas of replacing cryptography through assumptions in distributed systems, [19], and design and implement Domain Validation++ (DV++) - a distributed mechanism for authenticating ownership of Internet domains. We show that DV++ is resilient to MitM attackers, while retaining the benefits of DV.

PKI Security

A large research effort is focused on evaluating and improving security of PKI; see Section A for background on PKI. There are works that evaluate security of keys used to establish a secure communication [29], others showed how to exploit side channels to recover plaintext from encrypted communication [6], or launch downgrade attacks for recovering the encrypted communication [9]. There is also a history of attacks against PKI, typically by compromising a CA (we review these in Related Work, Section 5). However, no attention was given to the authentication of ownership used by the CAs. Correctly authenticating ownership of resources is a key element in certification and essential for building trust in the cryptographic material used for securing the communication.

The vulnerabilities along with the need to secure PKI motivated research on security mechanisms, most of which propose alternative models for PKI. Some proposals attempt to identify compromised certificates by using log servers which monitor CAs behaviour, for instance, Certificate Transparency [42], Sovereign Keys [21], Accountable Key Infrastructure [38], Attack-Resilient Public Key Infrastructure [12].

The proposals provide a good starting point and promising directions for design of future PKI. However, most are not adopted due to their prohibitive complexity and performance as well as the changes that they require to the existing infrastructure and the deployment overhead (e.g., introduction of multiple interacting entities). In this work we propose improvements to the existing PKI without changing the infrastructure or introducing new actors – we design and implement DV++ which replaces DV without requiring further modifications to the PKI. In contrast to DV, which we show is vulnerable even to off-path attacks, DV++ provides resilience against the strong (on-path) Man-in-the-Middle (MitM) attackers. Security against MitM attackers is essential since PKI needs to operate over untrusted networks. Hence it is prudent to assume that the attackers can eavesdrop, modify and inject messages. A critical property of DV++ is that it requires no changes to the existing CAs ecosystem. The interface and communication with the CA and with the verified domain are identical to DV. The difference is in the verification process which DV++ applies - which is transparent to the other actors in the CA ecosystem. We explain this in Section 4.

Attacker Model

For our attacks we use the weakest off-path attacker, which does not have access to the communication of legitimate parties. The attacker can generate packets and spoof source IP addresses. Notice that often off-path attackers can gain MitM capabilities, e.g., by launching BGP prefix hijacking attacks. Indeed, attackers are becoming more sophisticated and increasingly leverage BGP hijacking for DNS cache poisoning [44]. Such attackers obtain MitM capabilities for a short period of time and can efficiently launch the attacks described in this work. We demonstrate however, that even weaker (complete off-path) attackers, can subvert the security of DV validation in PKI.

Disclosure and Ethics

Our research shows that even weak off-path attackers can exploit vulnerabilities to issue fraudulent certificates. This puts at risk not only the vulnerable CAs but the entire PKI ecosystem, with services and clients. Nevertheless, we believe that this research is important: since the vulnerabilities exist, they may be exploited by attackers for malicious purposes without notifying the affected entities. Our goal is to expose and mitigate these issues. We are disclosing the vulnerabilities and are in contact with the affected CAs.

In the attacks that we demonstrate in this work we leverage DNS cache poisoning for injecting spoofed records into DNS caches. Hence to mitigate our attacks the immediate short term countermeasure is to fix the vulnerabilities in DNS that allow injection of records into caches. We notified the affected DNS vendors, DNS operators and service providers of the vulnerabilities.

Our attacks did not target any existing Internet clients and domains. Evaluation of our attacks against CAs' DNS resolvers were carried out using a domain that we own (for simplicity in this work we use domain `vict.im`). This ensured that the CA would not use the spoofed records for any "real" purpose. We setup a set of attacking hosts, which were issuing certificates for resources in domain `vict.im`. Our techniques can be applied to attack other domains (we survey the attack surface of popular domains that can be potential victims) hence adoption of mitigations is critical.

Our attacks were carried out ensuring that the normal CAs functionality is not affected. Part of our study was inferring different cache overwriting vulnerabilities, which we tested against the DV-supporting CAs. The study of caches introduces a larger volume of traffic, than say, merely running an attack with the goal of exploiting a single vulnerability. To avoid any potential load on the CAs infrastructure we distributed the study over a long period of time, with waiting intervals between the requests for issuing certificates.

[1]PKI is a set of roles, policies, procedures and entities for creating and managing certificates and public-key encryption.

Hence, guaranteeing that we do not generate an excessive traffic volume on the CAs.

Contributions

In this work we show that the DV mechanism, applied by the CAs to authenticate ownership of domains, can be circumvented by off-path attackers. In our attacks we leverage DNS cache poisoning for injecting spoofed records into caching DNS resolvers, mapping the resources in the target domain to attacker-controlled hosts. As a result, the DV performed by the CA is run against attacker's hosts (in this work these are our machines), which allows the off-path attacker to successfully pass the validation, and to receive fraudulent certificates (signed by a CA) for domains that the attacker does not own. We demonstrate the attack and evaluate it against DV supporting CAs. We successfully launched the complete attack against 7 of the DV-supporting CAs. Hence at least 7 CAs are vulnerable to our attack, but potentially more DV-supporting CAs are vulnerable; we explain the conditions for successful attack in Section 3. Unfortunately, even a single vulnerable CA is sufficient for subverting the security of PKI. This is due to the fact that the security of PKI is based on the security of the weakest link in the infrastructure - compromising a single CA allows attackers to issue fraudulent certificates for any domain, which would then be accepted by any operating system and any browser which have that vulnerable CA on a list of trusted CAs.

Our work is the first to weaponise off-path DNS cache poisoning for attacking a complex system, such as the certificates generation. Prior to our work, such off-path attacks were considered anecdotal and rather on a theoretical spectrum. DNS cache poisoning attacks are known to be launched in the wild, but these are done with a MitM attacker, which observes the requests and can efficiently craft malicious DNS responses, e.g., DNS cache poisoning for email hijacking [18] or for stealing digital cash [44]. In this work we provide the first demonstration of exploitation of off-path DNS cache poisoning for attacking a critical system - the certificate generation in PKI.

We discuss possible mitigations but argue that they do not solve the problem. Cryptographic protection of DNS would prevent the attacks but it is not clear when DNSSEC is going to be fully deployed. We follow the ideas of [19] for replacing cryptography through assumptions in distributed systems. We propose DV++, a modification to DV, which maintains the benefits of DV (it is efficient, automated, fits within the existing business model) while providing resilience even against MitM attackers. We discuss how DV++ can be useful also in other settings where mechanisms rely on correct and secure DNS functionality. We make the code of DV publicly available.

Organisation

In Section 2 we discuss the CAs ecosystem and explain which CAs we focus on in this work. In Section 3 we present the different modules in our attack and then show how to apply them to trick CAs to issue certificates for domains that the applicant does not own. We also survey the attack surface of domains that are potential victims. We provide recommendations and discuss their impact on clients. In Section 4 we propose DV++, and provide its implementation and experimental evaluations. In Section 5 we review related work. Finally, in Section 6 we conclude this work. Appendix, Section A provides background on DNS and PKI.

2 DV-SUPPORTING CA LIST

2.1 Selecting CAs

Although there are 122 root CAs, the 51 DV supporting CAs control more than 99% of certificates market share[2],[3]. The other CAs are resellers which use root CAs, device based CAs, e.g., for ID card or hardware, or country-specific CAs, which accept only specific country code for issuing certificates.

In our study we focus on CAs supporting DV, with which we could register and issue certificates. We list them in Figure 1. Root certificate programs can be extracted from the browsers and the Operating Systems (OS). We extracted CAs from the following OS: Internet Explorer with Windows, Apple with OS X/iOS, Mozilla with Linux.

2.2 Issuing Certificates

To issue a certificate an applicant should fill out a form called Certificate Signing Request (CSR) on CAs websites. The CSR contains information that is included in the certificate, such as organisation name, domain name, country, public key and more. A CA then uses the submitted CSR to authenticate the domain ownership by the applicant and subsequently to issue the certificate. When submitting the CSR the applicant should also specify which DV method it would like the CA to use for authenticating ownership of the domain. In what follows we describe the DV procedures that are supported by the CAs.

2.3 DV Methods

There are a number of methods for performing DV. We list them in Figure 1. Some CAs support more than one DV method. When a CA supports more than one method, the applicant can specify which method it wishes to use. All the methods rely on DNS, and can be subverted via DNS cache poisoning. We summarise which CA supports what DV methods in Figure 1, and below explain how the validation is performed:

2.3.1 *Email-Based DV.*

Upon filling out a CSR an email is issued to the administrative contact of the domain selected by the applicant out of email addresses registered for that domain in `Whois`. The email typically contains validation code and link, which the recipient has to click and enter the code to prove control over the domain. If the correct code is entered the code proceeds with the certificate issuance.

2.3.2 *WHOIS-Based DV.*

Similar to email-based DV, except that the client cannot select which email (out of those registered as administrative for the domain) will be used in the validation. During the DV procedure the CA selects itself the email address and can use any Admin, Registrant, Tech or Zone contact email address that appears in the domain's WHOIS record.

[2] https://w3techs.com/technologies/overview/ssl_certificate/all
[3] https://www.netcraft.com/internet-data-mining/ssl-survey/

2.3.3 DNS-Based DV. Upon submitting a CSR, a hash value is provided which has to be entered as a DNS CNAME Resource Record (RR) for the domain in the zonefile. For example, assume that applicant's domain is `vict.im` and CAs domain is `ca-domain.com`. The CNAME record would be:

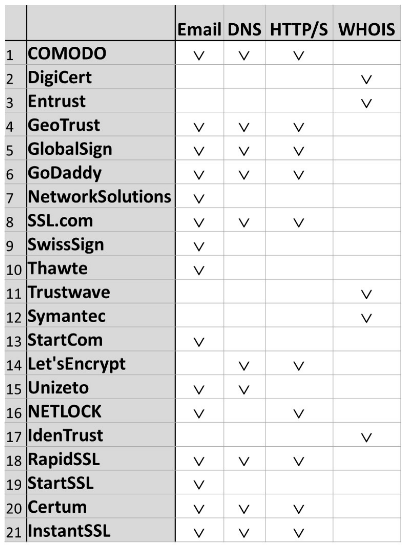
`hash1.www.vict.im.` CNAME `hash2.ca-domain.com.`

The DNS resolver of the CA queries the domain of the applicant and checks the presence of the CNAME record. If the correct record is present, the CA proceeds to issue the requested certificate.

2.3.4 HTTP/S-Based DV. Upon the submission of a CSR, a hash value is returned to the client. A file should be created and placed at the root of the web server with the hash value as its name, as follows: `http://www.vict.im/hash-value1.txt`.

The content of the file should contain `hash-value2` and the domain `ca-domain.com`.

The CA makes an HTTP/HTTPS request to retrieve the file. If correct, the CA proceeds to issue the certificate.



		Email	DNS	HTTP/S	WHOIS
1	COMODO	V	V	V	
2	DigiCert				V
3	Entrust				V
4	GeoTrust	V	V	V	
5	GlobalSign	V	V	V	
6	GoDaddy	V	V	V	
7	NetworkSolutions	V			
8	SSL.com	V	V	V	
9	SwissSign	V			
10	Thawte	V			
11	Trustwave				V
12	Symantec				V
13	StartCom	V			
14	Let'sEncrypt		V	V	
15	Unizeto	V	V		
16	NETLOCK	V		V	
17	IdenTrust				V
18	RapidSSL	V	V	V	
19	StartSSL	V			
20	Certum	V	V	V	
21	InstantSSL	V	V	V	

Figure 1: List of CAs and the supported DV methods.

3 OFF-PATH ATTACKS AGAINST DV

In this section we show that an off-path attacker can impersonate a victim domain to a CA and cause the CA to issue a spoofed (fraudulent) certificate binding the public key of an off-path attacker to a victim domain. The main ingredient in impersonating a victim domain is a DNS cache poisoning attack against a caching resolver of the CA. During the attack we inject a spoofed DNS record mapping the CA to an attacker controlled email and DNS server. As a result the DV process is performed against the attacker-controlled host, to which the attacker can respond, impersonating a victim domain.

The attack consists of a number of components. First we show how to cause the *victim* CA to issue a DNS request to the nameserver controlled by the attacker. Next we show how to match the challenge response authentication parameters in the *spoofed* DNS response, such that the response is accepted by the receiving caching DNS resolver of the CA. Finally, we explain how to construct the records in the spoofed DNS response, so that they are cached by the receiving caching resolver. We then present the attack and provide a measurement study of the CAs and the potential victim domains (from popular Alexa www.alexa.com domains) to estimate the number of clients and servers in the Internet vulnerable to our attack.

3.1 Triggering DNS Request

To initiate our study of cache poisoning vulnerabilities in CAs we need to trigger a DNS request from the *victim* DNS caching resolver to our nameservers. The problem is that the resolvers are configured not to respond to external requests, and since we are not on the network of the CA triggering a DNS request from the outside is a challenge. We trigger a query indirectly. To trigger a DNS request we upload a CSR – this initiates the DV procedure against the target domain (which we provide in the CSR). The "target domain" is the victim domain whose records the attacker wishes to poison in CA's DNS resolver. As a target domain in this work we use `vict.im`.

In the rest of this section, our study and attacks are performed in a black-box manner, using the requests that are sent from the caching DNS resolver of the CA to our nameservers. In response to the DNS request of the caching resolver we generate responses that cause the resolver to follow-up with DNS requests. This is achieved with, e.g., referral type responses, as well with responses with CNAME records. These interactions allow us to characterise the caches, identify vulnerabilities and eventually launch the attacks.

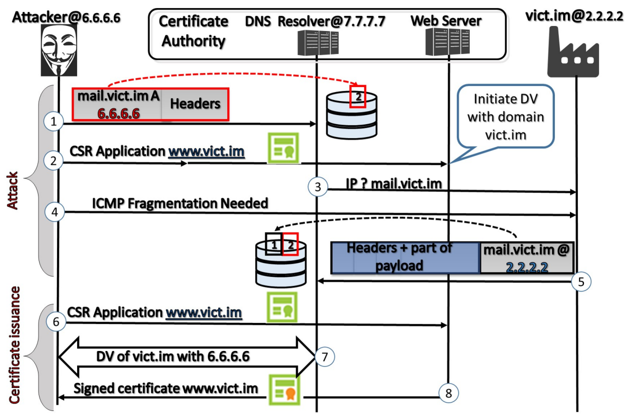


Figure 2: Defragmentation cache poisoning for subverting DV.

3.2 Defragmentation Cache Poisoning

The goal of our attacker is to spoof a DNS response packet from the nameserver to the CA's DNS resolver, which the resolver accepts as valid.

Once a DNS request is issued the time window for DNS cache poisoning attack is initiated. The window ends either when a timeout event happens or when a correct response arrives. To craft a correct response, the attacker has to guess the challenge-response authentication parameters in the DNS request. These are different values, most notably, source port and DNS Transaction Identifier (TXID), which are randomised by the DNS resolver in the request, and are validated in the response [RFC5452]; we provide background on DNS and security against cache poisoning in Appendix, Section A. Figure 3 illustrates a structure of a DNS request from DNS resolver at IP address 7.7.7.7 to a nameserver at IP address 2.2.2.2 sent from UDP source port 12345 and with TXID of 76543.

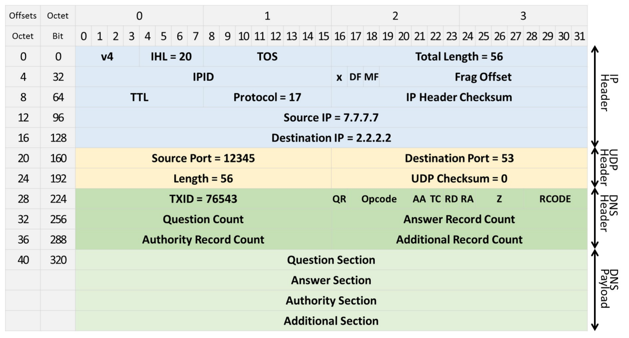

Offsets
Octet
0
1
2
3
Octet
Bit
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
0 0 v4 IHL = 20 TOS Total Length = 56
4 32 IPID x DF MF Frag Offset
8 64 TTL Protocol = 17 IP Header Checksum
12 96 Source IP = 7.7.7.7
16 128 Destination IP = 2.2.2.2
20 160 Source Port = 12345 Destination Port = 53
24 192 Length = 56 UDP Checksum = 0
28 224 TXID = 76543 QR Opcode AA TC RD RA Z RCODE
32 256 Question Count Answer Record Count
36 288 Authority Record Count Additional Record Count
40 320 Question Section
Answer Section
Authority Section
Additional Section
IP Header
UDP Header
DNS Header
DNS Payload

Figure 3: DNS request packet from DNS resolver at 7.7.7.7 to a nameserver at 7.7.7.7

Since source port and TXID are both 16 bits, they generate together a range of 2^{32} possible values. This is a lot of entropy hence it seems to make DNS robust to off-path attacks. Specifically, the attacker must spoof DNS responses with the correct source port and TXID which is about 2^{32} bits of entropy. In this section we show that this is not the case. We show how an off-path attacker can hijack the communication between the resolver and the nameserver. The key ingredient is overlapping IPv4 packet fragments. Our attacker does not attempt to guess the source port and TXID. Instead we use IPv4 packet fragmentation to overwrite the relevant fields in a real DNS response from the nameserver, with malicious values. In what follows, we explain fragmentation and Maximum Transmission Unit (MTU), we discuss how to force a server to send a fragmented response, and then describe how to combine these for attacking a defragmentation cache to inject a spoofed record into the caching DNS resolver of the CA.

3.2.1 Maximum Transmission Unit. The Maximum Transmission Unit (MTU) is the largest number of bytes that can be transmitted over a link in one datagram. The Path MTU between two end points is limited by the lowest MTU on the path [RFC791]. When packets exceed the path MTU, they will not be received at the destination. To cope with this, the Internet Protocol (IP) provides the possibility to fragment packets into smaller fragmented packets. Most networks support currently MTU of 1492 bytes [RFC2516]. According to [RFC791] the minimum MTU in the Internet is 68 bytes. The DNS responses typically do not exceed 1500 bytes, unless they contain a cryptographic material due to the use of DNSSEC [RFC4033-RFC4035]. Hence, most DNS responses would not fragment.

The idea of the attack is that the attacker "convinces" the nameserver to fragment responses to a specific destination. As a result, the nameserver responds with a fragmented packet, such that the second fragment contains either the additional or also the authority

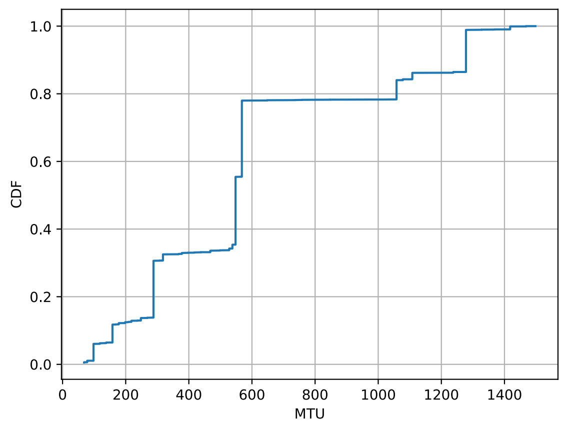


Figure 4: CDF of the packets' sizes in response to an ICMP fragmentation needed error message for servers that reduced their responses sizes.

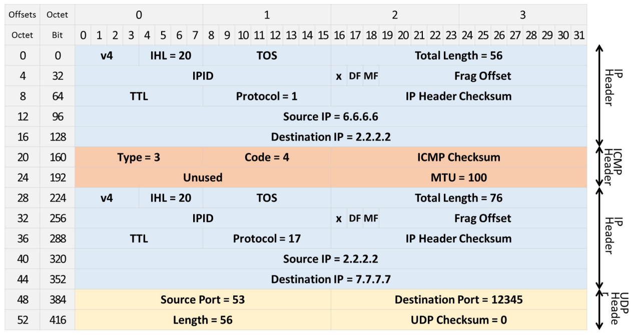
sections of a DNS response; see Appendix A for details on DNS packet structure.

3.2.2 ICMP Fragmentation Needed. To reduce fragmentation load on the routers, [RFC1191] proposes a mechanism to discover the MTU between two hosts. To do so, hosts make use of the Do Not Fragment (DF) bit in the IP header to instruct the routers along the path to not fragment the packet in case the packet exceeds the MTU of the next hop. Instead, intermediate hosts will discard the packet and issue an *ICMP Destination Unreachable* error message (type 3) to the originator with the code *Fragmentation Needed and DF set* (code 4). The information in the ICMP error message is stored by the receiving OS, e.g., 10 minutes by default on Linux 3.13. The ICMP error message can be originated by any Internet node on the path between the sender and the receiver and the receiver of the ICMP error message is not expected to know the IP addresses of the nodes on the path. Hence, we use an off-path attacker to issue an ICMP fragmentation needed packet to the nameserver, indicating that it should reduce the MTU when sending packets to the *victim* resolver.

We measured the fraction of servers among 5K-top Alexa that reduce the MTU size following ICMP fragmentation needed packet: 33, 4% of the servers reduce the packet size up to 296 bytes, and 11% reduce the fragment size to below 296 bytes. Figure 4 shows the CDF of packet sizes that were received in response to an ICMP fragmentation needed packet. While less than 15% of servers reduce their packet size below 296 bytes, almost 80% of servers are willing to reduce the packet size below 600 bytes (the two steps in Figure 4 represent 552 bytes, which is the default minimal value in the Linux kernel and 576, which is the suggested minimum MTU by the RFC).

The spoofed ICMP error message does not need to be sent from a spoofed source IP address - ICMP error message can originate from *any* node on the path. In contrast to ICMP with TCP headers, the OSes typically do not apply any checks on the received ICMP error messages with UDP headers, e.g., Linux 3.13. This is due to the fact that UDP is stateless. Hence, crafting a spoofed ICMP fragmentation needed error message is a simple task.

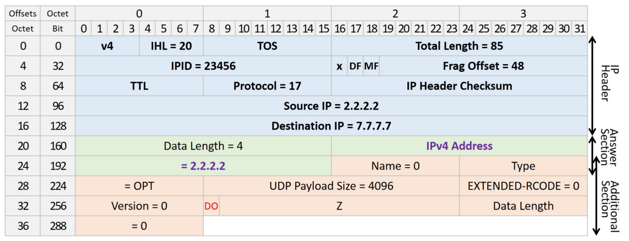
Figure 5 shows an example ICMP fragmentation needed error sent by an attacker at IP 6.6.6.6 to nameserver at IP 2.2.2.2 telling the nameserver to reduce the MTU to 100 bytes for all packets it sends to the DNS resolver at 7.7.7.7. The payload of the ICMP packet is the IP header and the first eight bytes of the original packet (that triggered the ICMP error message). The nameserver stores this information and uses it for limiting the size of IP packets to destination IP (7.7.7.7) and protocol (UDP).



Offsets / Octet	Octet / Bit	0 (bits 0–7)	1 (bits 8–15)	2 (bits 16–23)	3 (bits 24–31)
0	0	v4 IHL = 20	TOS	Total Length = 56	
4	32	IPID		x DF MF Frag Offset	
8	64	TTL	Protocol = 1	IP Header Checksum	
12	96	Source IP = 6.6.6.6			
16	128	Destination IP = 2.2.2.2			
20	160	Type = 3	Code = 4	ICMP Checksum	
24	192	Unused		MTU = 100	
28	224	v4 IHL = 20	TOS	Total Length = 76	
32	256	IPID		x DF MF Frag Offset	
36	288	TTL	Protocol = 17	IP Header Checksum	
40	320	Source IP = 2.2.2.2			
44	352	Destination IP = 7.7.7.7			
48	384	Source Port = 53		Destination Port = 12345	
52	416	Length = 56		UDP Checksum = 0	

Figure 5: ICMP fragmentation needed packet from attacker at 6.6.6.6 to nameserver at 2.2.2.2 indicating an MTU of 100 bytes for resolver at 7.7.7.7.

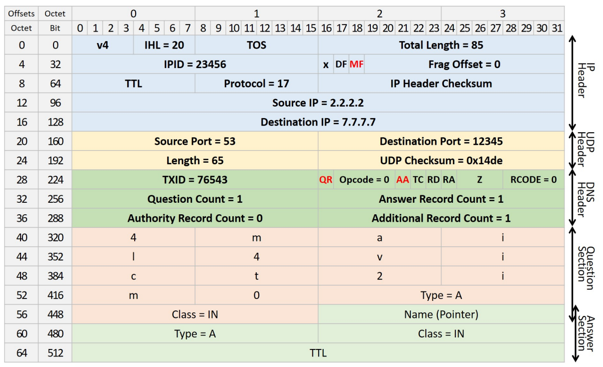
3.2.3 *IPv4 Fragmentation and Reassembly.* Upon arrival to the receiver the fragments of an IP packet are stored in an IP defragmentation cache, by default for 30 seconds. The receiver uses the IP ID value in the fragments to identify all the fragments of the same original IP packet (they all have the same IP ID value). Then uses the offset field to reassemble their payload together. The receiver knows that all the fragments of the original IP packet have arrived by checking that the fragment with the lowest offset has a More Fragments (MF) value of zero.



Offsets / Octet	Octet / Bit	0 (bits 0–7)	1 (bits 8–15)	2 (bits 16–23)	3 (bits 24–31)
0	0	v4 IHL = 20	TOS	Total Length = 85	
4	32	IPID = 23456		x DF MF Frag Offset = 48	
8	64	TTL	Protocol = 17	IP Header Checksum	
12	96	Source IP = 2.2.2.2			
16	128	Destination IP = 7.7.7.7			
20	160	Data Length = 4		IPv4 Address	
24	192	= 2.2.2.2		Name = 0	Type
28	224	= OPT	UDP Payload Size = 4096		EXTENDED-RCODE = 0
32	256	Version = 0	DO Z		Data Length
36	288	= 0			

IP Header
Answer Section
Additional Section

Figure 6: Malicious second fragment sent by attacker at 6.6.6.6 from a spoofed IP address 2.2.2.2 to the DNS resolver at 7.7.7.7, assuming MTU of 68 bytes.



Offsets / Octet	Octet / Bit	0 (bits 0–7)	1 (bits 8–15)	2 (bits 16–23)	3 (bits 24–31)
0	0	v4 IHL = 20	TOS	Total Length = 85	
4	32	IPID = 23456		x DF MF Frag Offset = 0	
8	64	TTL	Protocol = 17	IP Header Checksum	
12	96	Source IP = 2.2.2.2			
16	128	Destination IP = 7.7.7.7			
20	160	Source Port = 53		Destination Port = 12345	
24	192	Length = 65		UDP Checksum = 0x14de	
28	224	TXID = 76543		QR Opcode = 0 AA TC RD RA	Z RCODE = 0
32	256	Question Count = 1		Answer Record Count = 1	
36	288	Authority Record Count = 0		Additional Record Count = 1	
40	320	4	m	a	i
44	352	l	4	v	i
48	384	c	t	2	i
52	416	m	0	Type = A	
56	448	Class = IN		Name (Pointer)	
60	480	Type = A		Class = IN	
64	512	TTL			

Figure 7: First fragment sent by the nameserver at 2.2.2.2 to the DNS resolver at 7.7.7.7, assuming MTU of 68 bytes.

3.2.4 *Exploiting Fragmentation for Defrag. Cache Poisoning.* In this section we show how we exploit fragmentation for injecting a spoofed payload into a DNS response from the real nameserver. For our attack we exploit fragmentation, and trick the receiving resolver into reassembling the first fragment from the real response from the nameserver with the second fragment generated by the attacker. This allows us to bypass the challenge-response authentication fields used by the DNS resolvers, since they are echoed by the nameserver in the *first fragment.* In Section 3.2.2 we showed how we ensure that the response from the target nameserver is fragmented.

Assume that the attacker wishes to impersonate `vict.im`, and get a certificate for `vict.im` with a mapping to an attacker controlled IP address. The attack is initiated with an off-line preprocessing phase, during which the attacker needs to perform a measurement and a calculation, which it will use to set the values in the ICMP fragmentation needed error message in step (4) (Figure 2) and based on which it will construct the spoofed second fragment in step (1) (Figure 2). The attacker measures the responses' sizes from the nameserver of `vict.im` and calculates the offset where the fragmentation should occur. The MTU in the ICMP fragmentation needed error message is set accordingly. The goal is to ensure that the records, which it will replace with spoofed records, are in second fragment.

The attack proceeds as illustrated in Figure 2. In step (1) the attacker sends to the victim DNS resolver a spoofed second fragment (see spoofed second fragment illustrated in Figure 6). The fragment is cached by the receiving resolver in the IP defragmentation cache waiting for the remaining fragments to arrive. In step (2) the attacker uploads a CSR form requesting certification for domain `vict.im`. The attacker selects an email based DV, hence in step (3) the DNS resolver at the CA issues a DNS request to the nameserver of `vict.im` domain asking for an IP address of the email server `mail.vict.im`. The attacker issues an ICMP fragmentation needed packet in step (4) to ensure that the response is fragmented; this ICMP error message can be issued also before step (3). In step (5) the nameserver of `vict.im` sends a fragmented DNS response. The first fragment contains the entropy (resolver's source port and TXID are copied by the nameserver to the response) and some of the DNS payload, such as the question section, answer section, and probably part of the authoritative section (see first fragment in Figure 7). Some of the records from the authoritative section (or the complete authoritative section) as well as the additional section are in the second fragment. The first fragment is then reassembled with the spoofed second fragment that was already in the cache of the DNS resolver; they both leave the cache, and are passed on to the UDP layer. The legitimate second fragment will not have any first fragment to reassemble with, since when it arrives the legitimate first fragment have already been reassembled with the spoofed second fragment and left the buffer. The remaining steps (6)-(8) complete the DV process.

Notice that in step (1) when sending the spoofed fragment the attacker needs to ensure the correctness of UDP checksum and IP ID. In what follows we describe both challenges and explain how we ensure them.

3.2.5 UDP Checksum. The method to calculate UDP checksum is defined in [RFC768]. The UDP checksum is calculated as the 16-bit one's complement of the one's complement sum of the fields in IP header, the UDP header and the UDP payload. The UDP checksum value is stored in the UDP header. Upon receipt of a UDP packet the receiver calculates the UDP checksum (using the same procedure as the sender did) and then compares whether it is the same value as in UDP header. If the value differs - the UDP datagram is discarded.

If the checksum in the second spoofed fragment is not adjusted, When the spoofed second fragment is reassembled with the first real fragment, the overall value of the UDP checksum of both fragments is altered, and will differ from the UDP checksum value in the UDP header. Specifically, the spoofed second fragment contains different records than the real second fragment. Hence, the attacker needs to adjust the UDP checksum of the second fragment such that the computation of the UDP checksum using the payload in both fragments remains similar to the original value of UDP checksum in the UDP header.

The attacker can ensure this as follows: the attacker sends a request to the nameserver and receives a response in two fragments (possibly using the ICMP fragmentation needed packet to enforce fragmentation). The attacker calculates the UDP checksum value of the second fragment, then alters the second fragment and calculates the checksum of the altered second fragment. The difference value in the checksum of both second fragments the attacker needs to add (or remove) by altering two bytes (in even location) in the altered second fragment. This is a simple computation since the attacker knows the original value, and knows what bytes were modified, it can efficiently compute the difference and add (or remove) it to the value of UDP checksum of second fragment.

To adjust the value of two bytes the attacker can either modify two bytes in any record in the second altered fragment, or can add the two bytes at the end, after the EDNS [RFC6891] record (in this case the DNS software will ignore these two bytes but they will be included in the UDP checksum computation).

3.2.6 IP ID. The spoofed second fragment should contain the correct IP ID value – the same value as the original IP packet sent by the nameserver. The attacker must predict this value correctly, otherwise, when the first fragment and the spoofed second fragment have different IP ID values, the receiving OS will not reassemble them together.

Predicting the IP ID value can typically be done efficiently. There are three IP ID assignment algorithms: sequentially incrementing (the OS increments IP ID value by one for every sent packet), per-destination sequentially incrementing (the OS increments IP ID by one for every packet sent to a given destination), and random [24, 41]. We describe the three cases below.

Sequentially Incrementing IP ID. More than 60% of 10K-top Alexa domains use sequentially incrementing IP ID values assignment. Windows operating systems use a sequentially incrementing IP ID.

The attacker samples the IP ID value from the nameserver and samples the IP ID increase rate. Then the attacker calculates the IP ID value that will be assigned at the time the nameserver sends the DNS response to the victim DNS resolver. The attacker uses this value in the second spoofed fragment. The attacker can also send multiple fragments with different IP ID values to increase his chances to hit the correct IP ID value.

Per-Destination IP ID. A bit less than 40% of the nameservers use a per-destination incrementing IP ID. Linux versions are using a per-destination incrementing IP ID assignment. Our attacker uses the techniques presented in [41] for predicting the IP ID values. Since this is not the focus of our work, we do not describe the algorithm here and refer an interested reader to [24, 41] for details.

Random IP ID. Very few servers use random IP ID values, less than 1%. The reason is the overhead that it introduces on the server. Specifically, instead of maintaining a single counter (as in the case of globally incrementing) or one counter per destination (in the case of per destination incrementing IP ID), in this case the server has to maintain multiple counters, and to continually check for collisions, i.e., that it does not select an IP ID that was already allocated.

The success probability of the attack in case of a single fragment is $\frac{1}{2^{16}}$. The attacker can increase its success probability by sending multiple fragments, each with a different IP ID value. Assuming an infinite size defragmentation cache, if the attacker sends 2^{16} fragments, its success probability is 1. The existing OSes however limit the number of fragments that can be sent, e.g., Windows to 100 fragments, recent Linux versions to 64 and older allow several thousand fragments [37]. In Linux, the limit is set via *ip_frag_max_dist* parameter.

3.3 Overwriting Cached Records

Defragmentation cache poisoning allows to bypass the challenge response authentication as the entropy is in the first fragment and the attacker injects payload into the second fragment. A successful defragmentation cache poisoning allows to bypass the validation of the OS and the packet is transferred on to the DNS software. Now, the task is to ensure that the records (sent in the spoofed DNS response) are cached and served in responses to applications and clients. The problem is that often the records will already be present in the cache. For instance, records of domains of interest or popular domains, such as banks and social networks, are typically present in caches. The resolvers will not necessarily overwrite the already cached values, in fact, often they will silently ignore records for which there are cached copies but with other values. This poses a challenge - how to overwrite the already cached records with new values?

The tricky part here is that different DNS software assign different trust ranks to DNS records and apply different logic when overwriting the cached records with new values. The higher the rank the more difficult it is to overwrite the cached record with a new value. The ranking of DNS records is discussed in [RFC2181], and these recommendations are interpreted and implemented in each software differently. Since the records' overwriting behaviour is so different between the different DNS resolver software, it can be used to fingerprint the software of the DNS cache.

To understand how to overwrite cached records with new values we did a characterisation of DNS caches of CAs and modelled under which conditions the caches replace copies of cached records with new values. Our study builds on [40] which evaluated different approaches for overwriting cached records. We next describe the setup, the study and the results.

3.3.1 Setup. Based on a lab model of the caches' overwriting behaviour, we applied our study over the resolvers of CAs. We used the `vict.im` domain and its subdomains for our study. The zone file was configured with 6 nameservers each with an A record (IPv4 address) and AAAA record (IPv6 address), 3 MX records (incoming email servers), and other records, such as DNS specific records (e.g., SOA) and anti-spam records (e.g., SPF). The study proceeded in a black box manner - we cause the CAs to issue DNS requests for records within `vict.im` and monitor requests that arrive to our nameservers. We generate responses dynamically on the nameserver with program we built based on `Stanford::DNSserver` perl library. We then use the monitored queries to characterise the caches and to understand the overwriting behaviour.

In Appendix, in Figure 12 we recap a number of selected payloads (see full list in [40]) of the DNS responses and the cached records that they can overwrite, using records that are returned in answer, authority and/or additional sections. Our measurement of minimum MTU values in popular Alexa servers shows that the responses can be often reduced to even 68 bytes. Furthermore, fragmentation at the boundaries of authority or additional sections is common and can often be enforced.

In Figure 13 we recap the in-lab evaluation of selected payloads (from [40]) against popular DNS resolvers' software, appliances and public services. The indexes in the leftmost column correspond to indexes of the payloads in Figure 12. The values in cells indicate whether the cache in column j is vulnerable to overwriting by payload in row k: 0 means not vulnerable and 1 means vulnerable. The evaluation indicates that all the resolvers that were tested, except Unbound in hardened mode (which is resilient to all the payloads), are vulnerable to at least one type of cache overwriting payloads.

Caches' Overwriting Study. For each CA, our study proceeds for each payload p_i in Figure 12 in three phases: (a) seed the honey record, (b) overwrite the value of the honey record with a new value, (c) probe the value of the honey record.

We first plant the honey record in the cache of the DNS resolver of the CA. The honey record simulates a real value of our test domain `vict.im`. During phase (b) we attempt to overwrite the honey record with a new value. During phase (c) we send a DNS request for the honey record and check its value. If the value was modified following step (b), we mark the cache as vulnerable to payload p_i. Next iteration, increment i and evaluate with new payload.

Overwriting Vulnerabilities in CAs. We next list the CAs that were found vulnerable to overwriting attacks. Within our study we also identified CAs that share the same infrastructure (Email server and caching DNS resolver) - we grouped those CAs below.

- COMODO, InstantSSL, NetworkSolutions, SSL.com: these CAs use the same `MX` email server `mcmail1.mcr.colo.comodo.net` which uses the same caching DNS resolver. The results from our cache overwriting methods indicates that the DNS resolver software is `New BIND 9.x with DNSSEC-validation`.
- Thawte, GeoTrust, RapidSSL: use the same MX server and resolution platform.
- StartCom[4], StartSSL: both use the same email server and the same DNS resolver.
- SwissSign: uses `New BIND 9.x`.

Caches Overwriting Attacker. The study of caches overwriting is performed with a man-in-the-middle (MitM) type attacker – in this step we do not attempt to guess ports or TXID, just to characterise the caches of the CAs. In particular, we observe the requests from the caching DNS resolvers of the CAs and generate the responses dynamically "on the fly" as a function of the requests and the payloads in Figure 12. After characterising the caches of the CAs in this model, we run the complete attack in an off-path attacker model in Section 3.4.

3.4 Evaluation of Attack Against CAs

3.4.1 Setup. We evaluated the attack (with the components in Sections 3.1, 3.2 and 3.3) against the DNS resolvers of CAs (in Figure 1) using our own test domain `vict.im`. The nameservers hosting the test domain were located on one network (belonging to Internet Service Provider (ISP) A), while the attacking hosts were located on a different network (belonging to ISP B). Our nameservers running `vict.im` as well as its subdomains were set up with a globally sequential IP ID allocation. The attacking hosts are configured with a mapping of cache overwrite vulnerabilities per each CA on the list. Namely, when running an attack against CA ω, the attacker selects one of the payloads from the list 12 to which the caching DNS resolver of ω is vulnerable, and uses it to generate the payload of the spoofed second fragment. The attacking host queries the nameserver for IP ID value *ipid* and uses it to calculate the IP ID value that will be assigned to the response that we will be attacking with a spoofed second fragment.

3.4.2 Issuing Spoofed Certificate. The attacking host initiates the CSR process for victim domain `vict.im` (and its subdomains). After submitting the CSR the attacker selects the DV method that it wishes to pass (CAs typically support a number of DV methods, see Figure 1). Since the attacker cannot control the timing when the request is sent, it must periodically transmit spoofed second fragment, until the validation is initiated. Finally, if the attacker receives a signed certificate for the resource that it requested (i.e., a domain `vict.im` or its subdomains) we mark the CA as vulnerable.

[4]StartCom stopped issuing new certificates in January 2018.

3.4.3 Which Record To Attack? All the different DV methods (in Section 2) generate NS (nameserver) and A (IP address) type requests. In addition, DV with Email (or with WHOIS) also generate MX (email server exchanger) type requests. Responses to these requests are suitable attack targets. Specifically, the attacker hijacks the entire `vict.im` domain by injecting into a response packet a spoofed NS or A records or hijacks an email server by injecting an MX record (with the corresponding A record). Once a CA caches a record mapping the nameserver of `vict.im` to attacker's host, all the subsequent requests (e.g., for email server or webserver) will go to the attacker. The attacker passes any DV verification option listed in Section 2.

3.4.4 Constructing the Second Fragment. Essentially for many domains the attacker can control even the answer section of a DNS response, if fragmentation is at around 200 bytes. For instance, consider fragmentation at 68 bytes. Given a nameserver whose responses fragment so that the first fragment is 68 bytes long, the attacker can control the whole answer section. The headers take 40 bytes: IP header is 20 bytes, UDP header is 8 bytes, DNS header is 12 bytes. The query part (which is the first part in the response) copies the QNAME ($n + 1$ bytes, for QNAME of length n) together with the CLASS (2 bytes) and type (2 bytes). So a QNAME of length 23 or more bytes results in an Answer section getting fragmented before the answer part. Even with first fragment of size 200 bytes, the attacker controls the answer part using QNAME of 160 or more bytes (DNS names can be up to 253 characters). In other cases, where fragmentation occurs significantly above 200 bytes, the attacker controls the authority and additional sections. Depending on the DNS resolver software and the fragmentation restriction the attacker selects the suitable payload for cache poisoning attacks (selected payloads are listed in Figure 12).

3.4.5 Measuring the CAs (Off-Path) Attack Surface. All the CAs we measured use unpredictable source ports and TXIDs. Hence, we use the defragmentation cache poisoning to bypass the entropy in first fragment. As it turns out, many CAs block fragmented IP packets. We evaluated the complete attack against CAs that allow fragmented responses. These include COMODO, InstantSSL, NetworkSolutions, Let's Encrypt, SSL.com, CERTUM, NETLOCK.

3.5 Challenges and Conditions for Success

Our attack does not apply against all the DV-supporting CAs. In this section we explain why, and list the conditions that are required for our attack to succeed. We also discuss the challenges and hurdles in such off-path attack and how we overcame them.

3.5.1 Conditions for Success. Our attack succeeds against a DNS resolver of a given CA, for a specific target (victim) domain `vict.im` if the following conditions hold:

• The authentication of the client requesting a certificate is done via DV. The list of DV-supporting CAs that we could test is in Figure 1.

• The DNS resolver and the network of the CA allow fragmented responses. Some CAs block fragmented traffic (see list in Section 3.4.5).

• The domain `vict.im` does not filter ICMP fragmentation needed error messages and reduces the MTU based on the MTU indicated in ICMP fragmentation needed packet. The fraction of domains that do not filter ICMP fragmentation needed messages were measured by [27], see Figure 4.

• The caching DNS resolver is vulnerable to at least one of the overwriting attacks. As we show in Section 3.3 all tested resolvers except one (Unbound in hardened mode) are vulnerable to at least one overwriting payload.

3.5.2 Challenges and Hurdles. The attacks we launched are hard due to the following factors:

Putting it All Together. Putting all the modules into a working attack is a challenge. First the attacker has to synchronise all the modules: triggering the query, measuring the IP ID, crafting the spoofed second fragment with the correct payload (that will overwrite the value of a cached record with a new value), injecting the spoofed second fragment into the defragmentation cache. Carrying these steps out correctly requires off-line preprocessing prior to the attack.

Second, some servers fragment not exactly at the location specified in the MTU in ICMP fragmentation needed error message. For instance, when signalling MTU of 512 bytes, some servers may fragment at 500 bytes. To cope with this, depending on the situation, the attacker may need to add padding, or use records' names that fit within the fragment. The attacker can also play with compression in DNS (defined in [RFC1035]) and increase or decrease the size of the injected record. The compression uses pointers to locations where a given string already appeared, hence by using combinations of names that are already in the first fragment the attacker can reduce the size, or with strings that did not appear before, increase the size.

Off-path Attacker. Off-path attackers cannot see the communication between the DNS resolver and the nameservers. This introduces challenges with timing and synchronisation. To overcome this, we did a preprocessing phase, during which we performed a careful study of the certificate issuance procedure for a given CA: the time at which DNS queries are triggered, when validation is performed. The attack is then tailored for each CA based on the observed behaviour of the CA during the preprocessing phase.

Indirect Study. A key ingredient in our attacks is understanding the caching behaviour in the DNS resolver of the CA to identify and evaluate cache overwriting vulnerabilities. The challenge is that we do not have a direct communication to the resolver, and cannot trigger the queries ourselves. We trigger the queries indirectly, using the CSR submission form. This indirect communication introduces noise and prevents us from knowing when the queries will be sent.

Victim Domain. In this work we attacked our own domain. This allowed us to anticipate the traffic volume to the domain and the IP ID assignment method. In a real life scenario the attacker does not have visibility into the nameservers of some other domain, and hence has to learn this information using side channels. To identify the IP ID assignment the attacker sending queries to the nameserver from different hosts and checks the IP ID values in responses. Similarly, the attacker has to measure the IP ID increments by probing the value in intervals. If a random IP ID assignment is used, the attack becomes much more difficult since it needs to be repeated until the correct IP ID value is hit. Fortunately for the attackers (and unfortunately for security), random IP ID assignment is rare.

3.6 Mitigations and Recommendations

Our evaluations indicate that off-path attacks against CAs are feasible using DNS cache poisoning. We found a few vulnerable CAs, but even a single trusted CA that is attacked exposes to devastating consequences. We next discuss a few possible short and long term mitigations.

3.6.1 Blocking Fragmentation. Perhaps the simplest way to protect against defragmentation cache poisoning (and hence against issuance of spoofed certificates) is by blocking fragmented traffic; which many CAs indeed do. We considered this as a possible simple fix to the problem. However, as we show, MTU reductions are still common in the Internet, e.g., because of PPPoE on DSL lines. How common MTU values less than 1500 bytes in the Internet?

Is Fragmentation Common? We answer this question empirically, using the data captures that Center for Applied Internet Data Analysis (CAIDA) [14] offers. CAIDA operates two passive Internet monitors in the Equinix datacenters in Chicago and San Jose. The monitors capture packets, anonymise them, and provide them for download as *gzipped pcap* files, with the content limited to network and transport-layer protocol headers. Each dataset corresponds to one hour (between 12:59 and 14:01 UTC), split into chunks of one minute (between approx. 150 MB - 1.5 GB). In total, we analysed 14484 traces in 121 datasets, representing data from 9 years (2008 - 2016).

We observe 2.9 million ICMP fragmentation needed packets in total which is approximately one in 135k packets. On average, there are 24k ICMP fragmentation needed packets per dataset. Our measurement study indicates that blocking fragmented traffic would block access to many Internet clients.

The Issue Is More Significant. Nevertheless, even blocking fragments would not block the vulnerability. Off-path attackers may come up with other approaches to bypass the randomisation. Furthermore, there is an increasing tendency to launch short-lived Border Gateway Protocol (BGP) prefix hijacks, for redirecting DNS requests via attacker's network for launching DNS cache poisoning attacks. This allows the attacker to become a MitM (on-path) attacker for a short period of time, sufficient to inspect the DNS request, and to craft a correct DNS response (with valid challenge-response parameters), e.g., see [44]. In this recent attack, the attackers leveraged BGP prefix hijacking for launching DNS cache poisoning to redirect `myetherwallet` clients to attacker controlled servers.

Hence, the solution should be robust even against stronger MitM attackers, and should not rely on patches against off-path attackers.

3.6.2 DNSSEC. To mitigate DNS cache poisoning attacks even by MitM attackers, the IETF designed and standardised Domain Name System Security Extensions (DNSSEC) [RFC4033-RFC4035]. With DNSSEC, nameservers respond with signed DNS records and keys and the DNS resolvers validate the records prior to caching them. Fully deployed DNSSEC would prevent DNS cache poisoning attacks.

Nevertheless, although proposed and standardised more than two decades ago, DNSSEC is still not widely deployed. Measurements show that currently about 25% of the DNS resolvers validate DNSSEC (e.g., see `stats.labs.apnic.net/dnssec`) and only a bit more than 1% of domains are signed with DNSSEC, [51]. This means that DNSSEC validating DNS resolver gets no security benefit since most domains are not signed. More significantly, recent works found problems with DNSSEC keys generation and management procedures exposing a large fraction of signed domains (more than 35%) to attacks, [16, 47]. It is not clear when the problems are expected to be resolved since they involve practices used by many large registrars and DNS hosting providers.

Therefore, it is important to improve security of PKI independently of other defences. In the next section we introduce our proposal DV++ for a MitM-resilient PKI.

4 DOMAIN VALIDATION++

Our goal is to design a defence that preserves the benefits of DV while providing resilience against MitM attackers. We aim to ensure that the integration of the new mechanism would not require any changes to the CA infrastructure and functionality and that it should be easy to deploy. These properties ensure that the mechanism will have better changes to be used by the CAs.

In this section, we present the design, implementation, and simulations of our proposal.

4.1 Setup and Attacker Model

The main aspect of our proposal is to utilise distributed nodes which perform DV from multiple vantage points. The security against MitM attackers is achieved by placing the nodes in different networks, which do not traverse overlapping paths. In Figure 8 we provide an illustration of the attacker model, the CAs, and the DV++ agents.

In contrast to a cryptographic eavesdropping attacker, which is a global eavesdropper, a realistic MitM attacker can be present only on some networks but does not control the entire Internet. This serves as a premise for our design of DV++. The attacker is a malicious ISP, that passively collects the traffic that traverses its networks. The attacker can also actively try to attract traffic from other networks by performing BGP prefix hijacking.

4.2 Design

DV++ is a decentralised mechanism, that utilises the existing Internet infrastructure to validate claims of domain ownership. In contrast to the centralised validation performed by the CAs with DV, DV++ is based on a flat hierarchy with several equally trusted certification agents. To pass a DV++ validation domain owners must prove their ownership to a majority of the agents in a fully automated manner by responding to queries sent by the agents for the resource records in the domain. The agents are synchronised with an orchestrator module. The orchestrator is located on the CA network. The components of DV++ with the messages exchange are illustrated in Figure 9.

The orchestrator and the agents use HTTPS for their communication. During the domain validation process, all the agents receive from the orchestrator the domain and the record that need to be

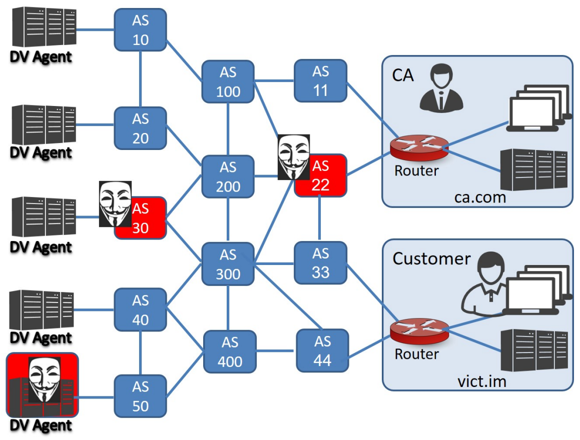


Figure 8: DV++ setup and attacker model.

queried. The agents send DNS requests to the nameservers in the domain requesting the record. As soon as the response arrives, the agent sends the response to the orchestrator. When more than 50% of the responses arrive, and they match, the orchestrator returns *success* otherwise *failure*. The number of the correct responses from the agents is a parameter that the CA can configure.

When sending a DNS request, each agent selects independently a source port at random as well as a random TXID. To launch a successful attack the attacker has to spoof correct responses to more than 50% of the agents. This is an impossible task even for strong, nation state attackers.

The agents are configured on different physical availability networks of AWS cloud. The selection of the cloud networks is done so that the agents are located in different networks, whose routes do not intersect. For selecting the networks to place the agents we use the empirically derived CAIDA AS-level graph [4] from 2016.

Similarly to DV, DV++ authentication is initiated by the CA following a submission of a CSR by the applicant. During this process, queries are sent to the agents, that perform look-ups in the target domain. Once majority of the responses are received by the orchestrator, they are processed.

The core idea of DV+ is that, even if the attacker can corrupt some agents, controlls some networks or path, it cannot control all the paths and all the networks in the Internet and cannot corrupt all the agents. For instance, even strong nation state attackers, such as security agencies, do not control all Internet networks and paths.

4.3 Implementation

The orchestrator and the agents are written in Go. This ensures good performance, easy configuration and cross-compilation of static executables. Static executables allow easy deployment without the need to install any runtimes or libraries. Integrating DV++ in CAs current infrastructure does not require replacing software or hardware nor any other modifications: DV++ uses the same interface and interaction as DV. To use DV++ a CA should merely configure the system to make a call to DV++ library instead of DV.

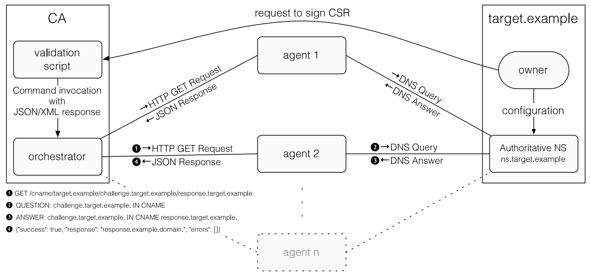

CA
validation script
Command invocation with JSON/XML response
orchestrator
request to sign CSR
agent 1
→HTTP GET Request
←JSON Response
→DNS Query
←DNS Answer
❶→HTTP GET Request
❹←JSON Response
agent 2
❷→DNS Query
❸←DNS Answer
agent n
target.example
owner
configuration
Authoritative NS
ns.target.example
❶ GET /cname/target.example/challenge.target.example/response.target.example
❷ QUESTION: challenge.target.example. IN CNAME
❸ ANSWER: challenge.target.example. IN CNAME response.target.example.
❹ {"success": true, "response": "response.example.domain.", "errors": []}

Figure 9: DV++ components and messages exchange.

The code, as well as configuration and execution instructions, are provided on GitHub https://github.com/dvpp/dvpp.

Using Our Setup. We have set up a pilot installation for evaluating our DV++ implementation. This can be used by the CAs, systems or users in the Internet, for verifying DNS responses and authenticating domain ownerships. We prepared a zip file containing everything needed to perform a DNS-based domain validation with a CNAME record. The zip file is available at http://45.76.90.74/orchestrator.zip, it contains a configuration file to use the local orchestrator with our agents. To get started, first extract the zip file and enter the extracted orchestrator directory with your terminal. Three parameters should be provided: (1) the domain requesting the certificate, (2) the CNAME to look up and (3) the expected response. For instance, to request a certificate for domain `example.com` with verification of `abcdef.example.com` that resolves to `ghijkl.example.com` the user should invoke (in one line without a linebreak):

```
  ./orchestrator_<platform>_<architecture> cname
example.com. abcdef.example.com. ghijkl.example.com
```

where platform is either Windows, Linux, or Darwin and architecture is either 386 or amd64. The command line tool will then return the status of the verification as a JSON object. If XML output is preferred a -x can be written in front of the CNAME.

4.4 Security Evaluations

We consider an attacker that tries to pass the authentication of DV++ to issue a fraudulent certificate. To succeed the attacker must provide correct spoofed responses for the majority of the DNS requests issued by the DV++ agents.

If the attacker is located on a network of the victim nameserver, it can hijack requests on the network of the nameserver and send spoofed responses to the DV++ agents. Luckily domains have more than one nameserver and the nameservers are placed in different networks; this is following best practices to avoid a single point of failure for domains. Our measurements of 1M-top Alexa domains show that an average number of nameservers per domain is 5, and minimal is 2. Furthermore, the nameservers belonging to the same domain are hosted in different networks. This ensures that the attacker cannot hijack and replace responses from all the nameservers.

We next evaluate attacks by passive MitM attacker, that controls a large ISP, and by an active attacker, that also additionally attempts to attract traffic from other networks. We run simulations using a different number of agents, and demonstrate that even 3 agents in networks of top-tier ISPs suffice to provide strong security guarantees against MitM attackers.

4.4.1 MitM Attacker. We quantify the ability of an on-path attacker to intercept majority of the DNS requests sent from the agents to the victim domain. We run simulations using Alexa nameservers in 1000-top domains and quantify over different attacker-agents pairs and the fraction of nameservers whose BGP routes to the victim traverse the attacker. We run simulations with the BGP route computations presented in [25, 26] to the empirically derived CAIDA AS-level graph [4] from 2016. The graph is annotated with bilateral business relationships. We average our measurements over 10^6 combinations of attacker-victim AS pairs, selecting them at random. To identify ASes that host the nameservers we mapped the IP addresses of the nameservers to AS numbers using RIPE `https://stat.ripe.net`.

The simulations evaluate all the possibilities for an on-path attacker to cover almost all the routes between the victim domain and the DV++ agents. We used the dataset from January 2018, which contained 60006 ASes. We categorise ASes into four classes: large ASes with more than 250 customers, medium ASes that have between 25 and 250 customers, small ASes with 1 to 25 customers and stub ASes that have no customers. AS graph has 56 large ASes, 615 medium ASes, 8329 small ASes and 50995 stub ASes. Our graph had in total 60006 nodes and 261340 edges.

In our simulation for each combination of attacker-victim ASes pairs, we measure the fraction of attacker-victim pairs in which the attacker can capture traffic from more than 50% of the DV++ agents. This would be a very strong attacker, nevertheless, we show that DV++ when using sufficient amount of agents ensures security. In practice, the attackers are of course much weaker. The simulations show that only 0.1% of the attackers can be on the path between the victim nameservers and 50% or more of the agents, and hence capture the traffic between the victim and the agents. No attacker can capture 50% of the traffic to the agents when one or more agents are in a large AS, and attackers that are small ISPs or stubs cannot launch successful attacks. We summarise the result of our simulation in Table 1, and plot the results in Figure 10. Figure 10 shows that even with agents in 3 vantage points located in either of top 10 ISPs, the probability of the attacker to launch a successful hijack of more than 50% of the agents drops to almost 0.

victim/attacker	Large	Medium	Small	Stub
Large	0.00	0.00	0.00	0.00
Medium	0.21	0.02	0.00	0.00
Small	0.34	0.12	0.00	0.00
Stub	0.52	0.07	0.00	0.00

Table 1: Evaluation of on-path attacker. % of attackers capturing traffic from $\geq$ 50% agents

4.4.2 Hijacking Attacker. The Border Gateway Protocol (BGP) is vulnerable to prefix hijack attacks [1–3, 10]. In prefix hijacks, the attacker hijacks all the traffic of a victim network. We evaluate attacker ability to exploit the insecurity in inter-domain routing protocol (BGP) to hijack traffic between the victim nameservers and the DV++ agents. In this case, both the victim nameservers and the attacker announce victims' BGP prefixes. We evaluate the fraction of agents that the attacker can attract. The simulation result in Figure 11 shows that probability that the attacker attracts more than 50% of the agents is 2%. The simulation also shows that the attackers that can hijack traffic from 50% of the agents, would disconnect the victim from the rest of the Internet. Only 0.20% of the attackers can successfully launch the attack while maintaining their route to the victim in order to relay packets between the victim nameservers and the rest of the Internet while avoiding detection. This is due to the fact that the fraction of agent nodes that the attacker hijacks is close to the fraction of the ASes in the Internet that the attacker attracts when announcing the victim's prefix.

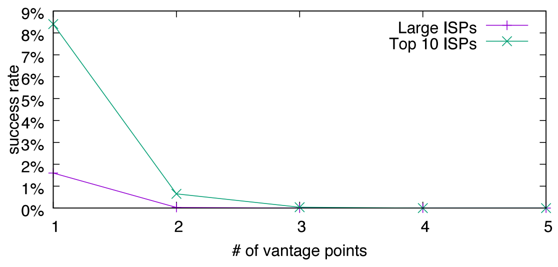


Figure 10: Passive Attacker. Large ISP with > 250 customers, and top ISP is tier 1 ISP.

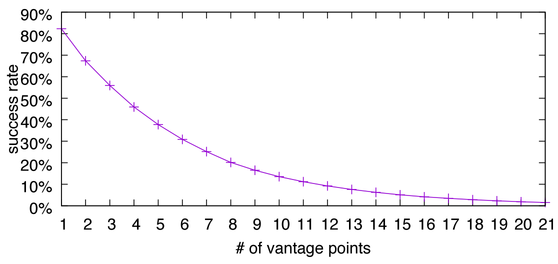


Figure 11: Simulations with active BGP prefix hijacking attacker.

4.5 Other Applications of DV++

DV++ can be used to bootstrap other mechanisms with security. For instance, our attacks apply against password recovery procedures in popular web services. In password recovery procedure the password or a link to reset the password is sent back to the email that initiated the password recovery. If the DNS resolver on the network of the service is attacked, and caches an incorrect record (mapping the email of the victim to attacker's controlled IP address) the password will be sent to the attacker.

DV++ can be used by the web services, to validate the DNS record of the email requesting the password recovery, hence blocking malicious requests that do not pass the verification of DV++.

5 RELATED WORK

5.1 DNS Security

Although researchers warned already in late 90s against the potential vulnerabilities to cache poisoning, DNS cache poisoning was first demonstrated in 2008 [36] against DNS resolvers that used fixed or incrementing source ports. Subsequently, [RFC5452] provided recommendations for patches, which included randomising the source ports in DNS requests, selecting the nameservers to which the requests are sent at random, and other patches. Nevertheless, shortly after attacks were found allowing to circumvent the [RFC5452] patches. Initially, [30, 32] showed that despite randomisation the source ports can be often predicted using side channels, such as timing. Followup works applied fragmentation for bypassing the recommendations in [RFC5452], [31, 33, 45].

However, predicting the source ports and other defences in [RFC5452] does not suffice for guaranteeing successful cache poisoning attacks. In particular, when the records are already in the cache, the DNS resolvers would often ignore and discard the DNS records which contain new values. To gain insights into caches behaviour [40, 48] performed a study of Which resolvers overwrite cached records with new values and under what conditions. Often, DNS resolution platforms consist of multiple actors and caches, these were studied in [39, 43] and multiple forwarders in DNS resolution chains which also expose to cache poisoning attacks [46].

DNSSEC [RFC4033-RFC4035] would prevent the cache poisoning attacks, however, recently vulnerabilities and misconfigurations were found in DNSSEC keys generation and management procedures exposing a large fraction of signed domains (more than 35%) to attacks, [16, 17, 47]. Furthermore, as [28] showed DNSSEC does not prevent attacks which replay DNSSEC-signed DNS response to redirect clients to incorrect servers in Content Distribution Networks (CDNs).

5.2 CA Compromises

A valid certificate signed by any of the trusted CAs in the browser, is accepted by the browsers. This property has spurred a proliferating market for certificates, it also has devastating implications for security: *any vulnerable or compromised CA can subvert the security of any domain.*

Indeed, there is a long history of attacks against the CAs. The compromise of small Dutch CA DigiNotar CA in 2011 was a critical point in history of PKI. For the first time a CA was removed from browser root stores because of poor infrastructure control and issuance of spoofed certificates. Subsequent works documented loopholes in PKI ecosystem, [7, 20, 34]. Other large CAs such as Comodo and Verisign have experienced breaches as well. In 2016 Chinese CA Wosign 'mistakenly' gave out certificates for GitHub domains due to a loophole in domain validation process. Specifically, if a user could prove ownership of a subdomain the CA would also issue certificate for the parent domain without applying DV to prove ownership of the base domain. In 2015 Comodo issued a certificate to an unauthorised party for live.fi – domain used by Microsoft to provide free email. This would have caused users to leak their credentials. Recently a Egyptian ISP (MCS holdings) obtained a certificate that was signed by CNNIC, that was included in root stores. Namely any certificate issued by MCS would be accepted as valid (even for domains MCS did not own). In 2008, a bug in Debian OpenSSL caused thousands of certificates to be issued for keys with only 15-17 bits of entropy [22]. In addition, many browsers accept (non-root) certificates for 1024-bit RSA keys, which are believed to be broken by nation state attackers [11].

The research community also pointed out that Border Gateway Protocol (BGP) prefix hijacking can be leveraged to bypass domain validation (DV), see poster [13].

5.3 PKI Defences and Alternative Proposals

Following CA compromises and the risk of MitM attacks, new security mechanisms have been added to SSL/TLS, and the PKI. These include certificate transparency (CT), HSTS and HPKP headers, SCSV for protection against downgrade attacks. A taxonomy in [12] lists the proposals for improving the security of PKI infrastructure. We briefly recap here for completeness.

There are attempts to create a new alternative PKI and proposals to use additional entities for storing and checking certificates. Some proposals are aimed at making compromises visible using log servers to monitor CAs behaviour, for instance, Certificate Transparency (CT) [42], Sovereign Keys [21], Accountable Key Infrastructure [38], Attack-Resilient Public Key Infrastructure (ARPKI) [12], DTKI [5, 15, 49, 50]. DNS-Based Authentication of Named Entities (DANE) [RFC7671] uses DNSSEC [RFC4033-RFC4035] to list trusted CAs for issuing domains certificates. Similarly DNS Certificate Authority Authentication (CAA) Resource Record [RFC6844] uses DNS to list acceptable CAs. The goal of these proposals is to replace or complement the existing certificates issued by CAs trusted by the browsers. The proposals provide a good starting point and promising options for design of future PKI. However, due to their complexity and efficiency overhead as well as the changes that they require to the existing infrastructure and the deployment overhead (e.g., introduce multiple interacting entities), most are not adopted. Since March 2016, the Chrome Browser uses CT and ceased displaying the green bar for EV certificates that are not registered in a log server.

After making our DV++ available in March 2017, a parallel similar direction was proposed by LetsEncrypt, called `multi-VA`. The difference is that in contrast to DV++, `multi-VA` uses fixed nodes (currently three). Which it uses to perform the validation. By corrupting the nodes, the attacker can subvert the security of `multi-VA` mechanism. DV++ selects the nodes at random from a large set. Furthermore, we ensure that the nodes' placement guarantees that the nodes are not all located in the same autonomous system (AS) and the paths between the nodes and that the validated domain servers do not overlap.

6 CONCLUSIONS

Automated, efficient and easy-to-use procedures pave success for many mechanisms and DV is no exception to it. Unfortunately, the benefits also come with reduced security due to reliance on an insecure DNS. This makes DV vulnerable to off-path attacks, allowing even very weak attackers to issue fraudulent certificates for domains they do not own. We demonstrate such attacks.

Since cryptography for DNS is far from being sufficiently deployed, we need another way to bootstrap security. The only alternative is utilising the distributed nature of the Internet and making structural assumptions about the attacker's capabilities. Based on this observation we design and implement DV++: an innovative distributed mechanism which comes with the same benefits as DV while providing resilience even against strong MitM attackers.

7 ACKNOWLEDGEMENTS

The research reported in this paper was supported in part by the German Federal Ministry of Education and Research (BMBF), by the Hessian Ministry of Science and the Arts within CRISP (www.crisp-da.de/) and co-funded by the DFG as part of project S3 within the CRC 1119 CROSSING.

REFERENCES

[1] [n. d.]. Hijack Event Today by Indosat. http://www.bgpmon.net/hijack-event-today-by-indosat. ([n. d.]).

[2] [n. d.]. New Threat: Targeted Internet Traffic Misdirection. http://www.renesys.com/2013/11/mitm-internet-hijacking. ([n. d.]).

[3] 2008. Renesys Blog - Pakistan Hijacks YouTube. http://www.renesys.com/blog/2008/02/pakistan_hijacks_youtube_1.shtml. (Feb. 2008).

[4] 2011. The CAIDA AS Relationships Dataset, 2011. http://www.caida.org/data/active/as-relationships/. (2011).

[5] Martín Abadi, Andrew Birrell, Ilya Mironov, Ted Wobber, and Yinglian Xie. 2013. Global Authentication in an Untrustworthy World.. In *HotOS*.

[6] Nadhem J Al Fardan and Kenneth G Paterson. 2013. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 526–540.

[7] Bernhard Amann, Matthias Vallentin, Seth Hall, and Robin Sommer. 2012. Extracting certificates from live traffic: A near real-time SSL notary service. *Technical Report TR-12-014* (2012).

[8] Daniel Anderson. 2012. Splinternet Behind the Great Firewall of China. *Queue* 10, 11 (2012), 40.

[9] Nimrod Aviram, Sebastian Schinzel, Juraj Somorovsky, Nadia Heninger, Maik Dankel, Jens Steube, Luke Valenta, David Adrian, J Alex Halderman, Viktor Dukhovni, et al. 2016. DROWN: Breaking TLS Using SSLv2.. In *USENIX Security Symposium*. 689–706.

[10] Hitesh Ballani, Paul Francis, and Xinyang Zhang. 2007. A Study of Prefix Hijacking and Interception in the Internet. (2007), 265–276 pages. https://doi.org/10.1145/1282380.1282411

[11] E. Barker and A. Roginsky. 2011. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. NIST Special Publication. (2011).

[12] David Basin, Cas Cremers, Tiffany Hyuni-jin, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. 2016. Design, Analysis, and Implementation of ARPKI: an Attack-Resilient Public-Key Infrastructure. *IEEE Transactions on Dependable and Secure Computing* (2016).

[13] Henry Birge-Lee, Yixin Sun, Annie Edmundson, Jennifer Rexford, and Prateek Mittal. 2017. Using BGP to acquire bogus TLS certificates. *HotPETSâĂŽ17* (2017).

[14] CAIDA. [n. d.]. Anonymized Internet Traces Dataset. ([n. d.]).

[15] Vincent Cheval, Mark Ryan, and Jiangshan Yu. 2014. DTKI: a new formalized PKI with no trusted parties. *arXiv preprint arXiv:1408.1023* (2014).

[16] Taejoong Chung, Roland van Rijswijk-Deij, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, and Christo Wilson. 2017. A longitudinal, end-to-end view of the DNSSEC ecosystem. In *USENIX Security*.

[17] Tianxiang Dai, Haya Shulman, and Michael Waidner. 2016. DNSSEC Misconfigurations in Popular Domains. In *International Conference on Cryptology and Network Security*. Springer, 651–660.

[18] Deploy260. 2014. Email Hijacking. https://www.internetsociety.org/blog/2014/09/email-hijacking-new-research-shows-why-we-need-dnssec-now/. (2014).

[19] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. 1993. Perfectly Secure Message Transmission. *J. ACM* 40, 1 (Jan. 1993), 17–47. https://doi.org/10.1145/138027.138036

[20] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. ZMap: Fast Internet-wide Scanning and Its Security Applications.. In *USENIX Security Symposium*, Vol. 8. 47–53.

[21] Peter Eckersley. 2011. Sovereign keys: A proposal to make https and email more secure. *Electronic Frontier Foundation* 18 (2011).

[22] P. Eckersley and J. Burns. 2010. An observatory for the SSLiverse. DEFCON'18. (2010).

[23] Hongyu Gao, Vinod Yegneswaran, Yan Chen, Phillip Porras, Shalini Ghosh, Jian Jiang, and Haixin Duan. 2013. An empirical reexamination of global DNS behavior. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*. ACM, 267–278.

[24] Yossi Gilad and Amir Herzberg. 2013. Fragmentation Considered Vulnerable. *ACM Transactions on Information and System Security (TISSEC)* 15, 4 (April 2013), 16:1–16:31. A preliminary version appeared in WOOT 2011.

[25] Phillipa Gill, Michael Schapira, and Sharon Goldberg. 2011. Let the market drive deployment: a strategy for transitioning to BGP security. In *SIGCOMM*, Srinivasan Keshav, Jörg Liebeherr, John W. Byers, and Jeffrey C. Mogul (Eds.). ACM, 14–25. http://doi.acm.org/10.1145/2018436.2018439

[26] Phillipa Gill, Michael Schapira, and Sharon Goldberg. 2012. Modeling on quicksand: Dealing with the scarcity of ground truth in interdomain routing data. *ACM SIGCOMM Computer Communication Review* 42, 1 (2012), 40–46.

[27] Matthias Gohring, Haya Shulman, and Michael Waidner. 2018. Path MTU Discovery Considered Harmful. In *38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018*. 866–874.

[28] Shuai Hao, Yubao Zhang, Haining Wang, and Angelos Stavrou. 2018. End-Users Get Maneuvered: Empirical Analysis of Redirection Hijacking in Content Delivery Networks. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association.

[29] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2012. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*. 205–220.

[30] Amir Herzberg and Haya Shulman. 2012. Security of Patched DNS. In *Computer Security - ESORICS 2012 - 17th European Symposium on Research in Computer Security, Pisa, Italy, September 10-12, 2012. Proceedings*. 271–288.

[31] Amir Herzberg and Haya Shulman. 2013. Fragmentation considered poisonous, or: One-domain-to-rule-them-all. org. In *Communications and Network Security (CNS), 2013 IEEE Conference on*. IEEE, 224–232.

[32] Amir Herzberg and Haya Shulman. 2013. Socket Overloading for Fun and Cache Poisoning. In *ACM Annual Computer Security Applications Conference (ACM ACSAC), New Orleans, Louisiana, U.S.*, Charles N. Payne Jr. (Ed.).

[33] Amir Herzberg and Haya Shulman. 2013. Vulnerable Delegation of DNS Resolution. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*. 219–236. https://doi.org/10.1007/978-3-642-40203-6_13

[34] Ralph Holz, Lothar Braun, Nils Kammenhuber, and Georg Carle. 2011. The SSL landscape: a thorough analysis of the x. 509 PKI using active and passive measurements. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 427–444.

[35] Margaret Hu. 2015. Taxonomy of the Snowden Disclosures. *Wash & Lee L. Rev.* 72 (2015), 1679–1989.

[36] Dan Kaminsky. 2008. It's the End of the Cache As We Know It. In *Black Hat conference*. http://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf.

[37] Kernel.org. 2011. Linux Kernel Documentation. http://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt. (2011).

[38] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perring, Collin Jackson, and Virgil Gligor. 2013. Accountable key infrastructure (AKI): a proposal for a public-key validation infrastructure. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 679–690.

[39] Amit Klein, Haya Shulman, and Michael Waidner. 2017. Counting in the Dark: Caches Discovery and Enumeration in the Internet. In *The 47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.

[40] Amit Klein, Haya Shulman, and Michael Waidner. 2017. Internet-Wide Study of DNS Cache Injections. In *INFOCOM*.

[41] Jeffrey Knockel and Jedidiah R Crandall. 2014. Counting Packets Sent Between Arbitrary Internet Hosts.. In *FOCI*.

[42] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. *Certificate transparency*. Technical Report.

[43] Kyle Schomp, Tom Callahan, Michael Rabinovich, and Mark Allman. 2013. On measuring the client-side DNS infrastructure. In *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 77–90.

[44] Sharon Goldberg. 2018. The myetherwallet.com hijack and why it's risky to hold cryptocurrency in a webapp. https://medium.com/@goldbe/the-myetherwallet-com-hijack-and-why-its-risky-to-hold-cryptocurrency-in-a-webapp-261131fad278. (2018).

[45] Haya Shulman and Michael Waidner. 2014. Fragmentation Considered Leaking: Port Inference for DNS Poisoning. In *Applied Cryptography and Network Security (ACNS), Lausanne, Switzerland*. Springer.

[46] Haya Shulman and Michael Waidner. 2015. Towards Security of Internet Naming Infrastructure. In *European Symposium on Research in Computer*

Security. Springer, 3–22.

[47] Haya Shulman and Michael Waidner. 2017. One Key to Sign Them All Considered Vulnerable: Evaluation of DNSSEC in the Internet.. In *NSDI*. 131–144.

[48] Sooel Son and Vitaly Shmatikov. 2010. The hitchhikerâĂŹs guide to DNS cache poisoning. In *Security and Privacy in Communication Networks*. Springer, 466–483.

[49] Pawel Szalachowski, Stephanos Matsumoto, and Adrian Perrig. 2014. PoliCert: Secure and flexible TLS certificate management. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 406–417.

[50] Dan Wendlandt, David G Andersen, and Adrian Perrig. 2008. Perspectives: Improving SSH-style Host Authentication with Multi-Path Probing.. In *USENIX Annual Technical Conference*, Vol. 8. 321–334.

[51] Hao Yang, Eric Osterweil, Dan Massey, Songwu Lu, and Lixia Zhang. 2011. Deploying cryptography in Internet-scale systems: A case study on DNSSEC. *Dependable and Secure Computing, IEEE Transactions on* 8, 5 (2011), 656–669.

A PKI AND DNS SECURITY

A.1 Public Key Infrastructure

Public Key Infrastructure (PKI) is a comprehensive system to provide public-key encryption and digital signature services. It includes a set of roles, policies and procedures to manage digital certificates and public-key encryption. It is used when there is a strict requirement to confirm the identity of parties involved in the communication or to validate the information transferred. The basic concept behind PKI is to bind public keys with respective identities of entities such as persons or organizations. To establish the binding, an entity should go through a process of registration and verification at and by a certificate authority (CA). Since there are different levels of binding, the process may vary, which can be fully automated or under human supervision. With a complete process of registration and verification, the CA would conclude the binding by issuing a certificate, which confirms the identity of the entity bound with the public key. Registration Authority (RA) accepts requests for digital certificates and authenticates the requesting entity. However, they do not actually sign the certificate that is issued. They would forward the certificate issuing to a CA after assuring valid and correct registration. After the issuance of a certificate, CA should publish it so that applications can retrieve it on behalf of users. Directory systems that are LDAP (Lightweight Directory Access Protocol)-compliant are believed to be the best technology for certificate repositories. Directories may be made publicly available or they may be private to a specific organization. Prior to each use of a certificate, the revocation status of a certificate must be checked. As a result, a PKI must incorporate a scalable certificate revocation system. The CA must securely publish the status of each certificate, while application software must verify the revocation information prior to each use of a certificate. Certificate Revocation List (CRL) is a popular way to maintain revocation information, which contains a list of digital certificates that have been revoked. CRLs can be published to a directory system by CAs. An alternative to CRLs is the certificate validation protocol, Online Certificate Status Protocol (OCSP), where a responder answers queries about the revocation information of a requested certificate.

A.2 Domain Name System (DNS)

Domain Name System (DNS), [RFC1034, RFC1035], is a distributed database containing mappings for resources (also called *resource records (RRs)*), from *domain names* to different values. The most popular and widely used mappings, [23], are for IP addresses, represented by `A` type RRs, that map a domain name to its IPv4 address, and name servers, represented by `NS` type RRs, that map a name server to domain name. The resource records in DNS correspond to the different services run by the organisations and networks, e.g., hosts, servers, network blocks.

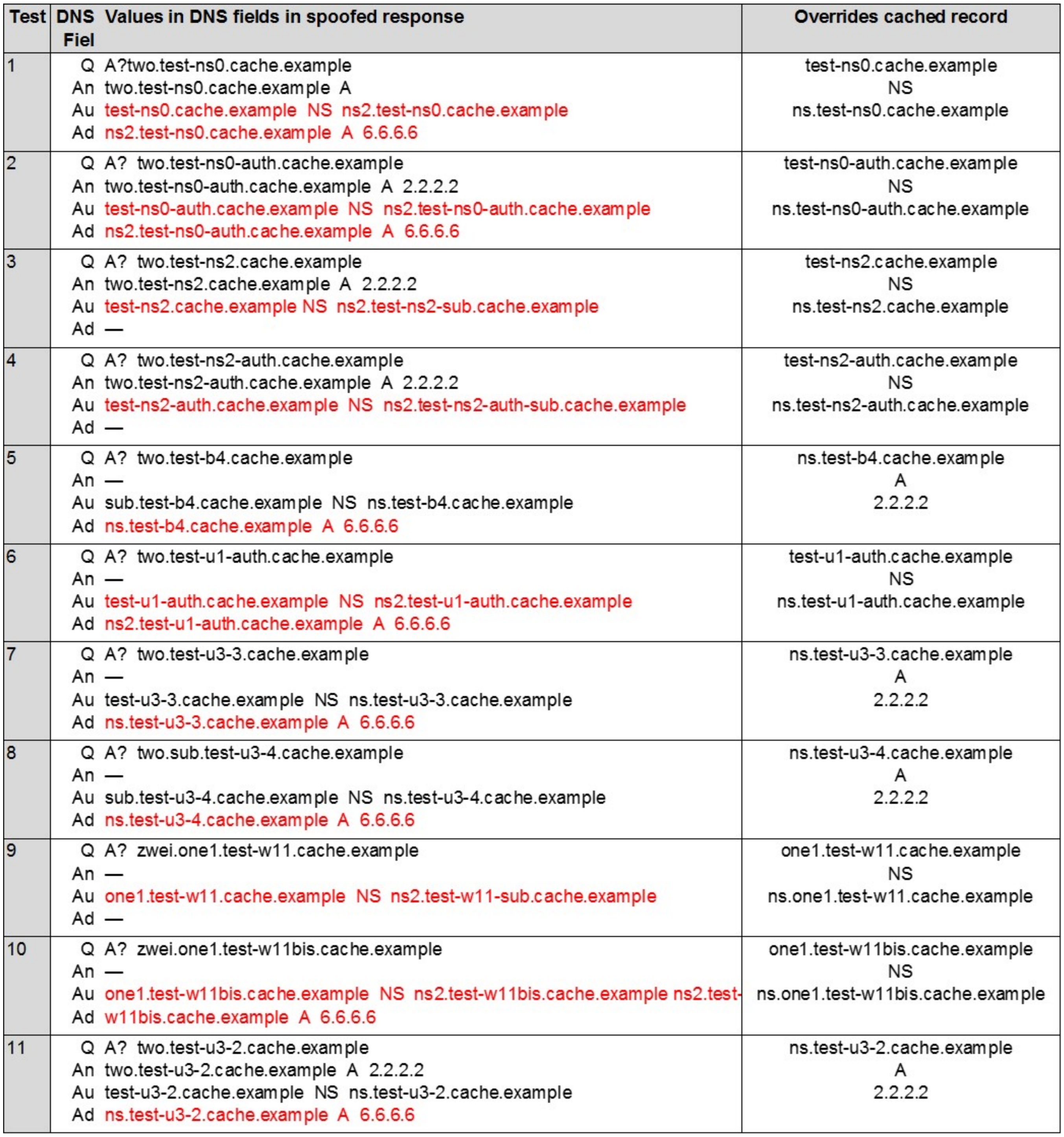
DNS is a client-server protocol, used by the resolvers to retrieve RRs stored in the zone files maintained by the name servers. The resolvers communicate to the name servers using a simple request-response protocol (typically over UDP); for instance, (abstracting out details) to translate `www.foo.bar` resolvers locate the name server `ns.foo.bar`, authoritative for `foo.bar`, and obtain the IP address of the machine hosting the web server of the website `www.foo.bar`. Resolvers store the DNS records, returned in responses, in their caches for the duration indicated in the Time To Live (TTL) field of each record set.

The zones are structured hierarchically, with the root zone at the first level, Top Level Domains (TLDs) at the second level, and millions of Second Level Domains (SLDs) at the third level. The IP addresses of the 13 root servers are provided via the *hints* file, or compiled into DNS resolvers software and when a resolver's cache is empty, every resolution process starts at the root. According to the query in the DNS request, the root name server redirects the resolver, via a `referral` response type, to a corresponding TLD, under which the requested resource is located. There are a number of TLDs types, most notably: *country code TLD* (ccTLD), which domains are (typically) assigned to countries, e.g., `us, il, de`, and *generic TLD* (gTLD), whose domains are used by organisations, e.g., `com, org`, and also by US government and military, e.g., `gov, mil`. Domains in SLDs can also be used to further delegate subdomains to other entities, or can be directly managed by the organisations, e.g., as in the case of `ibm.com, google.com`.

A.3 DNS Cache Poisoning and Defences

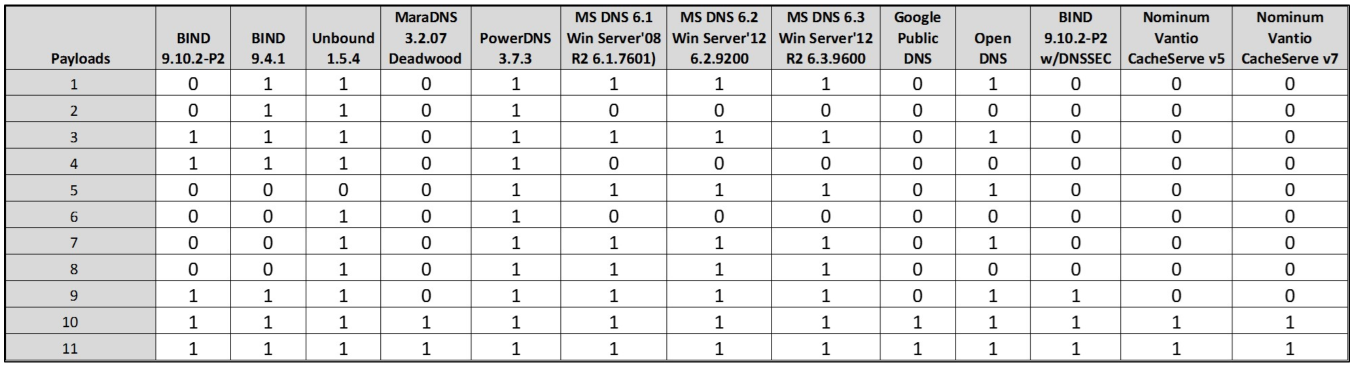
In the course of a DNS cache poisoning attack, the attacker sends spoofed DNS responses impersonating a real nameserver. The responses contain malicious DNS records, pointing legitimate services at incorrect addresses or names. If a victim DNS resolver accepts and caches the responses, it will redirect the services or clients using it to incorrect hosts. DNS cache poisoning expose to malware distribution, credentials theft, and can be leveraged for censorship [8] or for surveillance [35], as well as for financial gain by cyber criminals. The poisoned DNS responses are sent from a spoofed IP address (impersonating a legitimate nameserver) to the victim DNS resolver. Prior to accepting poisoned responses, the DNS resolvers validate that the responses contain the same source port and transaction identifier (TXID) as were in the corresponding DNS request. In accordance with the standard best practices [RFC5452] the DNS resolvers should select source ports and TXIDs at random, hence spoofing the correct responses, for attackers that do not see the DNS request packets, is not a simple challenge – the attacker has a $\frac{1}{2^{32}}$ success chance.

Following Kaminsky's cache poisoning attack, DNS resolvers were quickly patched to support challenge-response defences against cache poisoning. Most existing challenge-response mechanisms, [RFC5452], are 'patches', randomising and validating existing fields in the TCP/IP protocols. We next review standardised and



Test	DNS Fiel	Values in DNS fields in spoofed response	Overrides cached record
1	Q	A?two.test-ns0.cache.example	test-ns0.cache.example
	An	two.test-ns0.cache.example A	NS
	Au	test-ns0.cache.example NS ns2.test-ns0.cache.example	ns.test-ns0.cache.example
	Ad	ns2.test-ns0.cache.example A 6.6.6.6	
2	Q	A? two.test-ns0-auth.cache.example	test-ns0-auth.cache.example
	An	two.test-ns0-auth.cache.example A 2.2.2.2	NS
	Au	test-ns0-auth.cache.example NS ns2.test-ns0-auth.cache.example	ns.test-ns0-auth.cache.example
	Ad	ns2.test-ns0-auth.cache.example A 6.6.6.6	
3	Q	A? two.test-ns2.cache.example	test-ns2.cache.example
	An	two.test-ns2.cache.example A 2.2.2.2	NS
	Au	test-ns2.cache.example NS ns2.test-ns2-sub.cache.example	ns.test-ns2.cache.example
	Ad	—	
4	Q	A? two.test-ns2-auth.cache.example	test-ns2-auth.cache.example
	An	two.test-ns2-auth.cache.example A 2.2.2.2	NS
	Au	test-ns2-auth.cache.example NS ns2.test-ns2-auth-sub.cache.example	ns.test-ns2-auth.cache.example
	Ad	—	
5	Q	A? two.test-b4.cache.example	ns.test-b4.cache.example
	An	—	A
	Au	sub.test-b4.cache.example NS ns.test-b4.cache.example	2.2.2.2
	Ad	ns.test-b4.cache.example A 6.6.6.6	
6	Q	A? two.test-u1-auth.cache.example	test-u1-auth.cache.example
	An	—	NS
	Au	test-u1-auth.cache.example NS ns2.test-u1-auth.cache.example	ns.test-u1-auth.cache.example
	Ad	ns2.test-u1-auth.cache.example A 6.6.6.6	
7	Q	A? two.test-u3-3.cache.example	ns.test-u3-3.cache.example
	An	—	A
	Au	test-u3-3.cache.example NS ns.test-u3-3.cache.example	2.2.2.2
	Ad	ns.test-u3-3.cache.example A 6.6.6.6	
8	Q	A? two.sub.test-u3-4.cache.example	ns.test-u3-4.cache.example
	An	—	A
	Au	sub.test-u3-4.cache.example NS ns.test-u3-4.cache.example	2.2.2.2
	Ad	ns.test-u3-4.cache.example A 6.6.6.6	
9	Q	A? zwei.one1.test-w11.cache.example	one1.test-w11.cache.example
	An	—	NS
	Au	one1.test-w11.cache.example NS ns2.test-w11-sub.cache.example	ns.one1.test-w11.cache.example
	Ad	—	
10	Q	A? zwei.one1.test-w11bis.cache.example	one1.test-w11bis.cache.example
	An	—	NS
	Au	one1.test-w11bis.cache.example NS ns2.test-w11bis.cache.example ns2.test-	ns.one1.test-w11bis.cache.example
	Ad	w11bis.cache.example A 6.6.6.6	
11	Q	A? two.test-u3-2.cache.example	ns.test-u3-2.cache.example
	An	two.test-u3-2.cache.example A 2.2.2.2	A
	Au	test-u3-2.cache.example NS ns.test-u3-2.cache.example	2.2.2.2
	Ad	ns.test-u3-2.cache.example A 6.6.6.6	

Figure 12: Selected payloads for cache overwriting; see full list in [40]. Spoofed DNS records are marked in red.



Payloads	BIND 9.10.2-P2	BIND 9.4.1	Unbound 1.5.4	MaraDNS 3.2.07 Deadwood	PowerDNS 3.7.3	MS DNS 6.1 Win Server'08 R2 6.1.7601)	MS DNS 6.2 Win Server'12 6.2.9200	MS DNS 6.3 Win Server'12 R2 6.3.9600	Google Public DNS	Open DNS	BIND 9.10.2-P2 w/DNSSEC	Nominum Vantio CacheServe v5	Nominum Vantio CacheServe v7
1	0	1	1	0	1	1	1	1	0	1	0	0	0
2	0	1	1	0	1	0	0	0	0	0	0	0	0
3	1	1	1	0	1	1	1	1	0	1	0	0	0
4	1	1	1	0	1	0	0	0	0	0	0	0	0
5	0	0	0	0	1	1	1	1	0	1	0	0	0
6	0	0	1	0	1	0	0	0	0	0	0	0	0
7	0	0	1	0	1	1	1	1	0	1	0	0	0
8	0	0	1	0	1	1	1	1	0	0	0	0	0
9	1	1	1	0	1	1	1	1	0	1	1	0	0
10	1	1	1	1	1	1	1	1	1	1	1	1	1
11	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 13: Evaluation of selected payloads against popular DNS software, appliances and public services; see full list in [40].

most commonly used challenge-response authentication mechanisms.

DNS uses a random 16-bit TXID (transaction identifier) field that associates a DNS response with its corresponding request. DNS implementations additionally support a random selection of name servers each time they send a request. The main defence, that makes poisoning impractical is a (16-bit) source port randomisation recommended in [RFC5452], which together with a TXID result in a search space containing 232 possible values; a source port identifies the client side application in requests, and is echoed (as a destination port) in responses. Specific recommendations for port randomisation algorithms were recently provided in [RFC6056]. Due to the significance of port randomisation for preventing off-path attacks, e.g., cache poisoning and injections into TCP, multiple studies were conducted to measure support of port randomisation in the Internet, and it seems that many resolvers adopted port randomisation methods that were recommended in [RFC6056]. Currently the security of most DNS resolvers relies on these challenge-response mechanisms. However, relying only on TXID and source port randomisation is not believed to be sufficient against cache poisoning.

To mitigate the DNS cache poisoning attacks, the IETF designed and standardised Domain Name System Security Extensions (DNSSEC) [RFC4033-RFC4035]. Unfortunately DNSSEC requires significant changes to the DNS infrastructure as well as to the protocol, and although proposed and standardised already in 1997, it is still not widely deployed. The low adoption of DNSSEC in tandem with the recent wave of cache poisoning vulnerabilities and evidence for DNS injections in the Internet stimulated efforts within the operational and research communities to standardise alternative *easy to adopt* defences.