Off-Path Hacking:
The Illusion of Challenge-Response Authentication

Yossi Gilad*‡, Amir Herzberg†‡, Haya Shulman§‡

Abstract—Everyone is concerned about Internet security, yet most traffic is not cryptographically protected. Typical justification is that most attackers are *off-path* and cannot intercept traffic; hence, intuitively, *challenge-response* defenses should suffice to ensure authenticity. Often, the challenges re-use existing header fields to protect widely-deployed protocols such as TCP and DNS.

We argue that this practice may often give an *illusion of security*. We review recent off-path TCP injection and DNS poisoning attacks, enabling attackers to circumvent existing challenge-response defenses. Both TCP and DNS attacks are non-trivial, yet practical. The attacks foil widely deployed security mechanisms, and allow a wide range of exploits, such as long-term caching of malicious objects and scripts.

We hope that this review article will help improve defenses against off-path attackers. In particular, we hope to motivate, when feasible, adoption of cryptographic mechanisms such as SSL/TLS, IPsec and DNSSEC, providing security even against stronger Man-in-the-Middle attackers.

Keywords: off-path attacks, DNS cache poisoning, TCP injections, challenge-response defenses.

1 Introduction

Since 1989 [1], experts have been arguing that Internet security requires cryptographic protocols, ensuring security against *Man-in-the-Middle (MitM) attackers*. A MitM attacker is located on the path of the communicating parties, and can manipulate the communication between them in any way, i.e., intercept, modify, block and inject spoofed packets; see the *MitM Cookie Monster* in Figure 1.

The information security community invested significant efforts in developing cryptographic schemes and protocols, standards and products, providing security against MitM attackers, such as IPsec, SSL/TLS, Secure-BGP and DNSSEC. In spite of all these efforts, and although Internet security is well recognised to be critical, most Internet traffic is still not cryptographically protected. For example, we found that only about 6% of the TCP traffic is cryptographically protected with SSL/TLS (based on CAIDA dataset of 3 million packets [2]); and less than 1% of the DNS resolvers enforce DNSSEC (cryptographic) validation [3].

*mail@yossigilad.com
†amir.herzberg@gmail.com
§haya.shulman@gmail.com
‡Department of Computer Science, Bar Ilan University
§Fachbereich Informatik, Technische Universität Darmstadt/EC-SPRIDE

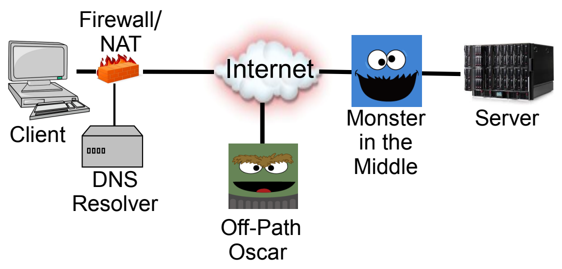


Fig. 1. Off-Path Attacker Network Model.

We believe that the main reason for the underutilisation of cryptography, is an *illusion of security* against network-based attacks, due to *two false beliefs*. The first false belief, is that in reality, attackers can rarely obtain MitM capabilities, and even when they can, they are reluctant to do so since such activities may lead to detection. We claim that this is incorrect; there are common scenarios where attackers may obtain MitM capabilities, e.g., by accessing wireless communication, by manipulations of the largely unprotected routing mechanisms, or by controlling some intermediate device. Furthermore, such attacks are often carried out, without detection and repercussions, e.g., route hijacking occurs frequently [4].

However, in this review, we focus on the second false belief, which is that current, non-cryptographic, Internet protocols already provide sufficient protection against *typical, common* attackers, and in particular, against *off-path attackers*.

Unlike a MitM attacker, an off-path attacker cannot observe or modify legitimate packets sent between other parties, however, he can transmit packets with a *spoofed* (fake) source IP address - impersonating some legitimate party, as illustrated by *Off-Path Oscar* in Figure 1. Spoofed packets are used in many attacks, most notably, in Denial of Service (DoS) attacks. Significant efforts are made to make spoofing less readily available to attackers, most notably ingress filtering ([RFC3704]). However, IP spoofing is still possible via many ISPs, see [5] and [6]; hence, the IP-spoofing ability is often available.

Our main goal in this review is to convince that this second belief is (also) false, and that *current Internet protocols are often vulnerable even to an off-path attacker*. Specifically, we discuss a few recent results, that allow off-path attacks on basic Internet protocols: traffic injection into TCP connections and DNS cache poisoning.

The key to the off-path attacks that we discuss is *circumvention of challenge-response defenses*. Challenge-response defenses are often relied upon to distinguish between (spoofed) packets from an off-path attacker and (legitimate) packets from legitimate communication end-point. In order to authenticate a response from a server, a client sends a random *challenge* with the request, which is echoed in the *response*. Since an off-path attacker, which we dub Oscar, cannot eavesdrop on packets exchanged between the server and the client, it appears that Oscar would have to guess the challenge; hence, the (sufficiently long, random) challenge allows to prevent Oscar from crafting a packet with a valid response.

The security of most Internet applications, e.g., email, web surfing, and most peer-to-peer applications, relies on challenge-response mechanisms, mainly as part of the underlying TCP and DNS protocols. For example, the widely-used web-security mechanisms based on cookies and other 'same origin policy' mechanisms, depend on the security of both TCP and DNS.

Trivially, *challenge-response mechanisms are ineffective against MitM attackers*, since they can eavesdrop on the challenges and send the matching response. The false sense of security is due to two false beliefs mentioned above: that MitM capabilities are 'rarely practical' and that existing challenge-response mechanisms, in particular, in TCP and DNS, provide sufficient defenses against the (weaker and common) off-path attackers. The goal/hope of prevention of off-path attacks is stated in RFC standards: [RFC4953] for TCP and [RFC5452] for DNS. The (false) belief is that TCP and DNS specifications and implementations (were enhanced to) provide security against off path adversaries. Indeed, since its early days, most Internet traffic is directed using DNS and carried over TCP - where both are protected only using challenge-response mechanisms. This is in spite of many warnings, e.g., [1], [7], [8].

Most existing challenge-response mechanisms are 'patches', reusing existing fields in the protocol as challenges; as we later show, this is often the root of the vulnerability. DNS uses a random 16-bit *TXID (transaction identifier)* field (which associates a DNS response with its corresponding request); the TXID, however, is too short to provide sufficient defense. TCP's main defense is the random 32-bit *ISN (initial sequence number)* [RFC6528] (which identifies where the data 'fits' within the transmission stream). Furthermore, many implementations additionally use a random (16-bit) *source port* (which identifies a client-side application) in requests, echoed (as destination port) in responses [RFC6056].

Many attacks, and in particular those we describe in the following sections, exploit the fact that each challenge field has a different original purpose in the protocol, and learn its correct response in a separate phase of the attack; this 'challenge-by-challenge' strategy allows the attacker to provide a valid response to all challenges in feasible time.

We review vulnerabilities, allowing off-path attacks on both TCP and DNS in (common) scenarios, i.e., where Oscar can circumvent the existing challenge-response mechanisms. Challenge-response defenses may fail in several ways:

Insufficient entropy: challenges may be insufficiently-long or non-uniform. Both types were abused in attacks against old implementations of DNS [9], [10] and TCP [11], [12].

Piggybacking: attacker may 'piggyback' fake content, onto valid responses (containing correct challenges), exploiting IP fragmentation. Such attacks were presented for DNS [13] and TCP [14], [15].

Side-channels: attacker may reduce the entropy of the challenge, by exploiting the fact that challenges mostly or wholly *reuse existing protocol fields*. Namely, challenges are fields which already exist in requests and are echoed in responses for some other purpose. Such attacks were presented for DNS [16], [17], [18] and TCP [8], [19], [20], [21], [22], [23], [24].

The root cause of many of these attacks is the attempt to retrofit security, and in this case incorporate a challenge-response mechanism, into an existing protocol. By reusing existing protocol fields, the defenses were deployed only by changing the clients, and without coordinated changes in servers (and the protocol itself). Such defenses are much easier to deploy - but also easier to attack. Specifically, we discuss attacks that allow an off-path attacker to *learn* the 'dual-use' challenge fields. This allows off-path TCP injection and DNS cache poisoning.

It seems that good defenses may require changes to the protocols themselves; this may be harder to deploy, but will ensure security. The changes can be via explicit and dedicated challenge-response mechanisms, e.g., [25], or via deployment of *cryptographic defenses*, such as signatures and MACs (Message-Authentication-Codes). The cryptographic defenses, e.g., MACs, are computed by the sender, using the sender's private signing key or a secret shared authentication key, and validated by the recipient (using the sender's corresponding public key or a shared secret key). In contrast to challenge-response defenses, such as [25], cryptography requires additional infrastructures, e.g., a public-key infrastructure (PKI) or other means to establish shared keys, as well as additional computations and communication, but protects even against MitM attackers, and is usually affordable as even relatively limited clients such as 'smartphone' devices are equipped with highly capable processors and broad-band communication capability.

1.1 History of Off-Path Attacks

TCP and DNS are basic protocols, and off-path attacks on their authenticity - TCP injection and DNS poisoning - impact almost all Internet applications. As such, it is a common belief that they ensure integrity against an

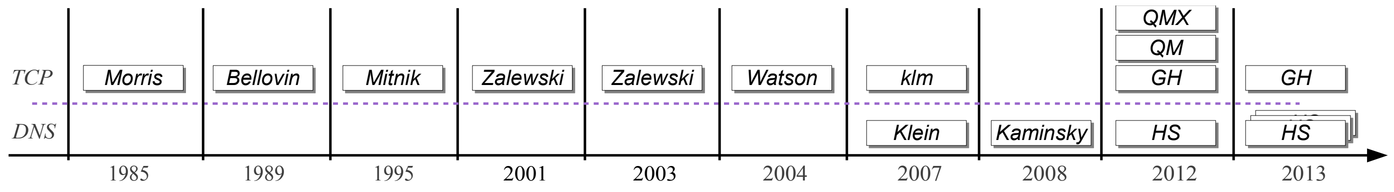


Fig. 2. Time-line of DNS Poisoning and TCP Injection attacks.

off-path attacker. However, security against off-path (or MitM) attackers was not of the original design goals of these protocols, and only minimal changes were done to the specifications to support challenge-response defenses, e.g., selecting identifiers at random.

Indeed, over the years, significant attention and efforts were dedicated to validating and improving the off-path security of TCP and DNS - and numerous off-path attacks were launched, some of them widely publicised. In Figure 2 we present a 'time-line' of important attacks and security improvements, for both TCP (upper row) and DNS (lower row).

The time-line begins in 1985, with publication of a TCP injection attack based on the use of predictable sequence numbers [8], and Bellovin's seminal paper from 1989 [1], pointing out that security should *not* be based on the presumed off-path protection of DNS and TCP. Bellovin presented vulnerabilities of (some) TCP implementations to off-path attacks, and discussed potential exploits and defenses.

Unfortunately, in spite of these warnings, until 1995 most TCP stacks still used trivially-predictable initial sequence numbers (ISN). This changed only after the notorious TCP injection attack by Mitnick on Shimomura [19]. After the attack, many implementations changed to 'less predictable' ISN choices. However, in 2001, Zalewski showed that most implementations are still 'sufficiently predictable', allowing off-path attacks; this motivated adoption of more random choice of ISNs in most operating systems, as standardised in [RFC6528].

In 2003, Zalewski also commented that 'piggybacking' on fragmented TCP traffic may allow injection attacks [14]; the piggybacking attack was improved in [15], and exploited for DNS poisoning in [13].

A special TCP injection attack was presented by Watson [11] in 2004. This attack only injected a 'RST' packet, breaking up a connection, and focused on long-lived connections using known client (and server) ports and addresses, as used at the time by the Internet routing protocol BGP. To address this concern, many TCP implementations also began using 'unpredictable' client ports.

In 2007, there were two surprising results: (1) a TCP injection attack presented by the pseudonym author *klm* in Phrack magazine [20], and (2) a DNS poisoning attack exploiting poor random-number generators [9]. Both attacks were clever and significant, although with limited scope. In particular, the attack on TCP worked only against Windows machines, connected directly to the Internet (rather than via firewall, as usually is the case), and did not handle concurrent connections.

Kaminsky presented an even more significant DNS poisoning attack in 2008, which allowed efficient off-path poisoning of most DNS resolver implementations at the time [10] (see Section 2). The response to this attack was rapid adoption of additional 'patches', mostly, more challenge-response fields, increasing the length of the random challenge and therefore (hopefully) making the attack impractical; the most notable patch was source port randomisation (SPR); see [RFC5452].

Following 2008, there were several years without additional off-path attacks; Kaminsky's attack was addressed by SPR and other 'patches' to DNS-resolvers, and klm's attack was impractical and not widely known. This changed dramatically in 2011-2013, with the publication of *ten* new off-path attacks. The first, in 2011, was an attack on fragmented IP traffic [15]. This was followed, by *four* new DNS poisoning attacks [16], [13], [17], [18], a connection-exposing attack [26] and *four* TCP injection attacks [21], [22], [24], [23].

1.2 Malicious Agents

Some of the off-path attacks require, in addition to the spoofing ability, also a *malicious agent* in the victim's network or host. We briefly explain the different agent models.

A *zombie* is a machine controlled by the adversary, e.g., compromised by malware, in the victim's network.

A *puppet* is weaker agent: a restricted malicious script or applet running in web-browser sandbox. Attacks relying on a puppet agent require (only) that a client in the victim network 'surfs' to the attacker's web-site, enabling the adversary to run such a script. The script is restricted by *same origin policy* (described in [RFC6454]), and can only communicate via the browser, i.e., request (and receive) HTTP objects (no access to TCP/IP packet headers).

2 DNS Cache Poisoning

The Domain Name System (DNS) provides name to address mapping for services in the Internet. DNS name servers maintain the mappings for services in the domains for which they are authoritative, and DNS resolvers are agents, used by clients, to retrieve the mappings from the name servers. Resolvers send requests to name servers, and receive responses. Prior to accepting and caching the responses, they are validated; widely deployed validations rely on challenge-response mechanisms. The resolvers send the challenges, e.g., in form of a random 16-bit TXID field within the request, and validate that the same values appear in responses. We next explain that challenge-response defenses may fail even against weak, off-path attackers.

2.1 Kaminsky's DNS Cache Poisoning

In 2008, Kaminsky [10] presented an efficient cache poisoning attack against resolvers which authenticated responses using a random TXID and used a known (fixed) source port, which at that time was 53. The steps of the attack, illustrated in Figure 3, are the following:

(1) the attacker triggers a DNS request for a random sub-domain of the victim domain `$1.foo.com`.

(2) DNS resolver receives the request and forwards it to the target name server.

(3) the attacker then sends 2^{16} responses with spoofed source IP (of the name server); each response is a referral mapping of the name server `ns.foo.com` to 6.6.6.6, an IP address controlled by the attacker.

(4) the response containing the correct TXID is accepted, cached and returned to the client.

(5) authentic DNS response is ignored, since there is no matching pending request.

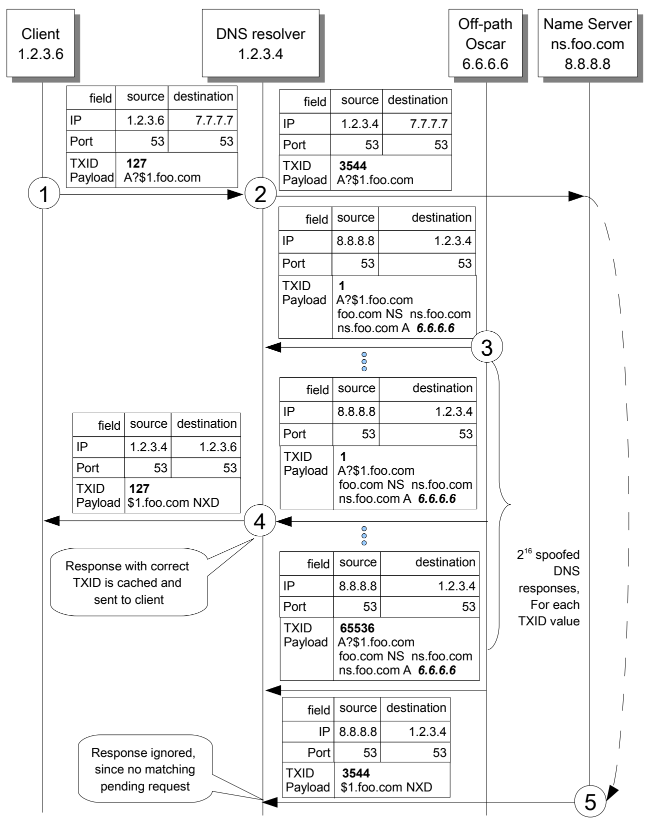
If the attack fails, i.e., an authentic response from the real name server arrived before the correct response from the attacker, the attack is repeated with a new random subdomain `$2.foo.com`.

Following Kaminsky's attack, additional challenge-response mechanisms were proposed to increase the entropy in DNS requests, see [RFC5452]. The most popular mechanism, supported by majority of the resolvers, is source port randomisation (SPR), which, in tandem with TXID, produce a search space of 2^{32} values and was believed to provide sufficient protection against poisoning by off-path adversaries. This reduced the motivation for deployment of DNSSEC [RFC4033-4035], the cryptographic defense against poisoning.

We found different techniques, [13], [16], [17], [18], that allow to circumvent the popular defenses against poisoning by off-path attackers. In this work we show a simple technique, from [16], that uses side-channels for ports' prediction and applies to common network scenarios, where the DNS resolvers are located behind a NAT device (as in Figure 1).

2.2 Vulnerability of Resolvers Behind NAT

Network Address Translation (NAT) devices are used to alleviate the problem of IPv4 addresses depletion in the Internet, by allocating non-unique addresses in local networks and sharing unique (external) addresses between a number of internal hosts. The NAT devices modify the source ports in outbound packets in order to



Client 1.2.3.6
DNS resolver 1.2.3.4
Off-path Oscar 6.6.6.6
Name Server ns.foo.com 8.8.8.8
field | source | destination
IP | 1.2.3.6 | 7.7.7.7
Port | 53 | 53
TXID 127
Payload A?$1.foo.com
1
field | source | destination
IP | 1.2.3.4 | 7.7.7.7
Port | 53 | 53
TXID 3544
Payload A?$1.foo.com
2
field | source | destination
IP | 8.8.8.8 | 1.2.3.4
Port | 53 | 53
TXID 1
Payload A?$1.foo.com foo.com NS ns.foo.com ns.foo.com A 6.6.6.6
3
field | source | destination
IP | 8.8.8.8 | 1.2.3.4
Port | 53 | 53
TXID 1
Payload A?$1.foo.com foo.com NS ns.foo.com ns.foo.com A 6.6.6.6
field | source | destination
IP | 1.2.3.4 | 1.2.3.6
Port | 53 | 53
TXID 127
Payload $1.foo.com NXD
4
Response with correct TXID is cached and sent to client
field | source | destination
IP | 8.8.8.8 | 1.2.3.4
Port | 53 | 53
TXID 65536
Payload A?$1.foo.com foo.com NS ns.foo.com ns.foo.com A 6.6.6.6
2^16 spoofed DNS responses, For each TXID value
field | source | destination
IP | 8.8.8.8 | 1.2.3.4
Port | 53 | 53
TXID 3544
Payload $1.foo.com NXD
Response ignored, since no matching pending request
5

Fig. 3. Kaminsky's Attack.

correlate between the inbound packets and the internal hosts.

NAT devices were patched to support port randomisation to prevent attacks. However, as we show, even systems that randomise ports in outbound packets may expose resolvers to attacks. Specifically, in [16] we tested NAT devices that support unpredictable port allocation algorithms and are reported as secure by DNS-checkers, yet are vulnerable to port derandomisation. Our attacks exploit a standard, and correct, behavior of the devices, e.g., recommended in [RFC6056]. What enables our attacks are *side channels*, which may have not been considered during the design of these algorithms. In this section we describe port derandomisation against *per-destination* ports allocation, implemented by many systems; see [16] for other popular algorithms.

PER-DESTINATION PORTS ALLOCATION. For a tuple defined by ⟨src-IP:src-port,dst-IP:dst-port,protocol⟩, a per-destination NAT selects the first port at random, and subsequent ports are increased sequentially (for that tuple).

PREDICT-THEN-POISON ATTACK. Off-path attacker, Oscar, controls a *zombie*, i.e., non-privileged malware, that runs on a client host in the LAN. The attack is composed of two phases, illustrated in Figure 4: *port prediction* and *poisoning*. During the port prediction

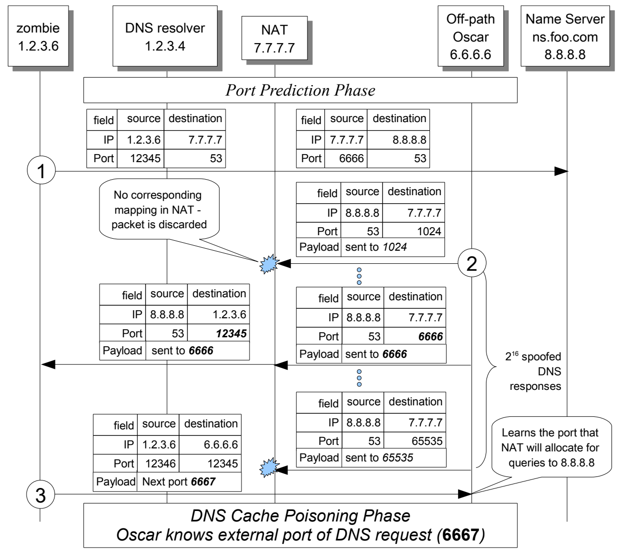

zombie 1.2.3.6
DNS resolver 1.2.3.4
NAT 7.7.7.7
Off-path Oscar 6.6.6.6
Name Server ns.foo.com 8.8.8.8
Port Prediction Phase
No corresponding mapping in NAT - packet is discarded
2^16 spoofed DNS responses
Learns the port that NAT will allocate for queries to 8.8.8.8
DNS Cache Poisoning Phase
Oscar knows external port of DNS request (6667)

Fig. 4. Predict-then-Poison Attack, assuming per-destination NAT.

phase, Oscar and the zombie expose the port that will be allocated to the request of the DNS resolver. Next, during the poisoning phase, Kaminsky's attack is launched (Figure 3).

(1) The zombie sends a packet to create a mapping in the NAT table; in the example in Figure 4 we assume arbitrarily that port 6666 was selected.

(2) Then, Oscar at address 6.6.6.6 sends 2^{16} packets with a spoofed source IP of 8.8.8.8, s.t. each is sent to a different port of the NAT and each contains the destination port in its payload; only the packet to port 6666 arrives at the zombie.

(3) The zombie increments this port by 1, in our example the result is 6667, and sends it to Oscar; this is the external port that will be allocated by the NAT to the subsequent DNS request of the resolver to the victim name server.

This phase allows to bypass the SPR defense.

2.3 Empirical Evaluation

The recent DNS poisoning results ([16], [13], [17], [18]) were validated empirically in many different network settings and for a large variety of devices. As a specific example, we focus on the attacks in [16] which include the predict-then-poison attack (described above); the viability of these attacks (and others) depends mainly on the port allocation method at the NAT.

We tested the attacks in [16] against eight popular NAT devices. Out of these, two used sequential port allocation (Linux Netfilter and Windows ICS), and hence were vulnerable to the predict-then-poison attack. Five NAT devices (Fedora, Wingate, FreeBSD, Cisco IOS and Cisco ASA) were vulnerable to the *trap* attacks, also from [16], and only one (Checkpoint FW-1) was immune to the attacks in [16]. All tests were done against a Bind9 DNS resolver, connected via the different NAT devices to the attacker.

We also checked port allocation methods in DNS requests (sent over UDP), using traces from two CAIDA datasets from 2012 [2], and found that 30% of the requests were sent from a fixed port and 54% of the requests were sent from incrementing ports.

3 TCP Injections

The Transmission Control Protocol (TCP) is the main transport protocol of the Internet, carrying most of the communication between clients and servers. The recent off-path TCP injection attacks operate in two phases:

(1) Learn Connection Four-Tuple. Oscar, the off-path attacker, learns the four parameters of a TCP connection between a client and a server, that is, their respective IP addresses and ports.

(2) Learn Sequence Number. Oscar learns the current sequence number, for packets sent from the server to the client or vise versa.

After the attacker learns the connection four-tuple and one of the sequence numbers, he can inject data into the TCP connection, impersonating as one of the participating peers to the other. Table 1 surveys the techniques used in recent injection attacks for both phases and their requirements. In the reminder of this section we present a simple implementation for each phase.

3.1 Learn Connection Four-Tuple

In order to launch an injection attack, Oscar must first identify a TCP connection between the victim client and server. Some methods for identifying this connection scan the victim's machine, either remotely ([20]) or locally ([24], [23]); see Table 1.

In this subsection we describe a simple method which uses a puppet (script restricted by browser sandbox) running on the client machine to *open* such a connection. Since Oscar opens the connection he chooses the server, and the server's IP address and port are known. To find the client's IP address, the puppet sends a request to Oscar's site; this request contains the client's IP address.

The final challenge of this phase is to detect the client port. In [22] we showed how to break the randomised port selection algorithm which was standartised in [RFC6056], and used by Linux and Android clients; this attack exploits the TCP state machine, which leaks to the off-path attacker information about the choice of the client port.

However, in many cases the attack is simpler since the clients (in particular, those running Windows) do not use a randomised algorithm, and assign ports to connections *sequentially*; we exploited this port selection method in [21], as follows: The puppet opens a connection to Oscar's remote site before and after opening the connection to the victim server; Oscar observes p_1

	Learn Connection Four-Tuple	Learn Sequence Number
klm [20]	Remote probing for connection (Windows client, no firewall)	Side channel (Windows client)
Qian et al. [24], [23]	Local probing for connection, e.g., with netstat (Malware)	Read client system-counters, (Malware; in [24] also seq. # checking firewall)
Gilad and Herzberg [21]	Establish connection, exploit sequential port allocation impl. (Puppet, Windows client)	Side channel (Puppet, Windows client)
Gilad and Herzberg [22]	Establish connection, client port derandomisation (Puppet, client behind firewall)	exploit HTTP-client state machine, (Puppet, no TLS/SSL)

TABLE 1
Off-Path TCP Injection Attacks: Building Blocks. In parentheses: requirements.

and p_2, the client ports used in the connection to his sites. If $p_2 = p_1 + 2$, then he learns that the connection to the server is via port $p_1 + 1$ (otherwise Oscar restarts the attack).

3.2 Learn Sequence Number

The next step after identifying the victim-connection, is learning one or both of the connection's sequence numbers. Off-path attackers use different methods to infer the sequence numbers (since they cannot observe them directly). We focus on the technique of [22] which exploits an *under-specification of HTTP* to learn the client's sequence number.

BACKGROUND. As of HTTP 1.1, clients can send multiple requests to the same web-server in *pipeline* over a single ('persistent') HTTP connection. In order to allow browsers to match between each response and the corresponding request, the server sends the responses exactly in the order in which it received the requests. The browser (client) keeps a FIFO queue of pending HTTP requests for each connection, and handles them one by one, as follows. To handle a request, the browser reads the bytes in the TCP connection's receive-buffer (when they become available). The browser expects to find the matching response in the beginning of TCP's receive-buffer and parses the response.

The HTTP standard does not specify what the browser should do when the receive-buffer contains data which is *not* a valid ('parsable') HTTP response. Browsers often handle this situation as follows: they treat *all available data* in the receive-buffer as payload of a response with the following default HTTP header:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=us-ascii
Content-Length: available-data-size
```

The browsers return this 'response' to the requesting module, normally, the rendering engine.

SEQUENCE NUMBER LEARNING TECHNIQUE. The learning phase has two steps: *Inject* and *Observe*, illustrated in Figure 5. In the inject step, Oscar injects data into the stream of HTTP responses that the server sends to the client. This data is read in the observe step, which allows Oscar to determine the server's sequence number.

(A) Inject step. Let *wnd* denote the browser's receive-buffer for the connection and $|wnd|$ denote its size. In order to inject the data, Oscar sends to the browser $\frac{2^{32}}{|wnd|}$ packets, spoofed to appear to be from the server (on its victim-connection with the client). The i^{th} packet has server sequence number $i \cdot |wnd|$; since the sequence number field is 32-bits long, exactly *one* of these packets has a 'valid' sequence number, which falls within the limits of *wnd*; all the other packets are discarded by the client. Each of Oscar's packets contains as payload *page*(i) which is a simple web-page defined as follows:

```
<HTML><BODY>
<iframe src = "oscar.com/i.html" />
</BODY></HTML>
```

(B) Observe step. In this step, the puppet makes prevalent requests to the server, until it reaches the data injected by Oscar in the previous step. Each server-response that arrives at the client shifts *wnd* forward; after several such responses arrive, there is no gap of unreceived bytes between the injected data and the beginning of *wnd*. Then, the browser reads the injected-response, assuming that it corresponds to the request.

The last server-response usually overwrites the beginning of the injected data (according to the TCP specification, in case of overlap, new data supersedes the old), therefore, the injected 'response' will usually be corrupt. However, as noted above, many browsers handle the injected data as payload 'wrapped' with a default header. When the browser renders *page(i)*, it tries to retrieve *i.html* from Oscar's web-site (see Figure 5); providing to Oscar the value of i. In order to keep *page(i)* intact despite the overwrite, when Oscar sends *page(i)* in the inject step, he prepends to it an easily removable pad. The value of i allows Oscar to compute the next server sequence number that the client expects.

3.3 Empirical Evaluation

We evaluated the Inject and Observe technique on connections with the 1000 most popular web-sites according

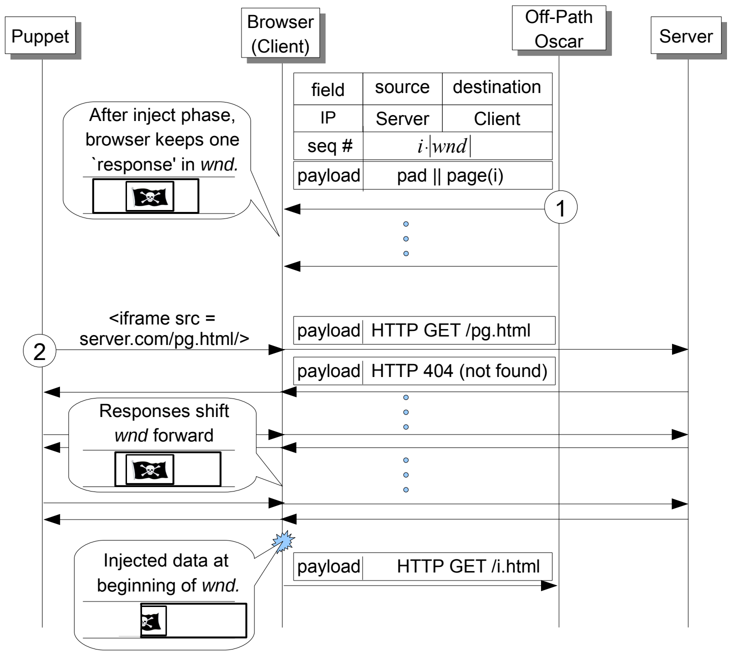

Puppet
Browser (Client)
Off-Path Oscar
Server
field | source | destination
IP | Server | Client
seq # | i·|wnd|
payload | pad || page(i)
After inject phase, browser keeps one `response' in wnd.
1
<iframe src = server.com/pg.html/>
2
payload | HTTP GET /pg.html
payload | HTTP 404 (not found)
Responses shift wnd forward
Injected data at beginning of wnd.
payload | HTTP GET /i.html

Fig. 5. Sequence Number Learning Technique.

to Alexa ranking. We placed the attacker and client machines in the same network, which allowed the attacker to send packets to the Internet using the client's IP address (in reality, the attacker would connect through one of the many ISPs that do not perform ingress filtering, see discussion in Introduction). The client and attacker connect through different physical interfaces of a network switch; hence, attacker is off-path, i.e., cannot observe packets to/from the client. The client and attacker connect through 10Mbps link to the Internet.

In order to identify a successful exposure of the sequence numbers, we used the puppet to request a non-existing object from the server and injected a spoofed HTTP OK response. If the puppet identified the spoofed OK response, then we determined that the data injection was successful and the attacker obtained the correct sequence numbers.

RESULTS. We performed three executions of the attack, using different popular browsers: Chrome (v23), Firefox (v16) and Internet Explorer (v9); all three browsers parse HTTP responses as described above, therefore, their TCP connections are vulnerable to the Inject and Observe attack. The attack had success rate of 78% on connections with the tested popular websites, and requires approximately 2.5 minutes to launch.

4 EXPLOITING INJECTION AND POISONING

To conclude our discussion of off-path TCP injection and DNS poisoning attacks, we briefly discuss some potential exploits.

Exploiting *DNS poisoning* is straightforward. Both users and programs use DNS extensively to resolve domain names; DNS poisoning allows circumvention of security mechanisms, e.g., SPF and blacklists, and 'hijacking' of connection requests to legitimate servers. In particular, 'hijacking' can allow *phishing*, where a user thinks that he interacts with a trusted site, while he actually deals with a fake site (exposing credentials, installing malware, etc.). The poisoned mapping is cached and hence can impact many users of the resolver.

Exploiting *TCP injections* is more challenging, since TCP is a transport protocol and does not involve caching. However, in common scenarios, TCP injections can allow critical exploits. In particular, TCP injections suffice to *circumvent the Same Origin Policy*, hijack 'cookies' and cause execution of malicious scripts (XSS). In order to cause long-term impact similar to DNS poisoning, attackers can exploit caching of objects by web caches. By crafting the HTTP headers of his injected packets, Oscar can cache spoofed objects (e.g., web-pages) for long time. When using a web-cache, this can impact many users (see [22]).

5 CONCLUSIONS

The vulnerabilities reviewed in this article show that relying on challenge-response mechanisms against off-path attackers can often be circumvented. Specifically, the techniques discussed in this article allow off-path attackers to circumvent the main challenge-response defenses: *source port randomisation* and *initial sequence number randomisation*.

Our message is that defenses should be designed and analysed carefully, and not 'patched' by reusing existing fields whose entropy may be insufficient or reduced by side-channels. In particular, in order to prevent these and other attacks, even by (stronger) MitM attackers, we recommend deployment of cryptographic defenses, in the common scenarios where the computational and communication overheads are acceptable.

ACKNOWLEDGEMENTS

We thank the anonymous referees for their comments and feedback on earlier version of this manuscript. This research was supported by grant 1354/11 from the Israeli Science Foundation (ISF).

REFERENCES

[1] S. M. Bellovin, "Security Problems in the TCP/IP Protocol Suite," *Computer Communication Review*, vol. 19, no. 2, pp. 32–48, apr 1989.

[2] CAIDA, "Anonymized Internet Traces 2012 Dataset," http://www.caida.org/data/passive/passive_2012_dataset.xml, 2012.

[3] O. Gudmundsson and S. D. Crocker, "Observing DNSSEC Validation in the Wild," in *SATIN*, March 2011.

[4] H. Ballani, P. Francis, and X. Zhang, "A Study of Prefix Hijacking and Interception in the Internet," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 265–276.

[5] Advanced Network Architecture Group, "Spoofer Project," http://spoofer.csail.mit.edu/summary.php, May 2013.

[6] R. Beverly, A. Berger, Y. Hyun, and K. C. Claffy, "Understanding the Efficacy of Deployed Internet Source Address Validation Filtering," in *Internet Measurement Conference*, A. Feldmann and L. Mathy, Eds. ACM, 2009, pp. 356–369. [Online]. Available: http://doi.acm.org/10.1145/1644893.1644936

[7] S. M. Bellovin, "A Look Back at "Security Problems in the TCP/IP Protocol Suite"," in *ACSAC*. IEEE Computer Society, 2004, pp. 229–249.

[8] R. T. Morris, "A Weakness in the 4.2BSD Unix TCP/IP Software," AT&T Bell Laboratories, Tech. Rep., Feb. 1985.

[9] A. Klein, "OpenBSD DNS Cache Poisoning and Multiple O/S Predictable IP ID Vulnerability," http://www.openbsdsupport.com.ar/books/OpenBSD_DNS_Cache_Poisoning_and_Multiple_OS_Predictable_IP_ID_Vulnerability.pdf, October-November 2007.

[10] D. Kaminsky, "It's the End of the Cache As We Know It," in *Black Hat conference*, August 2008, http://www.blackhat.com/presentations/bh-jp-08/bh-jp-08-Kaminsky/BlackHat-Japan-08-Kaminsky-DNS08-BlackOps.pdf.

[11] P. Watson, "Slipping in the Window: TCP Reset Attacks," Presented at CanSecWest, http://bandwidthco.com/whitepapers/netforensics/tcpip/TCP%20Reset%20Attacks.pdf, October 2004.

[12] M. Zalewski, "Strange Attractors and TCP/IP Sequence Number Analysis," http://lcamtuf.coredump.cx/newtcp/, 2001.

[13] A. Herzberg and H. Shulman, "Fragmentation Considered Poisonous: or one-domain-to-rule-them-all.org," in *CNS 2013. The Conference on Communications and Network Security. IEEE*. IEEE, 2013.

[14] M. Zalewski, "A New TCP/IP Blind Data Injection Technique?" BugTraq mailing list post, http://lcamtuf.coredump.cx/ipfrag.txt, 2003.

[15] Y. Gilad and A. Herzberg, "Fragmentation Considered Vulnerable," *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 4, pp. 16:1–16:31, April 2013, a preliminary version appeared in WOOT 2011.

[16] A. Herzberg and H. Shulman, "Security of Patched DNS," in *European Symposium on Research in Computer Security*, ser. LNCS, S. Foresti, M. Yung, and F. Martinelli, Eds., vol. 7459. Springer, 2012, pp. 271–288. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33167-1

[17] ——, "Vulnerable Delegation of DNS Resolution," in *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, September, 2013. Proceedings*, ser. Lecture Notes in Computer Science. Springer, 2013.

[18] ——, "Socket Overloading for Fun and Cache Poisoning," in *ACM Annual Computer Security Applications Conference (ACM ACSAC)*, December 2013.

[19] T. Shimomura and J. Markoff, *Takedown: The Pursuit and Capture of Kevin Mitnick, America's Most Wanted Computer Outlaws - by the Man Who Did It*, 1st ed. Hyperion Press, 1995.

[20] klm, "Remote Blind TCP/IP Spoofing," Phrack magazine, 2007.

[21] Y. Gilad and A. Herzberg, "Off-Path Attacking the Web," in *USENIX Workshop on Offensive Technologies*, 2012, pp. 41 – 52.

[22] ——, "When Tolerance Becomes Weakness: The Case of Injection-Friendly Browsers," in *Proceedings of the International World Wide Web Conference*, May 2013.

[23] Z. Qian, Z. M. Mao, and Y. Xie, "Collaborative TCP Sequence Number Inference Attack: How to Crack Sequence Number Under a Second," in *Proceedings of the ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2012, pp. 593–604. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382258

[24] Z. Qian and Z. M. Mao, "Off-Path TCP Sequence Number Inference Attack," in *IEEE Symposium on Security and Privacy*, 2012, pp. 347–361.

[25] Y. Gilad and A. Herzberg, "LOT: A Defense Against IP Spoofing and Flooding Attacks," *ACM Transactions on Information and System Security*, vol. 15, no. 2, pp. 6:1–6:30, Jul. 2012. [Online]. Available: http://doi.acm.org/10.1145/2240276.2240277

[26] ——, "Spying in the Dark: TCP and Tor Traffic Analysis," in *Privacy Enhancing Technologies Symposium*, ser. Lecture Notes in Computer Science, S. Fischer-Hübner and M. Wright, Eds., vol. 7384. Springer, 2012, pp. 100–119. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31680-7