Resilience of Deployed TCP to Blind Attacks

Matthew Luckie
University of Waikato
mjl@wand.net.nz

Robert Beverly
Naval Postgraduate School
rbeverly@nps.edu

Tiange Wu
CAIDA / UC San Diego
tiangewu@caida.org

Mark Allman
ICSI
mallman@icir.org

kc claffy
CAIDA / UC San Diego
kc@caida.org

ABSTRACT

As part of TCP's steady evolution, recent standards have recommended mechanisms to protect against weaknesses in TCP. But adoption, configuration, and deployment of TCP improvements can be slow. In this work, we consider the resilience of *deployed* TCP implementations to blind in-window attacks, where an off-path adversary disrupts an established connection by sending a packet that the victim believes came from its peer, causing data corruption or connection reset. We tested operating systems (and middleboxes deployed in front) of webservers in the wild in September 2015 and found 22% of connections vulnerable to in-window SYN and reset packets, 30% vulnerable to in-window data packets, and 38.4% vulnerable to at least one of three in-window attacks we tested. We also tested out-of-window packets and found that while few deployed systems were vulnerable to reset and SYN packets, 5.4% of connections accepted in-window data with an invalid acknowledgment number. In addition to evaluating commodity TCP stacks, we found vulnerabilities in 12 of 14 of the routers and switches we characterized – critical network infrastructure where the potential impact of any TCP vulnerabilities is particularly acute. This surprisingly high level of extant vulnerabilities in the most mature Internet transport protocol in use today is a perfect illustration of the Internet's fragility. Embedded in historical context, it also provides a strong case for more systematic, scientific, and longitudinal measurement and quantitative analysis of fundamental properties of critical Internet infrastructure, as well as for the importance of better mechanisms to get best security practices deployed.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques; C.2.0 [**Computer-communication Networks**]: Security

Keywords

TCP; security; blind attacks; middleboxes

1. INTRODUCTION

Despite the rich history of attacks against TCP [40, 38, 31, 14], and subsequent counter-measures [6, 36, 32, 15], little recent empirical data exists on the current resilience of *deployed* TCP implementations on the Internet. In this paper, we focus on attacks against TCP by a "blinded" attacker, i.e. an attacker that is off-path and does not observe the TCP connection. Of particular interest is the vulnerability of critical infrastructure, including web servers, routers, and switches. For example, long-lived BGP and OpenFlow TCP sessions are well-known to be vulnerable [8, 12], while recent work [14] demonstrated new blinded attacks that pollute web caches. As connection speeds increase and TCP receive windows grow, other long-lived TCP applications such as file transfers, ssh, and rsync may become vulnerable.

We therefore take a multi-faceted measurement examination of both commodity operating system TCP stacks as well as proprietary TCP implementations in routers and switches. We adopted an oracle-based approach by simulating a variety of blinded attacks. Our simulated exploits sent packets (RST, SYN, data, both in and out of window) that a blinded attacker might send as part of a brute-force attack. We also examined ephemeral port selection strategies in the wild, which impact the feasibility of blind attacks.

To test commodity TCP stacks as deployed in the wild, we established connections to websites in the Alexa list [2], and observed the behavior of their TCP implementations in response to our oracle's probing packets. We do not claim that our results are representative of any particular population; we tested websites to understand properties of currently deployed web operating systems and middleboxes. Nevertheless, we consider our results indicative of current likely behavior in other populations, including systems that support long-lived protocols including ssh and rsync.

In this population as measured in September 2015, we found: (1) 38.4% of systems tested were vulnerable to at least one blind in-window attack; (2) the in-window data attack is the most significant vulnerability, as 29.6% of systems accepted data with inadequate validation of the acknowledgment number; (3) systems that advertised a maximum segment size (MSS) of 1380 were almost never vulnerable to in-window reset and SYN packets (suggesting that middleboxes with this feature correctly filtered those suspicious packets) but incorrectly passed invalid in-window data packets, (4) in response to an in-window SYN, 1.1% of hosts established a parallel TCP connection using the same 4-tuple, which is unexpected behavior as a 4-tuple can only support a single TCP connection at any one time.

IMC'15, October 28–30, 2015, Tokyo, Japan.
© 2015 ACM. ISBN 978-1-4503-3848-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2815675.2815700.

We also investigated the TCP behavior of 14 network infrastructure devices (BGP routers and OpenFlow switches) in a laboratory setting to assess their resilience to these same off-path attacks. Of the devices we tested, 12 of 14 had some vulnerability in their handling of packets that could have been sent by a blinded attacker. In particular, 10 of 14 devices accepted data that could have been sent by a blinded attacker.

Finally, in addition to these active tests, we used passive captures of TCP connections to understand deployed ephemeral port selection algorithms. Within a March 2015 packet trace from an internal 10Gbps link between Chicago and Seattle belonging to a Tier-1 ISP, we observed 50% of selected ports within a 2K range – a fraction of the available ephemeral port range. Longitudinal analysis of an enterprise trace suggests that the slow adoption of new operating systems is slowly increasing the range of ports used in the wild.

We begin by describing blind in-window attack methods and defenses in section 2, and present our active measurement techniques to test for the vulnerability in section 3. In section 4 we present our findings from using our method to test TCP stacks deployed in a webserver environment; we found 22% of connections vulnerable to blind in-window SYN and reset packets, and 30% vulnerable to in-window data packets. In section 5 we report our findings from testing embedded TCP stacks in routers and switches; we found that while more recent stacks were not vulnerable to blind in-window SYN and reset attacks, they have not deployed best practices when validating data packets. In section 6 we explore deployed port selection algorithms over time using longitudinal connection summary logs from a campus intrusion detection system, and using a packet capture from a Tier-1 ISP backbone link. We end with a brief discussion on the current difficulty of mounting a blind attack in section 7, a comparison with related work in section 8, and conclude in section 9. We make our code publicly available as part of scamper [24], allowing vendors to test their deployed implementations of TCP for the mitigations recommended by RFC 5961 [32].

2. BACKGROUND

A TCP connection is defined by a four-tuple consisting of source and destination IP addresses, and source and destination port numbers. To interfere with a TCP connection, a "blinded" attacker (i.e. an attacker that is off-path and does not observe the TCP connection) has to guess (or infer) all four tuple values, as well as valid sequence (and sometimes acknowledgment) numbers for the connection. The TCP protocol has evolved, and vendors have strengthened implementations over time, to better resist attack. For example, the initial sequence number (ISN) was once predictable – allowing an attacker to infer acknowledgment numbers and forge entire TCP connections [6]. Modern TCPs instead generate unpredictable ISN values using a cryptographic hash function.

Despite unpredictable ISNs, Paul Watson demonstrated in 2004 that a blinded attacker could reset TCP connections by: (1) guessing a sequence number within the receive window; and (2) leveraging the small range of likely source port values [38]. Not only did operating systems at the time typically choose source ports from within a small range (1024-5000), they were selected sequentially. Watson noted two possible solutions. First, a host should choose an ephemeral port value randomly when it initiates a new TCP connection to increase the difficulty of blind-attacking a TCP connection by a factor of up to 2^{16}. Second, hosts should strictly validate the sequence and acknowledgment values, so that rather than requiring the packet to be merely in-window, it has to be exactly congruent with the receiver's position in the connection's window space. RFC 4953 [36], published in 2007, described blinded attack vectors in TCP known at that time. RFC 5961 [32], published in 2010, described attacks that a blinded attacker could conduct in addition to the well-known reset attack, and recommended that receivers strictly validate received packets for their position in the receive window.

Blind attackers rely on brute force to attack TCP connections, and are therefore constrained by network capacity. With the exponential growth of network capacity, attacks become easier. For example, a 100Mbps network link is capable of carrying 148,410 minimum-sized frames per second, and can be rented from some hosting networks for less than $100 per month. A blind attacker needs only small packets that fit entirely inside the minimum-sized Ethernet frame. Therefore, an attack that requires 2^{17} (131,072) packets to succeed can complete in less than one second at 100Mbps.

2.1 Slipping in the window

Operating systems retain myriad state about each active TCP connection. Beyond the source and destination IP addresses and port numbers that represent the 4-tuple, each end of the TCP connection has a 32-bit number corresponding to the next sequence number value it will use to send new data (*snd.nxt*), a 32-bit number recording the next in-order sequence number expected from the receiver (*rcv.nxt*), a 32-bit number recording the last sequence number for which it received an acknowledgment (*snd.una*), as well as a 16-bit number the receiver advertises which represents the amount of data it can accept (*rcv.wnd*). This 16-bit number can be scaled by up to 14-bits with the TCP window scale option, so that a TCP sender can have up to 1GB of data in flight provided the receiver advertises a sufficient window size [20].

The blind in-window reset attack described by Watson [38] relies on the victim's TCP stack following the original TCP specification from 1981, RFC 793 [30], which says that "a reset is valid if its sequence number is in the window." To reset a TCP connection with a given four tuple, a blinded attacker only has to try one packet in each possible window. Therefore, the number of packets required to complete the attack is inversely proportional to the size of the advertised window, i.e. the difficulty of an attack reduces the larger the receiver's window becomes.

RFC 5961 [32] suggests TCP should tighten the original specification's notion of an acceptable sequence number from "in the window" to exactly the next expected sequence number (*rcv.nxt*). Otherwise, the victim must send a *challenge ACK*, which is an acknowledgment reporting the current values of *snd.nxt* and *rcv.nxt*. The purpose of this ACK is to double check that the sender of the reset actually intended to reset the connection. If so, it will send a second reset packet, with the challenge ACK's *rcv.nxt* value copied into the sequence number space, confirming the reset and providing a valid reset that the receiver will act on. This approach increases the number of guesses the attacker must make from 2^{32} / *rcv.wnd* to 2^{32}.

RFC 5961 describes two additional attack vectors similar in nature to the well-known reset attack: a blind reset attack using the SYN bit, and a blind data injection attack. The SYN attack is similar to the reset attack, and relies on similar language in RFC 793 [30], which says that if a SYN is received in the window it is an error, and to send a reset. As with the reset attack, RFC 5961 says the receiver must send a *challenge ACK* to confirm the loss of the previous connection; this challenge ACK must be sent regardless of the sequence number in the SYN packet.

In a blind data injection attack, an attacker tries to inject data into an existing TCP connection by guessing a sequence number within the victim's receive window. Unlike the reset and SYN attacks, an attacker should also have to guess an acknowledgment value that is acceptable to the receiver. RFC 793 states that an acknowledgment number is acceptable as long as it is not acknowledging data that has not yet been sent. In practice, this means that the acknowledgment number in the packet can be any value less than the victim's *snd.nxt*, which is a range of nearly 2^{31} values. To reduce the likelihood of success, RFC 5961 introduces a variable to represent the maximum receive window the peer has advertised to the victim (*max.rcv.wnd*) and limits the acceptable acknowledgment range to lie between *snd.una* - *max.rcv.wnd* and *snd.nxt*.

RFC 5961 explicitly states that these mitigations are not required of TCP stacks – the mitigations to the blind reset and SYN attacks are recommended, and the mitigation to the blind data attack is optional because the attack is perceived to be twice as difficult (a blind attacker has to send the data twice, with two different acknowledgment values). However, RFC 5961 also notes that all three mitigations increase the robustness of TCP in general.

Finally, some connections are subject to a trivial blind attack that does not require an in-window packet. Until September 2014, all supported versions of FreeBSD were susceptible to an attacker being able to tear down any TCP connection by sending two packets with the SYN flag set that shared the same 4-tuple as an ongoing connection [1]. If an attacker has reason to believe there are connections between two IP addresses and a destination port, and the FreeBSD system has not been patched [1], the attacker could tear down all connections with 2^{17} packets (two packets for all 2^{16} ephemeral ports). While this vulnerability was fixed in all supported versions of FreeBSD, the recommendations in RFC 5961 have only been added to the development version of FreeBSD, and all supported versions of FreeBSD as of August 2015 are still vulnerable to some blind attacks.

2.2 Defenses to blind in-window attacks

Beyond strictly checking incoming packets per the advice in RFC 5961, there are several mitigations to blind in-window attacks that vendors can implement, ranging from those that make the problem harder to cryptographic solutions that make successful attacks practically impossible. The simplest approach is for a host to choose a random source port when establishing a TCP connection. Operating systems select port values from a configurable ephemeral range. Historically, versions of Windows and BSD used the range 1024 to 5000 – a range of 3976 port values. IANA now recommends the range 49152 to 65535 [10, 19] be used by all transport protocols – a range of 16K port values. Table 1 summarizes the ephemeral port ranges used by popular operating systems.

Port Range	Size	Operating System
1024–5000	3976	Windows XP and earlier FreeBSD ≤ 4.11 (Jan 2005) Linux ≤ 2.2
49152–65535	16384	FreeBSD ≥ 5.0 (Jan 2003) Windows Vista (Jan 2007) Apple MacOS X Apple IOS
32768–61000	28232	Linux ≥ 2.4
10000–65535	55535	FreeBSD ≥ 8.0 (Nov 2011)

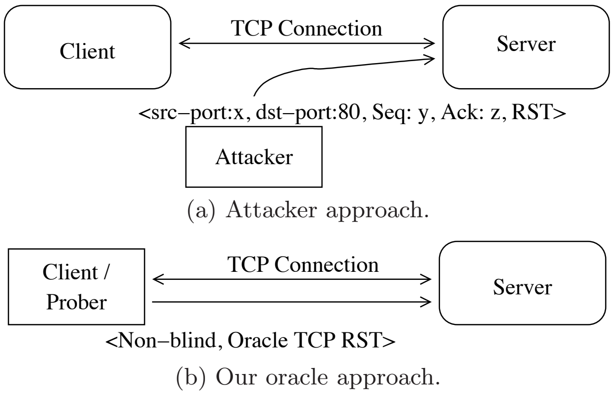
Table 1: Port ranges and sizes used by common operating systems to select ephemeral ports. Port selection strategies differ among systems; Windows and Apple currently assign ephemeral TCP ports sequentially, FreeBSD randomly, and Linux uses hash-based port selection.

It is tempting to believe that operating systems choose ports at random from within these ranges, making it difficult to guess which port values are likely to be used. This was one of the pieces of advice made by Paul Watson in 2004 when he reported on the feasibility of blind in-window attacks [38]. Researchers have since studied ephemeral port selection strategies [3] and port randomization is IETF best current practice [23] as of 2011.

Implementers made the well-known Kaminsky DNS cache poisoning attack [21] much more difficult to mount by using a random ephemeral port value selection strategy. However, for expediency, implementers made this fix in DNS server software, not in the operating systems, and while some operating systems have chosen ports at random since at least 2004 (FreeBSD, OpenBSD) current popular operating systems (Windows and MacOS) choose TCP ports sequentially from the global range of ephemeral ports. Linux kernels since version 2.6.15 choose ports sequentially from an offset in the ephemeral range based upon a cryptographic hash of the destination address and port with a secret, per [23].

Some protocols, such as BGP, typically operate between topologically adjacent systems. Because they are adjacent, the TTL value contained in the IP header will not be decremented by outside routers before packets arrive at the peer. The Generalized TTL Security Mechanism [15] relies on this fact; if the peer requires the received IP TTL to be 255 (the largest TTL value possible) then it will discard attack packets from outside the network because outside routers will have decremented the TTL and therefore the packets' TTL value cannot be 255. This mitigation does not protect Internet applications in the general case, though it does help in some situations.

It is also possible to protect TCP sessions by using a cryptographic authentication protocol between the hosts, to allow a receiver to validate that the packet must have come from its peer and not a third-party attacker. The three most common authentication options are the TCP MD5 option [16], the TCP authentication option [37], and the IP authentication option [22]. In all cases, the peers must establish a shared secret, or deploy certificates, before establishing a TCP connection. This approach is feasible for BGP, where sessions are configured between known endpoints, but


Client
TCP Connection
Server
<src–port:x, dst–port:80, Seq: y, Ack: z, RST>
Attacker
Client / Prober
TCP Connection
Server
<Non–blind, Oracle TCP RST>

(a) Attacker approach.

(b) Our oracle approach.

Figure 1: Comparing our oracle approach with that of an attacker. We do not attempt to disrupt third-party connections, rather we establish our own TCP connection and send test packets with perfect knowledge of the connection.

is more difficult in the general transport protocol case between arbitrary endpoints.

Application-layer encryption, such as that provided by TLS, does not prevent an attacker from disrupting a TCP session. A blind in-window reset, SYN, or piece of data can still cause the connection to terminate; the only advantage of TLS is that, in the case of the blind data attack, an attacker's accepted data will cause the application-layer decryption software to note the error and abort the TLS connection rather than corrupting the transmitted data.

Some TCP stacks use automatic buffer tuning [35] and increase the receive window (and *max.rcv.wnd*) when they infer the sender's transmission rate is receive window limited. A TCP stack that uses automatic buffer tuning can reduce their attack surface by using small initial receive windows, and only increase the window when they detect the sender is receive window limited. A study of TCP performance on a fiber-to-the-home network during 2011 and 2012 found that 20%-41% of TCP connections increased their window during the connection's lifetime [34], suggesting modest use of automatic buffer tuning.

Finally, since blind attacks require the attacker to use the source IP address of an ongoing TCP connection, blind attacks are possible because there is inadequate source address validation and hence IP address spoofing is allowed. RFC 2827 [13] describes best current practices for filtering spoofed packets, however previous work has found filtering not well deployed in the Internet [7] so blind attacks are unfortunately still possible.

3. ACTIVE MEASUREMENT METHOD

Our general approach is to actively open a normal TCP connection with some server and then simulate an attack on this connection. Our attacks were confined to our own experimental connections and at no point did we attempt to disrupt production traffic. Figure 1 compares our approach to that of a blind attacker. This methodology gives us perfect knowledge of the precise packets to use to determine if the TCP connection was susceptible to packets from a blind attacker who knows the four tuple of the target connection. Our specific tests follow from RFC 5961 [32]: we sent packets that have a sequence or acknowledgment number that should cause the receiver to reject or challenge the packet.

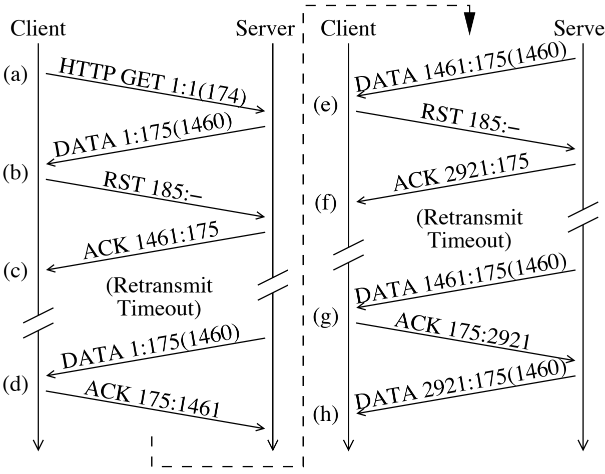


Figure 2: Overview of our blind reset and SYN test. After the TCP (and TLS) handshake, we send an HTTP request (a) to solicit a response to which we send a reset packet (b). If the server follows RFC 5961, it will send a challenge ACK (c, f) with the expected sequence number in the acknowledgment field. When the server re-transmits the unacknowledged data, we send an acknowledgment (d, g) and send the next blind reset (e) when the server sends new data. This method allows the TCP connection to progress while we send our three reset attempts; only two reset attempts (b, e) are shown.

3.1 Blind Reset and SYN Tests

Figure 2 illustrates our blind reset and SYN tests. Our tests began by establishing a TCP connection without the use of any TCP options. We established a TLS session in the HTTPS case, and sent a HTTP GET request specifying the HTTP "Host" header for the server. From there, we sent reset (or SYN) packets as if we were a blind attacker by using a sequence number 10 greater than the first byte of data the receiver is expecting from our client. We sent up to three reset (or SYN) packets to account for packet loss, interspersing each with an ACK that acknowledged a new segment of data to encourage the server to keep the TCP connection alive. If the server sent multiple packets, we used the first to trigger our test, and subsequent packets to send duplicate acknowledgments as if we had lost the first.

If we received a reset packet in response to a duplicate acknowledgment we sent, then we classified the server as vulnerable because it reset the connection in response to the reset or SYN. Likewise, if we received no other packets after sending the first reset packet, then we inferred the server reset the connection and was also vulnerable. If we received a challenge ACK, or the server ignored the packets and continued to send data, then we classified the server as not vulnerable because it ignored the packets. Otherwise, if we received a FIN before we could send all three packets, or the server went silent before we could send all three packets, then we classified the test as being inconclusive.

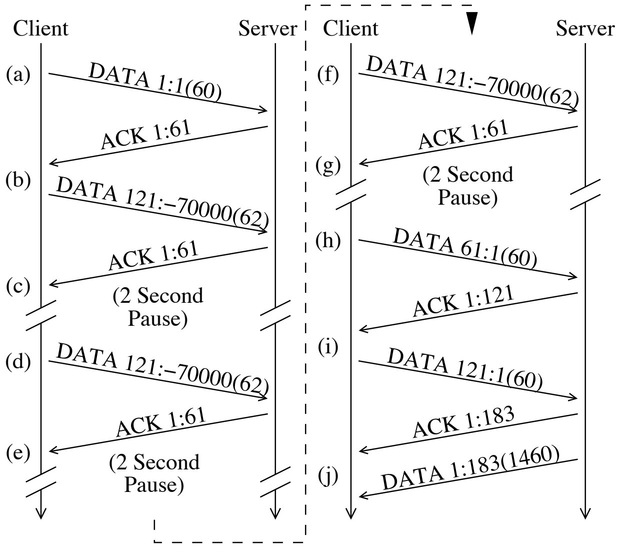


Figure 3: Overview of our blind data test. We break the first segment of data for the connection into three pieces. After the TCP handshake, we send the first piece (a) with an expected acknowledgment number, and then send the third piece (b) with an invalid acknowledgment number. We send the third piece twice more to account for potential packet loss (d, f), and then the second piece (h) with an expected acknowledgment number. If the server rejected the third piece, then we will not receive an ACK for it after the hole is filled by the second piece. We then send the third piece with an expected acknowledgment number (i) to allow the connection to complete (j). The acknowledgment packets (c, e, and g) are recommended by RFC 5961 though they are not challenge ACKs because they are not used by the server to confirm the loss of the connection.

The blind SYN test can reveal two additional behaviors. First, if the server sent a reset packet in response to the SYN (identified by the acknowledgment number in the reset being one greater than the sequence number in our reset packet) we inferred these systems were vulnerable. We confirmed these systems reset the original TCP connection, as any subsequent duplicate acknowledgments also solicited a reset. Second, if the server sent a SYN/ACK packet acknowledging the new sequence number, we classified these systems as establishing a parallel TCP connection using the same 4-tuple. This behavior was an unexpected finding; we confirmed that data continued to flow in the original TCP connection, and that the new TCP connection re-transmitted the SYN/ACK as if the server had state for both connections. We were able to associate most of the systems that established a parallel connection with a large content distribution network.

Finally, we also tested if the sequence number even had to be in-window for the reset or SYN packet to cause the connection to be reset. We begin a new test by sending packets with a sequence number 70,000 earlier than would be required for the packet to be in-window. We chose this value arbitrarily but it is a value that is outside the window and represents a position in the stream that our connection has not covered.

3.2 Blind data test

Devising a realistic methodology to test for vulnerability to the blind in-window data attack was not straight forward. In our testing, if we sent the first data packet for a connection with an unexpected acknowledgment number, we elicited a reset from 22% of the tested webserver population, regardless of the sequence number chosen. The same systems did not send a reset if the packet with an unexpected acknowledgment number was not the first data packet for the connection. In addition, we did not want to send the first segment with the exact sequence number expected by the receiver, in case the receiver's TCP has a special case for that specific segment.

Therefore, we chose a slightly complicated approach where we broke the first segment of data (the HTTP GET or the first packet in the TLS handshake) into three equal-sized pieces, and then sent the pieces as follows: (1) we sent the first piece with an acknowledgment number expected by the receiver, (2) we sent the third piece with a sequence number ahead of the receiver's *rcv.nxt* (leaving a hole for the second piece) and an acknowledgment number outside of the acceptable range defined by RFC 5961, and (3) we sent the second piece with an acknowledgment number expected by the receiver.

If the receiver accepted the third piece with an invalid acknowledgment number, the second piece will fill a hole and the receiver will send an acknowledgment for both pieces. We inferred the receiver is vulnerable to blind data attacks if we receive an acknowledgment for the third piece. Otherwise, we sent the third piece with the acknowledgment number expected by the receiver to avoid conflating an unresponsive TCP with a system that was ignoring our segments. Figure 3 illustrates our approach. We sent the third piece three times, and with a two second delay, regardless of the presence or absence of acknowledgments, and sent the first and second pieces up to three times each, to account for transient packet loss.

If we received an acknowledgment for any of the third pieces with an incorrect acknowledgment number, we classified these systems as not vulnerable and report them in the challenge ACK category, though these ACKs were not challenge ACKs because the sender was not using them to confirm the loss of the connection. If we did not receive an acknowledgment for any of the third pieces we sent with an incorrect acknowledgment number, we also classified these systems as not vulnerable because they ignored the data. If we received a reset quoting our invalid acknowledgment number, we classified these systems as vulnerable because they reset the connection, rather than discard the data packet.

Finally, we also tested if the acknowledgment number even had to acknowledge previously sent data for a blind data attack to be successful. We sent the third piece with an acknowledgment number 70,000 ahead of the receiver's position in the sequence number space, which we know from the SYN, and is beyond the feasible range of sequence numbers it could have sent given our advertised receive window.

3.3 Fingerprinting test

We completed probing with a test that established a TCP connection advertising support for TCP window scaling [20], timestamps [20], and Selective Acknowledgments [26]. We used this test to infer the normal behavior of a TCP connection with a peer, as well as to fingerprint the peer's operat-

Methods	Parameters
Blind reset: in window	Host's snd.nxt + 10
Blind reset: out of window	Host's snd.nxt - 70,000
Blind SYN: in window	Host's snd.nxt + 10
Blind SYN: out of window	Host's snd.nxt - 70,000
Blind data: behind	Peer's snd.una - 70,000 Host's snd.nxt + x
Blind data: ahead	Peer's snd.una + 70,000 Host's snd.nxt + x

Table 2: Summary of blind tests we conducted. We chose different sequence and acknowledgment values to test for different checks made by the receiver.

ing system with p0f [39]. p0f can infer the operating system using features found in the SYN/ACK packet, such as the ordering of TCP options, the window size advertised, and the IP-TTL value.

3.4 Implementation

Table 2 summarizes the blind TCP tests we implemented in scamper, an open-source parallelized packet prober [24]. For each measurement, our tool records, in a single data unit, meta-data about the test such as the URL and the server MSS seen, and all packets sent and received for that test. We also implemented a driver to coordinate scamper's measurements. Our driver first performs the sequence of blind tests towards the webserver one at a time, in random order, waiting at least one minute between tests to each unique IP address to avoid being a nuisance. Our driver concludes with the null test to fingerprint the operating system of the host. Our code is publicly available and available in scamper [24].

4. WEBSERVER VULNERABILITY

4.1 Targets

The first step in our methodology was to choose a list of web servers to test. We derived our targets from the Alexa Top 1,000,000 websites list [2]. Each of our measurement vantage points (see 4.2) independently chose 50K random web sites from the Alexa list. We sent a DNS query for each web site and used the first returned IPv4 address for the duration of our probing. Further, if multiple sites shared an IP address, we only probed the IP address once. This process yielded roughly 41K target IP addresses per vantage point. Additionally, some of our tests require a transfer to last long enough to conduct the test and therefore we must find a URL within each site that returns an object of at least 25KB. Therefore, we run a pre-probing step with wget against each IP address, as follows. If the default page for the given site is at least 25KB, we use that for our tests. Otherwise, we retrieved all objects required to display the default page and used the first that was at least 25KB in size in our tests.

4.2 Vantage Points

We used two of CAIDA's Archipelago vantage points (VPs) to conduct measurements [18]: cld-us, hosted by CAIDA in San Diego, and hlz-nz, hosted by the University of Waikato in New Zealand. We used a third machine hosted by the Massachusetts Institute of Technology (MIT) in Cambridge. We used three VPs to attempt to avoid an undetected middlebox in the hosting network from impacted our measurements by manipulating our test packets before they reached the destination network. We obtained permission from the operators of these three VPs to conduct our measurements from their network.

4.3 Results

Table 3 summarizes the results of our testing the webserver population from cld-us. The columns labeled in/out represent SYN and reset tests where the sequence numbers were in or out of window as per table 2, and the ahead/behind represent data tests where the acknowledgment number was behind or ahead of the range acceptable by RFC 5961. We found 22.2% of the connections tested were vulnerable to an in-window reset packet. Of these, most (18.8%) sent a reset in response to duplicate acknowledgments, while 3.4% went silent after we sent the reset packet. An additional 76.5% of the connections were not vulnerable to the in-window reset attack, with 71.4% issuing a challenge ACK as required by RFC 5961, and the remaining 5.1% simply ignoring the reset packets rather than confirming the loss of the connection.

22.4% of the connections we tested were vulnerable to an in-window SYN packet, with 17.1% sending a reset that acknowledged the in-window sequence number, and 5.3% sending a reset after we sent a duplicate acknowledgment. Interestingly, 35.9% of the connections we tested did not send a challenge ACK as required by RFC 5961, to confirm the loss of the previous TCP connection. Additionally, 1.1% of TCP connections acted in an unexpected way by appearing to create a second parallel TCP connection in response to the in-window SYN packet. These TCP connections were mostly with a large CDN provider, who appeared to terminate TCP connections on their systems, and relayed data from their customer's webserver obtained over a separate TCP connection.

30.3% of connections we tested were vulnerable to data injection attacks. In most of the cases (29.6%) the injected data was accepted by the stack, while the receiver sent a reset in the remaining cases. Most of the systems that reset the connection confirmed that they reset in response to the data packet because they quoted the invalid acknowledgment value in the reset packet. We also found that in the majority of the non-vulnerable connections, the server sent an ACK to the packets (37.1%) rather than just ignore the packets (29.3%).

Table 4 shows the overall results collected from the VPs we used to test webservers. Quantitatively, the results are similar, with overall classifications from each VP usually within 1% of each other, despite each VP testing a different set of random webservers. However the results of the blind in-window SYN tests were impacted by a stateful on-campus router at hlz-nz, which silently discarded outgoing SYN packets belonging to an existing connection. Because our results for each VP are quantitatively similar, we believe the behaviors observed occur close to the webserver end of the TCP connection.

Overall, 38.4% of the webservers tested from cld-us were vulnerable to at least one blind in-window attack. Figure 4 shows the overlap of vulnerability to blind in-window tests between tested systems. 12.4% of the webservers tested were vulnerable to all three blind in-window attacks tested. The vector with the largest exposure was the blind data injection

Result	Blind reset		Blind SYN		Blind data	
	in	out	in	out	behind	ahead
Accepted	3.4%	0.4%	-	-	29.6%	5.4%
Reset (ack-blind)	-	-	17.1%	0.0%	0.6%	0.6%
Reset (dup-ack)	18.8%	0.6%	5.3%	1.2%	0.1%	0.2%
Vulnerable	22.2%	1.0%	22.4%	1.2%	30.3%	6.2%
Challenge ACK	71.4%	1.1%	37.7%	57.0%	37.1%	8.1%
Ignored	5.1%	91.8%	35.9%	38.3%	29.3%	81.3%
Not Vulnerable	76.5%	93.0%	73.6%	95.3%	66.4%	89.4%
Parallel TCP	-	-	1.1%	1.1%	-	-
Early FIN	0.3%	3.3%	1.5%	1.6%	3.2%	3.7%
No Result	1.0%	2.7%	1.3%	0.9%	0.1%	0.7%
Other	1.3%	6.0%	4.0%	3.6%	3.3%	4.4%

Table 3: Overview of results for the webserver population testing from cld-us VP in September 2015. In our data, 22% of connections were vulnerable to in-window reset and SYN packets, and 30% of connections were vulnerable to an in-window data packet. A further 6.2% of connections were vulnerable to data packets with an acknowledgment ahead of the receiver's window; 5.4% accepted the data without correctly validating the acknowledgment number, and 0.8% reset the connection.

	cld-us	MIT	hlz-nz
Blind reset (in):			
Vulnerable	22.2%	22.1%	21.9%
Not Vulnerable	76.5%	76.0%	76.5%
Other	1.3%	1.9%	1.6%
Blind SYN (in):			
Vulnerable	22.4%	22.2%	0.3%
Not Vulnerable	73.6%	73.2%	94.2%
Other	4.0%	4.6%	5.5%
Blind data (behind):			
Vulnerable	30.3%	30.3%	30.3%
Not Vulnerable	66.4%	66.5%	66.2%
Other	3.3%	3.3%	4.5%

Table 4: Summary of the blind tests conducted from three VPs. Overall results are quantitatively similar, apart from the SYN tests from hlz-nz, which were blocked by a stateful campus router.

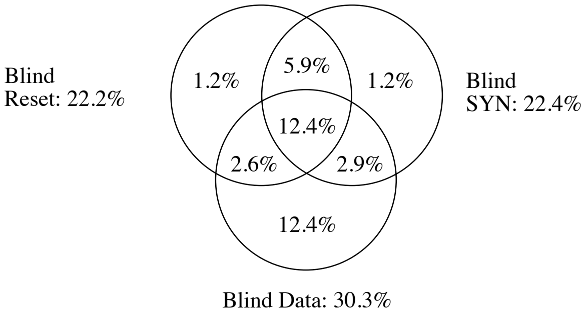


Figure 4: Overlap of vulnerability to blind in-window attacks of systems tested from cld-us. 38.4% of servers tested were vulnerable to at least one in-window attack, with 12.4% of the servers tested only vulnerable to the blind in-window data attack.

attack, with 12.4% of the webservers tested (32.3% of the vulnerable population) only vulnerable to that vector.

4.3.1 *Out-of-window tests*

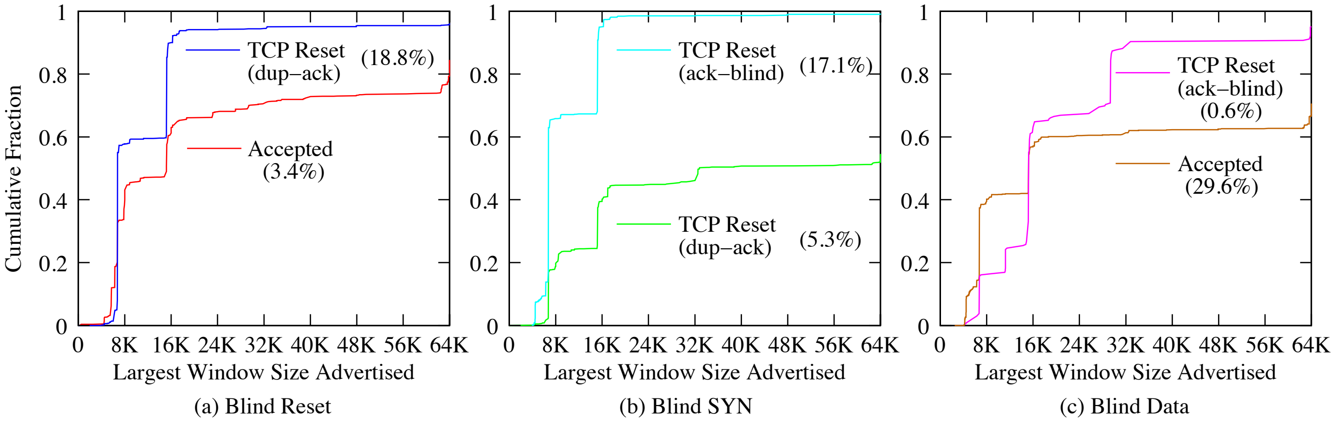
Table 3 also contains the results of our out-of-window tests. In our experiments, we found that 1.0% of tested systems incorrectly processed the reset and 1.2% incorrectly processed the SYN even though the sequence number we included was outside of the window. We are encouraged that most TCP connections were not falsely reset, though these represent systems where a blinded attacker with knowledge of the use of an application port between two IPs only has to brute force up to 65,536 ports to reset the connection, i.e. on average a successful attack requires 2^{15} packets.

More problematic is the in-window data attack, where 6.2% showed a vulnerability to packets containing an acknowledgment value ahead of the peer's send window that the peer could not have sent. In fact, 5.4% of servers acknowledged the data we sent, and sent a response anyway. A further 0.6% of connections were reset, quoting the invalid acknowledgment number we sent in our data packet. If an attacker's goal is simply to reset an arbitrary connection, then the attacker merely has to brute force a packet into the receive window, acknowledging data that has not been sent yet, a range of up to 2^{31} values, or half the sequence number space. Most (81.3%) systems simply ignored the data packet with an acknowledgment value ahead of their window, and 8.1% sent an acknowledgment before discarding the packet.

4.3.2 *Middlebox behavior*

In our previous 2010 work examining the behavior of Path MTU discovery (PMTUD) in the context of TCP [25], we found that servers that advertised an MSS of 1380 bytes were much more likely to fail PMTUD, suggesting a middlebox was interfering with PMTUD. We were curious if we could find evidence of middleboxes defending webservers from blind attacks. Table 5 shows the five server MSS values most frequently observed in our data and their vulnerability to blind attack. Because most (87.2%) of the TCP connections involved an MSS of 1460, the failure rate of these systems is within 2-3% of that observed for all connections.

However, systems with an MSS of 1380 were almost never vulnerable to blind in-window reset (2.0%) and SYN (0.5%) packets, but were vulnerable to in-window data packets in roughly half the cases. We found that 85.6% of TCP connections with an MSS of 1380 sent an ACK in response



(a) Blind Reset (b) Blind SYN (c) Blind Data

Figure 5: Largest window sizes observed for systems vulnerable to blind in-window attacks and their failure modes. A larger receive window makes it easier for an attacker to disrupt a connection. The small (8K) receive window we observed for most systems vulnerable to blind reset and SYN attacks makes it more difficult to attack those connections (2^{19} packets), but 40% of the systems vulnerable to accepting in-window data advertised a window of at least 64K.

Server MSS	Vulnerable Portion		
	Blind reset	Blind SYN	Blind data
1460 (87.2%)	23.9%	24.7%	28.1%
1380 (5.4%)	2.0%	0.5%	58.8%
8961 (2.3%)	2.3%	2.3%	4.7%
1440 (0.8%)	5.9%	4.7%	57.5%
1436 (0.7%)	22.2%	5.8%	32.5%

Table 5: Top five server MSS values advertised and the corresponding portion that were vulnerable to in-window reset, SYN, and data packets. Servers that advertised an MSS of 1380 were much less likely to be affected by in-window reset and SYN packets than the general population (MSS 1460), but were affected by in-window data packets.

to data with an acknowledgment ahead of the receiver's window, representing 57.6% of all connections that did so yet only 5.4% of the population; only 3.3% of connections with an MSS of 1460 behaved this way. Similarly, we observed 96% of TCP connections with an MSS of 1380 to send challenge ACKs to in-window SYNs and resets, and out of window resets; interestingly, 1380-MSS connections behaved identically to the general population, as 80% ignored out-of-window resets. We further investigated for the presence of middleboxes by searching for TCP connections where we received packets with different TTL values for the blind in-window tests, but not for the null test. We found that we received challenge ACKs with a different TTL value than the rest of the packets in the TCP flow where a 1380 byte MSS value was advertised, indicating that a middlebox intercepted the attack packets and correctly stopped them. Challenge ACKs with different TTL values were absent almost entirely from the other MSS values in table 5.

Systems with an MSS of 8961 were hosted by Amazon, which likely had a middlebox rewriting the server MSS value. It is unclear if this middlebox protected the TCP connections involved, or if the homogeneous nature of operating systems images is responsible for the low vulnerability rate.

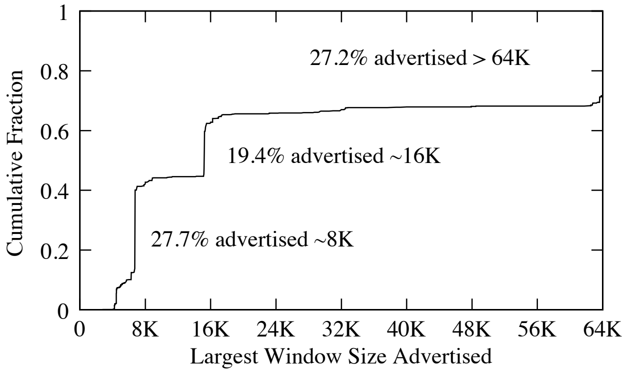


Figure 6: Largest window sizes observed for systems vulnerable to any blind in-window attack. 27% advertise a window of at least 64K, so an attacker requires a maximum of 2^{16} packets to disrupt a corresponding 4-tuple (twice as many in the data case).

4.3.3 Window sizes observed

The ease at which a blind attacker can inject a packet into a window depends on the size of the receiver's window. Figure 5 shows the largest window sizes observed to be advertised for the blind reset, SYN, and data tests. The blind in-window reset and SYN attacks appear to be the most difficult of the three attacks to accomplish, as most (60%) of potential victims used a receive window less than 8K bytes. A blind attacker would have to send more than 2^{19} packets to successfully disrupt a TCP connection that is using a receive window less than 8K bytes. However, a blind attacker might have more luck using data packets, as 50% of potential victims will accept data into a receive window of 64K. Because an attacker has to try each segment twice, with different acknowledgment values in different halves of the receiver's sequence number space, an attacker would require 2^{17} packets to successfully disrupt these TCP connections.

Figure 6 shows the landscape for systems vulnerable to any of the blind attacks tested. Because of the prevalence

Operating System	Blind reset		Blind SYN		Blind data		Total
	in	out	in	out	behind	ahead	
FreeBSD 8.x	19.2%	0.5%	**93.8%**	56.5%	**83.9%**	None	193 (0.5%)
FreeBSD 9.x	18.8%	1.0%	**88.1%**	22.2%	54.7%	None	612 (1.5%)
Linux 2.4-2.6	**87.4%**	3.0%	**83.6%**	0.4%	54.3%	40.5%	269 (0.6%)
Linux 2.6.x	**90.1%**	0.9%	**84.1%**	None	63.2%	35.8%	4903 (11.8%)
Linux 3.x	15.3%	0.6%	14.0%	0.1%	11.6%	0.6%	18021 (43.4%)
Windows 7 or 8	5.1%	2.1%	0.3%	0.3%	**88.7%**	0.9%	3877 (9.3%)
Windows XP	7.9%	6.1%	3.0%	1.8%	6.3%	3.5%	838 (2.0%)
Unknown	9.6%	0.8%	12.7%	1.4%	23.9%	3.2%	12543 (30.2%)

Table 6: Vulnerability to blind attacks based on operating system inferred by the p0f tool for the cld-us VP. Most vulnerable connections were Linux-based, and running older kernel releases. Many operating systems handled in-window data incorrectly by accepting the data that could have come from a blind attacker. We excluded operating systems that each accounted for fewer than 0.5% of observed connections (HP-UX, Linux 2.4, MacOS, OpenBSD, and Solaris).

of systems that will accept in-window data from a blind attacker and advertise a receive window of at least 64KB, 23% of tested connections could be disrupted with 2^{17} packets.

4.3.4 Operating systems inferred

Table 6 correlates the behavior inferred for each TCP connection with the OS inferred by the Passive OS Fingerprinting (p0f) tool [39] for data collected by the cld-us VP. The most vulnerable populations were inferred to be running Linux 2.6, with nearly all tested systems (84–90%) vulnerable to blind in-window reset and SYN attacks, and more than half vulnerable to in-window data attacks. The results for FreeBSD correlate with what is publicly known about their vulnerability to SYN packets matching an existing connection [1].

Table 6 only lists operating systems that made up at least 0.5% of the population. Beyond the operating systems in table 6, we also observed: (1) 0.3% of systems p0f classified were HP-UX 11.x, most of which were not vulnerable to blind reset or SYN attacks, but were vulnerable to blind data; (2) 3 Mac OSX systems that were vulnerable to the three blind in-window attacks tested, but not to the out/ahead of window tests; (3) 6 OpenBSD systems that were not vulnerable to any blind attacks.

4.3.5 Summary of findings for webservers

We found that 38.4% of currently deployed TCP stacks that support popular websites were vulnerable to at least one blind in-window attack, with the data attack the least well defended. Most TCP stacks correctly ignored reset and SYN attacks for out-of-window packets, but some incorrectly established a parallel TCP connection (1.1%) and some incorrectly reset the connection (1.0–1.4%). Perhaps most troubling was the 6.2% of tested systems whose connections were vulnerable to a blind data attack with an acknowledgment value ahead of the receiver's send window.

Systems advertising an MSS of 1380 were almost never vulnerable to in-window reset and SYN packets, because they represented TCP connections likely protected by a middlebox that sent challenge ACKs on their behalf. It is likely that the attack surface will reduce as the fraction of systems running older versions of Linux are upgraded. For example, we found approximately 3% fewer hosts vulnerable to in-window SYN and reset attacks in September 2015 than we found in April 2015.

5. INFRASTRUCTURE VULNERABILITY

Blind attacks have the potential to disrupt not just end stations, but also core infrastructure [8, 12]. For example, both BGP-speaking routers and OpenFlow-speaking switches establish TCP connections to exchange routing information. BGP and OpenFlow connections are especially susceptable to brute forcing, as they are long-lived. Long-lived flows permit the attacker to feasibly probe the entire sequence and portions of the tuple space. Furthermore, the four-tuples of control-plane flows may be easier to discover as BGP peers are well-known. An off-path attacker that is able to successfully disrupt a BGP or OpenFlow TCP connection can induce significant harm. When a BGP session with a peer that has advertised a large number of prefixes terminates, significant route recomputation and announcements may occur while the routers involved reconverge.

In recognition of the dangers of blind attacks on infrastructure TCP stacks, several mechanisms have been developed to protect TCP sessions. First, the generalized TTL mechanism (GTSM) [15] only accepts packets with a TTL of 255 (the maximum value), such that any packets not on the local subnetwork will arrive with a lower TTL and will not be accepted. While GTSM works for point-to-point eBGP sessions, it does not generalize. Second, TCP MD5 [16] and TCP-AO [37] options provide cryptographic authentication. Unfortunately, these strong authentication mechanisms require configuration of a shared secret, a manual and complex process that can discourage use. Finally, best common practices [12] dictate filtering of control plane messages via access control lists that admit only traffic from trusted networks. However, attackers able to employ IP source address spoofing [7] can circumvent such filtering.

5.1 Testing infrastructure stacks

Because routers and switches frequently use specialized operating systems, we sought to better understand the behavior of these non-commodity TCP stacks. We chose not to probe BGP or OpenFlow devices in the wild for several reasons. First, we did not want our probing to be perceived as an active attack by operators. In private discussions with a network operator, we were strongly discouraged from attempting to probe BGP routers in the wild. Second, many BGP routers employ source address filtering, yet may still be vulnerable to spoofed-source attacks, which we did not want to mount.

Device	OS	Blind reset		Blind SYN		Blind data		Port
	date	in	out	in	out	behind	ahead	range
Cisco 2610 12.1(13)	2002-01	× (A)	✓ (I)	× (R)	✓ (C)	× (A)	✓ (C)	seq.
Cisco 2610 12.2(7)	2002-01	× (A)	✓ (I)	× (R)	✓ (C)	× (A)	✓ (C)	seq.
Cisco 2650 12.3(15b)	2005-08	✓ (C)	✓ (I)	✓ (C)	✓ (C)	× (A)	✓ (C)	40785
Cisco 7206 12.4(20)	2008-07	✓ (C)	✓ (I)	✓ (C)	✓ (C)	× (A)	✓ (C)	54167
Cisco 2811 15.0(1)	2010-10	✓ (C)	✓ (I)	✓ (C)	✓ (C)	× (A)	✓ (C)	46166
Cisco 2911 15.1(4)	2012-03	✓ (C)	✓ (I)	✓ (C)	✓ (C)	× (A)	✓ (C)	39422
Juniper M7i 8.2R1.7	2007-01	× (A)	✓ (I)	× (R)	✓ (I)	× (A)	✓ (C)	181
Juniper EX9208 14.1R1.10	2014-06	✓ (C)	✓ (I)	✓ (C)	✓ (I)	× (A)	✓ (C)	13769
Juniper MX960 13.3	2015-05	✓ (I)	✓ (I)	✓ (C)	✓ (I)	× (A)	✓ (C)	13033
Juniper J2350 12.1X46-D35.1	2015-05	✓ (I)	✓ (I)	✓ (C)	✓ (I)	× (A)	✓ (C)	12481
HP 2920 WB.15.16.0006	2015-01	✓ (C)	✓ (C)	✓ (C)	✓ (C)	✓ (I)	✓ (I)	14273
HP e3500 K.15.16.0007	2015-06	× (A)	✓ (I)	× (R)	✓ (C)	✓ (I)	✓ (I)	15611
Brocade MLX-4 5.7.0bT177	2014-10	✓ (I)	✓ (I)	✓ (C)	✓ (C)	✓ (C)	✓ (C)	const.
Pica8 Pronto3290 v2.6	2015-05	× (A)	✓ (I)	× (R)	✓ (C)	× (A)	× (A)	HBPS

Table 7: Laboratory testing of blind TCP attacks against BGP-speaking router and OpenFlow-speaking switches. A cross-mark means the TCP connection accepted (A) or reset (R) in response to the packet that it should have rejected. A tick-mark means the TCP connection challenged (C) or ignored (I) the packet. The port range is reported from ten ephemeral port selections made by the device. Over time, implementations have generally become more resilient to blind reset and SYN attacks, but not to blind data attacks.

We therefore used our oracle approach on a variety of routers and switches available to us in a controlled laboratory environment. To perform this testing, we added a basic BGP application protocol [33] to scamper that sends a valid BGP OPEN message with the correct autonomous system number and no BGP options. We ensured that the BGP keepalive time was large enough so that the router under test did not prematurely terminate the TCP connection during testing. Similarly, we implemented the basic OpenFlow application protocol by sending an OpenFlow HELLO message. We further explicitly configured each device we tested to recognize our prober as a peer so that it would establish a BGP or OpenFlow session, allowing the prober to test the underlying TCP behavior. Concurrent to our probing, we captured packets from the device under test so that we could assess the device's choice of ephemeral ports from ten port selections, as we explain in the appendix.

Table 7 shows that while the overall protection mechanisms within Cisco devices improved across subsequent releases of the operating system, even versions released post RFC 5961 exhibit weakness to the blind in-window data attack. In fact, only 2 of 14 devices we tested were not vulnerable to any of the blind attacks tested (the HP 2920 and the Brocade MLX-4 switches). All of the Cisco and Juniper devices we tested accepted the data with an acknowledgment number behind the range recommended by RFC 5961.

Relatively obsolete operating systems (Cisco 12.1, 12.2, Juniper 8.2) used sequential or a small range of ephemeral ports, making these more predictable. While modern Cisco devices used a much wider ephemeral port range, modern Juniper and HP systems drew from a 16k range.

All systems we tested correctly ignored or challenged reset and SYN packets that were out of window, and all but one (the Pica8 switch) correctly rejected the data that used an acknowledgment value ahead of the sender's send window. The Pica8 OpenFlow switch was built on a Linux 2.6.27 kernel and selected ephemeral ports using a hash-based ephemeral port selection (HBPS) algorithm, but was vulnerable to four of the six tests.

5.2 Implications

Most devices we tested correctly ignored SYN and reset packets that could have come from a blind attacker, but few rejected data with a behind-window acknowledgment value. We encourage operators to remain vigilant in deploying available defense mechanisms such as GTSM [15] and TCP authentication options [16, 37] to ensure the security of their infrastructure, as well as ingress filter packets with a source address purporting to be from their infrastructure but arriving from outside of their network [4, 12].

6. PORT SELECTION OBSERVATIONS

Finally, we used passive network traces to infer current ephemeral port selection algorithms in deployed TCP stacks. Our goal was to understand the current degree to which TCP endpoints use an ephemeral port selection algorithm that is predictable, i.e. selecting ports in sequence using a central counter; as discussed in section 2.2, a predictable port selection algorithm can make a blind attacker's job easier. However, gaining an accurate picture of the deployment of port selection algorithms using passive data is difficult for a number of reasons. Figure 7 summarizes the two largest challenges. First, multiple devices may share a single global IP address if they are behind a router that does network address translation. In figure 7, hosts X and Y are selecting ephemeral port values sequentially in the range 1024–5000; X selects 1050 at time T-1, and Y selects 1030 at time T-2. Because these devices are behind a NAT device that rewrites their source IP addresses, the port sequence observed begins 1050, 1030, i.e. goes backwards and therefore although not predictable is not random.

Second, consider a passive tap at the edge of a network. If we include the internal population of the edge network, then we risk observing trends that do not hold in the general population. However, if we use only the external population who establish TCP connections to the internal population, we can only observe an unknown fraction of the TCP connections that each external host establishes. In figure 7, the passive tap does not observe connections from Y using

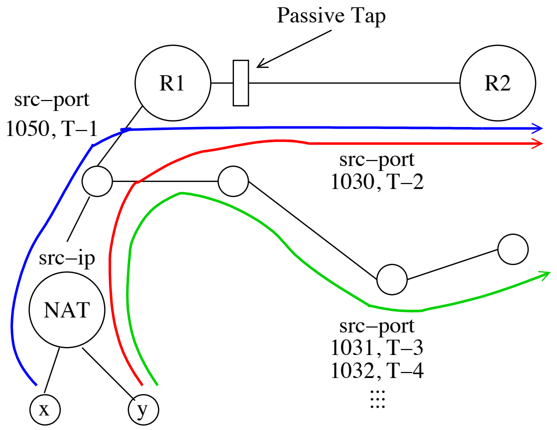


Figure 7: Challenges in using passive data to infer port selection algorithms. A passive tap topologically distant from the sources it measures observes an unknown fraction of the TCP connections established by hosts X and Y. Further, X and Y might use a predictable ephemeral ports assignment scheme, but be behind a NAT that rewrites the source address, which makes the predictable assignments indistinguishable from a hash-based counter.

source ports 1031, 1032, and so on. If the passive tap later observes a connection established by Y, there may be a large gap in the source port values chosen by Y.

Third, Linux hosts use a simple hash-based port selection technique to generate ephemeral ports, so the sequence of ephemeral port values chosen by Linux systems may appear predictable, yet an attacker would have to know the randomly-generated secret the system uses to select ephemeral port values to any given destination. To address this concern, we only considered a source if we observed it establishing connections to multiple destinations.

In this section, we consider two sources of passive data: (1) a passive tap operated by CAIDA on a Tier-1 ISP's backbone link between Chicago and Seattle, which periodically records 1-hour packet header traces, and (2) longitudinal data recorded by a Bro intrusion detection system (IDS) instance at ICSI containing logs of TCP flows since 2005.

6.1 Equinix Chicago Passive Header Trace

CAIDA operates a passive monitor located at Equinix Chicago, which since 2008 has captured a one-hour packet header trace per-month from a Tier-1 ISP's 10Gig Ethernet backbone link between Chicago and Seattle (there was an 18-month outage due to hardware failure from September 2011 to March 2013). From these files, we extracted ephemeral ports with the following algorithm.

For each SYN packet, we created a 4-tuple containing the source (the active opener) and destination IP addresses and ports, as well as a flag that records if the flow carried data. After two minutes had elapsed in the trace, we discarded the flow to allow for 4-tuple reuse. At this point we included the flow for further processing if we observed data transmission and discarded it if no data was observed (i.e. the SYN was a result of a scan or other anomalous behavior). Finally, we grouped the 4-tuples by source IP address, and only considered sources that established at least ten TCP connections.

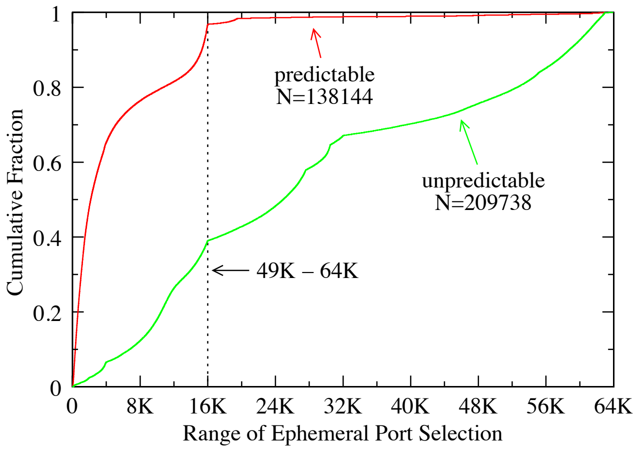


Figure 8: Range of ports observed (max - min) for one hour packet trace collected on 19th March 2015 at 1pm UTC from a Tier-1 ISP link between Chicago and Seattle. We observed that most port selections used a fraction of the available port space, especially the predictable port selections, where 50% of selections were in a range of 2K ports.

For each active opener, we classified the sequence of ports as predictable (derived from a central counter) by consulting a sliding windows of 3 ports at a time. We used windows of 3 ports at a time because these are the shortest window for which we can test for an unpredictable sequence. We deemed a single window to be "predictable" when ephemeral ports were generally increasing – with the possibility of a single wrap per window. For example, if we observed the ports [1, 2, 3], [2, 3, 1], or [3, 1, 2], then we would infer the use of a counter, but if we observed [2, 1, 3], [3, 2, 1] or [1, 3, 2] then we would infer the ports were selected unpredictably. Note that this method would have incorrectly inferred some busy active openers as unpredictable if we only observed a fraction of their SYNs on the Chicago-Seattle link. We also allowed for a small amount of packet reordering and SYN loss: enough to allow for the first retransmission of a SYN which was previously lost before the SYN could have crossed the link where we collected the passive header trace. Specifically, if the difference between two port values was less than 15 (a nominal value which is 0.4% of the smallest ephemeral range in the operating systems listed in table 1) and the difference in time was less than three seconds – then we assumed the port values were generated in order and the out-of-order observation was an artifact introduced by the network. For a source IP address with 10 port observations, we tested 8 windows; we inferred the address as generating a predictable ephemeral port sequence if all windows were inferred to be predictable.

Figure 8 shows the observed range (maximum - minimum) of ephemeral ports for each source that established at least 10 connections over this link over the course of an hour on 19 March 2015 to at least two destinations. As shown in the appendix, the expected port range given a host selecting ephemeral ports at random from the entire 2^{16} space given 10 observations is approximately 53,620. Of the hosts we inferred to select predictable ephemeral ports, 50% established all ports within a 2K port range – 3% of the total range of

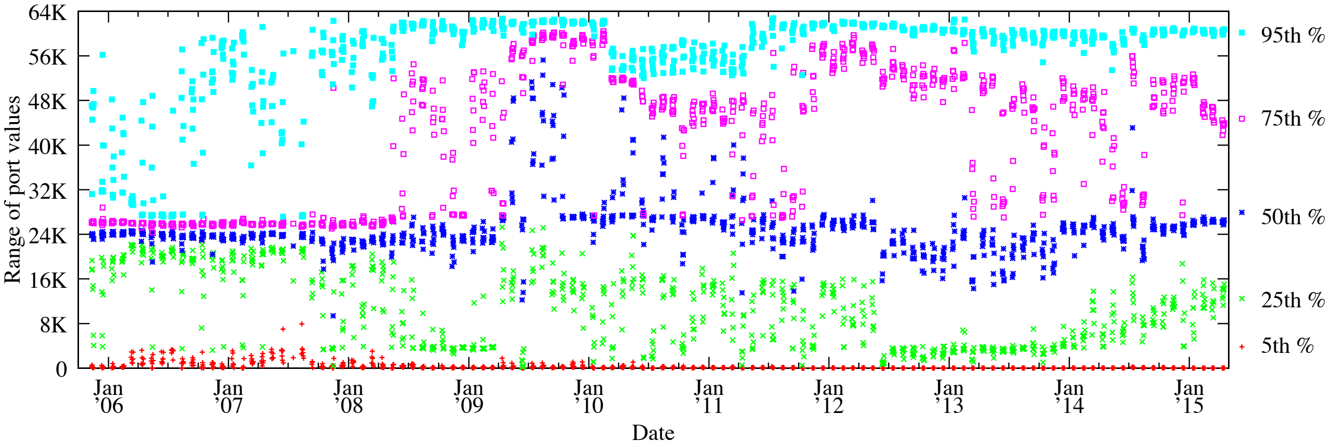


Figure 9: Range of ports observed (max - min) for one day of Bro logs collected one week per month since October 2005 at ICSI. The points represent the 5th, 25th, 50th, 75th, and 95th percentiles for one day's traffic within each week. Beginning January 2014, the range of ports selected for the 25th percentile has begun to widen.

port values potentially available, and 12% of the ephemeral port range recommended by IANA. Of the hosts we inferred to use an unpredictable strategy, a third chose ports from at least a range of 32K ports; a blind attacker would have to use other sophisticated means to infer port selection behavior of these hosts [14]. These results are consistent with the known port selection strategies of modern operating systems shown in table 1. We conclude that some operating system vendors continue to not follow best practices in ephemeral port selection [23].

6.2 Longitudinal ICSI Trace

Finally, we investigated longitudinal patterns in port selection using Bro logs collected at ICSI. We focused on inbound connections from external hosts, from sources that established at least 10 data-carrying TCP connections to internal hosts. Because of the sparsity of the data over the course of a day, rather than try to infer whether or not port selection is predictable or not, we instead focused on the ranges observed to have been used by individual IP addresses. Figure 9 plots the ranges of ports selected from October 2005 until May 2015. While the data is noisy, we can see two encouraging shifts in port selection behavior. First, between January 2006 and January 2008, the 95th percentile range rose from 32K to 62K, suggesting that systems were gradually upgraded to choose ephemeral ports from a larger range. Second, between October 2013 and May 2015, the 25th percentile of port ranges gradually increases from 4K to 12K, suggesting a different set of systems were gradually upgraded to choose ephemeral port ranges from a larger range. We conjecture this trend is due to the well-publicized end-of-life date for Windows XP systems, which chose port values between 1024–5000 (see table 1).

6.3 Implications

Beyond these trends, it is difficult to distill insight on port selection algorithms from either passive data source (CAIDA or ICSI) owing to many conflating factors. However, our experimental data indicates that as of March 2015 that most operating systems are choosing ports from a small fraction of the ephemeral port ranges (figure 8) which is consistent with MacOS and Windows operating systems choosing TCP ephemeral ports sequentially (table 1).

7. DISCUSSION

Holistically, the ability of an attacker to mount a blind attack is more difficult than it was 2004, as there has been some deployment of RFC 5961 into TCP stacks, and operating systems have been upgraded over time. In addition, some operating systems are using automatic TCP buffer tuning [35, 34] to increase the receive window as required, and using small receive windows by default (figure 5). Nevertheless, we found a surprising fraction of webservers acted inappropriately when they received a packet that could have come from a blind attacker. In particular, defenses to blind attackers forging data packets are less deployed than defenses to reset and SYN packets, suggesting further implementation and deployment effort is required. For example, middleboxes that correctly challenged reset and SYN packets did not discard invalid data packets (section 4.3.2). In addition, some popular operating systems (Windows and MacOS) are choosing ephemeral ports from a 16K range in a predictable fashion, allowing a blind attacker to optimize their efforts in relatively narrow port ranges.

8. RELATED WORK

Previous active measurement of TCP has focused on the deployment and behavior of features that improve the performance of TCP, such as algorithms used for congestion control and slow-start [29, 28]. In addition, there has been particular focus on the role that middleboxes play in hampering deployment and use of TCP features (e.g. [27, 25, 5, 17, 11]). Our work focuses on active measurement of TCP features that enhance TCP's resilience to blind attack.

More recently, researchers have shown it is possible for unprivileged malware deployed on a victim's computer to learn information that can help an otherwise blind attacker to focus their efforts [31], or even by visiting a malicious website that uses a script to predict TCP parameters, including source port values in the face of simple hash-based port selection algorithms [14]. While dangerous and similar to the attack we investigated, arguably the attackers in these cases are not completely blind, which aids the attacker. Our work holistically examines the state of TCP vulnerability to blind attackers, in terms of TCP implementation, port selection, and middlebox protections.

Convery and Franz examined the behavior of BGP speaking routers in the wild in 2003 [9]. They encouraged operators to deploy ingress filtering for router address space to make blind attacks impossible from outside of the network, and reported TCP enhancements (choosing ephemeral ports unpredictably, and requiring the reset to exactly match the receiver's position in the sequence number space) made TCP highly resistant to attacks. Our work examined the degree to which TCP enhancements that make blind attacks difficult have been deployed.

9. CONCLUSION

TCP is the most important transport protocol in the Internet, carrying the vast majority of the traffic volume on the Internet, including web browsing, streaming video (e.g., YouTube and Netflix), email, and BGP. Because of its importance, researchers and vendors are constantly developing and deploying features to improve the security and performance of TCP. However, for at least three reasons, little is empirically known about the current state of TCP deployment: TCP features have evolved over time, there are many vendors of operating systems and middleboxes, and there is no dedicated instrumentation or sustained effort to gather longitudinal measurements on the deployed ecosystem.

Complicating the situation is the fact that TCP was implemented with limited security functionality, and although several RFCs have recommended modifications to improve the security defenses of TCP, the adoption, configuration, and deployment of fielded TCP improvements can be slow. In this study we empirically assessed the resilience of *deployed* commodity TCP implementations to blind in-window attacks, where an off-path adversary can disrupt an established connection, causing data corruption or connection reset. We also characterized router and switch behavior – critical infrastructure where the impact of any TCP vulnerabilities is particularly acute. We tested operating systems (and middleboxes deployed in front) of webservers in the wild and found 22% of connections vulnerable to in-window SYN and reset packets, 30% vulnerable to in-window data packets, and 38.4% vulnerable to at least one of the three in-window attacks we tested. This surprisingly high level of extant vulnerabilities we found in the most mature Internet transport protocol in use today is a perfect illustration of the Internet's fragility. Given the largely unregulated and insecure TCP/IP network architecture society relies on for most of our communications, it also provides a strong case for more systematic, scientific, and longitudinal measurement and quantitative analysis of fundamental properties of critical Internet infrastructure, as well as for the importance of better mechanisms to get best security practices deployed.

Acknowledgments

We thank the UCSD systems and networking research group for early feedback. Special thanks to John Gibson, Tom Hutton, Bill Owens, and Brad Cowie for providing operational routers to test against. This work was supported in part by U.S. NSF grants CNS-1111449, ACI-1127506, and CNS-1237265, and by DHS S&T Cyber Security Division BAA 11-02 and SPAWAR Systems Center Pacific via N66001-12-C-0130 and Defence Research and Development Canada (DRDC) pursuant to an Agreement between the U.S. and Canadian governments for Cooperation in Science and Technology for Critical Infrastructure Protection and Border Security. This work represents the position of the authors and not of NSF, DHS, DRDC, or the U.S. government. Matthew Luckie conducted this work while at CAIDA, UC San Diego.

Appendix

In our analysis of ephemeral ports, we restricted our analysis to only those sources that established 10 or more connections and reported on the ephemeral port range. For the set of observed ephemeral ports $S = \{\}$, we compute $range = max(S) - min(S)$. In this appendix, we derive the expected value of *range* given a source that chooses ports from a uniform random distribution over the interval $(0, high)$. (The following trivially extends to a non-zero starting port range; we omit this detail for clarity). Given n flows from a source, the probability that the largest port observed is k is given by:

$$P\left(max(S) = k\right) = \frac{k^n - (k-1)^n}{high^n}$$

while the probability that the smallest port observed is k is:

$$P(min(S) = k) = P(max(S) = high - k)$$

Thus, the expected range is:

$$\begin{aligned} E[range] &= E[max(S)] - E[min(S)] \\ &= \sum_{k=1}^{high} k\Big(P\big(max(S) = k\big) - P\big(min(S) = k\big)\Big) \end{aligned}$$

For $n = 10$, we find (corresponding to ranges of ephemeral port numbers of operating systems in Table 1):

$$\begin{aligned} E[range|high = 2^{16}] &\simeq 53620 \\ E[range|high = 3976] &\simeq 3253 \\ E[range|high = 16384] &\simeq 13404 \\ E[range|high = 28232] &\simeq 23098 \\ E[range|high = 55535] &\simeq 45438 \end{aligned}$$

10. REFERENCES

[1] FreeBSD-SA-14:19.tcp: Denial of service in TCP packet processing. `https://www.freebsd.org/security/advisories/FreeBSD-SA-14:19.tcp.asc`.

[2] Alexa. Top 1,000,000 sites. `http://www.alexa.com/topsites`.

[3] M. Allman. Comments on selecting ephemeral ports. *ACM SIGCOMM Computer Communication Review*, 39(2):14–19, 2009.

[4] F. Baker and P. Savola. Ingress filtering for multihomed networks. RFC 3704, Mar. 2004.

[5] S. Bauer, R. Beverly, and A. Berger. Measuring the state of ECN readiness in servers, clients, and routers. In *IMC*, Nov. 2011.

[6] S. Bellovin. Defending against sequence number attacks. RFC 1948, May 1996.

[7] R. Beverly, A. Berger, Y. Hyun, and k claffy. Understanding the efficacy of deployed Internet source address validation. In *IMC*, pages 356–369, Nov. 2009.

[8] Cisco. TCP Vulnerabilities in Multiple IOS-Based Cisco Products, 2004. `http://tools.cisco.com/security/center/content/CiscoSecurityAdvisory/cisco-sa-20040420-tcp-ios`.

[9] S. Convery and M. Franz. BGP vulnerability testing: separating fact from FUD. In *Blackhat*, 2003.

[10] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire. Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry. RFC 6335, Aug. 2011.

[11] R. Craven, R. Beverly, and M. Allman. A middlebox-cooperative TCP for a non end-to-end Internet. In *SIGCOMM*, pages 151–162, 2014.

[12] J. Durand, I. Pepelnjak, and G. Doering. BGP Operations and Security. RFC 7454 (Best Current Practice), Feb. 2015.

[13] P. Ferguson and D. Senie. Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. RFC 2827, May 2000.

[14] Y. Gilad and A. Herzberg. Off-path TCP injection attacks. *ACM Transactions on Information and System Security*, 16(4), Apr. 2014.

[15] V. Gill, J. Heasley, D. Meyer, and P. Savola. The generalized TTL security mechanism (GTSM). RFC 5082, Oct. 2007.

[16] A. Heffernan. Protection of BGP sessions via the TCP MD5 signature option. RFC 2385, Aug. 1998.

[17] B. Hesmans, F. Duchene, C. Paasch, G. Detal, and O. Bonaventure. Are TCP extensions middlebox-proof? In *HotMiddlebox*, pages 37–42, 2013.

[18] Y. Hyun and k. claffy. Archipelago measurement infrastructure, 2015. `http://www.caida.org/projects/ark/`.

[19] Internet Assigned Numbers Authority (IANA). Service name and transport protocol port number registry. `http://www.iana.org/assignments/port-numbers`.

[20] V. Jacobson, R. Braden, D. Borman, and R. Scheffenegger. TCP Extensions for High Performance. RFC 7323, Sept. 2014.

[21] D. Kaminsky. Black Ops 2008: It's the end of the cache as we know it. *Black Hat USA*, 2008.

[22] S. Kent. IP authentication header. RFC 4302, Dec. 2005.

[23] M. Larsen and F. Gont. Recommendations for transport-protocol port randomization. RFC 6056, Jan. 2011.

[24] M. Luckie. Scamper: a scalable and extensible packet prober for active measurement of the Internet. In *IMC*, pages 239–245, Nov. 2010.

[25] M. Luckie and B. Stasiewicz. Measuring path MTU discovery behaviour. In *IMC*, pages 102–108, Nov. 2010.

[26] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. RFC 2018, Oct. 1996.

[27] A. Medina, M. Allman, and S. Floyd. Measuring interactions between transport protocols and middleboxes. In *IMC*, pages 336–341, Oct. 2004.

[28] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the Internet. *ACM SIGCOMM Computer Communication Review*, 35(2):37–52, 2005.

[29] J. Pahdye and S. Floyd. On inferring TCP behavior. In *SIGCOMM*, pages 287–298, 2001.

[30] J. Postel. Transmission control protocol. RFC 791, Sept. 1981.

[31] Z. Qian and Z. M. Mao. Off-path TCP sequence number inference attack: How firewall middleboxes reduce security. In *IEEE Symposium on Security and Privacy*, pages 347–361, May 2012.

[32] A. Ramaiah, R. Stewart, and M. Dalal. Improving TCP's robustness to blind in-window attacks. RFC 5961, Aug. 2010.

[33] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol 4 (BGP-4). RFC 4271 (Draft Standard), Jan. 2006.

[34] M. Sargent and M. Allman. Performance within a fiber-to-the-home network. *ACM SIGCOMM Computer Communication Review*, 44(3), July 2014.

[35] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. In *SIGCOMM*, pages 315–323, Sept. 1998.

[36] J. Touch. Defending TCP against spoofing attacks. RFC 4953, July 2007.

[37] J. Touch, A. Mankin, and R. Bonica. The TCP authentication option. RFC 5925, June 2010.

[38] P. Watson. Slipping in the window: TCP reset attacks, Apr. 2004.

[39] M. Zalewski. p0f v3 (version 3.08b). `http://lcamtuf.coredump.cx/p0f3/`.

[40] M. Zalewski. Strange attractors and TCP/IP sequence number analysis, 2002.