Resilience of Deployed TCP to Blind Attacks

Matthew Luckie
CAIDA / UC San Diego
mjl@caida.org

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 recommend mechanisms to protect against well-known weaknesses within TCP. Unfortunately, the adoption, configuration, and deployment of fielded 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 and found a quarter of connections vulnerable to in-window SYN and reset packets, 44.0% vulnerable to in-window data packets, and 53.1% vulnerable to at least one of the in-window attacks. We also tested out-of-window packets and found that while few deployed systems were vulnerable to reset and SYN packets, 2.7% of connections accepted in-window data with an invalid acknowledgment number. In addition to evaluating commodity TCP stacks, we found vulnerabilities in all of the routers and switches we characterized – critical infrastructure where the 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 vivid 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; Blind attacks; middleboxes

IMC'15, October 28–30, 2015, Tokyo, Japan.

DOI: http://dx.doi.org/10.1145/XXX.XXXXXXX.

1. INTRODUCTION

Despite the rich history of attacks against TCP [39, 37, 31, 15], and subsequent counter-measures [6, 35, 32, 16], 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 [15] demonstrates new blinded attacks that pollute web caches. As connection speeds increase and TCP receive windows grow, other common long-lived applications such as file transfers, ssh, and rsync may become vulnerable.

We therefore take a multi-faced measurement examination of both commodity operating system TCP stacks as well as the proprietary TCP implementations common to routers and switches. We adopt an oracle-based approach by simulating a variety of blinded attacks. Our simulated exploits send packets (RST, SYN, data, both in and out of window) that could have been sent by a blinded attacker as part of a brute-force attack.

To test commodity TCP stacks as deployed in the wild, we establish connections to websites in the Alexa list [3], and observe 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 test 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 those that support long-lived protocols including ssh and rsync as these systems represent commonly used systems software.

In this population as measured in April and May 2015, we found: (1) 53.1% of systems tested were vulnerable to at least one blind in-window attack; (2) the in-window data attack is the most significant vulnerability remaining, as 22.1% accepted data with inadequate validation of the acknowledgment number, and a further 20.5% reset the connection entirely; (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 in-window data packets, (4) in response to an in-window SYN, 1.2% of hosts established a parallel TCP connection, which is unexpected behavior as a 4-tuple can only support a single TCP connection at any one time.

In addition to these active tests, we further used passive captures of TCP connections to understand deployed ephemeral port selection algorithms. Within a March 2015 trace from an internal 10Gbps link between Chicago and Seattle in the USA belonging to a Tier-1 ISP, we observe 50% of the port selections being made within a range of 2K ports, while 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.

Finally, we investigated the TCP behavior of critical network infrastructure devices (routers and switches) in a laboratory setting to assess their resilience to these same off-path attacks. We tested both BGP routers and OpenFlow switches to packets that could have been sent by a blinded attacker and found that all exhibit some form of weakness.

Our contributions therefore include:

- We make our code publicly available, allowing vendors to test their deployed implementations of TCP for the mitigations recommended by RFC 5961 [32].
- An active measurement survey of the TCP stack behavior of web servers in the wild, suggesting that a significant portion are vulnerable to blind attacks.
- A laboratory survey of routers and switches where we find a variety of current weaknesses.
- An analysis of current ephemeral port selection behavior as observed on a exchange link.

We begin by thoroughly 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 a quarter of connections vulnerable to blind in-window SYN and reset packets, and 44% 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 difficulty of mounting a blind attack in section 7, a comparison with related work in section 8, and conclude in section 9.

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 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 [37]. Not only did operating systems at the time typically choose source ports from within a small range (1024-5000), they were used sequentially. Watson noted two possible solutions. First, a host should choose a 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 [35], published in 2007, thoroughly describes blinded attack vectors in TCP known at that time. RFC 5961 [32], published in 2010, describes attacks that a blinded attacker can conduct in addition to the well-known reset attack, and formally suggests 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 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 be completed in less than one second at 100Mbps.

2.1 Slipping in the window

To maintain TCP state, an operating system's *TCP stack* stores many variables that characterize a TCP connection. Beyond the source and destination IP addresses and port numbers that represent the 4-tuple, both ends of the TCP connection have 32-bit numbers corresponding to the next sequence number value it will use to send new data (*snd.nxt*) and the next in-order sequence number expected by the receiver (*rcv.nxt*), a 32-bit number recording the last sequence number for which we received an acknowledgment (*snd.una*), as well as a 16-bit number the receiver advertises to its peer that 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 [21].

The blind in-window reset attack described by Watson [37] relies on the victim's TCP stack following advice in the document that first defined TCP in 1981, RFC 793 [30], which says that "a reset is valid if its sequence number is in the window." To reset a TCP connection, a blinded attacker only has to try one packet in each possible window for a TCP packet defined by a 4-tuple; with larger windows, the difficulty of brute-forcing the reset attack reduces.

RFC 5961 [32] suggests that a TCP implementation should reset the connection only if the sequence number exactly matches the next expected sequence number from the peer (*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 check if the sender of the reset had actually reset the connection; if

it has, then 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 increases the challenge from 1 in 2^{32} / *rcv.wnd* to 1 in 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 that shared the same 4-tuple.

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 reduces the range to be no less than the current value of the victim's *snd.una* and the maximum receive window (*max.rcv.wnd*) the peer has ever advertised to the victim. In practice, most TCP stacks currently use automatic buffer tuning [34] and increase the receive window when they infer the sender's transmission rate is receive window limited.

Finally, some connections are subject to a trivial blind attack. 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 a current 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], it could tear down all connections with 2^{17} packets (two packets for all 2^{16} 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, 20] be used by all transport protocols – a range of 16K port values. Table 1 summarizes the ephemeral port ranges used by popular operating systems.

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 [37]. Researchers have since studied ephemeral port selection strategies [4] and port randomization is IETF best current practice [23] as of 2011.

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

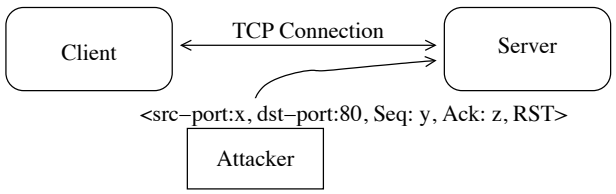
Table 1: Port ranges and sizes used by common operating systems to select ephemeral ports. Port selection strategies differ between systems; Windows and Apple currently assign ephemeral TCP ports sequentially, FreeBSD randomly, and Linux uses hash-based port selection.

In particular, the well-known DNS cache poisoning attack was made much harder by choosing port values at random. However, this fix was made in DNS server software, not at the transport layer, 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.

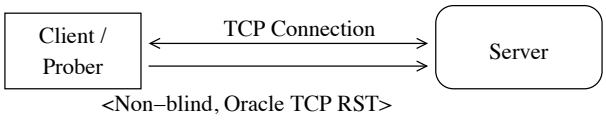
Some protocols, such as BGP, typically operate between adjacent systems. Because they are adjacent, the TTL value contained in the IP header will not be decremented before packets arrive at the peer. The Generalized TTL Security Mechanism [16] relies on this fact; if the peer requires the received IP TTL to be 255 (the largest TTL value possible) then it will discard attack packets from outside the network because their TTL value cannot be 255. This mitigation does not protect Internet applications in the general case.

It is also possible to protect TCP sessions by using an authentication protocol between the hosts, to allow a receiver to validate the packet must have come from its peer and not a third-party attacker. The three most likely authentication options are the TCP MD5 option [17], the TCP authentication option [36], and IP authentication option [22]. In all cases, the peers must establish a shared secret, or deploy certificates, before the TCP connection is established. This approach is feasible for BGP, where sessions are configured between known endpoints, but is not applicable to general transport protocol use between arbitrary endpoints.

Application-layer encryption, such as that provided by TLS, does not prevent an attacker from disrupting a TCP sessions where it is used. A blind in-window reset, SYN, or piece of data will still cause the connection to terminate; the only advantage provided by TLS is that the attacker's accepted data will cause the application-layer encryption software to note the error and abort the TLS connection.



(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.

Finally, blind attacks on Internet protocols are possible because there is inadequate source address validation. RFC 2827 [14] describes best current practices for filtering spoofed packets, however previous work has found filtering is not well deployed in the Internet [7] so blind attacks are unfortunately still possible.

3. ACTIVE MEASUREMENT METHOD

Our 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. We emphasize that at no point did we attempt to disrupt third-party connections. Figure 1 compares our approach to that of a blind attacker. Because we established our own TCP connection and then acted as if we were a blind attacker, we had perfect knowledge of the precise packets to use to determine if the TCP connection was susceptible to packets from a blind attacker.

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. From there, we sent up to three 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 three reset (or SYN) packets to reduce the chance that we will have false negatives caused by the reset (or SYN) being lost before arriving at the server; we sent these packets interspersed with an ACK that acknowledged a new segment of data from the sender to encourage the server to keep the TCP connection alive. If the server sent multiple packets, we use the first to trigger our test, and subsequent packets to send duplicate acknowledgements as if we had lost the first.

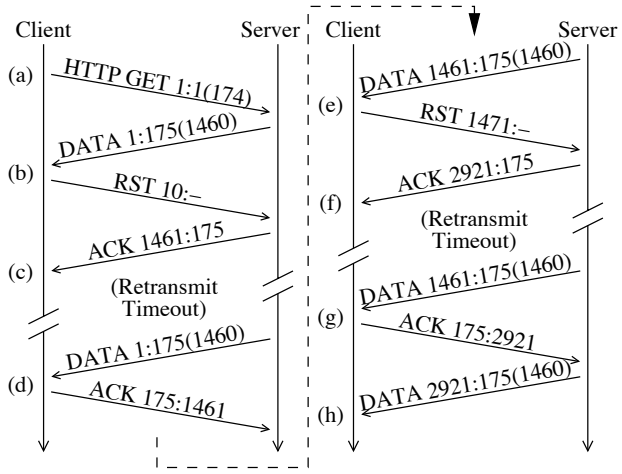


Figure 2: Overview of our blind reset and SYN test. After the TCP (and TLS) handshake, we send an HTTP request (a) to solicit a response for which we send a reset packet (b). The server will send a challenge ACK (c, f) if it follows RFC 5961 with the expected sequence number in the acknowledgment field. When the server retransmits 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.

If we received a reset packet in response to a duplicate acknowledgment we send, 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 the response, 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.

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 retransmitted 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 test if the sequence number even has to be in-window to cause the connection to be reset. We begin a

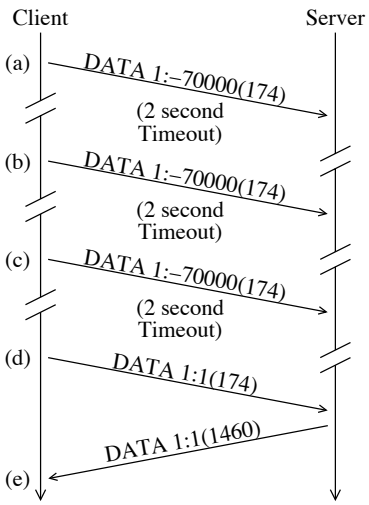


Figure 3: Overview of our blind data test. After the TCP handshake, we send the first data packet (a) with an acknowledgment value before the earliest acceptable value. The data is either the HTTP GET or the first packet in the TLS handshake. If we do not receive an acknowledgment for the data, we try two more times (b, c) with an invalid acknowledgment number, waiting two seconds between tries. We then send the data packet with the expected acknowledgment (d) to allow the server to respond with data (e).

new test by sending packets with a sequence number 70,000 earlier than would be required 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

Figure 3 illustrates our blind data test. Our test began by establishing a TCP connection without the use of any TCP options. We then sent the first data packet for the connection with an acknowledgment number that would place the acknowledgment before the earliest acceptable acknowledgment number per RFC 5961 (see section 2). This first data packet was the GET request for HTTP webservers, and the first packet in the TLS handshake process for HTTPS webservers. We sent the data packet up to three times at an interval of 2 seconds to reduce the chance that we would have false negatives caused by these packets being lost before reaching the server.

If we received an acknowledgment for the injected data then we classified the server as vulnerable because it accepted the data. Note that this definition does not require evidence that the peer passed the injected data to the application for processing; in our tests, we also received data from 99.3% of the servers that acknowledged the injected data, implying that the injected data was passed to the application. If we received a challenge ACK for the injected data then we classified the server as not vulnerable because it has implemented RFC 5961. If we received no response to the injected data then we classified the server as not vulnerable because it ignored the injected packets. If we received a reset in response to an injected packet then we classified the server as vulnerable because it reset the connection in response to the injected packet.

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

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

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

3.3 Fingerprinting test

We complete probing with a test that establishes a TCP connection advertising support for TCP window scaling and timestamps [21], and Selective Acknowledgments [26]. We used this test to infer the behavior of a TCP connection with a peer when we do not send blind packets, as well as to fingerprint the peer's operating system with p0f [38]. 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 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. It is simple to ensure inferences are valid given the packets recorded. 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.

4. TCP STACKS IN WEBSERVERS

4.1 Targets

We derive our targets from the Alexa Top 1,000,000 websites list [3]. For our tests, each vantage point independently derives its list of IP addresses to probe from a random selection of 50,000 entries. We do this so the nearest addresses for each vantage point will be tested if the location of the vantage point influences the addresses supplied by DNS. We test only one IPv4 address per domain, and an IP address with multiple domains attached is only tested once, so we end up testing approximately 44K addresses per VP. We used wget [2] to determine a suitable URL to use in our tests. If

Result	Blind reset		Blind SYN		Blind data	
	in	out	in	out	behind	ahead
Accepted	3.5%	0.4%	-	-	22.2%	2.7%
Reset (ack-blind)	-	-	20.3%	0.0%	21.8%	22.1%
Reset (dup-ack)	22.2%	0.6%	5.9%	1.3%	0.0%	0.1%
Vulnerable	25.8%	1.0%	26.2%	1.3%	44.0%	24.9%
Challenge ACK	67.7%	1.2%	38.4%	61.2%	17.8%	10.8%
Ignored	5.1%	90.5%	31.2%	33.6%	37.3%	63.3%
Not vulnerable	72.8%	91.7%	69.6%	94.8%	55.1%	74.2%
Parallel TCP	-	-	1.1%	1.1%	-	-
Early FIN	0.4%	4.7%	2.0%	2.1%	0.8%	1.0%
No Result	1.0%	2.6%	1.1%	0.7%	-	0.0%
Other	1.4%	7.3%	4.2%	3.9%	0.8%	1.0%
	41,353	41,354	41,351	41,338	41,342	41,339

Table 3: Overview of results for the webserver population testing from cld-us VP. In our data, a quarter of connections were vulnerable to in-window reset and SYN packets, and 44% of connections were vulnerable to an in-window data packet. A further 24.9% of connections were vulnerable to data packets with an acknowledgment ahead of the receiver's window; 2.7% accepted the data without correctly validating the acknowledgment number, and 22.1% reset the connection.

the default page for the domain is at least 25,000 bytes in size, we ask for it when we test for RFC 5961 compliance. Otherwise, we use wget to obtain all objects required to display the default page that are on that webserver, and select the first object that is at least 25,000 bytes in size, or the largest object available.

4.2 Vantage Points

We used three of CAIDA's Archipelago vantage points (VPs) to conduct our measurements [19]. Two of the VPs are located in the United States: cld-us, hosted by CAIDA in San Diego, and sjc2-us, hosted by Hurricane Electric in San Jose. The third, hlz-nz, is hosted by Waikato University in New Zealand. We used three to ensure there was no undetected middlebox in the hosting network that 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. Further, these VPs are running a recent version of FreeBSD (at least version 9) with a modern SSL library that supports TLS Server Name Indication [13] (SNI), and contain the IPFW firewall required by scamper to support its TBIT implementation.

4.3 Results

Table 3 summarizes the results of our testing for the webserver population from cld-us. In our data, 25.8% of the connections were vulnerable to an in-window reset packet; 3.5% went silent after we sent the reset packet, and 22.2% sent a reset in response to duplicate acknowledgments we subsequently sent. Nearly all of the rest of the TCP connections involved were not vulnerable; 67.7% sent a challenge ACK in response to the in-window reset as recommended by RFC 5961, and 5.1% simply ignored the packets entirely. We observed 26.2% of the connections being vulnerable to an in-window SYN packet; 20.3% sent a reset that acknowledged the in-window sequence number, and 5.9% sent a reset after we sent a duplicate acknowledgement. A further 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 appears to terminate TCP connections on their systems, and relay data from their customer's webserver obtained over a separate TCP connection.

	cld-us	sjc2-us	hlz-nz
Blind reset (in):			
Vulnerable	25.8%	26.2%	26.2%
Not Vulnerable	72.8%	72.4%	72.0%
Other	1.4%	1.4%	1.8%
Blind SYN (in):			
Vulnerable	26.2%	26.7%	0.3%
Not Vulnerable	69.6%	69.8%	93.0%
Other	4.2%	3.6%	6.8%
Blind data (behind):			
Vulnerable	44.0%	44.3%	44.1%
Not Vulnerable	55.1%	54.8%	55.0%
Other	0.8%	1.0%	0.9%

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

For the hosts that were vulnerable to an in-window data packet, half reset the TCP connection, while the other half accepted the packet. This result suggests there is significant work left to do, as the data attack is only twice as difficult to carry out than the other blind in-window attacks we tested. In particular, the data was accepted in 22.2% of the TCP connections, and a further 21.8% reset the connection in response to the data. 17.8% of the servers sent a challenge ACK to the in-window data packet, even though RFC 5961 does not call for this behavior.

Table 4 shows the overall results collected from all the VPs that we used to test webservers. Quantitatively, the results are very similar, with overall classifications from each VP being within 1% of each other for most classifications, despite each VP testing a different set of random webservers. The main difference, however, is that the results of the blind in-window SYN tests are impacted by a stateful on-campus

Server MSS	Vulnerable Portion		
	Blind reset	Blind SYN	Blind data
1460 (87.1%)	28.0%	28.9%	43.3%
1380 (5.7%)	1.9%	0.4%	50.1%
8961 (1.9%)	3.5%	3.7%	31.3%
536 (1.2%)	1.8%	0.6%	7.2%
1440 (0.7%)	5.6%	6.7%	65.8%

Table 5: Top five server MSS values advertised and the corresponding portion that were vulnerable to in-window reset, SYN, and data packets. TCP connections with a server that advertised an MSS of 1380 (likely Cisco ASA devices) 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.

router at hlz-nz (Waikato), which silently discards outgoing SYN packets belonging to an existing connection. Because our results for each VP are quantitatively very similar, we believe the behaviors measured occur mostly towards the webserver end of the TCP connection.

4.3.1 Out-of-window tests

Table 3 also contains the results of our out-of-window tests. In our data, we found that 1.0% of systems tested incorrectly processed the reset and 1.3% 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 expectation a successful attack requires 2^{15} packets.

More problematic is the in-window data attack, where 24.9% showed a vulnerability to packets containing an acknowledgement value ahead of the peer's send window that it could not have sent. 2.7% of connections acknowledged the data we sent and sent a response anyway. A further 22.1% of the connections were reset, likely because they followed advice in the original TCP specification [30] to reset a connection should such a packet should be received; if the attacker's goal is simply to reset an arbitrary connection, then it 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 (74.2%) systems simply ignored the data ahead-of-window acknowledgment value, and 10.8% sent an acknowledgement before discarding the packet.

4.3.2 Middlebox behavior

In our previous work examining the behavior of Path MTU discovery [25] (PMTUD) we found that servers that advertised a MSS of 1380 bytes were much more likely to fail PMTUD, suggesting a middlebox was interfering with PMTUD. Table 5 shows the five server MSS values most frequently observed in our data and their failure rate. Because most (87.1%) of the TCP connections involved an MSS of 1460, the failure rate of these systems is within 2% of that observed for all connections. However, there are several patterns observable. First, systems with an MSS of 1380 are almost never vulnerable to blind in-window reset (1.9%) and SYN (0.4%) packets, but are vulnerable to in-window data

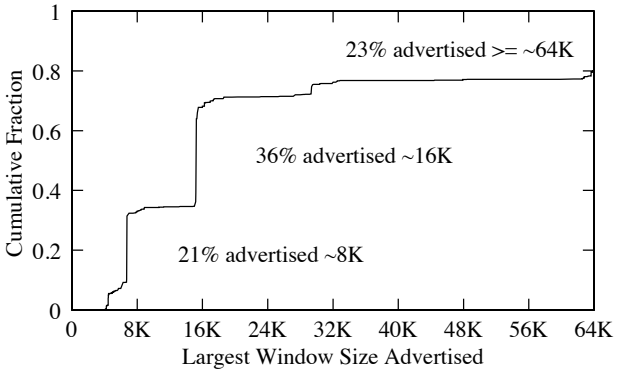


Figure 5: Largest window sizes observed for systems vulnerable to any blind in-window attack. 23% advertise a window of at least 64K, so an attacker requires a maximum of 2^{16} packets to disrupt a corresponding 4-tuple.

packets (50.1%). We hypothesize that this population corresponds to webservers behind a Cisco ASA security appliance, which protects TCP connections from being torn down by blind reset and SYN packets. Systems with an MSS of 8961 were observed to be hosted by Amazon, which likely has a middlebox rewriting the server MSS value. We do not believe this middlebox is protecting the TCP connections involved to Amazon, rather the homogeneous nature of operating systems images available for customers is responsible for the low rate of in-window reset and SYN vulnerability.

87.3% of TCP connections with an MSS of 1380 sent a challenge ACK in response to data with an acknowledgement ahead of the receiver's window, representing 46.4% of all connections that did so yet only 5.7% of the population; only 5.8% of connections with an MSS of 1460 behaved this way. Similarly, 96–97% of TCP connections with an MSS of 1380 were observed 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 90% 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 we received in TCP connections 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 other MSS values in table 5. We conjecture that most connections with MSS values of 536 and 1440 bytes represent connections with reverse proxy systems that terminate TCP connections with outside hosts, and coordinate connections with internal webservers.

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 4 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

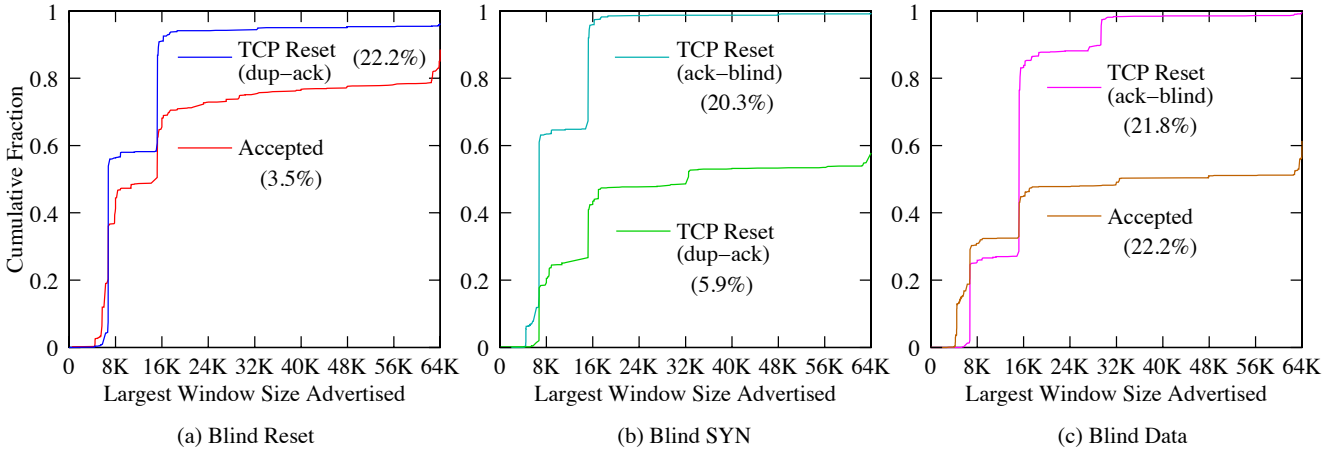


Figure 4: Largest window sizes observed for systems vulnerable to blind in-window attacks and their failure modes. A larger receive window makes it easier for an attacker to disrupt a connection. The small (8K) receive window we observed for most systems vulnerable to blind reset and SYN attacks makes it difficult to attack those connections (2^{19} packets), but half of the systems vulnerable to accepting in-window data advertised a larger window (at least 64K).

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 acknowledgement 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 5 shows the landscape for systems vulnerable to any of the blind attacks tested. Because of the prevalence of systems that will accept in-window data from a blind attacker and advertise a receive window of at least 64KB, 23% of connections to tested systems 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 behavior inferred by the Passive OS Fingerprinting (p0f) tool. The largest, most vulnerable populations were inferred to be running Linux 2.6, with nearly all tested systems (84–89%) vulnerable to blind in-window reset and SYN attacks, and 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) 0.2% of systems we tested 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 53.1% 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%). Systems advertising an MSS of 1380 were almost never vulnerable to in-window reset and SYN packets, because they represented TCP connections protected by a middlebox that sent challenge ACKs on their behalf.

5. TCP STACKS IN INFRASTRUCTURE

It is well-known that 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 amenable to brute forcing, as they are long-lived. Long-lived flows permit the attacker to feasibly probe the entire sequence and portions 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 router attempts to 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) [16] 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 and TCP-AO [17, 36] provide cryptographic authentication via signatures carried in TCP options. Unfortunately, these

Operating System	Blind reset		Blind SYN		Blind data		Total
	in	out	in	out	behind	ahead	
FreeBSD 8.x	21.8%	0.5%	**92.1%**	53.2%	**81.9%**	0.9%	216 (0.5%)
FreeBSD 9.x	19.1%	1.7%	**87.8%**	31.3%	61.2%	0.3%	598 (1.4%)
Linux 2.4-2.6	**84.0%**	1.6%	**81.8%**	0.6%	48.3%	42.9%	319 (0.8%)
Linux 2.6.x	**89.6%**	1.0%	**84.1%**	None	57.2%	49.7%	5772 (13.8%)
Linux 3.x	18.5%	0.6%	17.5%	0.1%	35.6%	29.3%	18286 (43.8%)
Windows 7 or 8	4.9%	1.4%	0.3%	0.2%	**92.9%**	1.6%	3729 (8.9%)
Windows XP	7.3%	5.0%	2.1%	0.8%	6.3%	3.3%	1135 (2.7%)
Unknown	11.1%	0.9%	14.7%	1.8%	34.2%	14.6%	11483 (27.5%)

Table 6: Vulnerability to blind attacks based on operating system inferred by the p0f tool. Most vulnerable connections were Linux-based, and running older kernel releases. Nearly all operating systems handled in-window data poorly. We excluded operating systems that each accounted for less than 0.5% of the connections (HP-UX, Linux 2.4, MacOS, OpenBSD, and Solaris).

strong authentication mechanisms require a shared secret to be configured, a manual, complex, and error-prone 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.

Convery and Franz informally examined the behavior of BGP speaking routers in the wild, however their study is more than a decade old and pre-dates current best practices [9].

5.1 Testing infrastructure stacks

Because router and switch infrastructure frequently uses proprietary 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 our 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.

We therefore performed testing of 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 ensure that that the BGP keepalive time is sufficiently large such that the router under test does not prematurely terminate the TCP connection during testing. Similarly, we implement the basic OpenFlow application protocol by sending an OpenFlow HELLO message. We further explicitly configure the device under test to recognize our prober as a peer such that the BGP or OpenFlow session is established, allowing the prober to test the underlying TCP behavior.

Concurrent to our probing, we capture packets from the device under test that is attempting to connect to our probing host, i.e., TCP SYN packets that reveal the device's choice of ephemeral ports.

Table 7 shows that while the overall protection mechanisms within Cisco devices improved across subsequent releases of the operating system, even versions released post RFC5961 exhibit weaknesses. Relatively obsolete operating systems (Cisco 12.1, 12.2, Juniper 8.2) use sequential or a small range of ephemeral ports, making these more predictable. While modern Cisco systems use a much wider range of port, the modern HP OpenFlow switch drew from only a 16k range.

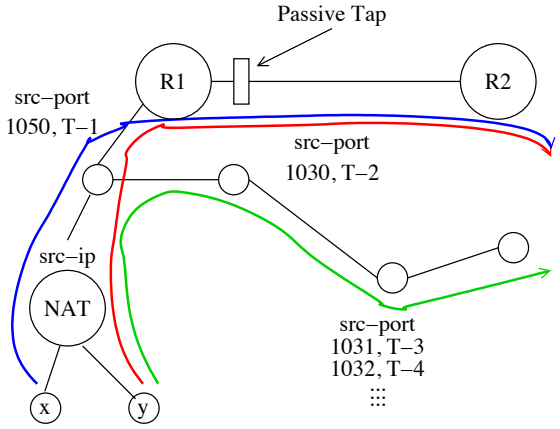


Figure 6: Challenges in using passive data to infer port selection algorithms. A passive tap topologically distant from the sources it measures observes an unknown fraction of the TCP connections established by hosts X and Y. Further, X and Y might use a predictable ephemeral ports assignment scheme, but be behind a NAT that rewrites the source address that makes the predictable assignments indistinguishable from a hash-based counter.

Similarly, old Cisco and Juniper systems accepted the blind RST that was ahead, but in-window, while modern systems send the challenge ACK. However, the behavior of blind RSTs out of window was more varied; the systems that sent a TCP FIN came from the keepalive timing out, implying that these systems correctly ignored the out-of-window reset.

Most significantly, all systems accepted the blind data that used an ACK that was behind the current window. Further, the modern HP switch accepted blind data whether the ACK was behind or ahead of the window.

6. PORT SELECTION OBSERVATIONS

Device	OS	OS date	Blind reset in	Blind reset out	Blind SYN in	Blind SYN out	Blind data behind	Blind data ahead	Port range
Cisco 2610	12.1(13)	2002-01	accepted	ignored	tcp-rst	challenge	accepted	challenge	seq.
Cisco 2610	12.2(7)	2002-01	accepted	ignored	tcp-rst	challenge	accepted	challenge	seq.
Cisco 2650	12.3(15b)	2005-08	challenge	challenge	challenge	challenge	accepted	challenge	40785
Cisco 7206	12.4(20)	2008-07	challenge	challenge	challenge	challenge	accepted	challenge	54167
Cisco 2811	15.0(1)	2010-10	challenge	challenge	challenge	challenge	accepted	challenge	46166
Cisco 2911	15.1(4)	2012-03	challenge	challenge	challenge	challenge	accepted	challenge	39422
Juniper M7i	8.2R1.7	2007-01	accepted	ignored	tcp-rst	ignored	accepted	challenge	181
Juniper EX9208	14.1R1.10	2014-06	challenge	ignored	challenge	ignored	accepted	challenge	13769
Juniper MX960	13.3	2015-05	ignored	ignored	challenge	ignored	accepted	challenge	13033
HP 2920	WB.15.16.0006	2015-01	challenge	challenge	challenge	challenge	accepted	accepted	14273

Table 7: Laboratory testing of blind TCP attacks against BGP speaking router and Openflow speaking switches. Over time, implementations have generally become more resilient to blind reset and SYN attacks, but not to blind data attacks.

Finally, we turn to observations of current ephemeral port selection algorithms in deployed TCP stacks using passive network traces. We attempt to understand the current degree to which TCP endpoints are using an ephemeral port selection algorithm that is predictable, i.e. selecting ports in sequence using a central counter. Gaining an accurate picture of the deployment of port selection algorithms using passive data is difficult for a number of reasons. Figure 6 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 6, 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 unpredictably.

Second, consider a passive tap at the edge of a network, of which there are many available and collecting data useful for researcher use. 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 6, the passive tap does not observe connections from Y using source ports 1031, 1032, and so on. If the passive tap later observes a connection established by Y, there may be an 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 system's randomly-generated secret it uses to select ephemeral port values to any given destination. To address this concern, we only consider a source if we observe 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 established 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 captures a one hour packet header trace most months beginning 2008 from a Tier-1 ISP 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 the case where the same 4-tuple is reused, and printed out the 4-tuple if data was observed for the flow (i.e. was not a result of a scan or other anomalous behavior). Then, we grouped the 4-tuples by source IP address, and only considered sources that established at least ten TCP connections.

For each active opener, we classified the sequence of ports as predictable by consulting a sliding window of 3 ports at a time; a single window of 3 ports is classified as predictable so long as there is only one wrap possible. 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 incorrectly infers some busy active openers as unpredictable if we only observed a fraction of their SYNs on the Chicago-Seattle link. We allow for a small amount of reordering; if the difference between two port values was less than 15, and the difference in time was less than three seconds (enough to allow for a retransmission of the SYN) then we assumed the port values were generated in order and the out-of-order observation was an artifact introduced by the network.

Figure 7 shows the observed range of ephemeral ports for each source that established at least 10 connections over the course of an hour on 19th March 2015 to at least two destinations. As shown in Appendix 9, 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. We find that, 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 port values potentially available, and 12% of the port range recommended by IANA for selecting ephemeral ports from. Of the unpre-

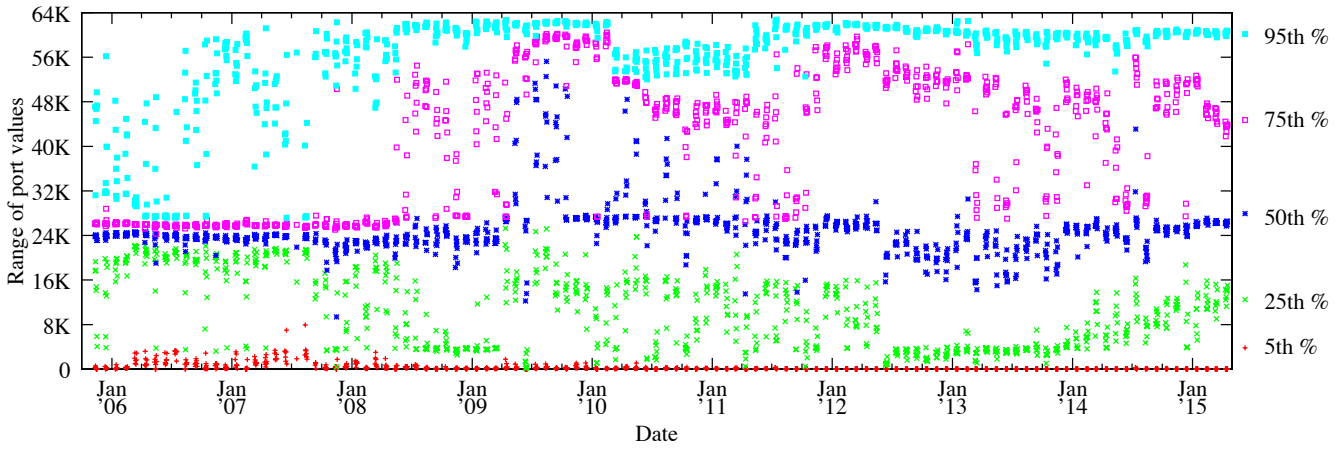


Figure 8: Range of ports observed (max - min) for one day of Bro logs collected one week per month since October 2005 at ICSI. The points represent the 5th, 25th, 50th, 75th, and 95th percentiles for one day's traffic within each week. Beginning January 2014, the range of ports selected for the 25th percentile has begun to widen.

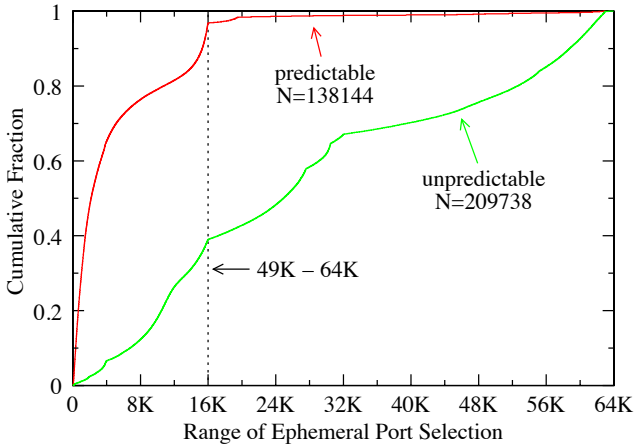


Figure 7: Range of ports observed (max - min) for one hour packet trace collected on 19th March 2015 at 1pm UTC from a Tier-1 ISP link between Chicago and Seattle. Most port selections were made using a fraction of the available port space, especially the predictable port selections, where 50% of the selections were made in a range of 2K ports.

dictable set, 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 of these hosts [15]. These results are consistent with the known port selection strategies of modern operating systems shown in table 1. We are concerned that some operating system vendors continue to not follow best practices in ephemeral port selection [23].

6.2 Longitudinal ICSI Trace

Finally, we attempt to investigate 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 TCP connections systems to internal hosts that carried data. 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 focus on the ranges observed to have been used by individual IP addresses. Figure 8 plots the ranges of ports selected since 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 some 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. We conjecture this is due to the well-publicized end-of-life date for Windows XP systems, which chose port values between 1024–5000, as listed in table 1.

Beyond these trends, it is difficult to distill insight on port selection algorithms from either passive data source (CAIDA or ICSI) owing to the many conflating factors present in this problem. However, we have evidence that most operating systems are choosing ports from a small fraction of the ephemeral port ranges (figure 7) 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, most operating systems are using automatic TCP buffer tuning [34] to increase the receive window as required, and using small receive windows by default (figure 4). 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 much 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, 18, 11]. Our work focuses on active measurement of TCP features that enhance TCP's resilience to blind attack.

More recently, authors have shown how it is possible for unpriviledged 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 [15]. Our work holistically examines (TCP software, port selection, and middlebox protections) the state of deployed TCP to blind attackers.

9. CONCLUSION

TCP is the most important transport protocol in the Internet, carrying nearly all traffic on the Internet, including web (including streaming video such as 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 a quarter of connections vulnerable to in-window SYN and reset packets, almost half vulnerable to in-window data packets, and more than half vulnerable to at least one of the in-window attacks. This surprisingly high level of extant vulnerabilities we found in the most mature Internet transport protocol in use today is a vivid illustration of the Internet's fragility. Given the largely unregulated and unsecure 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 Randy Bush and Stefan Savage 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 and ACI-1127506, and by DHS S&T Cyber Security Division BAA 11-02 and SPAWAR Systems Center Pacific via N66001-12-C-0130, by 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.

Appendix

In our analysis of ephemeral ports, we restrict our analysis to only those sources that establish 10 or more connections and report 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] GNU Wget. https://www.gnu.org/s/wget/.

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

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

[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 *Proceedings of ACM 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] D. Eastlake. Trasport layer security (TLS) extensions: Extension definitions. RFC 6066, Jan. 2011.

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

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

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

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

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

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

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

[21] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. RFC 1323, May 1992.

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

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

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

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

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

[39] M. Zalewski. Strange attractors and tcp/ip sequence number analysis, 2002.