Investigation of the 2016 Linux TCP Stack Vulnerability at Scale

Alan Quach†, Zhongjie Wang†, Zhiyun Qian
University of California, Riverside
{aquac005,zwang048}@ucr.edu, zhiyunq@cs.ucr.edu

ABSTRACT

To combat blind in-window attacks against TCP, changes proposed in RFC 5961 have been implemented by Linux since late 2012. While successfully eliminating the old vulnerabilities, the new TCP implementation was reported in August 2016 to have introduced a subtle yet serious security flaw. Assigned CVE-2016-5696, the flaw exploits the challenge ACK rate limiting feature that could allow an off-path attacker to infer the presence/absence of a TCP connection between two arbitrary hosts, terminate such a connection, and even inject payload into an unsecured TCP connection.

In this work, we perform a comprehensive measurement of the impact of the new vulnerability. This includes (1) tracking the vulnerable Internet servers, (2) monitoring the patch behavior over time, (3) picturing the overall security status of TCP stacks at scale. Towards this goal, we design a scalable measurement methodology to scan the Alexa top 1 million websites for almost 6 months. We also present how notifications impact the patching behavior, and compare the result with the Heartbleed and the Debian PRNG vulnerability. The measurement represents a valuable data point in understanding how Internet servers react to serious security flaws in the operating system kernel.

1. INTRODUCTION

In July 2016, researchers reported a serious vulnerability in Linux TCP implementations that subject all TCP connections to off-path/blind attacks [11], which raised significant awareness [8, 17, 32]. This TCP flaw, which we call the "challenge ACK vulnerability" [19], is particularly dangerous not only because TCP is one of the most widely used protocols, but also because it is completely remotely exploitable. The fact that Linux servers are dominating the server market makes matters worse. Simply put, the vulnerability allows a blind off-path attacker to infer if any two arbitrary hosts on the Internet are communicating using a TCP connection. Further, if the connection is present, such an off-path attacker can also infer the TCP sequence numbers in use, from both sides of the connection; this in turn allows the attacker to cause connection termination and perform data injection attacks. The last time a TCP flaw as serious as this dates back to 1985 which was discovered by Morris [28].

Despite being deployed since the inception of the Internet, TCP has been evolving steadily over the years to counteract various types of attacks [28, 29, 36, 38]. Interestingly and ironically, the recent TCP vulnerability was introduced as a defense against blind in-window attacks [36]. In particular, RFC 5961 [14] proposed in 2010 introduced the notion of challenge ACKs and how they should be rate limited. Since late 2012, Linux has fully implemented RFC 5961 and is the only operating system that is "fully compliant" to the standard. Unfortunately, Linux's rate limit on the challenge ACKs was a global one shared by all connections, effectively allowing an attacker to deduce information about a target connection by creating congestion on the *shared* challenge ACK counter and then measuring the changes by probing packets [19].

In this work, we study the impact of the vulnerability in the wild and the patching behaviors. Different from a user-space application or library vulnerability that is relatively easy to patch, *e.g.,* OpenSSL and Heartbleed [20], our measurement presents a unique data point on kernel vulnerability patching, which involves additional steps such as waiting for Linux distributions to backport the changes from upstream Linux.

Due to the nature of the vulnerability and the applied patches, we are able to clearly differentiate the servers that are vulnerable, non-vulnerable (those did not implement RFC 5961), or patched, allowing us to picture the community's reaction to this security event. Also, to facilitate large-scale scans, we present a highly efficient parallel scanning methodology that operates on a fixed period to accommodate the strict timing requirement.

Through extensive scanning on top Alexa 1 million websites (primarily Linux servers) on a daily basis for almost 6 months, we can picture a detailed and fine-grained patching behavior at scale. We estimate about half of the Alexa top 1 million websites were initially vulnerable, We find that only 19% of the IPs for the

†Both authors contributed equally.

Alexa top 100 and 41% of top 10,000 websites are patched 9 days after the vulnerability went fully public. Interestingly the top 100 websites eventually caught up with a higher patch rate (over two months after the disclosure). We also examine the various hosting services that are behind the websites and show a surprisingly diverse range of patching behaviors. For instance, some CDN providers (*e.g.,* CloudFlare) hosting the websites we studied have a perfect 100% patch rate from the first day of our measurement; while some other providers (*e.g.,* Amazon CloudFront) never patched their servers even 6 months after disclosure.

Finally, we survey the impact of the TCP vulnerability on other services, including Tor and telnet servers, and conclude with an vulnerability notification study.

Drawing upon the observations, we map out a comprehensive picture of the TCP stack vulnerabilities. By better understanding how Linux servers react to kernel vulnerabilities, we hope to shed light on what can be improved in the future in reacting to such Internet-wide security events.

2. BACKGROUND

A TCP connection is identified by a four-tuple: <source IP, source port, destination IP, destination port>. In addition, sequence numbers and acknowledgment numbers (32 bits each) are key TCP states. As a blind attacker who is not able to eavesdrop on the communication (off-path), it is necessary to guess or infer the four-tuple as well as the sequence/acknowledgment numbers to be able to launch any attacks against the connection. Once the 6 key states are known, an attacker can inject any malicious traffic to either the client or server by spoofing the IP of the server or client. The threat model is illustrated in Fig. 1. To resist simple attacks that attempt to predict these values (*i.e.,* source port and initial sequence number), modern TCP standards already produce randomized values [13, 15]. Unfortunately, it is proven that this is not sufficient against persistent attackers [19, 36].

2.1 Blind in-window attacks

A blind in-window attack is a blind TCP packet spoofing attack where an off-path attacker targets a particular client and server pair (running known services) to cause disruption on their ongoing connection [36]. In particular, RFC 5961 outlines three such types of blind in-window attacks:

- Spoofed RST, attempting to forcefully terminate a target connection.
- Spoofed SYN, attempting to fool the server into believing that the client restarted and thus close the current connection.
- Spoofed Data, attempting to corrupt data on either end of the transmission.

Before RFC 5961, all such spoofed packets will be accepted by the client and server as long as the packets satisfy the following criteria: (1) comes with the correct four-tuple; (2) has a sequence number that falls in the receive window. To target a connection between a particular client and server running known services (known destination port), an attacker only needs to guess the source port (or ephemeral port) and the sequence number. The source port is only 16-bit and not the entire range is utilized by default [19]. Even though the sequence number is 32-bit, it is only necessary to send one packet per receive window to exhaust the entire space (and ensure that at least one packet has an in-window sequence number). An attacker with sufficient network bandwidth can therefore bruteforce both the source port and the sequence number and perform the above blind in-window attacks [36].

Interestingly, in addition to end hosts being vulnerable to blind in-window attacks, TCP-aware middleboxes can introduce the same vulnerability as well. A stateful middlebox, such as the NAT of a firewall, may terminate a connection upon seeing an in-window RST, SYN, or even FIN packet [29].

2.2 Off-path attacks utilizing challenge ACK rate limit as a side-channel

RFC 5961 was proposed to set much more stringent rules on when incoming packets are considered valid. In particular, the high-level philosophy is that instead of blindly trusting an incoming packet, when in doubt, a challenge ACK packet can be sent to confirm its validity. Unfortunately, part of this proposal was demonstrated to result in a side-channel vulnerability in August 2016 [19]. We outline the key changes in RFC 5961 below. In addition to a matching four-tuple, the following changes are made for dealing with different incoming packets:

- Incoming RST - If the sequence number is outside the valid receive window, the receiver simply drops the packet (same as before). Only if the sequence number *exactly* matches the next expected sequence number (`RCV.NXT`), is the connection reset. If the sequence number is in-window but does not exactly match `RCV.NXT`, the receiver must send a challenge ACK to the sender, and drop the RST packet.
- Incoming SYN - Regardless of the sequence number, a challenge ACK is sent back to the sender to confirm the loss of the previous connection.
- Incoming Data - If the sequence number is in window, the ACK number of the incoming packet needs to be within a much smaller range: [`SND.UNA-MAX.SND.WND, SND.NXT`]. If the ACK number is smaller than the lower bound: [`SND.UNA-(`2^{31}`-1),`

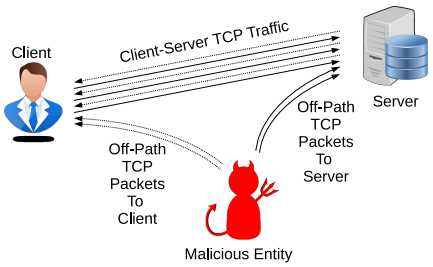

Figure 1: Threat model of an off-path attacker.

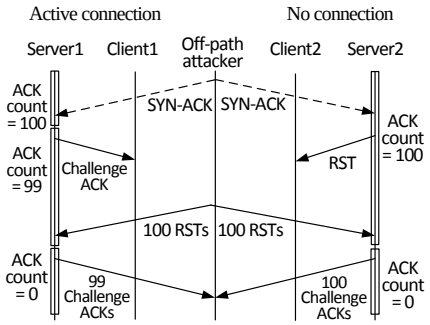

Figure 2: Connection (four-tuple) inference.

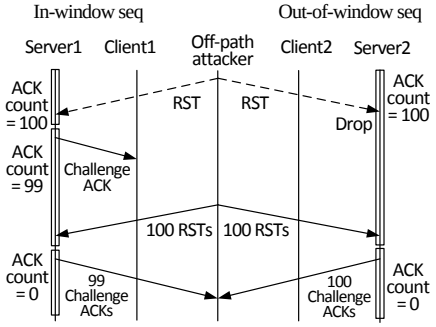

Figure 3: Sequence number inference.

`SND.UNA-MAX.SND.WND`], it is considered too old and a challenge ACK is sent back to the sender.

Besides the way packets are handled, a new subtle change is proposed in RFC 5961: in order to reduce the number of challenge ACK packets that waste CPU and bandwidth resources, an ACK throttling mechanism is introduced (only on the challenge ACK packets). Specifically, a counter is introduced to limit the maximum number of challenge ACKs (by default 100 on Linux) that can be sent out in a given interval (1 second on Linux). This counter is shared across *all* TCP connections, leading to a subtle side-channel as demonstrated in the recent research paper [19].

The key observation is that an attacker can establish a regular TCP connection to measure the remaining challenge ACK counter. This gives an attacker a reliable feedback channel on whether a spoofed packet has managed to trigger a challenge ACK. This side-channel allows an attacker to conduct three attacks in sequence: (1) Connection (four-tuple) inference; (2) Sequence number inference; (3) ACK number inference. In effect, instead of guessing all these unknowns simultaneously, this new attack can "divide and conquer" them and substantially reduce the difficulty of the guesswork.

We illustrate the idea of the first attack, connection (four-tuple) inference, in Fig. 2. The figure illustrates the sequence of packets that an off-path attacker can send to infer the presence or absence of an ongoing connection between the client and server. The first SYN-ACK packet, spoofing the source IP of the client, is represented by dashed line. The counter tracking the number of challenge ACKs that can be issued (100 initially) is shown on the timeline of the server. If the spoofed SYN-ACK hits the four-tuple of an ongoing connection between the client and server (corresponding to the left half of the figure), it will trigger a challenge ACK from the server (according to RFC 5961), and reduce the global rate limit counter from 100 to 99. Otherwise, the counter stays at 100. Next, the off-path attacker simply sends 100 non-spoofed RST packets to exhaust and measure the remaining challenge ACK counter (again leveraging the new behaviors of RFC 5961). As shown in Fig. 2, the attacker can successfully differentiate the "active connection" versus "no connection" cases by counting the number of observed challenge ACKs.

Once the four-tuple is known, an attacker can continue to launch the sequence number inference attack. A similar process is outlined in Fig. 3. Finally, an ACK number inference attack can be launched once the sequence number is known. We omit the details and refer the readers to the paper describing the attack [19].

2.3 Defenses to off-path attacks and timeline

As soon as the challenge ACK vulnerability is reported, defenses are immediately recommended. We list them below:

- Temporary fix [18] - Without upgrading the kernel, one can simply change the configurable rate limit from 100 to an extremely large value, disallowing the attacker to exhaust the limit in one second, making the attack practically impossible to launch.
- Randomize the global rate limit (version-1 patch) [11] - An initial patch to the Linux kernel is to randomize the limit every 1 second (from 500 to 1500 by default), making the attack much harder to execute.
- Enforcing a per-connection rate limit (version-2 patch) [9] - Later versions of Linux reflect the decision to completely eliminate the global rate limit and instead only enforce the rate limit per connection. The patch allows only 1 challenge ACKs by default per 1/2 second per connection.

We present the timeline of events related to the vulnerability and the patch process in Table 1. The vulnerability was originally reported to Linux security by researchers at UCR on July 5th. The issue received immediate attention and was publicly discussed on July 8th on the kernel netdev mailing list [10]. The researchers together with kernel developers iterated on an initial patch that was committed within five days (July 10th) [11]. Another patch was proposed five days later to further improve the security [9].

At this point, most of the technical details are al-

Date	Event
07/05	Researchers at UCR reports the vulnerability
07/08	The issue is publicly discussed on kernel mailing list
07/10	Linux fixes the problem with an initial patch
07/14	Linux fixes the problem with a newer patch
08/10	The vulnerability along with the temp fix goes public
08/10	Major vendors e.g., Akamai applied temporary fix
08/18	Red Hat releases patch for v7
08/19	We begin our scanning

Table 1: Timeline of events since July 2016 for the new side-channel vulnerability.

ready made publicly available, and certain vendors such as Red Hat had already started to prepare patches, although many vendors were likely not fully aware of this. Later on Aug 10th, the research paper was presented at USENIX Security and received significant press coverage, which prompted many more vendors to look into the matter. According to the information collected from the public sources on the same day, Cloud.gov and Akamai have patched the vulnerability. On Aug 16th, Verizon Edgecast claims to have patched it as well, using the temporary fix.

3. MEASUREMENT METHODOLOGY AND GROUND TRUTH

Our basic approach is to initiate a regular TCP connection with a server and then test whether the vulnerability is present by sending a series of probing packets. As will be shown later, since the signature of the vulnerability is distinctive, we are able to make an accurate determination on its vulnerability.

For each scanned server, we aim to find the following:
(1) whether it has the challenge ACK vulnerability,
(2) whether it is patched and how,
(3) whether it is not vulnerable due to other reasons, *e.g.,* server not compliant to RFC 5961.

To test the vulnerability on a high level, we need to confirm that there are indeed challenge ACKs in response to packets containing either the SYN, ACK, and RST flags. The absence of them would indicate that a server is not compliant to RFC 5961 and therefore not vulnerable. Second, we need to check if it has a default limit of 100 challenge ACKs per one second interval. The presence of the limit is indicative that it is a vulnerable and unpatched Linux kernel. The absence of any observed rate limit (higher than what we can measure) would indicate that the server is likely patched.

Challenges. There are three challenges we need to overcome: First, there exist a variety of operating systems and versions running on the Internet [33], we need to be able to reliably identify the vulnerable Linux hosts and exclude others. Second, we must also consider the possibility that middleboxes such as firewalls that may interfere or distort any measurements that we perform (*e.g.,* as described in [29]). Third, due to the nature of the vulnerability, a relatively large number of packets need to be sent, in order to test the rate limit behavior. If not managed carefully, packet losses can occur and servers can timeout.

Ethical Considerations. First of all, it is important to note that our scan merely tests for the presence of challenge ACK vulnerability without actually conducting a full-fledged attack, which has many more steps (*e.g.,* IP spoofing) and packets exchanged [19]. Secondly, to minimize the impact on the scanned server, we choose a very low scan intensity so as to not overwhelm the server or its network. Finally, for the vulnerable servers, the scan does require interacting with the challenge ACK rate limit mechanism, which may prevent challenge ACKs (pertaining to other connections on the server) from being sent out. However, considering that challenge ACKs are rarely triggered in regular connections and the fact that challenge ACKs are expected to not always be delivered (due to rate limit and packet loss), the negative impact on other connections is limited.

3.1 The Basic Scan

We start by describing a basic scan that can answer only the first question: whether a server has the challenge ACK vulnerability. Later we will expand the scan method to be able to answer other questions.

Scan Components. There are in total five components. Besides the handshake and termination, a scan consists of `ACK test`, `SYN/ACK test`, and `RST test`. Each test attempts to trigger and exhaust the challenge ACK rate limit (if any) by crafting packets according to RFC 5961.

Interleaved between all five components are data packets comprised of three packets (for redundancy) that use in-order sequence numbers, checking whether the connection is still alive. In the case of the data check after the handshake, it fulfills the additional goal of advancing the connection to the `ESTABLISHED` state in the event that the server uses TCP deferred accept [1].

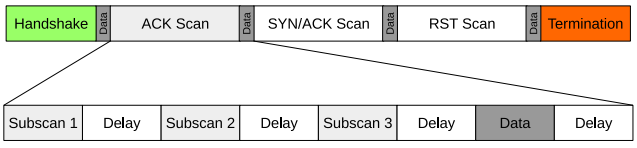

Figure 4: An overview of a single full scan.

Each of the three tests is comprised of three rounds of packet exchanges. Every round has 210 packets of the corresponding packet type to solicit challenge ACKs.

For `ACK test`, three rounds of 210 packets are sent with an in-window sequence number (`RCV.NXT`+10) and ACK number acknowledging old data (`SND.NXT`-2^{30}).

For `SYN/ACK test`, three rounds of 210 packets are sent with an out-of-window sequence number (`RCV.NXT`+2^{30}) and out-of-window ACK number

(`SND.NXT`-2^{30}).

For `RST test`, three rounds of 210 packets are sent with an in-window sequence number (`RCV.NXT`+10).

210 packets are chosen as it is a high enough number to distinctly differentiate whether a server responds with 100 challenge ACKs. If all tests return 100 challenge ACKs, we can safely assert that it contains the challenge ACK vulnerability, as Linux is the only operating system that has implemented the rate limit feature.

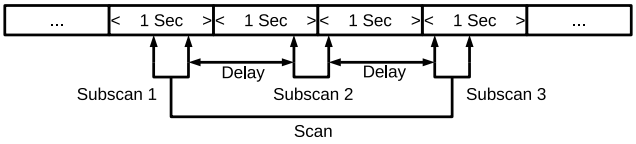

Figure 5: Timing of a specific type of test.

Scan Timing Instead of synchronizing the clock with the server (and wasting three seconds [19]) to ensure that all 210 packets fall in the one second interval, we arrange the packets such that it can avoid the synchronization delay while still providing reliable results. To do so, each round of 210 packets are sent over a 1/3-second interval with a 1 second delay before the next round 210 packets are sent.

The 1/3-second interval ensures that we are able to send all our packets fast enough within a 1 second time interval, yet is long enough to not cause congestion and packet losses. The 1 second delay ensures that we have enough time to collect all response packets from the server; it also allows us to skip over any "dirtied" 1 second intervals where the round of probing lands on the border of two 1 second intervals. With 3 rounds of tests, if a server is vulnerable, 2 out of 3 rounds are expected to receive 100 challenge ACKs, and the remaining round may have a number in between 100 and 200 if it happens to be crossing the two 1 second intervals. For brevity, we only illustrate this (instead of offering proofs) in Fig. 5.

3.2 Complete Scan

Building on the basic scan, we now expand it to a complete scan to be able to answer the two additional questions: (1) whether it is patched and how, (2) whether it is not vulnerable due to other reasons. The complete scan has branches and effectively translates to a decision tree as depicted in Fig. 6.

3.2.1 Patch and baseline behaviors

To understand how we can differentiate the patching behavior from non-patched/other operating systems, we profile a set of major operating systems offline in a lab environment to obtain a baseline. Fortunately, during our study, we find responses to be distinct from operating system to operating system. Based on this, we can identify patched cases reliably as shown in Fig. 6.

Note that we are less interested in the specific OS versions. Instead, we classify the OSes based on their challenge ACK behaviors (whether RFC 5961 is implemented), and list them in Table 2. For instance, upon inspecting the Linux kernel source code, we are able to distill two types of vulnerable challenge ACK behaviors (before patching): (1) old vulnerable Linux which does not have a per-socket challenge ACK rate limit. (2) new vulnerable Linux which does have a per-socket rate limit. Note that the per-socket rate limit does not eliminate the vulnerability as it can be bypassed as long as there is at least one byte in the payload (and it does not carry the SYN flag). This means that only `SYN/ACK test` is affected and will receive 1 challenge ACK while the other two will still see 210. These behaviors are verified through a range of Linux distributions including the default Ubuntu 12.04, 14.04, 16.04, Red Hat 7.1, SUSE Linux 12, and CentOS 6,7 released prior to the patch date.

Type	OS	ACK	SYN/ACK	RST
RFC5961 compliant	Unpatched Linux (early*)	100	100	100
	Unpatched Linux (late*)	100	1	100
	Patched Linux (Temp fix)	210	210/1	210
	Patched Linux (V1)	210	210/1	210
	Patched Linux (V2)	1	1	1
Non-RFC5961 compliant	FreeBSD (Mac†)	210‡	210	0
	Windows	0	210	1
	Solaris	210‡	210	0
	OpenBSD	0	0	0
	Others, old Linux/Windows	0	0	0

*: Early = Prior to 4.0. Late = 4.0 and above.
†: The TCP stack of Mac OS is based on FreeBSD.
‡: These OS's can be differentiated through a forward ACK test.

Table 2: Operating system responses.

For patched hosts, as we discussed, there are in total three types of patching behaviors; two derived from the kernel patches [9, 11] and the temporary fix recommended by the researchers and the industry [18, 35]. They can all be clearly differentiated from non-patched cases. As the temporary fix and V1 patch yield the same results to our tests (both appearing to be raising the rate limit to higher than 210), we always see 210 challenge ACKs during `ACK test`. For `SYN/ACK test` however, it is possible that we see 210 or 1 challenge ACKs, depending on whether the kernel prior to patch already had the per-socket rate limit. The only case that comes close is the recent FreeBSD versions that have partially implemented RFC 5961 (tested on FreeBSD 10.3). On the surface, it behaves identically to that of a patched Linux server. However, we are able to differentiate them from Linux servers due to how they respond to ACK packets that are acknowledging data in the future (*i.e.,* ACK number too advanced). In Linux servers, such ACK packets are silently dropped while FreeBSD will send a regular ACK packet in response. Exploiting this unique behavior, we follow up cases during the initial `ACK test` (shown in Fig. 6) where we see a potentially patched server with a second round of `Forward ACK`

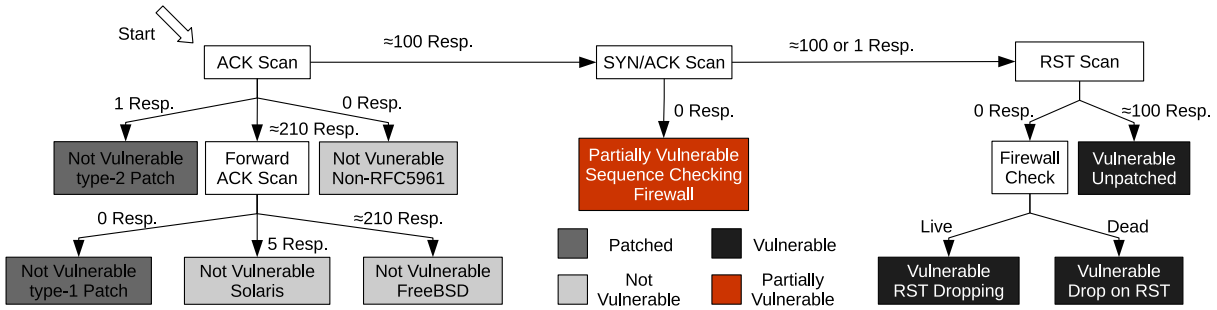

Figure 6: The decision tree of the expanded scan.

`test`. FreeBSD will return 210 responses while Linux-based servers always return 0. Interestingly, Solaris is similar to FreeBSD except it will only respond with 5 ACK packets. For Windows, we tested Windows 7, 8, 10, and Windows Server 2003 R2, 2008 R2, 2012 R2, and 2016 DataCenter (all are fresh installs) and the results are consistent. They can be easily differentiated from the unpatched and patched Linux cases.

In addition, it is evident that V2 patched Linux is unique enough by looking at just `ACK test` result, as only 1 challenge ACK can be observed. The reason is that V2 patch has a per-connection rate limit which allows only 1 challenge ACK to be sent every 0.5 second. This is because our `ACK test` sends 210 packets in 1/3 of a second (less than 0.5 second).

Finally, in some cases during the `RST test` as shown in Fig. 6, we may encounter two interesting corner cases that are likely caused by firewalls. By the time we reach the `RST test`, it is already evident that it is a vulnerable Linux host (no other major OS fingerprints match it). However, we note that the in-window RST packets either are discarded (after which the connection with the server is still alive), or terminate the connection directly (a classic stateful firewall behavior [29]). In both cases, they are still considered vulnerable cases according to the alternative attack strategy designed in [19] which can replace the RST-based probing with ACK-based.

3.2.2 *Other Non-Vulnerable Cases*

As we see in Table 2, if a host is not even RFC 5961 compliant, it is definitely not vulnerable. This includes Windows, FreeBSD, Solaris, and OpenBSD. In the decision tree (Fig. 6), we can see that FreeBSD cases can be uniquely captured and other non-RFC5961-compliant cases will be discovered after the initial ACK test as 0 challenge ACKs are received.

In addition, there is a partially vulnerable case caused by firewall interference. As briefly discussed in §3.1, a certain type of stateful firewalls check TCP sequence numbers [29] and drop TCP packets that have out-of-window TCP sequence numbers. In this case, if the server is behind such a firewall, the probing SYN/ACK packets (with out-of-window sequence numbers) will be dropped by the firewall, resulting in 0 challenge ACKs. This defeats the connection (four-tuple) inference attack (discussed in §2.2) as it is now impossible to craft SYN or SYN/ACK packets soliciting challenge ACKs without knowing a valid in-window sequence number. However, an attacker already knowing the four-tuple can still perform sequence number inference using the SYN/ACK packets where only those with in-window sequence numbers can solicit challenge ACKs (exactly the requirement of sequence number inference §2.2); ACK number inference is also possible as `ACK test` still sees 100 challenge ACKs; hence this case is labeled as partially vulnerable.

3.3 Parallelizing the Scan

Even though the complete scan based on the decision tree can correctly classify a TCP stack, we need something more efficient than probing a single IP at a time to carry out a large-scale scan. A parallel scanner aims at maximizing network utilization by scheduling probing packets for multiple targets at appropriate times and recording/maintaining the state for each scanned target. In the past, scanners such as ZMap [22] are able to achieve this by simply scheduling the probing packets for different targets without any constraint (except not to saturate the bandwidth). However, in our scan, we need to not only maintain the state for each probed target IP address, but also enforce the strong timing requirement as described in § 3.1, *i.e.,* sending 210 packets spread out in one third of a second in each round of test, and one full second of delay in between rounds. This unique timing constraint makes it difficult to schedule the probing packets for multiple targets and achieve high bandwidth utilization.

To this end, we develop a general probing methodology to overcome the challenge. Similar to ZMap, we operate with only two dedicated threads: one for sending and one for receiving. A single sender thread allows us to precisely schedule the timing of each outgoing packet. Now the challenge is that we need to manage a timer for each IP somehow. Our solution is as follows: (1) instead of managing a timer for each IP, we can group IPs into fixed-size batches and manage a timer for each batch; (2) each batch will have a one-second idle pe-

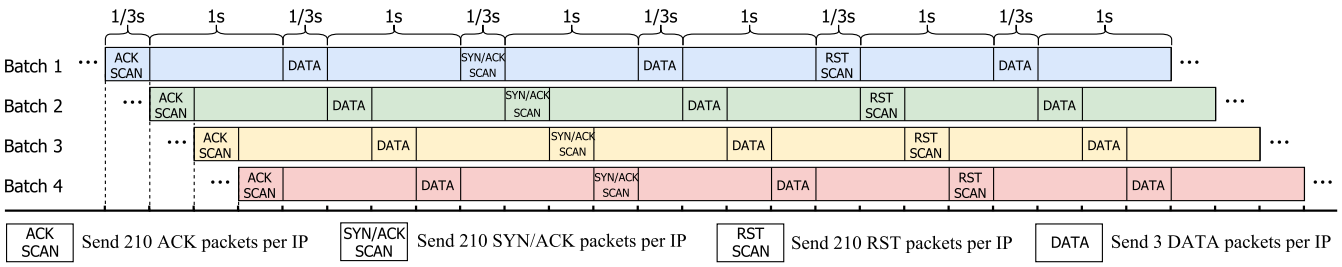

Figure 7: Scheduling of parallel scanning. The figure is simplified with each test (*e.g., ACK test*) done once instead of three times. The connection establishment and teardown phases are omitted.

riod in between tests which can be time-multiplexed for scheduling N other batches (N=3).

Specifically, we scan different IP batches in a time-multiplexed fashion as shown in Fig. 7. Each time slot is 1/3 second, allocated one IP batch (210 probing packets will be sent per IP). The scan has a 4/3-second period that allows 4 different IP batches to be time-multiplexed, where probing packets for each batch will receive a gap of a full second (as intended). The time slots are assigned to different batches in rotation This ensures that we always have some IPs to scan in each time slot.

The batch in its assigned time slot is managed together where $210 \times k$ packets are sent in total; here k is the number of IPs in each batch. To avoid congestion, these packets are scheduled evenly (one IP after another) over the 1/3-second duration. Besides the timer state maintained for each batch, a more fine-grained state for each IP in the batch is maintained indicating where it is on the decision tree; this allows the sender to know what packets to send for a specific IP (or if it is already done).

With this strategy, the scanning speed is roughly $4 \times k$ times the speed of a sequential scanning (one IP after another). In practice, on our network, $k = 40$ (therefore 160X speedup) is the maximum number we can go without experiencing significant packet losses.

Our parallel scanning tool can also be easily applied to measurement scans sharing similar characteristics:
(1) **Packet-intense** scan that requires sending multiple packets in a short period of time for each IP. If the scanner has a poor scheduling decision, packet losses may occur, *e.g.,* bursty packets scheduled sent to the same destination at once. Our batch design allows us to pace packets for each IP while still fully utilizing the bandwidth resource by rotating among different IPs.
(2) **Stateful** scan that consists of multiple rounds of tests, and the scan state changes according to the response of each round of test. To track the scan state, we maintain a fine-grained per-IP state using a finite state machine.
(3) **Fixed-time-period** scan that has strict requirement on when packets need to be delivered. This allows us to perform the time-multiplexed scan with multiple time slots designed to accommodate the time requirement. Otherwise, we have to seek suboptimal solutions such as per-IP time management, which is difficult to administer.

In theory, any large-scale scans that have strong timing requirements and fixed time period may leverage a similar design.

3.4 Data Cleaning

Due to the sheer volume of traffic sent at a relatively high rate, we are bound to experience losses which make it challenging to interpret and classify the results. To do so, we cross-validate results across days to correct mis-classified data when possible. Specifically, we conduct two types of data cleaning: (1) forward cleaning and (2) backward cleaning. In forward cleaning, whenever we classify an IP as patched reliably, it cannot go back to vulnerable in the future (we argue that the chances that it does go back are extremely small). Similarly, in backward cleaning, whenever we classify an IP as vulnerable reliably, it cannot be possibly patched in the earlier days. Through our manual analysis, we discover that these rules are effective in eliminating mis-classified results due to packet losses.

4. IMPACT ON TOP 10K WEBSITES

Since the vulnerability was introduced in all Linux kernels since late 2012, we expect a large percentage of the Internet servers to be vulnerable, as it was estimated that more than 96.6% of the top 1 million Alexa websites use Linux servers [6].

4.1 First day of scan

We conducted our first scan on the IPs that host the top 10,000 websites of Alexa's top sites on Aug 19th, 9 days after the vulnerability went public. We excluded the duplicate IPs and those that timeout before we can finish our scan, which leaves us to 7,484 unique IPs. Since the vulnerability received significant attention and is quite severe, it is likely that many of the websites would have patched their Linux servers by the time we started our scan. For instance, as we noted, popular CDN services like Akamai have even patched their servers before the vulnerability went public.

Dataset	RFC 5961 Vulnerable	RFC 5961 Patched	Non-RFC 5961	Unknown
Top 100	32.91%	18.99%	37.97%	10.13%
Top 10K	26.39%	41.09%	30.85%	1.67%

Table 3: Results of the first day of scan on top 100/10K sites

.

1-15	Stat.	Note	16-30	Stat.	Note
google.com	V		msn.com	N	Non-RFC
youtube.com	V		yahoo.co.jp	V	
facebook.com	N	Non-RFC	weibo.com	V	
baidu.com	N	Non-RFC	linkedin.com	V	
yahoo.com	Unk	Unk	vk.com	V	
amazon.com	N	Non-RFC	yandex.ru	N	Non-RFC
wikipedia.com	N	Patched	hao123.com	N	Patched
qq.com	N	Patched	instagram.com	N	Non-RFC
google.in	V		ebay.com	N	Non-RFC
twitter.com	N	Non-RFC	google.ru	V	
taobao.com	N	Patched	amazon.co.jp	N	Non-RFC
live.com	N	Non-RFC	reddit.com	N	Patched
sina.com.cn	N	Non-RFC	360.cn	N	Patched
google.co.jp	V		t.co	N	Non-RFC
bing.com	N	Non-RFC	pinterest.com	V	

Table 4: Top 30 vulnerable URLs with unique IPs. *V* for Vulnerable, *N* for Non-Vulnerable and *Unk* for Unknown.

Top 10K vs. Top 100. In Table 3, we see that 26.39% of the 7,484 unique IPs for the top 10K websites were vulnerable, and 41.09% were patched already, indicating significant patching has indeed been performed prior to our first day of scan. Note that a large fraction (30.85%) of IPs are running extremely old TCP stacks (at least 4 years old), indicating that they may have other kernel vulnerabilities.

We were surprised to find that the Alexa top 100 websites actually have a much smaller fraction of patched servers than the top 10,000. As shown in Table 3, only 18.99% of the 79 unique IPs corresponding to the top 100 were patched and 32.91% were vulnerable. As the top 100 websites all belong to large Internet companies, it is extremely unlikely that they were unaware of the vulnerability. Instead, we hypothesize that it is because they have a larger share of global traffic which makes it more challenging to find time to apply patches to the kernel and reboot the server. In Table 4, we list the vulnerability status of the top 30 websites on the first day of scan. It is apparent that Google servers are all vulnerable even though the developer who submitted the initial Linux patch was from Google [11]. In addition, *yahoo.co.jp*, *weibo.com*, *linkedin.com*, *vk.com*, and *pinterest.com* were found to be vulnerable. Upon further inspection, Google servers are patched finally between Oct 26th and Nov 3rd gradually.

4.2 Patching

Linux distribution patch timeline. Unlike Windows or Mac OS, upstream Linux kernel patches often need to be backported and tested on various Linux distributions (*e.g.,* Red Hat) who then finally push the update to their users. This can be a relatively long delay as most Linux distributions are operating on old base kernel versions. We list the timeline of Linux distribution patch releases in Table 5 (some are still operating on Linux kernel 2.6.X).

Date	Distribution patch	Kernel base	Patch version
07/19	Fedora 23, 24 server	4.6.4	V1
08/17	Red Hat 6 server	2.6.32	V1
08/18	Red Hat 7 server	3.10	V2
08/23	CentOS 6, 7 server	3.18	V1
08/29	Ubuntu Xenial 16.04	4.4	V1
08/29	Ubuntu Trusty 14.04	3.13	V1
08/29	Ubuntu Precise 12.04	3.13	V1
08/31	Debian Wheezy 7	3.2	V1
09/03	Debian Jessie 8	3.18	V1

Table 5: Timeline of Linux distribution patch releases.

Coincidentally, Fedora 23 and 24 running kernel 4.6.4 are the fastest to push out patches, only 9 days after the initial patch (V1) was proposed by the Linux kernel developers. It is likely because the kernel is so new that it is the easiest to incorporate the patch from upstream Linux (which fixed the vulnerability in 4.7). They are ahead of the next fastest distribution – Red Hat (patch released on 08/17) – by almost a month, even though Red Hat was involved in the patching process early on (publicizing the CVE [31]). Overall, the fastest release (Fedora) and the slowest release (Debian) are 47 days apart, leaving some Linux servers vulnerable for much longer than others.

In addition, as described earlier, there are two different upstream Linux patches (V1 and V2) which were committed on 07/10 and 07/14 respectively. Interestingly, almost all Linux distributions except Red Hat 7 picked up the first patch instead of the second even though the second is more secure, indicating that they were eager to pick up the earliest possible patch.

Finally, to mitigate the vulnerability, even though Linux distributions can be slow in releasing patches, system administrators do have the option to apply a temporary fix which will look similar to V1 patch (in our scan results, they both look like raising the global rate limit) [18].

Patching behaviors. We continued our scan for almost six months, ending on Feb 10th, 2017. The scans were conducted mostly daily in the beginning and every two days towards the end. We end up with a set of 6994 unique IPs, all of which are successfully scanned on all days. When we look at the beginning few days in Fig. 8, it is evident that the fraction of vulnerable IPs for the top 10K dataset reduced dramatically, likely representing the tail end of the initial burst in patching. The spike in patch rate also correlates roughly with the patch releases of several popular Linux distributions, including Red Hat and CentOS (shown in Table 5). How-

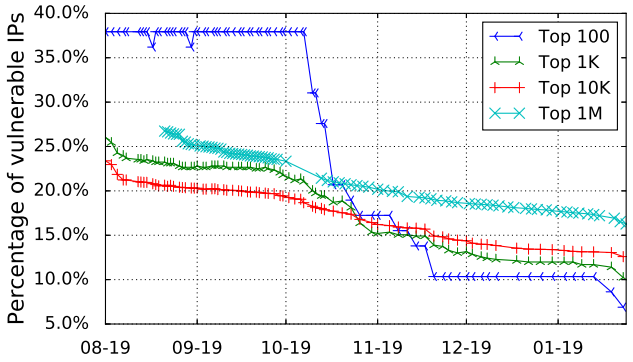

Figure 8: Percentage of vulnerable IPs over time

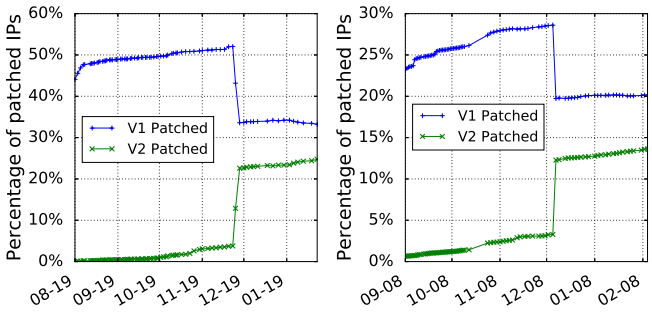

Figure 9: Percentage of V1 patched/V2 patched IPs over time (Left: Top 10K. Right: Top 1M)

ever, since some distributions pushed out updates late, it is likely that a significant portion of vulnerable IPs applied the temporary patch, *e.g.,* Akamai claimed that they did so [18].

Interestingly, after the initially burst, the number of vulnerable hosts is mostly steady, except three notable drops starting mid October, early November, and early December respectively (*e.g.,* including Google server patches).

Finally, in Fig. 9, we show that V2 patches are catching up extremely slowly until a surge from Dec 12 to Dec 14 caused by CloudFlare upgrading their already V1 patched servers. Besides CloudFlare, however, most V1 patches never turned into V2 patches during the 6-month period. This indicates that either these companies are aware of the fact that the 2nd patch is non-critical and intentionally delayed it, or their patching cycle is simply very slow in general (on the order of several months). We can thus infer that patching kernel vulnerabilities is indeed a painful process and the frequency of patching is being minimized by most.

Top 100 vs. Top 1K vs. Top 10K. Fig. 8 provides a clear view on how the patching behaviors differ for top websites at various ranks. As mentioned earlier, top 100 websites generally patch more slowly compared to top 10K. We suspect that it is because many top 100 websites maintain their own infrastructure (*e.g.,* Google) that is not only large in scale but also customized to tailor for their own service needs. In contrast, the top 10K

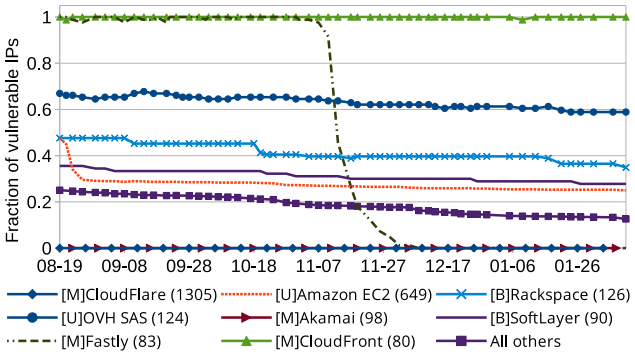

Figure 10: Percentage of vulnerable IPs of top hosting companies. Derived from the result of top 10k. *U* for Unmanaged, *M* for Managed and *B* for Both. The numbers in the legend indicate unique IPs belonged to the company in the dataset.

are mostly hosted by third-party hosting services such as Amazon EC2 (more detailed on this later). However, if we look at the patching behavior over time, top 100 websites eventually catch up from late Oct to mid Nov (more than 2 months after disclosure). During the catch-up period, the list of patched IPs in top 100 includes 8 google IPs, and 1 IP each for stackoverflow, github, ok.ru, pinterest, yahoo.co.jp, adobe, imgur *etc.*

Between top 1K and 10K, a similar (although not as obvious) trend shows that top 1K websites initially have lower patch rate, but eventually catch up from a similar time frame. As expected, as will be described in more details in the next section, top 1M websites ultimately have the lowest patch rate.

Managed vs. Unmanaged hosting. Besides Internet giants such as Google that have dedicated infrastructure to host their own websites, most top websites are hosted on infrastructures managed by professional hosting companies such as Amazon EC2 and various CDNs. We divide the hosting services into two types (managed and unmanaged) according to whether the operating systems are managed by a particular hosting company. For instance, Amazon EC2 and OVH allow users to run guest operating systems of their choice inside virtual machines, which means tenants are responsible for managing and patching their own operating systems (and hence we call them unmanaged hosting). In contrast, CDNs such as Cloudflare never expose the operating system details to tenants, which implies that CDNs themselves are responsible for managing the underlying operating system. Besides CDNs, some hosting services also offer managed hosting, *e.g.,* Rackspace [12], which takes care of patching.

We are interested in the patching behaviors among the managed and unmanaged hosting services. For each scanned IP address, we map it to their corresponding corporation using the Whois database, which usually allows us to determine whether it is a managed or unman-

aged hosting (or both). To further distinguish Amazon EC2 and Cloudfront CDN (both of which belong to Amazon), we consult the published IP ranges provided by Amazon [2]. As shown in Fig. 10, we show the top 8 companies in the top 10K dataset; all of them are hosting companies (Google did not have enough unique IPs). It is interesting to see that the patch rate of different hosting services varies significantly (even among CDN providers). CDNs such as CloudFlare and Akamai are fully patched (0% vulnerable) from the very first day of our measurement. Surprisingly, the Amazon Cloudfront CDN have never patched any of their servers even six months after disclosure. The Fastly CDN is somewhere in between where initially all their servers are vulnerable and they get fully patched over a period of about 20 days, starting early Nov. This shows that applying kernel patches to a large set of servers is very challenging, as only a subset of servers can afford to have their services shutdown and reboot.

On the other hand, unmanaged hosting such as Amazon EC2 has significantly less patching. This is understandable as it is completely up to tenants on Amazon EC2 to decide their patch schedule. Note that many unmanaged hosting companies sometimes do optionally provide managed hosting at an extra cost (*e.g.,* AWS Managed Services [3]). For Amazon EC2, we observe a significant drop in the first few days in the number of vulnerable IPs, and it is likely because either a small portion of tenants chose the managed services, or they patched their servers themselves. However, given that Amazon CloudFront CDN did not patch any of their servers, it seems unlikely that they would be patching the same vulnerability for their EC2 customers.

Finally, for hosting services such as Rackspace and SoftLayer that offer both managed and unmanaged services, irregular and small patching is observed.

4.3 Comparison to Debian Weak Keys and Heartbleed

There were two major historical security vulnerabilities that are documented in the literature for their corresponding Internet-wide patching responses: (1) the weak keys generated in the Debian OpenSSL package [37] reported in 2008; (2) the Heartbleed vulnerability (CVE-2014-0160) which is an information leakage bug in the OpenSSL cryptographic library that can leak private keys [16, 20]. Both security vulnerabilities are extremely dangerous as they can cause the cryptographic keys to be guessed easily or even directly read by a remote attacker.

The most interesting aspect for comparison is the timeliness of patching in all three events (including the TCP stack vulnerability). The three vulnerabilities are all different: (1) Debian weak key is a flaw that requires replacement of keys (arguably more effort than regular patching). (2) Heartbleed is an application-layer vulnerability which is relatively easy to patch. (3) The TCP vulnerability here is a kernel vulnerability which requires rebooting to apply the patches.

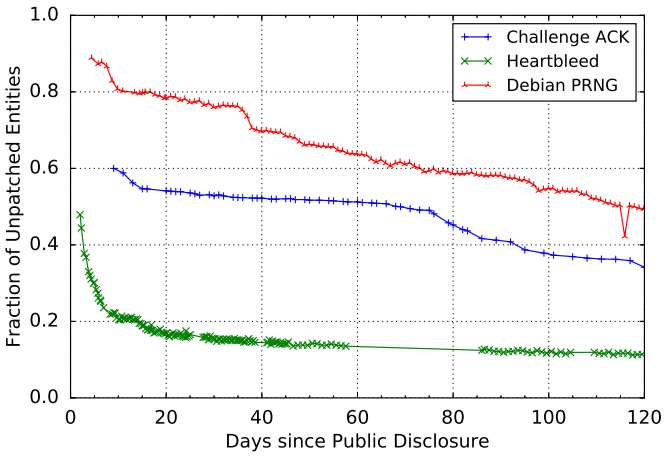

Figure 11: Comparison to Heartbleed and Debian PRNG. The initial number of vulnerable hosts of Heartbleed and Debian PRNG at day 0 are estimated based on data in [20, 37].

We acknowledge that it is difficult to perform a direct "apple-to-apple" comparison as there are several differences in the way datasets are collected: (1) the Debian weak key study has a small sample of affected hosts (only 751 out of 50,000 servers displayed weak keys); (2) our TCP scan is against the Alexa top 10,000 while the Heartbleed is against the top 1 million; (3) we started our scan 9 days after the vulnerability disclosure while Debian and Heartbleed scans started 4 days and 2 days after, respectively. Nevertheless, we believe the data can already shed some light on answering the following question — how fast the affected hosts get fixed in respect to the disclosure date.

To this end, we use the metric of *fraction of affected hosts*, which is computed as "the number of affected hosts" divided by "the total number of hosts that were vulnerable prior to disclosure". If the number drops quickly early on, it means that patching is very responsive. Fig. 11 captures this trend. For Heartbleed, only 11% of the hosts are reported to be vulnerable on their initial scan conducted 2 days after the disclosure. Given that it is estimated that the total fraction of vulnerable hosts is around 24% to 55% prior to disclose [20], this means that the vulnerable rate after 2 days is from 11%/55% = 20% to 11%/24% = 46%, and more than half of the hosts (54% to 80%) have already been fixed 2 days after disclosure. Even if we use 54%, it is an impressive patch rate from the very beginning. For the TCP case, we consider all patched hosts to be vulnerable prior to disclosure. Clearly, the patch rate is not as impressive as Heartbleed. In our study, we see that still

60% of the affected hosts remain 9 days after disclosure, while only about 20% affected hosts remain for Heartbleed on the same day. For Debian weak key, no estimate is available on the total number of affected hosts prior to disclosure. We there retroactively generate the number according to the relatively stable curve which started 4 days after disclosure. We admit that theoretically there could be more affected hosts than what we estimated, so the starting point for Debian curve needs to be taken as a grain of salt. Nevertheless, the patch trend since day 4 remains accurate. Overall, we can see that both TCP kernel vulnerability and Debian weak key have a more steady patching behavior than Heartbleed. In contrast, the patching for Heartbleed is much more aggressive early on and dies off very quickly.

5. IMPACT ON TOP 1M WEBSITES

To measure the impact on a larger scale, we began to conduct scans on the Alexa Top 1 million websites since Sep 8, 2016, almost a month since the vulnerability went fully public, until Feb 11, 2017. Again, we excluded the duplicate IPs (which lead to 537,049 unique IPs remaining) and those that timeout before we can finish our scan, which leaves us to 474,093 unique IPs.

In terms of vulnerable websites, on the day of Sep 8, 26.7% of IPs were vulnerable for the top 1 million (shown in Fig. 8) as opposed to 22.4% for the top 10,000. This is perhaps expected as the top 10,000 websites are typically operated by larger companies and possibly CDNs. Interestingly, there is a large fraction of servers (34.8%) that are not compliant to RFC 5961. We further verified that most of them in fact are vulnerable to traditional blind in-window attacks [14], even though they are not vulnerable to the challenge ACK attacks.

Patching behavior. Since our 1 million websites measurement started on Sep 8, a large proportion (23.1% of all IPs) of the RFC 5961 compliant Linux hosts are already patched before our scan started. As shown in Fig. 8, over the course of the scan, we find mostly steady and measurable patching throughout, except a few notable drops in vulnerable hosts that are somewhat more significant than others. Upon closer inspection, they correspond to large companies such as Google patching their servers during the same period.

Interestingly, towards the end of the scan (6 months after disclosure), we see that the patched hosts keep increasing with no signs of leveling off. At this point, it is unclear if the hosts are really patching against the specific vulnerability. It is more likely that the system administrators simply have a very loose maintenance schedule for kernel updates. In general, it is worrisome to see that kernel vulnerabilities are being patched this slowly as opposed to application-layer vulnerabilities such as heartbleed, especially considering that slow kernel patching could leave many other kernel vulnerabilities open as well.

Also, Fig. 9 shows that, similar to the top 10K case, the V2 patch has a surge from Dec 12 to Dec 14, again due to CloudFlare upgrading all of their already V1 patched servers.

Middlebox behaviors. According to our measurement, the most common middleboxes that may affect the challenge ACK attack is the TCP sequence number checking firewall [29]. As shown in the decision tree (Fig. 6), such firewalls block SYN/ACK packets that come with out-of-window sequence numbers, and makes the hosts partially vulnerable. Overall, around 2% of the IPs (10741 in total) are behind such sequence checking firewalls (we conservatively exclude them from the set of vulnerable IPs). Interestingly, as shown in [29], sequence number checking firewalls themselves can introduce security vulnerabilities; this is worth looking into in the future.

In addition, we observe a very small fraction of firewalls (affecting 0.03% or 161 IPs) that terminate connections upon seeing an in-window RST, which renders the server vulnerable to the traditional blind in-window attacks, even when the servers are already patched against them. We also confirm another type of firewalls (affecting 0.02% or 108 IPs) simply drop in-window RST packets silently, whose behavior was not publicly known. However, both such firewalls do not prevent the challenge ACK attack as suggested by the researchers [19].

6. OTHER IMPACT

6.1 Impact on other services

Tor services. The challenge ACK attack is particularly dangerous for Tor services [19], as users may be forced to go through attacker-controlled relays when connections with other relays are repeatedly shut down. We extract a full list of publicly advertised online Tor relays on Sep 16 which consists of 6,767 IPs. Among them, 37.9% are vulnerable, 21.9% are patched, 35.0% are non-RFC 5961 compliant, and the rest are unknown.

Telnet services. Hosts that run telnet services incur significant risks against off-path attacks who can inject malicious commands. We conducted a one-time scan on Oct 04 over telnet servers in the IPv4 range, based on the IP set with an open port of 23 from Censys [5]. We have measured a total of 2,797,179 unique IPs. As expected, telnet services are not commonly enabled on Linux servers, only 1.8% of them are found to be vulnerable to the challenge ACK attack. Interestingly, we do observe that 14.8% are patched Linux, a much better patch rate compared to web servers.

6.2 Notifications

We conjectured that it is more difficult to patch ker-

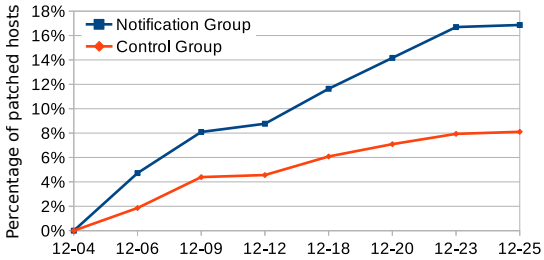

Figure 12: Notification effects.

nel vulnerabilities; however, we are curious if notifications will in fact make any difference in improving the patching behavior. On Dec 4, we started our notification study following a similar methodology described in [26, 34]. We constrain the study to the top 10,000 URLs. Out of them, we randomly select 1185 vulnerable URLs, among which are further divided into into our notification set (593) and control set (592). Specifically, we send a verbose email every week to the IPs in the notification set, describing the vulnerability and the associated IP addresses to the WHOIS abuse contacts obtained from Abusix. We then measure whether these IPs have patched the vulnerability over the course of three weeks. In the case that we have multiple URLs for the same email contact, we performed aggregation when possible to reduce the number of emails we send.

From our observations, we noticed that sending emails to abuse contacts generally helped in increasing patch rates. As of December 25th, we found that patching rates were double that of the control group - 16.86% for the notification group and 8.11% for the control group, despite some emails sent to the notification group not reaching their destinations (bounced back). Interestingly, this improvement is generally in line with the prior studies [26,34] even though the vulnerabilities used in their studies are not kernel ones. The underlying factors that determine how successful vulnerability notifications will be (*e.g.,* severity, who manages the affected services, and how easy is it to patch) remain an important research question.

7. RELATED WORKS

Off-path TCP attacks. The oldest off-path TCP attack can be dated back to 1985 [28], which allows an attacker to predict the initial sequence number of the SYN/ACK packet from the server without actually observing the packet. This effectively allows an attacker to establish connections with the server using spoofed IP addresses. In 1999, Linux had an interesting vulnerability [7] where it allows an off-path attacker to create a legitimate connection using any spoofed IP address. In 2004, the famous blind in-window attack is discovered [36] where an off-path attacker can craft various types of spoofed packets to interfere with a victim connection. The attack succeeds as long as any such packet has a sequence number within the receive window of the receiver. In recent years, a number of new off-path TCP attacks have been reported that can infer (instead of guessing blindly) what sequence number can be used to carry out attacks that can either forcefully terminate a connection or inject arbitrary content to the victim connection. Most attacks require executing malicious code on the client side [4, 19, 23–25, 29, 30], either in the form of malware [29, 30] or malicious javascript [23–25]. The most recent one reported in Aug 2016 [19] is the only one that does not have any requirement of executing malicious code, rendering it the most powerful attack to date. It is also the only off-path TCP attack that primarily targets servers, which prompts many Internet-facing services to patch their Linux servers. In our study, we developed a fast and scalable methodology for conducting the measurement of this latest vulnerability to measure their impact on Internet servers.

Large-scale measurement on Internet servers and security patching. Recent advances in Internet measurement has allowed much more to be learned about the behaviors of Internet servers at scale. ZMap is one of the pioneering tools that enabled Internet-wide (IPv4 range) scans [22]. Subsequently, significant research has been performed leveraging ZMap or similar concepts [21, 33]. A recent Internet measurement studied the resilience of TCP stacks against off-path/blind in-window attacks [27]. The conclusion is that there is still a significant fraction of TCP stacks (of top web servers) on the Internet that are using extremely old and vulnerable implementations (which are confirmed in our study as well, *e.g.,* the servers whose TCP stacks are prior to RFC 5961). There are two other recent Internet measurement was on the matter of Heartbleed and Debian weak keys [20, 37]. We compare our measurement results extensively with them in §4.3. Recently, researchers have also started to explore Internet-wide vulnerability notification as a proactive approach to improve the security patch rate [26, 34]. We also conduct a notification study on the challenge ACK vulnerability and show that the patch rate does improve drastically.

8. CONCLUSION

In this work we analyzed the impact of the recent challenge ACK vulnerability in Linux kernel, including (1) who was initially vulnerable, (2) patching behavior over time, by hosting services, and by network services (*e.g.,* Tor and telnet), (3) how notification affects the patching behavior. In general, we find that many in the top websites were vulnerable and remain vulnerable for an extended period of time. Interestingly, compared to top 1K and 1M websites, a larger fraction of Top 100 websites were initially vulnerable but eventually they caught up and have smaller fraction of vulnerable servers. We find that the hosting services behind many of the top websites in fact have a surprisingly diverse

and sometimes opposite patching behavior. We show that Linux kernel patching has some interesting differences from the recent Heartbleed and the Debian weak key event. The lessons and data collected will hopefully help the community better react to future Internet-wide security events.

Acknowledgment

Research was sponsored by National Science Foundation under grant #1464410 and grant #1528114. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Science Foundation or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

9. REFERENCES

[1] TCP protocol - Linux man page. http://man7.org/linux/man-pages/man7/tcp.7.html.

[2] Amazon AWS IP Address Ranges. http://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html.

[3] AWS Managed Services. https://aws.amazon.com/cn/managed-services/.

[4] Blind TCP/IP Hijacking is Still Alive. http://phrack.org/issues/64/13.html.

[5] Censys Scan Data Repository. https://censys.io/data.

[6] CVE-2016-5696 and its effects on Tor. https://blog.patternsinthevoid.net/cve-2016-5696-and-its-effects-on-tor.html.

[7] Linux Blind TCP Spoofing Vulnerability. http://www.securityfocus.com/bid/580/info.

[8] Linux bug leaves USA Today, other top sites vulnerable to serious hijacking attacks. http://arstechnica.com/security/2016/08/linux-bug-leaves-usa-today-other-top-sites-vulnerable-to-serious-hijacking-attacks/.

[9] [PATCH net] TCP: enable per-socket rate limiting of all 'challenge acks'. https://www.mail-archive.com/netdev@vger.kernel.org/msg119411.html.

[10] [PATCH net] TCP: make challenge acks less predictable. https://www.mail-archive.com/netdev@vger.kernel.org/msg118677.html.

[11] [PATCH v2 net] TCP: make challenge acks less predictable. https://www.mail-archive.com/netdev@vger.kernel.org/msg118918.html.

[12] Rackspace Managed Hosting Services. https://www.rackspace.com/en-us/managed-hosting.

[13] RFC 1948. https://tools.ietf.org/html/rfc1948.

[14] RFC 5961. https://tools.ietf.org/html/rfc5961.

[15] RFC 6056. https://tools.ietf.org/html/rfc6056.

[16] The Heartbleed Bug. http://heartbleed.com/.

[17] The TCP "challenge ACK" side channel. http://lwn.net/Articles/696868/.

[18] Vulnerability in the Linux kernel's TCP stack implementation. https://blogs.akamai.com/2016/08/vulnerability-in-the-linux-kernels-tcp-stack-implementation.html.

[19] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel. Off-path TCP exploits: Global rate limit considered dangerous. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[20] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer, and V. Paxson. The matter of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, 2014.

[21] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, 2013.

[22] Z. Durumeric, E. Wustrow, and J. A. Halderman. Zmap: Fast internet-wide scanning and its security applications. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, 2013.

[23] Y. Gilad and A. Herzberg. Off-Path Attacking the Web. In *USENIX WOOT*, 2012.

[24] Y. Gilad and A. Herzberg. When tolerance causes weakness: the case of injection-friendly browsers. In *WWW*, 2013.

[25] Y. Gilad, A. Herzberg, and H. Shulman. Off-Path Hacking: The Illusion of Challenge-Response Authentication. *Security Privacy, IEEE*, 2014.

[26] F. Li, Z. Durumeric, J. Czyz, M. Karami, M. Bailey, D. McCoy, S. Savage, and V. Paxson. You've got vulnerability: Exploring effective vulnerability notifications. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[27] M. Luckie, R. Beverly, T. Wu, M. Allman, and k. claffy. Resilience of deployed TCP to blind attacks. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, IMC '15, 2015.

[28] R. Morris. A Weakness in the 4.2BSD Unix TCP/IP Software. Technical report, 1985.

[29] 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*, 2012.

[30] Z. Qian, Z. M. Mao, and Y. Xie. Collaborative TCP sequence number inference attack: How to crack sequence number under a second. In *CCS*, 2012.

[31] Redhat. Bug 1354708 - (CVE-2016-5696) CVE-2016-5696 kernel: challenge ACK counter information disclosure. `https://bugzilla.redhat.com/show_bug.cgi?id=1354708`.

[32] Redhat. CVE-2016-5696. `https://access.redhat.com/security/cve/cve-2016-5696`.

[33] Z. Shamsi, A. Nandwani, D. Leonard, and D. Loguinov. Hershel: Single-packet OS fingerprinting. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '14, 2014.

[34] B. Stock, G. Pellegrino, C. Rossow, M. Johns, and M. Backes. Hey, you have a problem: On the feasibility of large-scale web vulnerability notification. In *25th USENIX Security Symposium (USENIX Security 16)*, 2016.

[35] UCR Today. Study Highlights Serious Security Threat to Many Internet Users. `https://ucrtoday.ucr.edu/39030`.

[36] P. Watson. Slipping in the window: TCP reset attacks, Apr. 2014.

[37] S. Yilek, E. Rescorla, H. Shacham, B. Enright, and S. Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, IMC '09, 2009.

[38] M. Zalewsk. Strange attractors and TCP/IP sequence number analysis. Technical report, 2001. `http://lcamtuf.coredump.cx/oldtcp/tcpseq.html`.