Off-Path TCP Hijacking Attacks via the Side Channel of Downgraded IPID

Xuewei Feng, Qi Li, *Senior Member, IEEE*, Kun Sun, *Member, IEEE*, Chuanpu Fu, and Ke Xu, *Senior Member, IEEE*, *Member, ACM*

***Abstract*— In this paper, we uncover a new off-path TCP hijacking attack that can be used to terminate victim TCP connections or inject forged data into victim TCP connections by manipulating the new mixed IPID assignment method, which is widely used in Linux kernel version 4.18 and beyond. Our attack has three steps. First, an off-path attacker can downgrade the IPID assignment for TCP packets from the more secure per-socket-based policy to the less secure hash-based policy, thus building a shared IPID counter that forms a side channel in the victim. Second, the attacker detects the presence of TCP connections by observing the side channel of the shared IPID counter. Third, the attacker infers sequence and acknowledgment numbers of the detected connection by observing the side channel. Consequently, the attacker can completely hijack the connection, e.g., resetting the connection or poisoning the data stream. We evaluate the impacts of our attack in the real world, and we uncover that more than 20% of Alexa top 100k websites are vulnerable to our attack. Our case studies of SSH DoS, manipulating web traffic, and poisoning BGP routing tables show its threat on a wide range of applications. Moreover, we demonstrate that our attack can be further extended to exploit IPv4/IPv6 dual-stack networks on increasing the hash collisions and enlarging vulnerable populations. Finally, we analyze the root cause and develop a new IPID assignment method to defeat this attack. We prototype our defense in Linux 4.18 and confirm its effectiveness in the real world.**

***Index Terms*— Side-channel, off-path exploit, hash collisions, IPID assignment.**

Manuscript received May 24, 2021; revised September 1, 2021; accepted September 15, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Ren. Date of publication October 1, 2021; date of current version February 17, 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB0803405, in part by China National Funds for Distinguished Young Scientists under Grant 61825204, in part by the National Natural Science Foundation of China under Grant 61932016 and Grant 62132011, in part by the U.S. Office of Naval Research (ONR) under Grant N00014-16-1-3214 and Grant N00014-18-2893, in part by the U.S. Army Research Office (ARO) under Grant W911NF-17-1-0447, in part by Beijing Outstanding Young Scientist Program under Grant BJJWZYJH01201910003011, and in part by Beijing National Research Center for Information Science and Technology (BNRist) under Grant BNR2019RC01011 and Grant BNR2020RC01013. *(Corresponding author: Ke Xu.)*

Xuewei Feng, Chuanpu Fu, and Ke Xu are with Beijing National Research Center for Information Science and Technology (BNRist), Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: fengxw18@mails.tsinghua.edu.cn; fcp20@mails.tsinghua.edu.cn; xuke@tsinghua.edu.cn).

Qi Li is with the Beijing National Research Center for Information Science and Technology (BNRist), Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing 100084, China (e-mail: qli01@tsinghua.edu.cn).

Kun Sun is with CSIS, Department of Information Sciences and Technology, George Mason University, Fairfax, VA 22030 USA (e-mail: ksun3@gmu.edu).

Digital Object Identifier 10.1109/TNET.2021.3115517

I. Introduction

SINCE the transmission control protocol (TCP) was first presented in RFC 793 in 1981 [1], more than 100 TCP related RFCs have been released to improve the protocol [2]. Consequently, it becomes difficult for off-path attackers to hijack TCP connections, mainly due to the challenge of inferring the 32-bit random sequence numbers and acknowledgment numbers of a targeted TCP connection [1], [3]–[5]. When launching a brute-force attack, the attacker has to flood more than 300 million spoofed packets at a time to the target systems that support both RFC 793 [1] and RFC 5961 [6]. Hence, off-path TCP attacks mainly rely on discovering side channel vulnerabilities to facilitate the inference of the sequence and acknowledgment numbers [7]–[11]. Fortunately, most of the uncovered vulnerabilities have been fixed or constrained by the security community [7]–[9].

In this paper, we uncover a new off-path TCP hijacking attack that exploits the mixed IPID assignment method in the latest Linux kernels (i.e., version 4.18 and beyond) to either terminate victim TCP connections or inject malicious data into victim TCP connections. First, our attack tricks the victim Linux machine into adopting the hash-based IPID assignment policy, instead of the by default more secure per-socket-based IPID assignment policy. Once the IPID assignment policy for socket protocols (TCP in our attack) is downgraded, it builds a side channel based on the IPID hash collisions of the globally shared 2048 hash counters, i.e., identifying a shared IPID counter on the victim by leveraging hash collisions. Second, by observing the shared IPID counter, an off-path attacker can detect the presence of TCP connections on the victim. Third, the attacker infers sequence and acknowledgment numbers of the victim connection to completely hijack the connection. This new attack does not need any assistance of puppets, i.e., unprivileged applications or sandboxed scripts controlled by attackers on victim hosts [7], [12], [13].

The `Identification` field of IP protocol (IPID) is used to indicate the uniqueness of a packet [14], [15]. After abandoning two previous vulnerable IPID assignment methods (i.e., global IPID assignment and per-destination IPID assignment) [7], [16]–[19], Linux currently assigns IPID to packets based on a mixed method [20]–[22]. If a packet is generated from socket protocols such as TCP, Linux uses the per-socket-based IPID assignment policy that assigns IPID to the packet based on the counter recorded in the protocol socket. Otherwise, Linux adopts the hash-based IPID assignment policy that assigns IPID based on one of the 2048 globally shared hash counters. In practice, Linux uses the `DF` (Don't Fragment) flag in packet header [14] to choose between the two policies, since only socket protocols can set this flag to TRUE to perform the path MTU discovery (PMTUD) mechanism [23], [24]. In other words, if the DF flag is set to TRUE, it uses the per-socket-based policy; otherwise, it chooses the hash-based policy. However, our study shows that the mixed IPID assignment in Linux implementations has vulnerabilities that can be exploited to launch a new off-path TCP hijacking attack.

Since Linux uses the more secure per-socket-based IPID assignment by default for TCP connections, an off-path attacker first tricks the victim into assigning IPID for its TCP packets using the less secure hash-based IPID assignment. This goal can be achieved by pretending to be a router and sending a forged ICMP "Fragmentation Needed" error message [25] to a victim. Since the ICMP error message informs the victim that the packets issued from the victim need to be fragmented and the DF flag is set, the victim will be tricked into cleaning the DF flag of TCP packets and thus uses the hash-based IPID assignment. Next, the victim chooses one IPID counter from the 2048 hash counters to assign IPID for its TCP packets. Among the 2048 globally shared hash counters, the target counter is decided by the hash value of four components, i.e., three fields of the packet (*source IP address*, *destination IP address*, *protocol number*) and a *random value* generated on system boot. Due to the small-sized hash counter pool, the attacker may identify the target hash counter used in a victim TCP connection via hash collisions, namely, alternating IP addresses to collide with the target counter.

Once the shared IPID counter is known, attackers can use the challenge ACK mechanism [6] as trigger conditions to change the shared IPID counter, facilitating the next two attack steps, i.e., to detect the presence of the victim TCP connection and infer the sequence and acknowledgment numbers. The attacker sends forged TCP packets to the victim, and the triggered challenge ACK packets will alter the shared IPID counter under different situations. It helps the attacker to determine if the specified values in the forged TCP packets are correct. Note that our attack only leverages the challenge ACK mechanism as trigger conditions to assist the inference of a victim TCP connection, instead of directly exploiting vulnerabilities in the challenge ACK mechanism to hijack TCP connections [8], [9].

Our attack does not suffer from traditional noise challenges that other works have to address [16], [17], [26], [27]. Since, in our attack, irrelevant TCP traffic using per-socket-based counters, instead of the hash-based counters, will not interfere with the attack traffic. Moreover, we measure that non-TCP traffic also rarely interferes with the attack. We evaluate the impacts of the new off-path TCP vulnerability on the Internet. We find that more than 20% of the Alexa (www.alexa.com) top 100k websites are vulnerable to our off-path attack. Those websites can be tricked into cleaning the DF flag and downgrading the IPID assignment from the per-socket-based policy to the hash-based policy for their TCP packets after receiving forged ICMP "Fragmentation Needed" messages. We implement a PoC and perform case studies on a wide range of applications, e.g., HTTP, SSH and BGP, to validate the effectiveness of the attack in the real world. The experimental results show that our off-path TCP attack can be constructed within 215 seconds on average and the success rate is over 88%.

Note that the attacks in IPv4 networks require that the off-path attackers control 1,837 IPv4 addresses on average to construct hash collisions and build the side channel. However, this requirement may not be easily fulfilled in the IPv4 networks for individuals. We resolve this limitation by exploiting IPv4/IPv6 dual-stack networks that hold adequate IP addresses to build the side channel. Particularly, for Linux systems running IPv4/IPv6 dual-stack, we show that attackers can more easily construct hash collisions by leveraging the adequate IPv6 addresses. Since the globally shared 2048 hash counters on the victim are also used by IPv6 addresses [22], an off-path attacker can misuse its adequate IPv6 addresses to construct hash collisions and identify the target hash counter used in a victim TCP connection. Hence, in IPv4/IPv6 dual-stack networks, the off-path attacker can more easily hijack victim TCP connections between the vulnerable server's IPv4 interface and an arbitrary IPv4 client.

Finally, we propose countermeasures that aim to eliminate the root cause of the newly discovered off-path TCP attack. We fix the mixed IPID assignment in Linux kernels by determining if a packet is originated from TCP protocol on the Protocol field in IP header, instead of the DF flag. We implement a prototype of our countermeasure in Linux 4.18 and confirm its effectiveness through experimental evaluation on the Internet.

Contributions: Our main contributions are the following:

- We uncover that the new mixed IPID assignment method can still be exploited to hijack TCP connections.
- We uncover a new side channel in IPID assignment in the latest Linux kernels. We demonstrate that the side channel can be exploited to learn the presence of victim TCP connections and infer the sequence and acknowledgment numbers of the connections.
- We discover that more than 20% of Alexa top 100k websites are vulnerable to our off-path attack. We also perform case studies on a wide range of applications and confirm the effectiveness of the attack.
- We analyze the root cause of the new attack and develop countermeasures that use new IPID assignment methods for TCP packets. Our prototype in Linux 4.18 validates its effectiveness.

II. Background

A. IPID Assignment in Linux

There are two basic IPID assignment policies in current Linux, i.e., IPID based on 2048 hash counters or IPID based on per-socket counters, where the latter is specific to socket related protocols such as TCP. Figure 1 illustrates the procedure of IPID assignment in Linux version 4.18 and beyond. When a packet is generated, the IP protocol first checks whether the packet is a TCP RST packet. If yes, then the IPID of the packet is set to 0 directly. This assignment is due to Geoffrey *et al.*'s disclosure of a side channel in previous assignment methods, i.e., IPID of the RST packet was assigned based on one of the 2048 hash counters before version 4.18, which can be exploited to detect the presence of TCP connections [21].

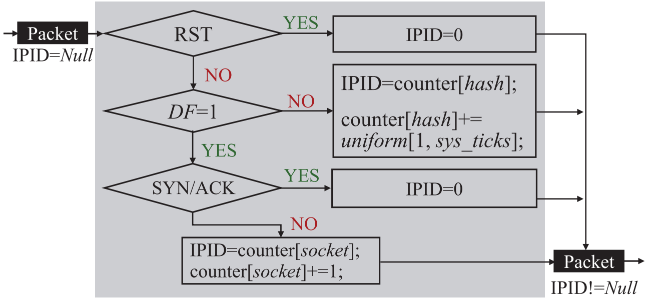


Fig. 1. IPID assignment in Linux version 4.18 and beyond.

If the packet is not a TCP RST packet, IP protocol checks the DF flag of the packet. IF the DF flag is set to FALSE, the IPID will be assigned based on a hash counter. There are totally 2048 hash counters in Linux. IP will select one from these counters according to the hash value of 4 variables, i.e., *source IP address* and *destination IP address* of the packet, the *protocol number* of the packet, and a *random value* generated on system boot. After the IPID value is copied from the selected counter, the counter will increase by a uniform distribution value between 1 and the number of system ticks since the last packet transmission that used the same counter. The system tick is usually measured in milliseconds.

When the DF flag is set to TRUE, except for the TCP SYN/ACK (both the SYN flag and the ACK flag in TCP header are set to TRUE) packet whose IPID is assigned to 0, IP assigns IPID for other packets based on the second policy that is based on a per-socket counter unique to each connection. The per-socket counter is usually initialized to a random value. Then each time after a packet is transmitted using this counter, the counter increases by 1. The per-socket-based assignment policy is considered to be more secure and can avoid being observed from off-path attackers. Since the DF flag of TCP packets is be default set to TRUE to enable the path MTU discovery mechanism, TCP packets follow this per-socket assignment policy. Through studying the IPID assignment in Linux, we find that if the DF flag of TCP packets can be cleared (i.e., set to FALSE), then the IPID assignment to TCP packets will be downgraded from using the per-socket-based policy to the hash-based policy.

B. Path MTU Discovery

To avoid IP fragmentation, RFC 1191 [23] and RFC 1981 [24] propose a mechanism to discover path MTU (PMTU) between two end hosts, i.e., the minimum of all hops' MTUs in the entire packet transmission path. PMTUD relies on the DF flag. Before sending a packet, the originator sets the DF flag of the packet to TRUE, indicating that the packet is not allowed to be fragmented by intermediate routers. If the packet exceeds a router's next-hop MTU, the intermediate router discards it and issues an *ICMP Destination Unreachable* message (type 3) to the originator with the code *Fragmentation Needed and DF set* (code 4) in IPv4 or an ICMPv6 *Packet Too Big* message in IPv6, along with the router's next-hop MTU value carried in the ICMP message. After receiving the ICMP message, if the embedded packet in the message passes the originator's check, then the originator reduces the size of subsequent packets according to the carried next-hop MTU value in the message. The originator repeats the sending process until a packet with certain size could be forwarded to the destination, and it then sets the size as PMTU.

However, during this procedure, if an intermediate router's next-hop MTU is smaller than the originator's acceptable minimum PMTU *min_pmtu* that is a system variable in PMTUD implementations, the originator will resize the packet size to *min_pmtu*, clear the DF flag of subsequent packets, and then send them out. In RFC 1191 [23], *min_pmtu* is recommended as 576 octets. However, it varies in different implementations, e.g., 256 octets in FreeBSD, 296 octets in Mac OS, 552 octets in Linux, and 596 octets in Windows.

In most PMTUD implementations, hosts do not validate the source and transmission path of ICMP "Fragmentation Needed" messages (e.g., Linux kernel version 3.9 and beyond). Therefore, an off-path attacker can pretend to be a router and forge such an ICMP message specified with an extremely small next-hop MTU value. Actually the specified next-hop MTU value can be even set to 68 octets, the minimum of PMTU value on the Internet. After sending such a forged ICMP message to the originator, if the embedded packet in the forged ICMP message can pass the originator's check, the originator will be tricked into clearing the DF flag, thus downgrading the IPID assignment for TCP packets. According to RFC 792 [25], the forged ICMP message should embed at least 28 octets data to pass the originator's check. We will show that an ICMP echo reply packet can be embedded in the forged ICMP message to deceive the originator's check.

C. Challenge ACK Mechanism

To defeat blind in-window attacks on TCP, the challenge ACK mechanism was proposed as RFC 5961 [6]. In a nutshell, the challenge ACK mechanism requires that the sender of packets triggering the challenge conditions replies with the exact sequence number, not just one within the receive window. Thus, it can prevent an off-path attacker's blind injection unless the attacker is extremely lucky to be able to guess the exact sequence number with a probability of $1/2^{32}$. The challenge ACK mechanism is designed to enhance the security of TCP; however, we show that it can be abused to infer the state of a victim TCP connection.

Our attack exploits the challenge conditions in three aspects. First, if a receiver[1] sees an incoming SYN segment, regardless of the sequence number in the segment, it sends back an challenge ACK to the sender to confirm the loss of the previous connection. Only the legitimate remote peer will send a RST segment with the correct sequence number (derived from the ACK field of the challenge ACK packet) to prove that the previous connection is indeed terminated. Off-path attackers cannot answer this challenge with correct sequence number. We will show that this challenge condition can be abused to detect victim TCP connections.

Second, when a receiver sees an incoming RST segment, if the carried sequence number is outside the receive window, the receiver simply discards the segment. Instead,

[1] TCP is a full duplex protocol, hence the receiver may be the server peer or the client peer of the connection. We use the receiver here to refer to the peer that receives an incoming TCP segment.

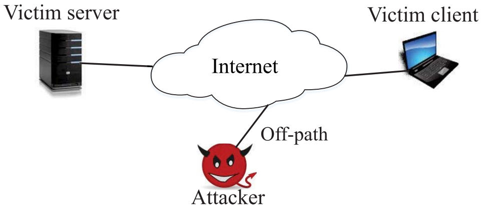


Fig. 2. Threat model.

if the sequence number is in-window but does not exactly match the expected next sequence number (i.e., $RCV.NXT$), the receiver will send a challenge ACK to the sender to confirm the reset action. We will show that this challenge condition can be abused to judge the correctness of the guessed sequence number.

Third, if a receiver sees an incoming ACK segment, it validates the acknowledgment number of the segment ($SEG.ACK$) with a window of $SND.UNA - SND.MAX.WND <= SEG.ACK <= SND.NXT$, where $SND.UNA$ is the sequence number of the first unacknowledged octet, $SND.MAX.WND$ is the maximum window size that the receiver has ever seen from its peer. The receiver considers that the acknowledgment number is legal and accepts it if the acknowledgment number is in this range. If $SEG.ACK$ is in the range of $[SND.UNA - (2^{31} - 1)$, $SND.UNA - SND.MAX.WND]$, i.e., the challenge ACK window, the receiver responds with a challenge ACK packet. We will show this challenge condition can be abused to judge the correctness of a guessed acknowledgment number.

III. Attack Overview

A. Threat Model

Figure 2 illustrates the threat model of our off-path TCP exploit. It involves three hosts, i.e., a victim client, a victim server, and an off-path attacker. The server and the client communicate based on a TCP connection, while the off-path attacker aims to hijack the connection. The off-path attacker cannot eavesdrop the traffic transferred between the server and the client as the man-in-the-middle attacker does. However, the attacker is capable of sending spoofed packets with the IP addresses of the server and the client. This capability assumption is practical, since at least a quarter of the Autonomous Systems (ASes) on the Internet do not filter packets with spoofed source addresses leaving their networks [28].

B. Attack Procedure

Our off-path TCP exploit consists of three main steps to hijack a victim TCP connection.

Step 1: Detecting Victim Clients. The attacker downgrades the server's IPID assignment from the per-socket-based policy to the hash-based policy. Then, through hash collisions, the attacker detects victim clients who share the same IPID counter with the attacker on the server side, i.e., the server uses the same hash-based IPID counter to assign IPID for TCP packets to the victim client and for packets to the attacker.

Step 2: Detecting TCP Connections. Once a potential victim client is detected, the attacker impersonates the victim client and sends spoofed SYN/ACK packets to the server. Then, by observing the change of the shared IPID counter, the attacker can determine the correctness of the specified source port number in the spoofed SYN/ACK packets and thus detect the presence of the TCP connection between the server and the victim client.

Step 3: Inferring Sequence and Acknowledgment Numbers. After a victim TCP connection is identified, the attacker sends spoofed RST packets and ACK packets to the connection, and triggers challenge ACK mechanism on the connection. By observing the changes of the shared IPID counter, the attacker can determine the correctness of the specified sequence number and acknowledgment number in the forged packets.

After correctly identifying the sequence numbers and acknowledgment numbers of the victim connection, the attacker can forge malicious TCP segments with the identified values and inject the segments into the victim connection to either reset the connection or poison the data stream. In the next three sections, we will detail the above three steps.

IV. Detecting Victim Clients

A. Downgrading the IPID Assignment

Linux assigns IPID for packets based on the DF flag. If the DF flag is set to TRUE, Linux will assign IPID for the packet based on a per-socket IPID counter; otherwise, based on a hash IPID counter. However, we observe that the DF flag can be maliciously cleared by off-path attackers, thus downgrading the IPID assignment. The attacker pretends to be a router and sends a forged ICMP "Fragmentation Needed" message to the victim server, indicating that a router between the server and the client has a smaller next-hop MTU and the packet is not allowed to be fragmented.

In order to trick the server into accepting the forged ICMP "Fragmentation Needed" message and clearing the DF flag of TCP packets sent to the client, the forged ICMP message needs to satisfy two conditions. First, the server does not validate the source of the ICMP message, i.e., the forged ICMP message from off-path attackers will not be discarded by the server. In practice, the validation requires extra functionality support from hardware devices [29], since major OSes, e.g., Linux 3.9 and beyond, do not perform the validation but directly accept the message. Second, the data embedded in the forged ICMP message must be able to evade the server's checks. RFC 792 [25] states that ICMP error messages should be embedded at least 28 octets (i.e., the IP header plus at least the first 8 octets) of the triggering packet, which is used by the server to match the message to the appropriate process. Moreover, according to the newer standard RFC 1812 [30], ICMP error messages should be embedded as much of the triggering packet as possible, but not exceeding 576 octets. Hence, the attacker has to craft and embed feasible data into the forged ICMP error message to evade the server's check.

To evade the server's check, we can embed the ICMP echo reply data into the forged ICMP "Fragmentation Needed" message, as shown in Figure 3. When servers equipped with Linux 3.9 and beyond receive such an ICMP "Fragmentation Needed" message embedded with an echo reply, the server does not check whether it sent the embedded echo reply data earlier. Instead, it directly responds to the forged ICMP error

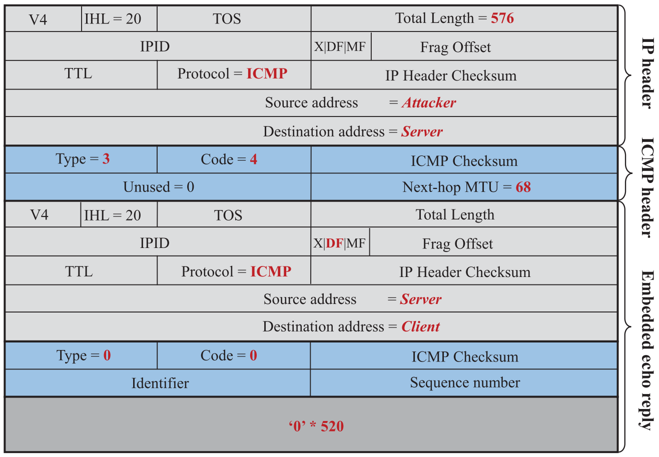


Fig. 3. Structure of the forged ICMP error message.

message and clears the DF flag of subsequent packets sent to the client whose IP address is specified in the embedded echo reply. Even if the server checks on the embedded echo reply data, it is easy for attackers to circumvent this checking. For example, the attacker can impersonate the client and initiate an ICMP echo request to the server, triggering the server to send an echo reply message. Then, the attacker pretends to be a router and sends an ICMP "Fragmentation Needed" message embedded with the known echo reply data to the server, tricking the server into accepting the forged message. Note the next-hop MTU value specified in the forged ICMP "Fragmentation Needed" message should be smaller than the server's acceptable minimum PMTU *min_pmtu*, as described in Section II-B. Actually, the value can be set to 68 octets, which is always smaller than the system variable of *min_pmtu* in various IP implementations. Besides, we find that a forged ICMP error message embedded with a GRE data [31] can also be used to trick the server into clearing the DF flag.

In a nutshell, it is difficult to verify the legitimacy of ICMP error messages on the Internet. Therefore, an attacker can forge an ICMP message and trick the server into accepting it. The forged ICMP error messages can force the server to clear the *DF* flag of packets sent to the victim client. Thus, the IPID assignment can be easily downgraded by attackers.

B. Constructing Hash Collisions

The server will assign IPID to the packets by using one of 2048 hash counters once the TCP packet's DF flag is cleared. The counter is selected based on four factors, i.e., *source IP address*, *destination IP address*, *protocol number* (e.g., 1 represents ICMP protocol, 6 represents TCP protocol) of the packet, and a *random value* generated on system boot. A hash index computed from the four factors is used to select one counter from the 2048 hash IPID counters. Hence, if the TCP packets sent to the victim client have the same hash value as the packets sent to the attacker, the server will use the same IPID counter to assign IPID for those packets to different destinations. Therefore, by constructing hash collisions using Equation 1, the attacker can detect the victim clients who share the same IPID counter on the server side. In other words, the shared IPID counter forms a side channel, which can be exploited to infer TCP connections between the server and the detected client. Our attack uses the ICMP protocol to detect hash collisions due to its simplicity and observability.

The procedure of detecting victim clients by constructing hash collisions is shown in Figure 4. First, the attacker pretends to be a router and sends a forged ICMP "Fragmentation Needed" message to the server, who will be tricked into clearing the *DF* flag and downgrading the IPID assignment of packets to the client whose IP address is specified in the forged ICMP message. Second, the attacker initiates ICMP echo requests to the server and observes the IPID of the reply packets. Then the attacker impersonates the client and sends a spoofed SYN packet to the server's listening port (e.g., 80). Following the three-way handshake process of TCP, the server will respond an SYN/ACK packet to the client. Here, the key difference is that if the attacker's IP address collides with the client (i.e., the attacker and the client share the same hash-based IPID counter at the server side), the IPID assignment for the SYN/ACK packet will incur an additional increment to the shared IPID counter, which can be observed by the attacker.[2] Otherwise, if there is no collision, the IPID observed by the attacker will be continuous distribution, i.e., without additional increment. Using this method, the attacker can identify victim clients who collide with its IP address and share the same IPID counter.

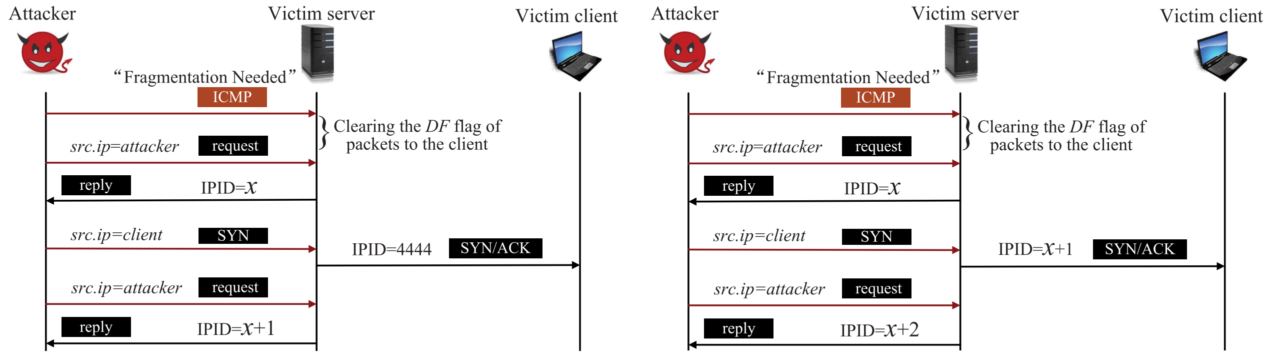
$$\begin{aligned} &hash(server\_IP, \boldsymbol{client\_IP}, TCP, Boot\_key) \\ &= hash(server\_IP, \boldsymbol{attacker\_IP}, ICMP, Boot\_key) \quad (1) \end{aligned}$$

The hash-based IPID counter does not increase linearly. Instead, the increment is a random value in a uniform distribution between 1 and the number of system ticks since the last packet transmission that used the same counter. Hence, if the attacker wants to force the counter to increase linearly and facilitate the observation, it needs to restrict the increment of system ticks related to the IPID counter. Usually, if more than 3 packets are sent to the server under 10 ms, the random value added to the IPID counter will never be larger than one [21]. We send ICMP request packets to the server in parallel and restrict the increment of system ticks. Our experiments show that if the round-trip time (RTT) from the attacker to the server is within 200 ms, the attacker only needs to send less than 300 packets per second to force the hash-based IPID counter increasing linearly.

There are totally 2048 hash-based IPID counters in Linux, and the probability of hash collisions between the attacker and the client is a geometric distribution. When the protocol is specified as ICMP (see Eq. 1), if the attacker has k IP addresses, the probability of collisions between the attacker and the target client is $1-(1-p)^k$, where p equals 1/2048. To construct an attack in practice, the attacker has two strategies to detect victim clients by leveraging hash collisions.

Attacking Potential Targets: If the attacker has only one or a few IP addresses, the attacker can detect potential victim clients who collide with the attacker. In theory, if the attacker has only one IP address, on a target server, the number of

[2] A special case is that the source port in a spoofed SYN packet happens to match the source port of a TCP connection from the client to the server. In this case, the spoofed SYN packet will trigger a challenge ACK packet, instead of a SYN/ACK packet. However, the attacker can still observe an additional increment to the shared IPID counter.



(a) No hash collisions with the client

(b) Hash collisions with the client

Fig. 4. Detecting potential victim clients through hash collisions.

clients conflicting with the attacker is $2^{32}/2048 = 2^{21}$. Since the detection only depends on the server side, the attacker can create a list of IP addresses and select one from the list as the client's IP address each time. Following the procedure in Figure 4, the attacker can determine if the selected one is a victim. In this way, the attacker can identify and enumerate all the potential victim clients who are vulnerable to its IP address. In our test, an attacker can detect more than 20 victim clients within 6 minutes using one IP address (see Section VII-A).

Attacking Arbitrary Targets: If the attacker has enough number of IP addresses, it can attack arbitrary TCP connections by alternating its IP addresses to generate the hash collision. According to the geometric probability distribution, if the attacker has more than 2048 IP addresses, it can collide with any clients with above 63.2% probability. Especially, for servers running IPv4/IPv6 dual-stack, attackers can use adequate IPv6 addresses to construct hash collisions with arbitrary target clients, since both IPv4 and IPv6 addresses share the same 2048 hash-based IPID counters (see Section VII-D for implementations and evaluations of our attack in dual-stack networks).

The detected hash-based IPID counter shared with the victim client is stable. That is, if the server does not restart (i.e., the *Boot_key* in Eq. 1 is not altered), the client's TCP connection will always share this IPID counter with the attacker. In practice, servers (e.g., web servers and BGP routers) do not restart frequently. Hence, the attacker can detect shared IPID counters and victim clients in advance, regardless of if TCP connections have been established.

Note Linux assigned IPID to RST packets based on hash counters before version 4.18, and thus an attacker can observe its IPID distribution to determine if it shares the same counter with the client by spoofing SYN/ACK packets [21]. This vulnerability has been fixed since Linux 4.18 by always setting the IPID of RST packets to 0, incurring no changes on any IPID counters. However, we find that after the critical step of downgrading the IPID assignment, an attacker can still detect hash collisions through forging SYN packets and triggering the server to respond SYN/ACK packets and then identify a victim client. Moreover, we will show that the fix of assigning 0 to RST packets introduces yet another vulnerability, which can be exploited by a pure off-path attacker to detect the presence of victim TCP connections (see Section V).

V. Detecting TCP Connections

Once a victim client is identified, the attacker can learn the presence of TCP connections between the client and the server. A TCP connection is identified by a four-tuple, i.e., [source IP address, source port number, destination IP address, destination port number]. Usually, the destination IP address, and port number are public known, so an attacker only needs to infer the source IP address and source port number. In our attack, since the victim client can be detected by using hash collisions, the only missing tuple is the source port number.

Assuming that a TCP connection from source port *y* has been established earlier by a legal user in the victim client, an off-path attacker can identify this port number by sending out probing packets. First, the attacker continuously sends ICMP echo request packets to the server and observes the IPID values of the reply packets from the server. Then, the attacker impersonates the victim client and sends a forged SYN/ACK packet with a guessed source port number to the server. If the source port number specified in the SYN/ACK packet does not equal *y*, according to the TCP specification [1], the server will respond a RST packet to the client. Due to the patch fixing the vulnerability identified by Alexander *et al.* [21], Linux kernel versions 4.18 and beyond assign an IPID of 0 to the RST packet, which will not incur an increment to the shared IPID counter. Hence, the IPID values in the reply packets observed by the attacker are continuous.

If the guessed source port number specified in the forged SYN/ACK packet equals *y*, the challenge ACK mechanism [6] makes the server send a challenge ACK packet to the victim client for confirming the legitimacy of the SYN/ACK packet. The IPID in the challenge ACK packet will be assigned based on the shared IPID counter, which will incur an additional increment to the counter. Thus, from the view of the attacker, the IPID values in the reply packets from the server will not be continuous.

The attacker repeats the above procedure, i.e., changing the source port number specified in the forged SYN/ACK packet and then observing the IPID of the reply packets, until the correct port number *y* is identified. Finally, the TCP connection running on the identified four-tuple is all known to the attacker. In practice, the attacker can adopt a parallel approach to facilitate the identification of the source port, i.e., to search the source port number by sending multiple probing packets in a certain range during a period. If the source port is in the range, the shared IPID counter will have an additional increment, so the attacker can further narrow the range. Otherwise, the attacker can detect another port range in parallel. In addition, the attacker can use a binary-search-like algorithm [8], [9] to further reduce the detection time cost.

VI. Inferring Sequence and Acknowledgment Numbers

A. Inferring Acceptable Sequence Number

To infer the acceptable sequence numbers on the server side, the attacker continuously sends ICMP request packets to the server and observes the IPID values of the reply packets. Then, the attacker impersonates the victim client to send a spoofed RST packet to the server. The RST packet is specified with the guessed sequence number *seq*. We need to consider two cases: (i) *seq* not in the server's receive window and (ii) *seq* in the server's receive window. According to the challenge ACK mechanism described in Section II-C, in the first case, the server will discard the spoofed RST packet directly, so it does not influence the shared IPID counter. In the second case when the guessed *seq* is in the server's receive window, the server will respond to this RST packet and send a challenge ACK packet to the victim client to confirm the legitimacy of the packet. The IPID of this challenge ACK packet is assigned based on the shared IPID counter, and it will incur an additional increment to the counter. The attacker can observe the increment and then determine that the guessed *seq* is located in the server's receive window.[3]

In practice, in order to reduce the time cost of sequence number inference, the attacker can divide the sequence number space into multiple blocks whose sizes are equal to the default receive window size in Linux (87380 octets), probing only once per block. Besides, the attacker can apply parallel search methods similar to those used in connections detection to further reduce the time cost.

B. Locating the Challenge ACK Window

According to RFC 5961, when a segment arrives at the server, the server also checks the segment's acknowledgment number even if its sequence number is in the server's receive window. There are three cases in the whole acknowledgment number space: (i) the acknowledgment number in challenge ACK window, (ii) in the acceptable ACK range, and (iii) invalid acknowledgment numbers. In the first case, the server will issue a challenge ACK packet to confirm the legitimacy of the triggering segment. In the second case, the server will accept the segment directly. Otherwise, if the segment carries an invalid acknowledgment number, the server will discard it silently. The last two cases cannot be differentiated directly because it cannot be observed from an off-path attacker. However, the attacker can first identify the challenge ACK window of the server and then infer the acceptable ACK numbers.

When locating the challenge ACK window, the attacker observes and records the shared IPID counter. Then the attacker impersonates the victim client and sends a spoofed ACK packet with a guessed acknowledgment number ack to the server, the packet is also specified with an acceptable sequence number detected previously. If ack is in the challenge ACK window of the server, a challenge ACK packet will be issued, incurring an additional increment to the shared IPID counter. Instead, if ack is not in the challenge ACK window, the observed IPID will be continuous from the view of the attacker. In practice, the challenge ACK window size is always between 1G and 2G [8], [9], [32], i.e., one quarter of the entire acknowledgment number space. Hence, to facilitate the detection, the attacker can divide the entire space into 4 blocks and probe each block to check which block the challenge ACK window falls in.

C. Detecting the Exact Sequence Number

Now we present the method of detecting the exact sequence number (i.e., $RCV.NXT$, the lower boundary of the server's receive window) based on the previous inferred results. The attacker can forge multiple ACK packets with a constant acknowledgment number $ack\_challenge$ in the challenge ACK window and the specified sequence number in each ACK packet set to $seq\_acceptable - i$, where $seq\_acceptable$ is an acceptable sequence number inferred previously. Then the attacker impersonates the victim client to send these forged ACK packets to the server. In the beginning, the server will be triggered to send challenge ACK packets at a rate of one packet per 500 ms due to the rate limit of challenge ACK, so the triggered challenge ACK packets will incur regular increments to the shared IPID counter. However, once the specified sequence number $seq\_acceptable - i$ reaches $RCV.NXT$ (the lower boundary of the server's receive window), the server will switch to send duplicate ACK packets, which is not enforced by any rate limit. Thus, the shared IPID counter will have a jitter,[4] and the attacker can observe this jitter and then detect the exact sequence number. The detecting procedure has no side effects, e.g., resetting the connections, on the connections.

D. Detecting Acceptable ACK Number

Once an acknowledgment number $ack\_challenge$ in the challenge ACK window is identified, the attacker can also detect the boundary of the challenge ACK window by sending multiple probing ACK packets and then observing the shared IPID counter, similar to detecting the lower boundary of the server's receive window. The forged probing ACK packets are specified with a constant sequence number $seq\_acceptable$, and the acknowledgment number of each ACK packet is set to $ack\_challenge - i$. In turn, the attacker sends these forged ACK packets to the server. Challenge ACK packets will be triggered until $ack\_challenge - i$ reaches the lower boundary of the challenge ACK window. Once this boundary is detected, then $SND.UNA$ can be easily inferred, i.e., adding $2G$ to the detected boundary. $SND.UNA$ is in the acceptable ACK range. When all the data sent earlier has been acknowledged, $SND.UNA$ equals $SND.NXT$. Instead, if the server has an amount of data to be sent to the client, $SND.NXT$ can also

[3] In a special case when *seq* exactly matches the server's $RCV.NXT$, the server will reset the connection directly. However, the probability of this case occurring is $1/2^{32}$, which is negligible.

[4] The increments to the shared IPID counter become 20 per 500 ms in our experiments, instead of 1 per 500 ms.

be inferred by adding the typical size of the send window to $SND.UNA$, e.g., 16384 octets in Linux by default.

VII. Implementation and Evaluation

Ethical Considerations: In order to avoid causing real damages or negative impacts on the Internet, we choose not to directly attack real users and their hosts. All the hosts involved in the experiments are our machines. We evaluate the impacts of our off-path TCP attacks on the Internet, e.g., measuring Alexa top 100k websites to identify potential victim servers that are vulnerable to our attack. However, we do not exploit the vulnerability of these web servers for real attacks.

A. Identifying Victim Clients

We show two scenarios of identifying victim clients via hash collisions after downgrading the server's IPID assignment. First, we show how to detect potential victim clients using one IP address. Second, we illustrate that the attacker can attack arbitrary victim clients if having enough IP addresses.

Experimental Setup: Two types of hosts are used in this experiment. A server listening on port 80 and waiting for TCP connection requests is equipped with Ubuntu 18.04 (kernel version 5.5) with a prefix of 152.136.0.0/16. Attack machines locate in different positions with independent IP addresses. We use these IP addresses to detect victim clients to the server. The attack machines are equipped with Ubuntu 18.04 (kernel version 4.15) and are able to send packets to the server with spoofed IP addresses.

Results With One Attacker IP Address: When having only one IP address, an attacker can detect more than 2 million victim clients to the server. In this scenario, we deploy one attack machine and assign IP address from a target IP list, which contains several prefixes owned by different organizations. The attack machine clears the DF flag of the server's TCP packets to IP addresses in these prefixes via forging ICMP "Fragmentation Needed" messages, and then it scans the target prefixes to identify potential victim clients that share the same hash-based IPID counter with the attack machine on the server. The experimental results are shown in Table I. By using only one attack machine (i.e., one attacker IP address), the attacker can detect a considerable number of potential victim clients in different organizations. For instance, as shown in the last row of Table I, the number of victim clients that collide with the attack machine are 179 in the prefixes of 3.208.0.0/12 (owned by Amazon). The time costs of identifying these victim clients are 30.0 minutes, i.e., it takes 10.1 seconds on average to detect a victim client. The average outbound traffic of the attack machine is 125.14 KB/s (i.e., around 584 packets/s). Thus, it is difficult to detect the malicious probing, e.g., by leveraging network traffic monitoring systems.

Results With Multiple Attacker IP Addresses: When the attacker has multiple IP addresses, it aims to attack an arbitrary client to the server using these IP addresses. After selecting the target client, the attacker clears the DF flag of the server's TCP packets to the client by forging an ICMP "Fragmentation Needed" message. Then, the attacker detects addresses in its address pool to find a correct one that collides with the target client, i.e., sharing the same hash-based IPID counter.

TABLE I

Detecting Victim Clients Using One IP Address

Prefix owner	Prefix	Location	Victim clients	Time (min)
eBay	209.140.128.0/18	US, CA	4	1.32
Yahoo Japan	124.83.128.0/17	JP, TKY	9	2.70
Google	74.125.0.0/16	US, CA	23	5.56
Tencent	162.14.0.0/16	CH, BJ	27	5.54
Facebook	157.240.0.0/16	US, CA	33	5.53
Alibaba	47.56.0.0/15	CH, HK	57	11.3
Amazon	50.16.0.0/14	US, WA	121	22.9
China Telecom	101.80.0.0/12	CH, SH	156	30.0
Amazon	3.208.0.0/12	USA, VA	179	30.0

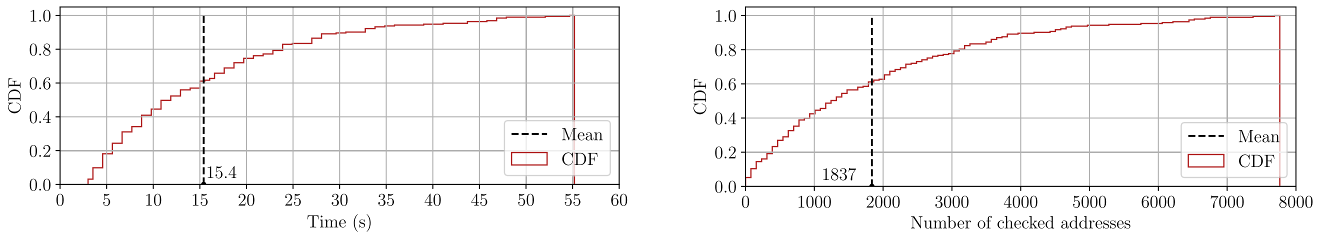
In this experiment, we select different target clients and repeat the detecting process 200 times. The empirical cumulative distribution function (CDF) of the time cost and the number of required attacker IP addresses are shown in Figure 5. For an arbitrary target client, the average time cost to detect a correct IP address in the attacker's address pool is 15.4 seconds, and the number of IP addresses needed is 1,837 on average. Furthermore, after spending 24 seconds to check 3,000 addresses, the attacker has a probability of 80.0% to identify a correct IP address that can be used to attack an arbitrary client. The measured probability is higher than the theoretical one, i.e., $1-(1-1/2048)^{3000} \approx 76.9\%$.

To evaluate the threats of our off-path TCP attack on the Internet, we measure Alexa top 100k websites to identify how many websites suffer from the vulnerable IPID assignment. We observe that 22,953 websites are vulnerable to forged ICMP "Fragmentation Needed" messages from off-path attackers and thus can be tricked into clearing the DF flag of TCP packets and downgrading the IPID assignment. These websites are vulnerable to our attack. We cannot confirm the effectiveness of the attack against 22,803 websites that are unreachable from our vantage point in California. The unreachability of these websites is mainly caused by censorship [33] and ISP filter rules [34]. Moreover, we suspect that the rest resist to our attack due to two reasons, i.e., the OS versions of the websites are invulnerable (e.g., old Linux kernel versions or Windows), or the forged ICMP error messages are blocked.

B. TCP DoS Attacks

Experimental Setup: This attack involves 3 hosts, namely, an SSH server equipped with Ubuntu 18.04 (kernel version 4.18 or beyond), OpenSSH 7.6 and OpenSSL 1.0.2, a victim client who accesses the server based on SSH connections, and an attack machine equipped with Ubuntu 18.04 (kernel version 4.15) and a prefix of 152.136.0.0/16 that contains 2000 IP addresses in this prefix. The attack machine can use these IP addresses to detect hash collisions with the target client. The attacker attempts to reset the connection via sending TCP RST packets to the server.

Attack Procedure: In this attack, the 3-tuple [client IP address, server IP address, server port] is known. First, the attacker identifies an IP address in its prefix which collides with the client IP address. Second, based on the identified attacker IP address, the attacker infers the correct client port number and the exact sequence number ($RCV.NXT$ on the



(a) Empirical CDF of time cost

(b) Empirical CDF of the number of attacker IP addresses

Fig. 5. Empirical CDF of time cost and the number of attacker IP addresses needed to detect a hash collision.

TABLE II
EXPERIMENTAL RESULTS OF SSH CONNECTION RESET

Server address	Linux version	Time cost (s)	Bandwidth cost (KB/s)	Success rate
62.234.203.×	4.19	148.7	25.02	9/10
152.136.49.×	4.20	150.2	25.03	10/10
152.136.59.×	5.3	160.1	24.95	9/10
62.234.182.×	5.5	156.3	25.20	44/50

server side) of the victim SSH connection. Finally, a spoofed `RST` packet specified with the inferred value is sent to the server, and the server will be tricked into resetting the SSH connection from the victim client. In this attack, the acceptable acknowledgment numbers are not needed.

Experimental Results: On average, the time cost of identifying a correct attacker IP address is 15.4 seconds, and the correct one will be identified after checking 1,837 IP addresses on average. Table II illustrates our experimental results. We test the attack against 4 servers that are equipped with Linux kernel version 4.19, 4.20, 5.3, and 5.5, respectively. The diversity of servers ensures the feasibility and effectiveness of the attack. The average time cost of resetting an SSH connection is 155 seconds, and the success rate is over 88%. TCP connections DoS attack is particularly applicable to compromising applications secured by encrypted traffic, e.g., HTTPS and SSH.

C. TCP Manipulation Attacks

In this case, we perform two attacks to demonstrate that the newly discovered IPID side channel can be exploit to manipulate a TCP connection maliciously, thus causing serious damage to the upper applications including HTTP and BGP.

(1) Manipulating Web Traffic: We demonstrate that under the typical web application scenario, an off-path attacker can detect a victim client connecting to the target web server and then hijack the connection between the server and the client.

Experimental Setup: This attack involves 3 hosts. A web server is equipped with Linux kernel version 5.5 and a popular real-time communication web application called Rocket.Chat [35]. An attack machine is equipped with Ubuntu 18.04 (kernel version 4.15), and it is able to send packets to the server with a spoofed IP address. A client can access the web server based on HTTP. Note the OS type or version of the client is unrestricted in our attack. The attacker attempts to identify the potential victim client and hijack the TCP connection between the server and the client. For instance, the attacker may impersonate the victim client to inject malicious segments into the server and then inject fake messages into the chatting group. Here, the server IP address and server port are publicly known.

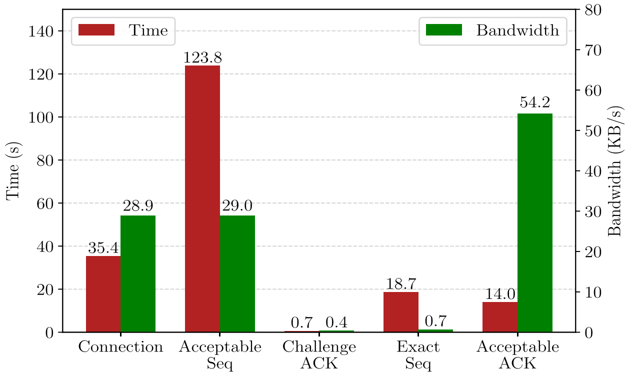
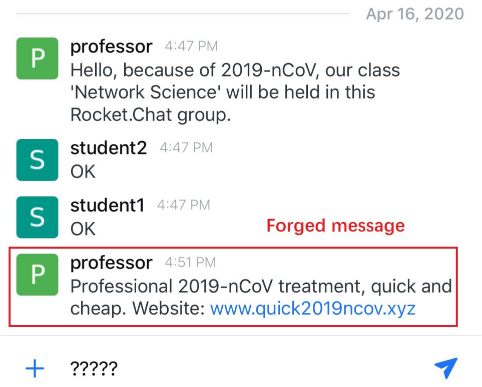


Fig. 6. Time/Bandwidth overheads of web manipulation.

Attack Procedure: The attacker first downgrades the server's IPID assignment and detects potential victim clients who share the same hash-based IPID counter with the attacker. Next, the attack can be constructed in the following four steps: (1) detecting whether the client has a TCP connection to the server, i.e., identifying correct source port number to obtain the TCP 4-tuple information, (2) inferring the exact sequence number, i.e., $RCV.NXT$ on the server, which can slide the server's receive window, leading to that the segment can be delivered to HTTP immediately, (3) inferring the acceptable acknowledgment numbers, and (4) injecting forged segments specified with the inferred values into the server and pushing fake messages into the chatting group.

Experimental Results: It takes 14.0 seconds to detect a potential victim client and 35.4 seconds to identify the correct source port number of the TCP connection. Figure 6 shows that the time cost in inferring the acceptable sequence numbers, locating the challenge `ACK` window, detecting the exact sequence number and an acceptable acknowledgment number are 123.8 seconds, 0.7 seconds, 18.7 seconds and 14.0 seconds, respectively. On average, the overall time cost of this attack is 206.6 seconds, including the time cost of detecting the victim client that can be performed in advance. 64.3% of the overall time is spent on inferring the acceptable sequence numbers. The reason is that the server's receive window is relatively narrow and the attacker must sample a large number of sequence numbers. The average bandwidth overhead of this attack is 23.55 KB/s. When detecting acceptable ACK numbers, the probing packets are padded with crafted application data, hence they consume a large portion of the bandwidth. Finally, when the server accepts the forged segment, the fake messages in the segment will



Apr 16, 2020

P professor 4:47 PM
Hello, because of 2019-nCoV, our class 'Network Science' will be held in this Rocket.Chat group.

S student2 4:47 PM
OK

S student1 4:47 PM
OK

Forged message

P professor 4:51 PM
Professional 2019-nCoV treatment, quick and cheap. Website: www.quick2019ncov.xyz

Fig. 7. Snapshot of messages viewed by group members.

be stored on the server and advertised to group members, as shown in Figure 7. Overall, the success rate of this attack is over 90%.

(2) Manipulating BGP Routing Table: Customized security mechanisms for BGP have not been widely deployed yet [36], hence BGP messages undertaken by long-lived TCP connections may be poisoned by leveraging our attacks.

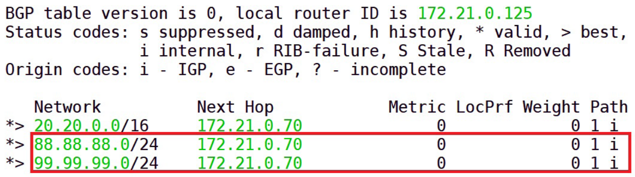
Experimental Setup: Due to ethical concerns, we do not probe and attack BGP systems on the Internet. Instead, we evaluate our attacks against BGP systems in our test-bed. This attack involves 3 hosts. A BGP server is equipped with Linux kernel version 5.5, listening on its port 179. A BGP client is equipped with Ubuntu 18.04. Both the server and the client run the BGP suite of Quagga [37] with version 1.2.0. After the client initiates a BGP connection, the two peers advertise BGP messages to each other and update their BGP route tables. An attack machine is equipped with Ubuntu 18.04, and it is able to send packets with a spoofed IP address. The attacker aims to identify the potential victim client and hijack the BGP connection. We show that the attacker can impersonate the client and manipulate the server's BGP route table. We assume that the server IP address and server port are publicly known.

Attack Procedure: Similar to the HTTP hijacking attack, after downgrading the server's IPID assignment and identifying a victim client, the attacker first learn the presence of a BGP connection between the server and the identified client. Then, it infers the exact sequence number and an acceptable acknowledgment number to the server. Finally, the attacker sends forged BGP messages to the server based on the inferred values to poison the routing table.

Experimental Results: Figure 8 presents a snapshot of the poisoned BGP routing table. The Network Layer Reachability Information (NLRI) of network "99.99.99.0/24" and "88.88.88.0/24" is fake, which are not advertised by router 172.21.0.70 but injected by the attacker. On average, the attacker can finish BGP routing table poisoning in 214.3 seconds (including the time cost of identifying the victim client), with a success rate over 90%.

D. Implementation and Evaluation in Dual-Stack Networks

As we presented in Section VII-A, if the attacker aims to hijack the victim TCP connection between the vulnerable server and an arbitrary client, the attacker needs 1,837 IP addresses on average to construct the hash collision and identify the shared IPID counter. In practice, it is not easy for an attacker to hold such a number of IPv4 addresses, limiting the effect of the proposed attack [38]. However, we discover that if the vulnerable server enables IPv4/IPv6 dual-stack, the attacker can easily use adequate IPv6 addresses to construct hash collisions and identify the hash counter used in the victim TCP connection.



```
BGP table version is 0, local router ID is 172.21.0.125
Status codes: s suppressed, d damped, h history, * valid, > best,
              i internal, r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*> 20.20.0.0/16     172.21.0.70              0             0 1 i
*> 88.88.88.0/24    172.21.0.70              0             0 1 i
*> 99.99.99.0/24    172.21.0.70              0             0 1 i
```

Fig. 8. Snapshot of the poisoned BGP routing table.

On Linux systems running IPv4/IPv6 dual-stack, the 2048 globally shared hash IPID counters are also used by IPv6 addresses. Therefore, once the outgoing IPv6 packets are fragmented (IPID is needed only when IPv6 packets are fragmented at the originator), Linux will assign IPID for the fragments by choosing one from the 2048 global counters. Hence, once the IPID assignment for the target TCP connection between the vulnerable server and the victim client undertaken by IPv4 is downgraded to hash-based policy, the attacker can use its IPv6 addresses to construct hash collisions and identify the target hash counter used in the victim connection. Next, by observing the side channel of the shared IPID counter, the attacker can hijack the victim connection and overcome the lack of IP addresses.

It is worth noting that the attacker needs to force the IPv6 echo reply packets (from the server to the attacker) to be fragmented when observing and recording the server's IPID, since only fragmented IPv6 packets will contain an IPv6 extension header where an IPID field is carried. The attacker can impersonate intermediate routers to issue ICMPv6 "Packet Too Big" messages [39] to the vulnerable server and trick the server into accepting the forged messages. After the attacker fill the IPv6 echo request packets greater than 1500 octets (supposing the vulnerable server is connected to Internet via Ethernet), the subsequent reply packets will be fragmented and IPID will be assigned before being sent to the attacker. As a result, the attacker can observe the server's IPID assignment using its IPv6 addresses, thus successfully constructing hash collisions and identifying the target hash IPID counter shared with the victim client on the server side (via varying the source IPv6 address) to perform our off-path TCP hijacking attack.

We first measure vulnerable IPv4/IPv6 dual-stack servers in Alexa top 100k websites. We discover that 73,216 (more than 73.2%) of the top 100k websites enable IPv4/IPv6 dual-stack, and 14,329 (more than 14%) of them are vulnerable to our attacks. Considering the previous measurement result, i.e., a total of 22,953 websites in Alexa top 100k are vulnerable, we can see that 8,264 vulnerable websites only run IPv4. Aiming at vulnerable IPv4/IPv6 dual-stack servers, we can use 1,627 IPv6 addresses on average to identify the target hash IPID counter shared between our attack machine and a victim TCP connection to a IPv4 client under our control. In practice, IPv6 users are usually assigned an IPv6 prefix of 64-bit, which

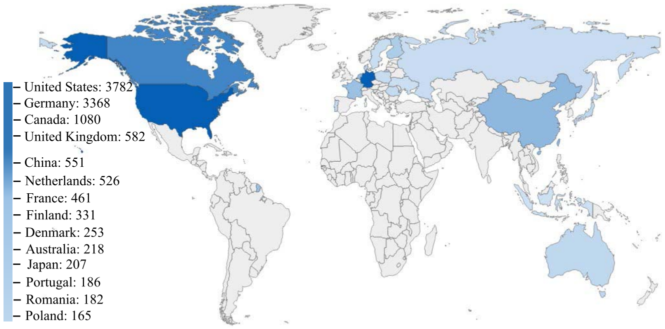


Fig. 9. Geographical distribution of vulnerable dual-stack websites.

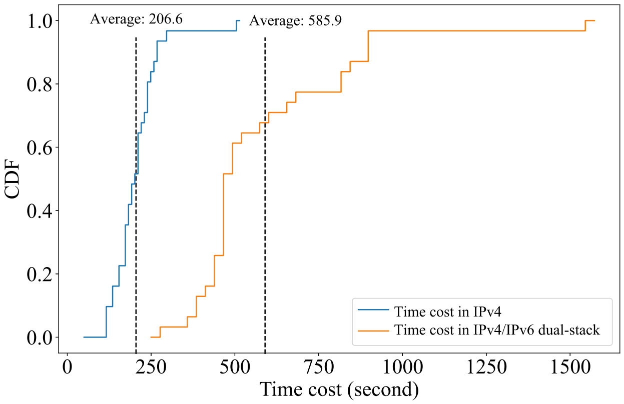


Fig. 10. CDF of the overall time costs in IPv4/IPv6 dual-stack and in IPv4.

can easily meet the requirement for adequate IPv6 addresses to construct hash collisions. Figure 9 presents the geographical distribution of the vulnerable IPv4/IPv6 dual-stack websites we detected.

We also compare the costs of implementing our attack in IPv4/IPv6 dual-stack with those in single IPv4 stack. Figure 10 shows the CDF of the time costs when implementing our attack in IPv4/IPv6 dual-stack and in single IPv4 stack. The average time costs of implementing our attack in IPv4/IPv6 dual-stack stack is about 585.9 seconds, i.e., 380 seconds more than the average time costs in single IPv4 stack, which is mainly due to the time-consuming packet fragmentation and defragmentation we triggered on IPv6 echo reply packets in order to observe the IPID in IPv6.

Figure 11 compares the attack traffic bandwidth in IPv4/IPv6 dual-stack and in single IPv4 stack. The attack traffic bandwidth in IPv4/IPv6 dual-stack is much higher than that in single IPv4 stack, mainly because in IPv4/IPv6 dual-stack, the echo request and reply packets that are used to identify and observe the shared hash IPID counter in the vulnerable server are always filled greater than 1500 octets (under which IP fragmentation will happen and thus IPID is enabled and assigned in the reply packets). In contrast, when performing our attack in single IPv4 stack, the size of echo request and reply packets can be less than 60 octets, since the IPID field is always enabled and assigned in IPv4 networks.

VIII. Discussion and Countermeasure

A. Attack Robustness

(1) IPID Noises: Unlike the global IPID counter that is noisy due to the sharing between all outgoing traffic, the downgraded

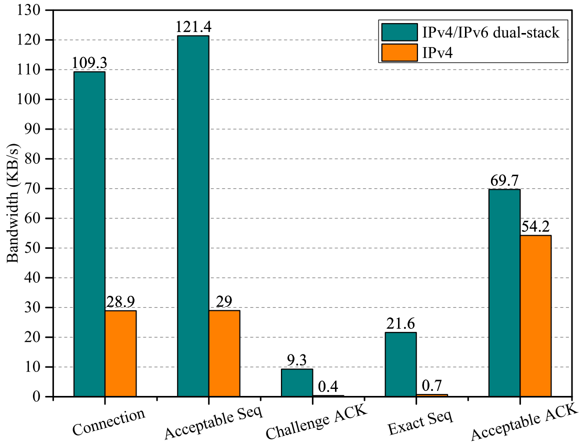


Fig. 11. Bandwidth of attack traffic in IPv4/IPv6 dual-stack and in IPv4.

hash-based IPID counter shared between the attacker and the victim client on the TCP server side is reliable, and hence our attack does not suffer the traditional noise issue [16], [17], [26], [27].

Since the outgoing TCP traffic directed to irrelevant clients uses per-socket-based IPID counters instead of the hash-based IPID counters, irrelevant TCP traffic will not disturb our attack. We measure and evaluate the disturbances from the noises of non-TCP traffic in the real world and find that the impact is also limited. We find that the hash-based IPID counters of more than 91% vulnerable websites in the Alexa top 100k websites list are not disturbed at all (within 5 minutes in our experiment), which means the IPIDs are always contiguous and there is no outgoing traffic sharing the same hash-based IPID counter with the attacker at the server in this time window. Note that, less than 9% vulnerable websites are disturbed by non-TCP traffic, e.g., the ICMP traffic generated by these websites, that happens to share the same hash-based IPID counter with the attack traffic during the period. Considering that our attack can be finished within 215 seconds on average, the real disturbance is negligible. Moreover, other types of noises that specific to certain application/network scenarios, e.g., packet loss, can be effectively mitigated by re-running the attack multiple times. For example, in our experiments, when we detect a potential victim, we usually conduct the detection process again to enhance the confidence, which incurs around 6 seconds additional delay but can almost eliminate the false positives. We confirm the results by generating random packet loss in our experiments.

(2) More Victim Clients Under the NAT Scenario: Network Address Translation (NAT) is a widely used technique to overcome the shortage of IPv4 addresses [40]. Under this scenario, multiple hosts share a public IP address. As a result, if the attacker identifies a potential victim client who accesses the Internet via NAT technique, it indicates that all hosts behind the same NAT gateway are potential victims. Hence, in practice, the actual number of victim clients is far greater than the number of being identified, and the NAT technique incurs a more wide attack surface.

(3) Shifting Sequence and Acknowledgment Numbers: A circumstance that may affect the success rate of our attack is the shifting of the sequence and acknowledgment numbers, i.e., if

the victim TCP connection has ongoing traffic, the acceptable sequence and acknowledgment numbers may shift as the attack is in progress. This problem can be solved by the repeated inference of the acceptable sequence and acknowledgment numbers. We confirm that if the receive window does not slide very quickly, e.g., under the scenarios of SSH and BGP, the success rate of the attack will not be affected obviously. Even if the receive window slides quickly enough to break the attacker's inference, the attacker can choose to exploit the other side of the TCP connection where the receive window slides more slowly.

B. Countermeasures

We have reported the newly discovered IPID side channel to the Linux community. Meanwhile, we also propose to throttle the exploit via eliminating the root cause.

(1) Assigning IPID Based on the `Protocol` Field: The root cause of the attack is that Linux can be tricked into choosing an incorrect IPID assignment policy for TCP packets. When Linux assigns IPID to TCP packets, it decides which policy to be chosen based on the `DF` flag in IP header, rather than the `Protocol` field. Therefore, attackers can clear the `DF` bit of the TCP packets by forging ICMP "Fragmentation Needed" messages, which causes hash collisions and build a side channel. To address this issue, we propose to assign IPID by evaluating if a packet is originated from TCP based on the field of `Protocol` in IP header, instead of the `DF` flag. If the packet's `Protocol` field is specified as TCP, we assign IPID for the packet based on the per-socket assignment policy. As a result, all TCP packets issued from the server will no longer share IPID counters with the attacker and the side channel can be eliminated, no matter the server enables the IPv4/IPv6 dual-stack or only the IPv4 stack. We implement the mechanism in Linux 4.18 and confirm its effectiveness through real evaluation.

(2) Enhancing IPID Assignment for `RST` Packets: Another countermeasure is to change the IPID assignment of `RST` packets. Since Linux kernel version 4.18, Linux directly sets the IPID of `RST` packets to 0. When an attacker learn the presence of a TCP connection between the server and the identified victim client, the attacker can forge `SYN/ACK` packets. If there is no connection initiated from the specified source port, the server responds with a `RST` packet, otherwise, with a challenge `ACK` packet. The IPID of the `RST` packet is 0, and hence it will not cause an increment to the shared IPID counter. It will be different from the behavior of the challenge `ACK` packet, which enables an indicator for the attacker to judge the existence of the connection.

Thus, we propose to modify the IPID assignment for `RST` packets. Note we cannot assign IPID for `RST` packets based on a socket preserved counter, since the `RST` packets may be generated and issued independently of a TCP connection. Also, we cannot assign IPID for `RST` packets based on hash IPID counters, since this assignment is vulnerable to previous TCP/IP connections detecting attacks [21]. An empirical method is to assign IPID for `RST` packets based on the destination of the packet. If there is a TCP connection to the destination (the victim client in our scenario) and the counter preserved in the socket will be selected, it can avoid the differences on the counter.

IX. Related Work

IPID Side Channels: IPID ensures the uniqueness of a packet for packet fragmentation and reassembly [14], [15]. However, IPID has been widely abused to conduct off-path attacks due to the vulnerable assignment methods. Ensafi *et al.* performed idle port scan and network protocol analysis by leveraging the side channel of global IPID counters [16]. They also suggested that the global IPID counters can be used to detect intentional packet drops [17]. By leveraging the side channel of global IPID counters, Pearce *et al.* measured the reachability between any two Internet locations without controlling a measurement vantage point [26], [27], and the blind TCP hijacking attacks was demonstrated can be successfully performed against early OSes of Windows 2K, Windows XP and FreeBSD 4 [41]. Jeffrey *et al.* showed that per-destination IPID counters are also vulnerable, which can be exploited to infer the number of packets between two machines with UDP and ICMP and even learn the presence of a TCP connection by launching off-path attacks [19]. Alexander *et al.* detected TCP connections via IPID hash collisions. They leveraged the IPID of the triggered `RST` packets to determine the presence of a victim TCP connection [21]. Their method can only detect TCP connections, but not hijacking a TCP connection. Moreover, the vulnerabilities they used have been fixed since Linux kernel version 4.18. In this paper, we identified a new vulnerability of abusing IPID, which can be exploited to perform an off-path TCP hijacking attack.

TCP Hijacking Attacks: Cao *et al.* found that an off-path attacker can infer whether two arbitrary hosts on the Internet are communicating using a TCP connection by utilizing a side channel in the challenge `ACK` mechanism, identify the sequence and acknowledgment numbers of the connection, and then hijack the connections [8], [9]. The side channel vulnerability has been eliminated by setting a random challenge `ACK` count limit. A timing side channel has been uncovered in the half-duplex IEEE 802.11 or Wi-Fi technology, which can be exploited by an off-path attacker to inject data into a TCP connection and force the browser to cache malicious objects [10]. By exploiting the global IPID counter which was adopted by the previous Linux and Windows systems, Gilad *et al.* inferred if two hosts have established a TCP connection identified by a specific four-tuple and then launch off-path TCP injection attacks [7], [42]–[44].

Besides, unprivileged applications (called puppets) controlled by attackers running on victim hosts can also be leveraged to perform off-path TCP attacks [12], [13], [45]. Qian *et al.* uncovered that the middlebox of firewall can be abused to perform the TCP sequence number inference attack [13], and conducted a collaborative TCP sequence number inference attack by exploiting the packet counter side channels [12]. Gilad *et al.* identified that attackers can conduct web cache poisoning attacks by leveraging a restricted script in the user's browser sandbox [45]. Compared with these attacks, our off-path TCP attack does not need any assistance of puppets. Moreover, our attack leverages a new side channel vulnerability appearing in the interactions among IP, ICMP, and TCP, which cannot be unearthed by the principled methods [11].

TCP DoS Attacks: TCP SYN flooding is a major threat that is difficult to be identified due to the similarity to the legitimate establishment of TCP connections [46]–[49]. Besides, more sophisticated and stealth DoS attacks have also been proposed to cause resource decreases or exhaustion of the target TCP connections, such as low-rate TCP-targeted DoS attacks [50]–[53], congesting intermediate links attacks [54], [55] and pulsing DoS attacks that manipulate the victim's TCP congestion window to decrease the performance [56].

X. Conclusion

In this paper, we uncover a new off-path TCP hijacking attack that leverages a subtle side channel in the new mixed IPID assignment method of Linux kernel version 4.18 and beyond. We find that a pure off-path attacker can downgrade the IPID assignment for TCP packets from the more secure per-socket-based policy to hash-based policy, thus building a shared IPID counter that can be exploited to infer the state of a victim TCP connection. We evaluate the impacts of our attack on the Internet and implement the exploit under different scenarios. Our experiments show that off-path attackers can perform various attacks by exploiting the newly discovered IPID side channel, e.g., resetting SSH connections, manipulating web traffic and poisoning BGP routing tables. We also propose to eliminate the root cause of the exploit via repairing the IPID assignment. We implement our countermeasure and confirm its effectiveness in practice.

References

[1] J. Postel, *Transmission Control Protocol*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 793, Sep. 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc793.txt

[2] M. Duke, R. Braden, W. Eddy, E. Blanton, and A. Zimmermann, *A Roadmap for Transmission Control Protocol (TCP) Specification Documents*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 7414, Feb. 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7414.txt

[3] J. Touch, *Defending TCP Against Spoofing Attacks*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 4953, Jul. 2007. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4953.txt

[4] F. Gont and S. Bellovin, *Defending Against Sequence Number Attacks*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 6528, Feb. 2012. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6528.txt

[5] M. Larsen and F. Gont, *Recommendations for Transport-Protocol Port Randomization*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 6056, Jan. 2011. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6056.txt

[6] A. Ramaiah, R. Stewart, and M. Dalal, *Improving TCP's Robustness to Blind In-Window Attacks*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 5961, Aug. 2010. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5961.txt

[7] Y. Gilad and A. Herzberg, "Off-path TCP injection attacks," *ACM Trans. Inf. Syst. Secur.*, vol. 16, no. 4, p. 13, 2014.

[8] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel, "Off-path TCP exploits of the challenge ACK global rate limit," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 765–778, Apr. 2018.

[9] Y. Cao *et al.*, "Off-path TCP exploits: Global rate limit considered dangerous," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, 2016, pp. 209–225.

[10] W. Chen and Z. Qian, "Off-path TCP exploit: How wireless routers can jeopardize your secrets," in *Proc. 27th USENIX Secur. Symp. (USENIX Secur.)*, 2018, pp. 1581–1598.

[11] Y. Cao, Z. Wang, Z. Qian, C. Song, S. V. Krishnamurthy, and P. Yu, "Principled unearthing of TCP side channel vulnerabilities," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 211–224.

[12] Z. Qian, Z. M. Mao, and Y. Xie, "Collaborative TCP sequence number inference attack: how to crack sequence number under a second," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 593–604.

[13] Z. Qian and Z. M. Mao, "Off-path TCP sequence number inference attack-how firewall middleboxes reduce security," in *Proc. IEEE Symp. Secur. Privacy*, May 2012, pp. 347–361.

[14] J. Postel, *Internet Protocol*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 791, Sep. 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc791.txt

[15] J. Touch, *Updated Specification of the IPv4 ID Field*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 6864, Feb. 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6864.txt

[16] R. Ensafi, J. C. Park, D. Kapur, and J. R. Crandall, "Idle port scanning and non-interference analysis of network protocol stacks using model checking," in *Proc. USENIX Secur. Symp.*, 2010, pp. 257–272.

[17] R. Ensafi, J. Knockel, G. Alexander, and J. R. Crandall, "Detecting intentional packet drops on the Internet via TCP/IP side channels," in *Proc. Int. Conf. Passive Active Netw. Meas.* Berlin, Germany: Springer, 2014, pp. 109–118.

[18] E. Dumazet. (2014). *Inetpeer: Get RID of IP_ID_Count*. [Online]. Available: https://lore.kernel.org/patchwork/patch/490770/

[19] J. Knockel and J. R. Crandall, "Counting packets sent between arbitrary internet hosts," in *Proc. 4th USENIX Workshop Free Open Commun. Internet (FOCI)*, 2014, pp. 1–14.

[20] Bootlin. (2020). *IPID Assignment in Linux Kernel*. [Online]. Available: https://elixir.bootlin.com/linux/latest/source/include/net/ip.h

[21] G. Alexander, A. M. Espinoza, and J. R. Crandall, "Detecting TCP/IP connections via IPID hash collisions," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 4, pp. 311–328, Oct. 2019.

[22] X. Zhang, J. Knockel, and J. R. Crandall, "ONIS: Inferring TCP/IP-based trust relationships completely off-path," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 2069–2077.

[23] J. Mogul and S. Deering, *Path MTU Discovery*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 1191, Nov. 1990. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1191.txt

[24] J. McCann, S. Deering, and J. Mogul, *Path MTU Discovery for IP Version 6*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 1981, Aug. 1996. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1981.txt

[25] J. Postel, *Internet Control Message Protocol*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 792, Sep. 1981. [Online]. Available: http://www.rfc-editor.org/rfc/rfc792.txt

[26] P. Pearce, R. Ensafi, F. Li, N. Feamster, and V. Paxson, "Augur: Internet-wide detection of connectivity disruptions," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 427–443.

[27] P. Pearce, R. Ensafi, F. Li, N. Feamster, and V. Paxson, "Toward continual measurement of global network-level censorship," *IEEE Secur. Privacy*, vol. 16, no. 1, pp. 24–33, Jan. 2018.

[28] M. Luckie, R. Beverly, R. Koga, K. Keys, J. A. Kroll, and K. claffy, "Network hygiene, incentives, and regulation: Deployment of source address validation in the internet," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 465–480.

[29] B. Wu *et al.*, "Enabling efficient source and path verification via probabilistic packet marking," in *Proc. IEEE/ACM 26th Int. Symp. Qual. Service (IWQoS)*, Jun. 2018, pp. 1–10.

[30] F. Baker, *Requirements for IP Version 4 Routers*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 1812, Jun. 1995. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1812.txt

[31] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, *Generic Routing Encapsulation (GRE)*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 2784, Mar. 2000. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2784.txt

[32] D. Borman, B. Braden, and V. Jacobson, *TCP Extensions for High Performance*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 7323, Sep. 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7323.txt

[33] R. S. Raman, A. Stoll, J. Dalek, R. Ramesh, W. Scott, and R. Ensafi, "Measuring the deployment of network censorship filters at global scale," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, 2020, pp. 1–16.

[34] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, San Diego, CA, USA, 2019, pp. 1–15.

[35] Rocket.Chat. (2020). *Web Application*. [Online]. Available: https://rocket.chat/

[36] Y. Gilad, A. Cohen, A. Herzberg, M. Schapira, and H. Shulman, “Are we there yet? On RPKI’s deployment and security,” in *Proc. NDSS*, 2017, pp. 1–15.

[37] K. Ishiguro. (2020). *Quagga Routing Suite*. [Online]. Available: https://www.quagga.net/

[38] X. Feng, C. Fu, Q. Li, K. Sun, and K. Xu, “Off-path TCP exploits of the mixed IPID assignment,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 1323–1335.

[39] A. Conta, S. Deering, and M. Gupta, *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 4443, Mar. 2006. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4443.txt

[40] P. Srisuresh and M. Holdrege, *IP Network Address Translator (NAT) Terminology and Considerations*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 2663, Aug. 1999. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2663.txt

[41] LKM. (2007). *Blind TCP/IP Hijacking is Still Alive*. [Online]. Available: http://phrack.org/issues/64/13.html

[42] Y. Gilad and A. Herzberg, “Spying in the dark: TCP and Tor traffic analysis,” in *Proc. Int. Symp. Privacy Enhancing Technol. Symp.* Berlin, Germany: Springer, 2012, pp. 100–119.

[43] Y. Gilad, A. Herzberg, and H. Shulman, “Off-path hacking: The illusion of challenge-response authentication,” *IEEE Secur. Privacy*, vol. 12, no. 5, pp. 68–77, Sep. 2014.

[44] Y. Gilad and A. Herzberg, “Off-path attacking the web,” in *Proc. WOOT*, 2012, pp. 41–52.

[45] Y. Gilad and A. Herzberg, “When tolerance causes weakness: The case of injection-friendly browsers,” in *Proc. 22nd Int. Conf. World Wide Web*, 2013, pp. 435–446.

[46] W. Eddy, *TCP SYN Flooding Attacks and Common Mitigations*, Internet Requests for Comments, Internet Engineering Task Force, document RFC 4987, Aug. 2007. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4987.txt

[47] R. Bani-Hani and Z. Al-Ali, “SYN flooding attacks and countermeasures: A survey,” in *Proc. Int. Conf. Inf. Commun. Syst.*, 2013, pp. 149–155.

[48] H. Wang, D. Zhang, and K. G. Shin, “Detecting SYN flooding attacks,” in *Proc. 21st Annu. Joint Conf. IEEE Comput. Commun. Societies*, vol. 3, Jun. 2002, pp. 1530–1539.

[49] A. Aborujilah, M. N. Ismail, and S. Musa, “Detecting TCP SYN based flooding attacks by analyzing CPU and network resources performance,” in *Proc. 3rd Int. Conf. Adv. Comput. Sci. Appl. Technol.*, Dec. 2014, pp. 157–161.

[50] A. Kuzmanovic and E. W. Knightly, “Low-rate TCP-targeted denial of service attacks: The shrew vs. the mice and elephants,” in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2003, pp. 75–86.

[51] A. Shevtekar, K. Anantharam, and N. Ansari, “Low rate TCP denial-of-service attack detection at edge routers,” *IEEE Commun. Lett.*, vol. 9, no. 4, pp. 363–365, Apr. 2005.

[52] A. Herzberg and H. Shulman, “Stealth DoS attacks on secure channels,” in *Proc. NDSS*, 2010, pp. 1–19.

[53] S. Jero, M. E. Hoque, D. R. Choffnes, A. Mislove, and C. Nita-Rotaru, “Automated attack discovery in TCP congestion control using a model-guided approach,” in *Proc. NDSS*, 2018, pp. 1–15.

[54] J. M. Smith and M. Schuchard, “Routing around congestion: Defeating DDoS attacks and adverse network conditions via reactive BGP routing,” in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 599–617.

[55] M. Tran, M. S. Kang, H.-C. Hsiao, W.-H. Chiang, S.-P. Tung, and Y.-S. Wang, “On the feasibility of rerouting-based DDoS defenses,” in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 798–813.

[56] X. Luo *et al.*, “On a new class of pulsing denial-of-service attacks and the defense,” in *Proc. NDSS*, 2005, pp. 1–19.

Xuewei Feng received the B.E. degree from the Department of Computer Science and Technology, Xi’an Jiaotong University. He is currently pursuing the Ph.D. degree with Tsinghua University. His research interests include network security and software vulnerability detection.

Qi Li (Senior Member, IEEE) received the Ph.D. degree from Tsinghua University. He has worked with ETH Zurich and The University of Texas at San Antonio. He is currently an Associate Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. His research interests include network and system security, particularly in internet and cloud security, mobile security, and big data security. He is an Editorial Board Member of the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING and *ACM DTRAP*.

Kun Sun (Member, IEEE) received the Ph.D. degree in computer science from North Carolina State University. He has more than 15 years of working experience in both industry and academia. He serves as the Director for the Sun Security Laboratory (SunLab) and the Associate Director for the Center for Secure Information Systems (CSIS). He has published more than 100 peer-reviewed conference papers and journal articles. His research focuses on systems and network security.

Chuanpu Fu received the B.E. degree from the Department of Networking Engineering, Dalian University of Technology, in 2020. He is currently pursuing the Ph.D. degree with Tsinghua University. His research interests include machine learning for security, and network and system security.

Ke Xu (Senior Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China. He serves as a Full Professor for the Department of Computer Science and Technology, Tsinghua University. He has published more than 200 technical papers and holds 11 U.S. patents in the research areas of next-generation internet, blockchain systems, the Internet of Things, and network security. He is a member of ACM. He served as the Steering Committee Chair for IEEE/ACM IWQoS and has guest-edited several special issues in IEEE and Springer journals. He is an Editor of IEEE INTERNET OF THINGS JOURNAL.