Exploiting Sequence Number Leakage: TCP Hijacking in NAT-Enabled Wi-Fi Networks

Yuxiang Yang[*], Xuewei Feng[*], Qi Li[†§], Kun Sun[‡], Ziqiang Wang[¶], and Ke Xu[*§✉]
[*]Department of Computer Science and Technology & BNRist, Tsinghua University
[†]Institute for Network Sciences and Cyberspace & BNRist, Tsinghua University, [§]Zhongguancun Lab
[‡]Department of Information Sciences and Technology & CSIS, George Mason University
[¶]School of Cyber Science and Engineering, Southeast University
{yangyx22@mails, qli01@, xuke@}tsinghua.edu.cn, fengxw06@126.com, ksun3@gmu.edu, ziqiangwang@seu.edu.cn

***Abstract*—In this paper, we uncover a new side-channel vulnerability in the widely used NAT port preservation strategy and an insufficient reverse path validation strategy of Wi-Fi routers, which allows an off-path attacker to infer if there is one victim client in the same network communicating with another host on the Internet using TCP. After detecting the presence of TCP connections between the victim client and the server, the attacker can evict the original NAT mapping and reconstruct a new mapping at the router by sending fake TCP packets due to the routers' vulnerability of disabling TCP window tracking strategy, which has been faithfully implemented in most of the routers for years. In this way, the attacker can intercept TCP packets from the server and obtain the current sequence and acknowledgment numbers, which in turn allows the attacker to forcibly close the connection, poison the traffic in plain text, or reroute the server's incoming packets to the attacker.**

We test 67 widely used routers from 30 vendors and discover that 52 of them are affected by this attack. Also, we conduct an extensive measurement study on 93 real-world Wi-Fi networks. The experimental results show that 75 of these evaluated Wi-Fi networks (81%) are fully vulnerable to our attack. Our case study shows that it takes about 17.5, 19.4, and 54.5 seconds on average to terminate an SSH connection, download private files from FTP servers, and inject fake HTTP response packets with success rates of 87.4%, 82.6%, and 76.1%. We responsibly disclose the vulnerability and suggest mitigation strategies to all affected vendors and have received positive feedback, including acknowledgments, CVEs, rewards, and adoption of our suggestions.

I. Introduction

Wi-Fi has emerged as one of the most popular technologies for providing Internet access, being widely used in restaurants, offices, coffee shops, airports, and other public places. However, Wi-Fi networks are often exploited by malicious attackers to launch various attacks. In addition to exploiting vulnerabilities to break the protection of encryption [55], [56], [57], a lot of prior works have already been conducted on session hijacking in Wi-Fi networks [47], [53], [51], [2], [19], e.g., injecting forged wireless frames via vulnerabilities in WPA2 implementations [47], [53], eavesdropping on wireless channels [37], intercepting packets via side channels in VPN tunnels of wireless routers [51], creating a rogue clone (i.e., Evil-Twin) of the network [2], or just abusing the classic ARP poisoning attack [19] to hijack the communication between victim clients and servers, thus disrupting normal user usage, stealing confidential information, and potentially causing financial losses. Fortunately, most of the prior attacks have been repaired or mitigated and targeted defense measures have been proposed as well [54], [35], [53], [57]. Nowadays, with the widespread deployment of wireless security mechanisms (e.g., WPA2 and WPA3) and the adoption of protection strategies (e.g., AP isolation, ARP prevention, and Rogue AP detection), it is increasingly difficult for an off-path attacker (i.e., with no control over the router) to obtain the communication information between other clients in the same Wi-Fi network and outside servers.

In public Wi-Fi networks, network address translation (NAT) is widely used to save IPv4 address space and protect internal clients from being identified by external attackers. After attaching to the same Wi-Fi network enabling NAT, clients share the external IP address to access the Internet. When it takes the upper protocols (e.g., TCP and UDP) into consideration, the router will create NAT mappings to keep track of the connections, which record the IP addresses, upper-level information such as protocol, ports, timeout, and reply status, etc. In most cases, the router tries to keep the layer-4 information the same as the originators, such as the TCP source port, which is the so-called *port preservation* strategy [14]. However, cases are that some clients in the LAN may communicate with the same remote server with the same source port at the same time as they have no idea about each other. Although with very little probability, the router has to deal with these cases and it will assign a new TCP source port, change the IP address, and port at the same time when TCP packets pass through it. Besides, due to reasons such as performance considerations, the router will not record all of the session information in the NAT mappings, such as tracking the current TCP window. Thus, it will not check the sequence and acknowledgment numbers strictly when TCP packets arrive.

In this paper, we uncover a new off-path TCP hijacking attack in Wi-Fi networks that exploits vulnerabilities in the NAT mapping strategies of routers. The attack includes three steps. First, the attacker probes the router's external IP address, identifies whether AP isolation is enabled and scans to find potential victims in the same network when it is disabled. Second, the attacker infers the presence of TCP connections

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23419
www.ndss-symposium.org

between any client and a remote server by sending fake TCP `SYN` and `SYN/ACK` packets. Third, the attacker evicts the original NAT mapping of the victim connection with forged `RST` packets and replaces it with a new mapping at the router by sending a TCP data packet to the server. After that, it can intercept the `ACK` packet from the server that is meant to send to the victim and thus obtain the sequence and acknowledgment numbers within it so as to completely hijack the TCP connection. The attacker only needs to connect to the same network as the victim client, and it does not need any assistance of malicious puppets, i.e., unprivileged applications or sandboxed scripts deployed on victim clients. Compared with prior attacks, our work sheds light on the vulnerabilities existing in the abusing peculiarities of NAT strategies and behaviors of routers instead of flaws in TCP specifications, and our attack is not limited to specific scenarios or applications (e.g., WPA2/WPA3, or VPNs). Besides, the OS types or versions of the clients and servers are unrestricted in our attack in contrast to previous TCP hijacking attacks that can only target servers or clients with specific operating systems[17], [43], [44].

In our investigations, most Wi-Fi routers (e.g., Asus, Netgear, Linksys, TP-Link, Huawei, and Xiaomi) adopt the *port preservation* strategy when creating new NAT mappings for TCP connections initiated by internal clients [22], [18]. The attacker can intentionally initiate a connection, i.e., sending a `SYN` packet, to the target server with a guessed client's port and distinguish the guess by observing whether the port will be changed at the router as a collision will happen if it is a right guess. The attacker can send a spoofed `SYN/ACK` packet with a source address of the remote server, a destination address of the external IP of the router, and a destination port of the guessed port as a response to verify if the port is changed. If the router disobeys the RFC recommendation to enable the reverse path validation with a strict mode [48], [4], the forged `SYN/ACK` packet cannot be detected and will not be dropped by the router, which is often the case in most routers we tested. If there is any connection from the LAN to a target remote server with the guessed source port, the router will choose another source port to initiate the connection, and then the `SYN/ACK` packet will be forwarded to the victim. Yet if there is no connection with this source port from the LAN to the server, the router will keep the port to initiate the connection, and then the `SYN/ACK` packet will be forwarded back to the attacker. In this way, the attacker can infer whether any client is communicating with the server and the source port of the client if there is such a TCP connection.

After identifying a target TCP connection, the attacker can directly get the sequence and acknowledgment numbers of the connection by exploiting a new vulnerability arising in the disabled TCP window tracking strategy of Wi-Fi routers. As routers pursue higher performance, they choose to disable TCP window tracking by default, i.e., they will not check the sequence and acknowledgment numbers strictly in TCP packets. So the attacker can send forged TCP reset packets to clean the NAT mapping of the victim connection. After waiting for the timeout of the NAT mapping (i.e., 1 second or 10 seconds), the attacker can send a TCP data packet using its private IP address and the same source port to the server with arbitrary sequence and acknowledgment numbers. The router will only translate the IP address of the packet except for the source port, as there is no port collision anymore. And the packet will match the victim connection from the perspective of the server, which will return a TCP `ACK` packet carrying the exact sequence and acknowledgment numbers of the victim connection upon seeing the packet with wrong numbers [42]. When arriving at the router, this `ACK` packet will be routed to the attacker as the NAT mapping has been falsified, and thus it steals the sequence and acknowledgment numbers of the victim connection easily, i.e., without traversing the 32-bit space to infer these numbers as previous methods [6], [10], [7]. It should be noted that the NAT mapping timeout will be refreshed if there are related packets traveling through the router, which may interfere with the attack. We will analyze the detailed influence in Section VI-A.

Once the sequence and acknowledgment numbers are obtained by the attacker, it can choose to launch three types of attacks: (i) **TCP Denial-of-Service (DoS) attack** to terminate victim TCP connections directly by sending `RST` packets. (ii) **TCP hijacking attack** to take over the NAT mapping and replace the victim by itself since the router will continue forwarding packets (intended for the victim client) to the attacker instead. (iii) **TCP injection attack** to poison the victim TCP traffic by sending crafted data packets after restoring the mapping for the victim client via issuing spoofed TCP `RST` and `ACK` packets. Note that traffic encryption (e.g., HTTPS) may disturb the attacker's poisoning. However, about 20% of websites still transmit traffic in plaintext according to the reports on HTTPS adoption[1]. AP isolation may also influence the TCP injection attack due to the requirement of reconstructing the client's original NAT mapping. However, the other two attacks (i.e., TCP DoS and hijacking attacks) are not affected, which will be illustrated in Section VI-A.

We conduct a large-scale empirical study to demonstrate that the attack can be performed to cause potential damage in the real world. First, we investigate the default settings of routers on the market and have tested 67 widely used router models from 30 vendors and find that 52 of them from 24 vendors are vulnerable to the attack. Moreover, our empirical measurement results show that the attacks can be successfully performed in various real-world Wi-Fi networks. We evaluate 93 Wi-Fi networks in six months, including most of the popular Wi-Fi scenarios (e.g., Wi-Fi networks in coffee shops, hotels, bookstores, and enterprises). The experimental results show that 75 (81%) out of these evaluated Wi-Fi networks are fully vulnerable to our attacks. We implement a PoC and perform case studies on applications like SSH, FTP, and HTTP to validate the effectiveness of the attack. In our experiments, an off-path attacker can detect and terminate an SSH connection in 17.5 seconds with a success rate of 87.4%, download private files from an FTP server within 19.4 seconds with a success rate of 82.6%, and manipulate web traffic within 54.5 seconds with a success rate of 76.1%, on average. These results demonstrate that this attack is feasible and may throw potential threats to normal Wi-Fi users.

Finally, we identify the root cause and suggest mitigation to void this attack with the intuitive idea of breaking the conditions of the attack. Besides, we have responsibly disclosed the vulnerability to the affected router vendors and the OpenWrt

[1] See https://w3techs.com/technologies/details/ce-httpsdefault for daily statistics on HTTPS adoption.

community with affirmative feedback. At the time of writing, researchers from the OpenWrt community and 7 of these vendors have confirmed the vulnerability and are repairing it in their products according to our suggestions. In addition, 10 CVE numbers have been assigned for this vulnerability from different vendors (i.e., from CVE-2023-30305 to CVE-2023-30314). The rest vendors are still in the process of investigating the vulnerability.

Contributions. Our main contributions are the following:

- We uncover a new side channel vulnerability of the NAT behaviors in Wi-Fi networks that can be exploited to attack TCP connections by off-path malicious insiders.
- We perform a large-scale measurement and reveal a number of routers vulnerable to the attack. Our extensive evaluations against 67 widely used router models and case studies in 93 various Wi-Fi networks show that our attacks can cause potential damage in the real world.
- We suggest three countermeasures by eliminating the conditions to fight back the attack, and some of them have been adopted by the affected manufacturers.

II. Background

A. NAT and Port Allocation Strategies

Network Address Translation (NAT) is a technology developed to solve the shortage of IPv4 addresses and hide the network topology from an external entity [5], which is widely used by routers in Wi-Fi networks. When packets traverse through the router, it has to translate the IP addresses of the packets between internal and external addresses[2] and record the other necessary information of the related connection. The router maintains a NAT mapping table to keep track of the internal IP addresses and ports associated with each corresponding external IP address and port, which allows incoming packets to be directed to the correct host on the private network. Since our work focuses on the TCP protocol, we will illustrate the NAT behavior of TCP mappings henceforth.

When an internal host initiates a connection to an external server, i.e., sending a `SYN` packet, the router will create a new mapping in the table, which is called a binding in NAT terminology [20]. Besides, we find that not only `SYN` packets but also packets with `PUSH`, or `ACK` flags can incur new NAT mappings at the router. The mapping will record the source IP addresses and ports translated before and after, the destination IP address and port, protocol, session state, and corresponding mapping timeout. After the replies from the external host arrive, the router forwards the packets to the internal host according to the mapping and updates its state simultaneously. Since the related RFCs have not proposed a fixed strategy for the translation behavior of source ports, it can be different depending on the implementation of NAT devices, which includes the following strategies [18]:

(1) *port preservation*, where the NAT device attempts to preserve the source port if possible. When a collision happens, i.e., different internal hosts choose the same source port to communicate with the same external host of the same port, the NAT device should resolve the collision by selecting a new port (e.g., another random unused port). (2) *random selection*, where the NAT device translates the source port to another random port from a pool of available ports. (3) *sequential selection*, where the NAT device selects a random port for the first connection to each destination and translates the ports of subsequent packets to that destination consecutively based on the first port. (4) *port overloading*, where the NAT device always uses *port preservation* even in the case of collision. In this case, new connections will take over the original mapping, and the old connection will be disturbed, which is not recommended in RFC 5382 [14].

As with any stateful middle device, routers have to manage the state of mappings and track active flows. Generally, the routers often rely on both the states of connections and timeouts of mappings to prune unnecessary NAT mappings. RFC 5382 recommends that the minimum timeout for the `ESTABLISHED` state is 2 hours and 4 minutes, which is faithfully implemented in most routers [14], and the routers also set timeouts for other states (e.g., 1 second or 10 seconds for the `CLOSE` state which the mapping will turn into upon seeing corresponding `RST` packets).

B. TCP Window Tracking in Routers

As a middle device between the client and server, the router has to record the connection information of the related hosts for subsequent packet delivery. However, as the TCP protocol was originally designed for end-to-end communication and did not take the middle devices into consideration, the router cannot and will not record all of the information due to many reasons (e.g., performance considerations). For instance, the router will choose not to track the current TCP window of the connection, and thus it will not check the sequence and acknowledgment numbers of TCP packets strictly. The open-sourced router operating systems, i.e., OpenWrt and AsusWrt, both have related options to reduce CPU overhead, i.e., the `nf_conntrack_tcp_no_window_check` option in OpenWrt and the `ip_conntrack_tcp_be_liberal` option in AsusWrt. These options are set to true by default, and once they are set, Netfilter [25] will not perform TCP window tracking in contrast to the original Linux kernel. The difference between the two systems is that OpenWrt does not check the sequence number of the packet at all, while AsusWrt only checks if the sequence number is beyond the current sequence number in a 2G space. Besides, we found most of the routers in the market also disable the TCP window tracking strategy by default and have similar behaviors to the two systems above.

We will show that routers disabling TCP window tracking can be abused by an off-path attacker to clean the NAT mappings of other clients with forged `RST` packets. For OpenWrt-based routers and those with similar settings, the attacker can use one forged `RST` packet with any arbitrary sequence number to clean the mapping, and for AsusWrt-based routers and those with similar settings, the attacker can forge two `RST` packets specified with two sequence numbers in the gap of 2G to bypass the range check easily and effectively.

C. Reverse Path Validation

To prevent IP spoofing attacks and promote the process of source address validation, RFC 2827 and RFC 3704 propose

[2] Since our work considers multiple levels of NAT, the router's external IP address may not be a public IP address.

the concept of reverse path validation, which verifies the authenticity of inbound traffic by checking whether the source IP address can be routed back via the interface on which packets are received against the routing table, to ensure they come from an authorized sender [48], [4]. With this strategy enabled, only if the packets can be routable back from the incoming interface will they be processed by the kernel and routed to their destinations. Otherwise, they will be dropped. Most Linux-based systems control the strategy through the *rp_filter* kernel variable, which offers three options [26]:

- **0:** In this mode, the source address validation is disabled.
- **1:** Strict Mode as defined in RFC3704. In this mode, the device should compare the source address of incoming packets to the Forwarding Information Base (FIB). If the incoming interface is not the best reverse path, packets will be dropped.
- **2:** Loose Mode as defined in RFC3704. In this mode, the device compares the source address of incoming packets against the FIB, and only if the packets are not reachable via any interface will they be dropped.

RFC 3704 recommends using the strict mode to prevent IP spoofing attacks. The loose mode is recommended if the device uses asymmetric routing (e.g., a mobile phone with a Wi-Fi interface and multiple interfaces for receiving packets from cell towers) or other complicated routing strategies. Previous research [51] has shown that in the VPN scenarios, the lack of reverse path validation on client devices allows a blind in-path attacker (e.g., a router controlled by an attacker) to spoof packets to learn the virtual IP used by the tun0 interface for the VPN connection and infer the necessary fields to hijack the active connection. By contrast, we find that most routers also do not obey the recommendation, and they will not drop packets with spoofed source addresses matching a connection in the NAT mappings and will accept them on any interface.

We will show that an off-path attacker in the LAN can abuse routers without reserve path validation to forward spoofed `SYN/ACK` packets with the server's IP address as the source and the router's external IP address as destination, which can be leveraged to infer source ports of connections used by other clients through observing the whereabouts of these `SYN/ACK` packets. Additionally, the attacker can also send forged `RST` packets to the router's external IP address. Though the source address specified in the packets is the server, the router without reverse path validation will process them in the kernel mistakenly and thus change the state of the NAT mappings to `CLOSE`, leading to our attack.

III. Threat Model

Figure 1 illustrates the threat model of our off-path TCP attacks in Wi-Fi networks. The model consists of three hosts and one router, namely, a remote server, a victim client, an off-path attacker, and a vulnerable router. The remote server may be a web application, an SSH or FTP server in different attack scenarios. The victim client (e.g., a mobile phone or a laptop) is connected to a wireless access point to communicate with the remote server on the Internet, i.e., visiting web pages, downloading files through FTP, or using the SSH service to control remote hosts. The off-path attacker is a malicious client who can access the same Wi-Fi network as the victim client. A router acts as the gateway of clients in the LAN to provide Internet services for the Wi-Fi network.

Existing studies [2], [39] demonstrate that a malicious insider can create an evil twin of the network and trick the victims into connecting to it by broadcasting the same SSID in the open (with no encryption) or home mode (accessed through pre-shared key) Wi-Fi networks, thus hijacking the traffic in the network. However, these attacks can be throttled by existing defenses, e.g., Rogue AP detection [21], [24]. It is widely believed that only AP isolation enabled enterprise mode Wi-Fi networks can effectively protect clients from each other, whereas open and home mode Wi-Fi networks face challenges in preventing insider threats. In this work, we propose a novel attack that can evade all defenses above in Wi-Fi networks. As a result, our attack holds particular significance for enterprise mode Wi-Fi networks, differentiating it from the rogue clone attacks in open and home mode Wi-Fi networks. Moreover, our attack can serve as an alternative method to compromise open and home mode Wi-Fi networks. In our attack, we assume that with the deployment of security mechanisms (e.g., WPAs) and the usage of security protection strategies (e.g., ARP prevention, AP isolation, and Rogue AP detection), an off-path attacker would not be able to discern if any client is communicating with a specific remote server. Furthermore, the attacker would not be able to ascertain the source port of the TCP connection, if it exists, and the sequence and acknowledgment numbers.

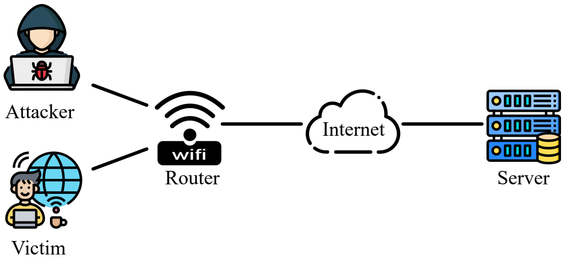


Fig. 1. Threat model of TCP hijacking attacks in Wi-Fi networks.

To successfully launch our attacks, there are some requirements to be fulfilled. First, the attacker should be able to probe the external IP address of the router. We will illustrate our methods in Section IV-B. Then the attacker tests whether AP isolation is enabled in the network[3] [32]. The attacker can successfully carry out the TCP DoS and TCP hijacking attacks regardless of whether AP isolation is enabled. When AP isolation is enabled, the attacker will not be able to probe potential victim clients within the network using scanning tools (e.g., Nmap [36]). Thus, the TCP injection attack will be thwarted with AP isolation. We will discuss the impacts of AP isolation on our attacks in Section IV-D and Section VI-A. Besides, the attacker has to target the remote server that the client is communicating with or will connect to, which can be set as those providing popular services as previous works [6], [10], e.g., famous servers, web search engines, or social sites.

[3] Previous research has revealed that nearly 89% of the public Wi-Fi networks allow clients to communicate with each other [9].

Second, the router adopts the *port preservation* strategy. Our investigations show that most routers adopt this strategy except for an enterprise router model from Huawei and the open-sourced routing firmware of pfSense [40]. Also, the router disables the reverse path validation strategy. We find that routers from 24 out of 30 vendors will forward forged packets except for Asus, Aruba, Cisco Meraki, Netgear, pfSense, and ZTE. Besides, some models from TP-Link, Mercury and Huawei also enable this strategy. Moreover, the router disables the TCP window tracking strategy. In our measurement, most routers have disabled it by default, with the exception of Cisco Meraki.

Third, the victim client does not communicate with the server frequently. The state of the NAT mapping will transfer from `ESTABLISHED` to `CLOSE` state after receiving corresponding TCP `RST` packets, and the mapping will be removed completely after its timeout (1 second or 10 seconds in our test). It should be noted that if the client's communication continues during this period, it may interfere with the attack as the mapping will be refreshed. As there are many long-lived TCP connections that clients periodically retrieve new data from the server in minutes and 42% of the tested routers set the timeout to only 1 second, the attacker has been provided with enough time to finish its attack. We will analyze its influence in detail in Section VI-A.

IV. Attack Procedure

A. Attack Overview

To perform our attacks, the attacker has to carry out the following three steps:

1. Probe the router's external IP address and identify whether AP isolation is enabled, thus finding potential victim clients.
2. Make inferences about whether there is any active connection from the LAN to the server.
3. Evict and construct NAT mappings at the router and then intercept the sequence and acknowledgment numbers from the replies to unsolicited packets from the server.

After the above steps, the attacker can terminate the connection directly or hijack the connection by replacing the victim client. Besides, when AP isolation is disabled, the attacker can restore the original NAT mapping of the victim client at the router and send fake response packets to the client.

B. Phase 1: Probing the Network

In this step, the attacker prepares the attack in two aspects, namely, identifying the status of AP isolation in the network and probing the external IP address of the router. Firstly, the attacker detects whether AP isolation is enabled via network scanning tools (e.g., Nmap [36], MacStealer [32]). If it is disabled, the attacker records the scanning results of potential victim clients for the futural TCP injection attack. Note that the attacker does not need to know the specific private IP address of the victim client (i.e., which IP is the victim), as we will show that it only needs to send related packets to the router, the server, or all of the clients in the subsequent attack phases, which is different from previous works that they will choose a target victim client beforehand, i.e., identifying whether a given client is communicating with the server [6], [10] or redirecting the victim's traffic to the attacker [9].

Secondly, the attacker probes the router's external IP address. With the widely deployed carrier-grade NAT [45], the Wi-Fi networks in the real world may consist of multiple levels of NAT [15], which means that the router's external IP address is not always a public IP address that can be obtained easily by querying its own public IP. We adopt the following methods to deal with this problem. (i) First, the attacker gets the gateways along the way to any outside host (e.g., 8.8.8.8) through Traceroute [52]. Second, the attacker issues the ping command to the second gateway with the *RECORD_ROUTE* option, which will record the passed routes [41], and then all the IP addresses of the passed interfaces will be returned. The result snapshot of the method is provided in Appendix A (refer to Figure 7). (ii) In certain scenarios, the aforementioned method may encounter failure, as the passed routes might not be returned when pinging the second gateway. In such cases, the attacker can opt to scan the subnet of the second gateway to identify live hosts' IP addresses. Subsequently, it can proceed to ping these IPs using the *RECORD_ROUTE* option. When the ping reaches the external IP of the router, the previously passed routes will be returned. However, when pinging other IPs, the routes will not be returned. Besides, the attacker can access these IPs via a web browser. When accessing the external IP of the router, the router's setting page (Web GUI) will be displayed, whereas accessing the other IPs will lead to different pages.

C. Phase 2: Making Inferences about Active Connections

Assuming that the attacker has connected to a Wi-Fi network, in which one of the normal users has established a TCP connection with a remote server from source port `m`. The router has a corresponding NAT mapping to keep track of the connection. The attacker intends to infer which source port is used by the victim client of the connection.

Figure 2 illustrates the side-channel vulnerability that leverages the NAT *port preservation* strategy and insufficient reverse path validation of the router. First, the attacker sends a `SYN` packet targeting the server with its own IP address and a guessed port number as the source. If the source port number (e.g., `n`) specified in the `SYN` packet does not equal `m`, the router will create a new NAT mapping with source port `n` that records this new TCP connection. Note that with the wide deployment of NIDS at the server side [29], a large amount of `SYN` packets arriving at the server may be detected, and the server may find it attacked and take corresponding fightback. The attacker can set the `TTL` of the `SYN` packet to a small number (e.g., 2 in our test), and thus the packet will be dropped quickly at the intermediate routers. In most cases, the routers do not deploy detection systems typically.

Then, the attacker impersonates the server and sends a forged `SYN/ACK` packet whose destination is the router's external IP address and whose destination port is the guessed source port `n`. According to RFC 3704 recommendation, the reverse path of packets received should be strictly checked so as to prevent IP spoofing attacks. In this case, as the `SYN/ACK` packet is received from the router's internal interface while its source address is a public IP address and it actually cannot

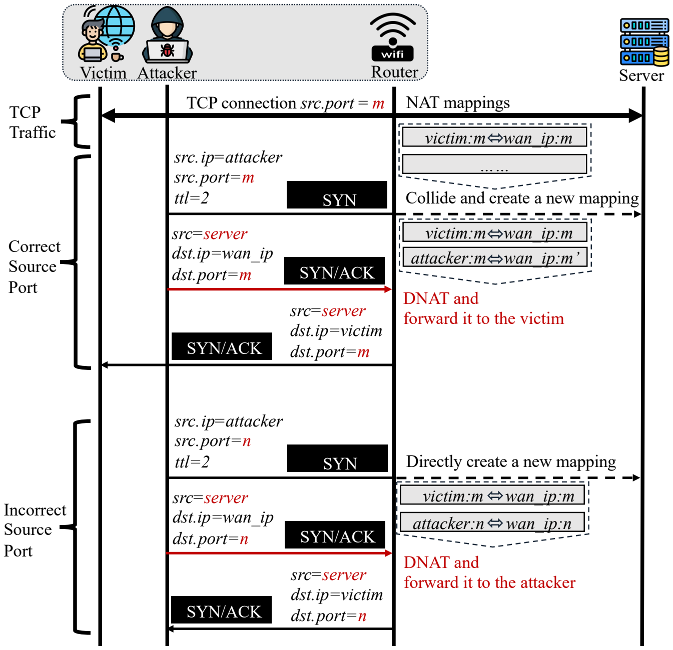


Fig. 2. Inferring the source port of the victim TCP connection

be routed from this incoming interface for responding to the packet, then it should be dropped by the router. However, many routers in the real world do not adopt the RFC recommendation and will not check the reverse path of packets received. So when the forged packet arrives, it will be processed by the router's kernel and forwarded according to NAT mappings. Since there is a NAT mapping that translates the router's external IP to the attacker's private IP when the source is the remote server and the destination port is `n`, the forged packet will match this mapping and be forwarded to the attacker. In this way, the attacker can receive the forged packet that is sent from itself again if the guessed source port `n` is not equal to the victim client's source port `m`.

If the attacker guesses the right source port, i.e., `m`, when the `SYN` packet arrives at the router, it will translate the source port of the new mapping to another port due to the collision. Let us say that the changed source port is `m'`. In the second step, when the forged `SYN/ACK` arrives at the router, however, it will be forwarded to the victim according to the client's NAT mapping as the port specified in it is `m` instead of `m'`. Thus, from the view of the attacker, it cannot receive the forged `SYN/ACK` packet again if the port it guesses is right, i.e., previously occupied. In this way, the attacker can infer that there is a connection from some local host to the target server with the source port `m`.

The attacker repeats the above procedure, i.e., changing the guessed source port number specified in the forged `SYN` and `SYN/ACK` packets and then observing if it can receive `SYN/ACK` back until the correct port `m` is identified, which will be used for the subsequent attacks.

D. Phase 3: Hijacking Active Connections

Once the attacker has determined an active TCP connection to a given remote server with the source port `m`, it will attempt to obtain the current sequence number *SEQ* and the acknowledgment number *ACK* of the server. Note that as TCP is a bidirectional symmetric full-duplex protocol, the sequence and acknowledgment numbers of the victim client are symmetrical to the server. Previous works mostly infer these values by exploring the entire possible 4G space via leveraging some side channels [10], [6], [7], [51]), which are rather time-consuming and largely impact the success rate of hijacking the short-period TCP connections. However, in this work, we demonstrate a new method to obtain these two values directly and precisely, which abuses vulnerable routers without TCP window tracking. We assume that there will be some intervals in the communication between the client and the server. Depending on the scenarios, the client periodically initiates a request and waits for responses, or the server proactively pushes notification messages, which are often the cases in real-world services.

Figure 3 shows our method for the attacker to hijack the TCP connection between the victim client and server. Firstly, the attacker cleans the router's NAT mapping of the victim connection by sending spoofed TCP `RST` packets whose source is the server and destination IP is the router's external IP, and destination port is the previously inferred port `m`. The sequence numbers specified in these packets are crafted for various brands and types of routers due to their different behaviors of disabling the TCP window tracking strategy. Generally, there are two popular behaviors as stated in Section II-B that the first type of router does not check the sequence number at all, and the second type of router will check if the sequence number is beyond the exact sequence number in a 2G space. For the first type of router, the attacker can specify an arbitrary sequence number in one crafted `RST` packet to clean the TCP NAT mapping at the router directly. And for the second type of router the attacker can send two `RST` packets, one with sequence number *x* and the other with *(x + 2G) % 4G*, which ensures that one of them will fall within the required range.

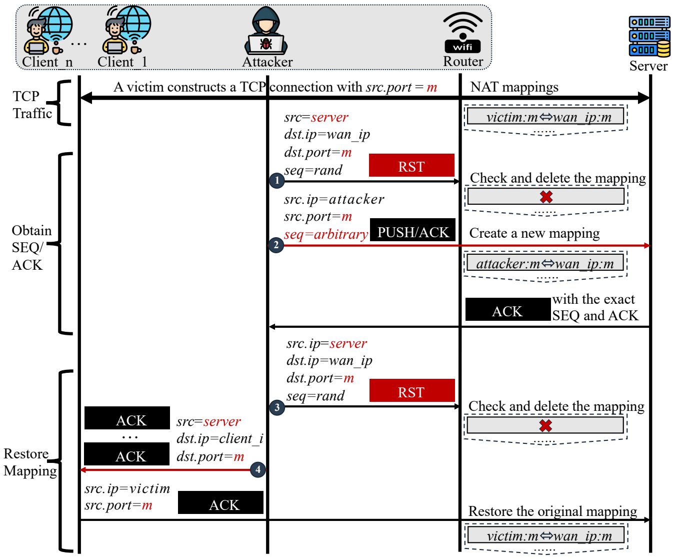


Fig. 3. Hijacking active connections

To simplify the description, we will only elaborate on the details of the attack on the first type of router. After receiving the RST packet, the router will falsely process the packet as it does not perform reverse path validation, and the state of the recorded TCP mapping will transfer from ESTABLISHED to CLOSE. The CLOSE state will last for a timeout according to a kernel variable (e.g., *ip_conntrack_tcp_timeout_close* in OpenWrt-based systems). In our empirical investigation, 41% of the tested routers set this value to as short as 1 second, and the other vulnerable routers set it to 10 seconds. After the countdown of the timeout, the original TCP NAT mapping at the router will be completely cleared.

Secondly, after the original NAT mapping has been fully evicted, the attacker replaces the victim client by constructing a new mapping at the router via sending a forged data packet with PUSH/ACK flags to the remote server, using the port m as the source port and its private IP as the source IP address. The sequence and acknowledgment numbers specified in the packet can be arbitrary as the router does not verify them. With the translation of IP addresses at the router, the packet will be routed to the server. From the perspective of the remote server, this packet is from the public IP address of the client with source port m and it will match the victim connection. Since the sequence and acknowledgment numbers specified in the packet are wrong, an ACK packet with the server's exact sequence and acknowledgment numbers will be triggered back [42]. When the ACK packet arrives at the router, it will be translated and routed to the attacker according to the new NAT mapping created just now. Then it can obtain the sequence and acknowledgment numbers of the victim connection directly.

After the above two steps, the attacker can decide on the follow-up procedures according to the purpose of the attack. In this work, we will illustrate three types of possible attacks. **(i) TCP DoS attack.** If the attacker intends to forcibly close the connection in the scenario of encrypted tunnels (e.g., SSH or HTTPS), it can just send forged TCP RST packets to the server with the information obtained before, thus causing the connection terminated at the server side. After that, the client will not receive any response when it sends requests to the server, which leads to a denial of service attack. **(ii) TCP hijacking attack.** If the attacker intends to hijack the traffic from the server to the victim client, it can take over the NAT mapping and impersonate the client again with the exact sequence and acknowledgment numbers to launch requests to the server and wait for responses from the server. For instance, at the beginning, the victim client logins into the FTP server and requests personal files from the server. After the attack, the attacker can bypass the initial verification stage by replacing the victim client to send requests to the server, which may lead to permission bypass and privacy leakage. **(iii) TCP injection attack.** If the attacker intends to send forged responses by impersonating the server when the victim client initiates a new request later, it needs to restore the original NAT mapping of the victim client at the router so as not to interfere with the client's normal communication. We are going to elaborate on this case as shown in the last two steps in Figure 3.

The attacker repeats the first step to clean the mapping of itself and waits for another NAT mapping timeout of CLOSE state. To restore the original mapping of the victim, as the attacker does not know the victim client's exact private IP, it can send forged ACK packets to all of the local hosts probed in the network probing phase (see Section IV-B) when AP isolation is disabled. The source of the ACK packets is the server, and the destination port is m. After arriving at the irrelevant hosts with no corresponding connections, the packets will be dropped. Yet the victim will send an ACK packet back to the server, which restores the NAT mapping of the victim when it travels through the router. Then the attacker can inject forged responses to the client by sending them to the external IP of the router via abusing the disabled reverse path validation and NAT mappings (the same as it did when inferring the source port). However, the attacker cannot restore the mapping if AP isolation is enabled within the network, as it cannot send packets to other clients, and thus this attack is thwarted.

Note that in the above attack phases, there are timeouts in which the NAT mappings are in the state of CLOSE. We have found that if related packets are traveling through the router at this period, the countdown will be refreshed, which may lead to interference with the attack. We will discuss the influence in detail in Section VI-A.

V. Empirical Study

We conduct extensive real-world evaluations to measure the impacts of the attack. We first investigate the default settings of routers on the market. Next, we conduct case studies to evaluate the effectiveness of the attack in various real-world Wi-Fi networks.

Ethical Considerations. As it is essential to respect the privacy and security of others when engaging in authorized hacking experiments, our experiments in real-world Wi-Fi networks require careful consideration of ethical issues. We addressed the ethical issues of our real-world experiments from the following perspectives. First, we provided the Wi-Fi network administrators with detailed explanations of our experimental plans and obtained their approval before conducting the experiments. Second, with the help of the administrator, we ensured that no other users were accessing the Wi-Fi network during our experiments, thus avoiding potential risks or side effects for other users. We then deployed our machine (a laptop or a cellphone) as the victim client in the Wi-Fi network, thus ensuring that all the machines involved in our experiments were under our control and would not affect other machines. Finally, after completing the experiments, we provided feedback on the results to the administrator. Moreover, we recommended that they restart the Wi-Fi router and clear the cache to restore the network to a safe state.

A. Analysis of Routers

The attack leverages the strategies adopted by the router, and only if all of the conditions are fulfilled can our attack succeed. In order to explore the coverage of vulnerable routers, we perform tests on real router models from lots of vendors, including 360, Aruba, ASUS, Amazon, Cisco Meraki, China Mobile, Comfast, D-Link, GL.iNet, Google, H3C, Huawei, IP-COM, iKuai, JdCloud, Linksys, Mercury, Netgear, Netcore, Ruijie, Skyworth, Tenda, TP-Link, Ubiquiti, Volans, Wavlink, WiMaster, Xiaomi, and ZTE. To our best knowledge, the operating systems of most routers we tested are based on Linux with custom modifications, except for some router models from TP-Link and Mercury, which are based on VxWorks.

Therefore, we also build a soft routing environment with a FreeBSD-based firmware, i.e., pfSense 2.7.0 [40]. In total, we perform tests on 67 mainstream router models (acting as the gateway to provide Internet services) from 30 vendors. For each router model, we test if it fits all attack conditions proposed in Section III. Here we list the detailed test results of 33 representative routers from these 30 vendors in Table I.

First, the router has to take the *port preservation* strategy. In our test, most of the routers adopt this strategy by default. Only the enterprise wired router model "AR6140E-9G-2AC" produced by Huawei and the soft routing machine with pfSense which co-work with a wireless AP to provide Wi-Fi service, take the *random selection* strategy that prevents the attacker from inferring the source port of clients' connections. Besides, there is no router model which takes the *sequential selection* or *port overloading* strategy as stated in Section II-A

Second, we investigate the deployment of the reverse path validation strategy in the routers. Among these routers, Netgear and Asus set the kernel variable *rp_filter* to 1 by default, which means they are secure to check the packet received strictly. And some of the old-styled models of TP-Link and Mercury (i.e., VxWorks-based) will also validate the received packets. However, their newest models (e.g., designed for Wi-Fi 6) and some enterprise routers will not validate them anymore. In addition, ZTE routers, the router model "AR6140E-9G-2AC" from Huawei, and the soft routing machine with pfSense will also not forward the forged packets. The routers from the other vendors all disable reverse path validation, which results in the vulnerability of inferring active connections.

Third, the router has to disable the TCP window tracking strategy. In our test, only one enterprise wired router model "Meraki 64" produced by Cisco Meraki will check the sequence number strictly and all of the other routers disable TCP window tracking while the processing logic is slightly different. Asus, Google, Netgear, Tenda, Wimaster, and ZTE routers will check if the sequence number is in the 2G space beyond the exact sequence number. And the other routers do not check the sequence number at all.

Fourth, the timeout of TCP `CLOSE` state of NAT mappings may influence the time cost and success rate of our attack. The shorter the timeout is, the easier the attack can succeed. Among the 66 router models without TCP window tracking, 28 of them will clean the NAT mapping in only 1 second, 37 of them will be tricked to clean the mapping in 10 seconds, and the default setting of pfSense is 90 seconds.

Due to the limited space, the detailed information of the 67 tested routers is listed in Appendix B (see Table IV). We take the first row as an instance to analyze the results. The Linux-based router model "TL-XDR6020" produced by TP-Link, provides the latest generation of Wi-Fi 6 for network services. As for the four metrics mentioned above, this model takes the *port preservation* strategy, does not validate the reverse path of received packets, disables the TCP window tracking, and sets the TCP `CLOSE` timeout to 10 seconds by default. In this way, it is vulnerable to our attacks. In conclusion, 52 of the 67 tested routers are vulnerable, and 15 models are immune to the attack as they do not fulfill all of the conditions.

B. Attack Evaluation

To evaluate the impacts of the attack in the real world, we also conduct thorough experiments of the attack in 93 various Wi-Fi networks. We investigate whether the conditions of the attack are fulfilled in each network by taking three case studies of attacks on SSH, FTP, and HTTP applications and measuring the time cost and success rate of each attack.

Experimental Setup. Our experiments consist of four types of devices, i.e., a router, a victim client, a remote server, and an attacker.

- *Router.* The router in Wi-Fi networks works as the gateway to provide Internet access and forward packets between local clients and outside servers.

- *Remote Server.* For the DoS attack, we set up an SSH server equipped with Ubuntu 22.04 (kernel version 5.15.0), OpenSSH 8.9, and OpenSSL 3.0.2. For the hijacking attack, we set an FTP server equipped with Ubuntu 22.04 (kernel version 5.15.0) and vsftpd version 3.0.3. And for the injection attack, we pick a well-known finance website *www.ANONYMOUS.com* (anonymized for ethical consideration) in which the client initiates a long-lived TCP connection that periodically retrieves data updates every minute.
- *Victim Client.* Though the OS type or version of the client is unrestricted in our attack, we still deploy victim clients equipped with five typical OSes (i.e., Windows, Linux, Mac, iOS, and Android). Each victim client has connected to the Wi-Fi network. In the case of DoS and hijacking attacks, the victim client will communicate with the remote server through SSH and FTP. And in the case of the injection attack, the victim client will access the website above to get the newest future index of stocks (e.g., `HSI`, `HSCEI`).
- *Attacker.* An attack machine is equipped with Linux 5.15.0, which is capable of crafting packets. The attacker aims to forcibly close the victim client's SSH connection with the remote server, steal private files from the FTP server, or inject fake HTTP responses to the client by performing the attack.

Attack Procedure. The attacker first tries to get the router's external IP address and test whether AP isolation is enabled and other hosts can be detected in the Wi-Fi network. Next, the attack can be constructed in the following steps: (1) detecting whether there is any TCP connection from the LAN to the given server, i.e., identifying the correct source port, (2) evicting the router's original NAT mapping of the client with forged `RST` packets and constructing a new one by sending a TCP data packet to the server, which in turn incurs an `ACK` packet from it, (3) terminating the SSH connection or requesting an FTP file download, (4) restoring the original NAT mapping of the client and answering the client's HTTP requests with forged segments specified with the inferred values.

For the SSH DoS attack, we define the result that the client or the server cannot receive messages from each other as a successful attack, which includes two cases. First, after the attacker receives the `ACK` packet from the server, it can send a TCP `RST` packet with the exact sequence number to the server, resulting in the connection terminated at the server

TABLE I. PARTIAL TESTED ROUTERS FROM 30 VENDORS

No.	Router Model	Vendor	OS	Generation	Port Preservation	Reverse-path Validation Disabled	TCP Window Tracking Disabled	TCP Close Timeout (second)	Vulnerable
1	TL-XDR6020	TP-Link	Linux-based	Wi-Fi 6	✔	✔	✔	1	✔
2	TL-WDR7620	TP-Link	Vxworks-based	Wi-Fi 5	✔	✘	✔	1	✘
3	AX3 Pro	Huawei	EMUI (Linux-based)	Wi-Fi 6	✔	✔	✔	10	✔
4	AR6140E-9G-2AC*	Huawei	VRP (Linux-based)	-	✘	✘	✔	10	✘
5	V6G	360	360OS(Linux-based)	Wi-Fi 6	✔	✔	✔	1	✔
6	Magic R365	H3C	Comware(Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
7	W30E	Tenda	Linux-based	Wi-Fi 6	✔	✔	✔	1	✔
8	RAX1800Z	China Mobile	AOS(Linux-based)	Wi-Fi 6	✔	✔	✔	10	✔
9	X32 Pro	Ruijie	RGOS(Linux-based)	Wi-Fi 6	✔	✔	✔	1	✔
10	Redmi RA81	Xiaomi	MiWiFi(Linux-based)	Wi-Fi 6	✔	✔	✔	1	✔
11	MW300R	Mercury	Vxworks-based	Wi-Fi 4	✔	✘	✔	1	✘
12	X30G	Mercury	Linux-based	Wi-Fi 6	✔	✔	✔	1	✔
13	RAX50	Netgear	DumaOS(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
14	RT-AX89X	ASUS	AsusWrt(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
15	E9450	Linksys	Linux-based	Wi-Fi 6	✔	✔	✔	10	✔
16	QUANTUM D2G	Wavlink	Linux-based	Wi-Fi 5	✔	✔	✔	10	✔
17	CF-616AC	Comfast	OrangeOS(Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
18	DI-7003GV2*	D-Link	Linux-based	-	✔	✔	✔	1	✔
19	AX3000	ZTE	ZXR10ROS(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
20	M80*	IP-COM	Linux-based	-	✔	✔	✔	1	✔
21	SK-WR6640X	Skyworth	Linux-based	Wi-Fi 6	✔	✔	✔	10	✔
22	VE5200G*	Volans	Linux-based	-	✔	✔	✔	1	✔
23	NBR1009GPE	Netcore	NOS(Linux-based)	-	✔	✔	✔	1	✔
24	Wimaster*	Wimaster	Linux-based	-	✔	✔	✔	10	✔
25	IK-Enterprise*	iKuai	iKuaiOS(Linux-based)	-	✔	✔	✔	10	✔
26	Instant On AP22	Aruba	ArubaOS(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
27	EdgeRouter X*	Ubiquiti	Linux-based	-	✔	✔	✔	10	✔
28	AX1800	JdCloud	Linux-based	Wi-Fi 6	✔	✔	✔	10	✔
29	Cisco Meraki 64*	Cisco Meraki	Linux-based	-	✔	✘	✘	-	✘
30	eero pro	Amazon	Linux-based	Wi-Fi 5	✔	✔	✔	10	✔
31	Google Wi-Fi	Google	ChromeOS(Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
32	GL-MT3000	GL.iNet	Linux-based	Wi-Fi 6	✔	✔	✔	10	✔
33	pfSense 2.7.0*	pfSense	FreeBSD-based	-	✘	✘	✔	90	✘

✔means that the router is satisfied with the condition, and ✘means that the router is dissatisfied with the condition.
✔means that the router is vulnerable to our attack, and ✘means that the router is immune to our attack.
* means that the model is an enterprise router which does not support Wi-Fi by itself and needs to work together with wireless access points.

side. Second, as the attacker has replaced the NAT mapping at the router, the source port of the packet will be translated if it happens that the client sends a packet to the server at this stage. When the packet arrives at the server, it will incur a RST packet as there is no corresponding connection, which will be routed to the client, resulting in the connection terminated at the client side. In the context of the FTP hijacking attack, the attack can be deemed successful when the attacker manages to download files from the FTP server that belong to the victim client. And for the HTTP injection attack, we define the result that the client receives forged packets, and the falsified data is displayed on the web page as a successful attack. Compared with SSH DoS and FTP hijacking attacks, the conditions are more difficult to meet. As the attacker only knows that the request interval is 60 seconds for the *www.ANONYMOUS.com* website while it does not know when the client will request an update. The client may request new data during the attack, which results in the connection being terminated, and we strictly take this case as a failure.

We repeat the experiments 20 times in each tested Wi-Fi network. Each experiment is conducted independently with a renewed connection between the client and server. In order to limit the time of experiments, we take an experiment as a failure if the attacker cannot terminate the connection, download private files, or the forged data does not show up on the client's web page in 5 minutes. As mentioned before, it takes time (mostly 1 second or 10 seconds) for the mapping to disappear completely. The countdown will be refreshed if the client sends packets during this period, which may interfere with the time cost and success rate of our attack. To simulate real-world situations, we require the tested client to send requests to the server for random times, and we set the interval between two requests as a random number from 5 to 30 seconds during the 5 minutes of an experiment. We will further investigate the impacts of communication intervals between the client and server and the timeout of NAT mappings in Section VI-A.

Experimental Results. We evaluate our attack against 93 real-world Wi-Fi networks to cover the most typical Wi-Fi scenarios, e.g., Wi-Fi networks in coffee shops, hotels, shopping malls, campuses, and office buildings. As the attacker can sniff the non-encrypted packets on the air in open networks directly, we mainly launch our experiments under networks protected by WPAs from home mode networks and enterprise mode networks, e.g., 45 with WPA2-Personal enabled (home mode), 22 with WPA2-Enterprise enabled (enterprise mode), and 26 with WPA3-Personal enabled (home mode) and we do not find any network with WPA3-Enterprise enabled. The experimental results illustrate that more than 81% of the real-world Wi-Fi networks (i.e., 75 out of the 93 evaluated networks) are fully vulnerable that they satisfy all of the conditions of our attacks. For the other 18 Wi-Fi networks, 9 of them have AP isolation enabled, which prevents the detection of potential victims and thwarts the HTTP injection attack. However, the SSH DoS and FTP hijacking attacks remain unaffected. Our attack fails in 7 networks as they do not use the vulnerable routers, and we cannot get the router's external IP as described in Section IV-B in the rest two networks. We successfully acquire the external IP addresses of routers using the route-recording method in 80 networks, involving router models from 22 vendors (i.e., Ubiquiti, Amazon, Google, Tenda, ASUS, Netgear, Huawei, Linksys, Xiaomi, Ruijie, ZTE, H3C, Wavlink, Comfast, IP-COM, Skyworth, Netcore, iKuai, WiMaster, GL.iNet, JdCloud, and China Mobile). Additionally, we employ the scanning method in 11 networks, utilizing router models from 6 vendors (i.e., D-Link, Volans, pfSense, and some models of 360, Mercury, and TP-Link). In the remaining two networks using the router models from Cisco Meraki and Aruba, we fail to get the external IP address unless we log in to the control page of the router with the help of the network administrators, and we take these networks as failures.

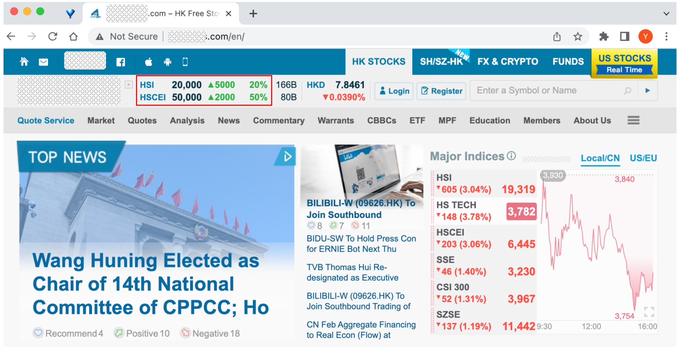
TABLE II. EXPERIMENTAL RESULTS IN OUR TESTS (ON AVERAGE).

Attack Type	Inferring Port(s)	Getting SEQ/ACK(s)	Finishing Attacking(s)	Total Time(s)	BW (pkts)	Success Rate
SSH DoS	8.1	8.4	1.0	17.5	4000	87.4%
FTP Hijacking	9.1	9.2	1.1	19.4	4000	82.6%
HTTP Injection	9.4	15.2	29.9	54.5	4000	76.1%

Next, we elaborate on our experimental results in the Wi-Fi networks as shown in Table II. In the case of the SSH DoS attack, the average time cost of identifying the client's source port is 8.1 seconds with a bandwidth of 4000 packets per second, which is much shorter than previous methods [10], [7], [6] as we only need to transfer packets in the same LAN and we are not restricted by rate limits. And the average time cost of obtaining the exact sequence and acknowledgment numbers is 8.4 seconds, as this step mainly relies on the default settings of the timeout of `CLOSE` state in NAT mappings. Besides, the communication between the client and the server may also influence the time cost. Finally, the average time cost of totally terminating an SSH connection is 17.5 seconds, and the average success rate is 87.4%. The failure cases in the tests are due to continuous communications between the client and the server (e.g., the client requests a file download and related packets will always refresh the NAT mapping). After the attack succeeds, the client's SSH terminal will be stuck for a period of time, which greatly affects the user experience.

For the FTP hijacking attack, the average time costs of identifying the client's source port and getting the sequence and acknowledgment numbers are 9.1 and 9.2 seconds, respectively, which results in a time cost of 19.4 seconds for the entire attack to get a private file from the server with a success rate of 82.6% on average. The failure cases in the tests are due to two reasons. The first is the same as the cases in the SSH DoS attack, i.e., continuous communications. The second is that the attacker happens to begin the attack when the victim connection has been constructed while the victim has not logged in.

As for the HTTP injection attack, the average time costs of identifying the client's source port, obtaining the sequence and acknowledgment numbers, and the whole attack to inject forged data to the victim's web page are 9.4 seconds, 15.2 seconds, and 54.5 seconds, respectively, with an average success rate of 76.1%. Compared with the SSH DoS and FTP hijacking attacks, it is more time-consuming as the attacker has to wait for more time to inject fake responses until the next request, and it has a lower success rate as the connection may be terminated if the client sends a request when the NAT mapping has been occupied by the attacker. Another scenario of failure occurs when the attacker fails to win the condition race of returning responses with the server, i.e., the client accepts the right data from the server. Figure 4 shows the snapshot of our HTTP injection attack against *www.ANONYMOUS.com*. The original website shows that the `HSI` number is 19,319, and it has reduced by 605 with a drop rate of 3.04%. After the attack, the victim will find that the `HSI` number is 20,000 and it has increased by 5000 with a growth rate of 20%. The same is true for the data of `HSCEI`. The attack may lead to wrong stock purchase or sale, affecting the financial status of the victim.


[redacted].com – HK Free Sto

Not Secure | [redacted].com/en/

HSI	20,000	▲5000	20%	166B	HKD	7.8461
HSCEI	50,000	▲2000	50%	80B		▼0.0390%

Wang Huning Elected as Chair of 14th National Committee of CPPCC; Ho

Recommend 4 Positive 10 Negative 18

BILIBILI-W (09626.HK) To Join Southbound

8 7 11

BIDU-SW To Hold Press Con for ERNIE Bot Next Thu

TVB Thomas Hui Re-designated as Executive

BILIBILI-W (09626.HK) To Join Southbound Trading of

CN Feb Aggregate Financing to Real Econ (Flow) at

HSI	▼605 (3.04%)	19,319
HS TECH	▼148 (3.78%)	3,782
HSCEI	▼203 (3.06%)	6,445
SSE	▼46 (1.40%)	3,230
CSI 300	▼52 (1.31%)	3,967
SZSE	▼137 (1.19%)	11,442

Fig. 4. Snapshots of web poisoning

The experimental results of 30 Wi-Fi networks in our investigations are listed in Table III. As shown in the first row, an enterprise mode Wi-Fi network with the SSID of "Campus 1[4]" is located in a campus. The router is produced by the vendor of Huawei, and it supports the generation of Wi-Fi 6. We take the experiment of SSH DoS attack in this network, and it takes the attacker 15.43 seconds to terminate an SSH connection with a success rate of 90%.

VI. DISCUSSION

In this section, we discuss the factors that affect the attack's effectiveness. We also compare our attack with existing attacks in WLANs. Besides, we extend our attack model to launch a remote TCP DoS attack from an attacker on the Internet.

A. Factors Impacting the Attack

Impacts of Traffic Load. We analyze the impact of traffic load on the attacks from two aspects: 1) the bandwidth between the client and the server, and 2) the communication interval between the client and the server. First, we extend the experiments of the FTP hijacking attack with varied bandwidths (i.e., 10KBps, 100KBps, 1000KBps) and set the

[4] We anonymized the real SSIDs of the Wi-Fi networks due to ethical considerations.

TABLE III. EXPERIMENTAL RESULTS OF TCP ATTACKS IN 30 WI-FI NETWORKS.

No.	Network Mode	SSID	Router Vendor	Wi-Fi Generation	WPA2/3 Enterprise/Personal	Attack Result	Time Cost (s)	Success Rate
1	Enterprise mode	Campus 1	Huawei	Wi-Fi 6	WPA2-Enterprise	SSH DoS	15.43	18/20
2	Enterprise mode	Campus 2	TP-Link	Wi-Fi 4	WPA2-Enterprise	FTP Hijacking	10.32	18/20
3	Enterprise mode	Campus 3	H3C	Wi-Fi 6	WPA2-Enterprise	HTTP Injection	48.87	15/20
4	Enterprise mode	Enterprise 1	TP-Link	Wi-Fi 6	WPA2-Enterprise	SSH DoS	11.56	16/20
5	Enterprise mode	Enterprise 2	TP-Link	Wi-Fi 5	WPA2-Enterprise	FTP Hijacking	11.43	18/20
6	Enterprise mode	Enterprise 3	Netcore	Wi-Fi 6	WPA2-Enterprise	HTTP Injection	87.20	15/20
7	Enterprise mode	Office building 1	TP-Link	Wi-Fi 5	WPA2-Enterprise	SSH DoS	9.56	18/20
8	Enterprise mode	Office building 2	iKuai	Wi-Fi 6	WPA2-Enterprise	FTP Hijacking	21.46	17/20
9	Enterprise mode	Office building 3	Mercury	Wi-Fi 6	WPA2-Enterprise	HTTP Injection	31.14	15/20
10	Enterprise mode	Hotel 1	Netcore	Wi-Fi 5	WPA2-Enterprise	SSH DoS	15.75	18/20
11	Enterprise mode	Hotel 2	D-Link	Wi-Fi 6	WPA2-Enterprise	FTP Hijacking	9.45	19/20
12	Enterprise mode	Hotel 2	iKuai	Wi-Fi 6	WPA2-Enterprise	HTTP Injection	71.32	16/20
13	Home mode	Restaurant 1	TP-Link	Wi-Fi 5	WPA2-Personal	SSH DoS	8.95	17/20
14	Home mode	Restaurant 2	Comfast	Wi-Fi 5	WPA2-Personal	FTP Hijacking	21.56	18/20
15	Home mode	Restaurant 3	Skyworth	Wi-Fi 6	WPA2-Personal	HTTP Injection	62.35	13/20
16	Home mode	Coffee shop 1	Mercury	Wi-Fi 4	WPA2-Personal	SSH DoS	8.98	17/20
17	Home mode	Coffee shop 2	TP-Link	Wi-Fi 4	WPA2-Personal	FTP Hijacking	9.29	18/20
18	Home mode	Coffee shop 3	Wavlink	Wi-Fi 5	WPA2-Personal	HTTP Injection	45.22	13/20
19	Home mode	Shopping mall 1	Tenda	Wi-Fi 6	WPA3-Personal	SSH DoS	24.23	18/20
20	Home mode	Shopping mall 2	TP-Link	Wi-Fi 4	WPA2-Personal	FTP Hijacking	11.44	19/20
21	Home mode	Shopping mall 3	Huawei	Wi-Fi 6	WPA3-Personal	HTTP Injection	78.44	15/20
22	Home mode	Bookstore 1	360	Wi-Fi 5	WPA2-Personal	SSH DoS	19.45	18/20
23	Home mode	Bookstore 2	Xiaomi	Wi-Fi 6	WPA3-Personal	FTP Hijacking	10.61	18/20
24	Home mode	Bookstore 3	H3C	Wi-Fi 6	WPA3-Personal	HTTP Injection	56.12	14/20
25	Home mode	Experience store 1	Xiaomi	Wi-Fi 6	WPA3-Personal	SSH DoS	16.97	17/20
26	Home mode	Experience store 2	Huawei	Wi-Fi 6	WPA3-Personal	FTP Hijacking	23.98	18/20
27	Home mode	Experience store 3	Xiaomi	Wi-Fi 5	WPA2-Personal	HTTP Injection	52.14	16/20
28	Home mode	Cinema 1	Ruijie	Wi-Fi 5	WPA2-Personal	SSH DoS	8.89	19/20
29	Home mode	Cinema 2	Mercury	Wi-Fi 6	WPA3-Personal	FTP Hijacking	11.31	18/20
30	Home mode	Cinema 2	Huawei	Wi-Fi 6	WPA3-Personal	HTTP Injection	54.26	16/20

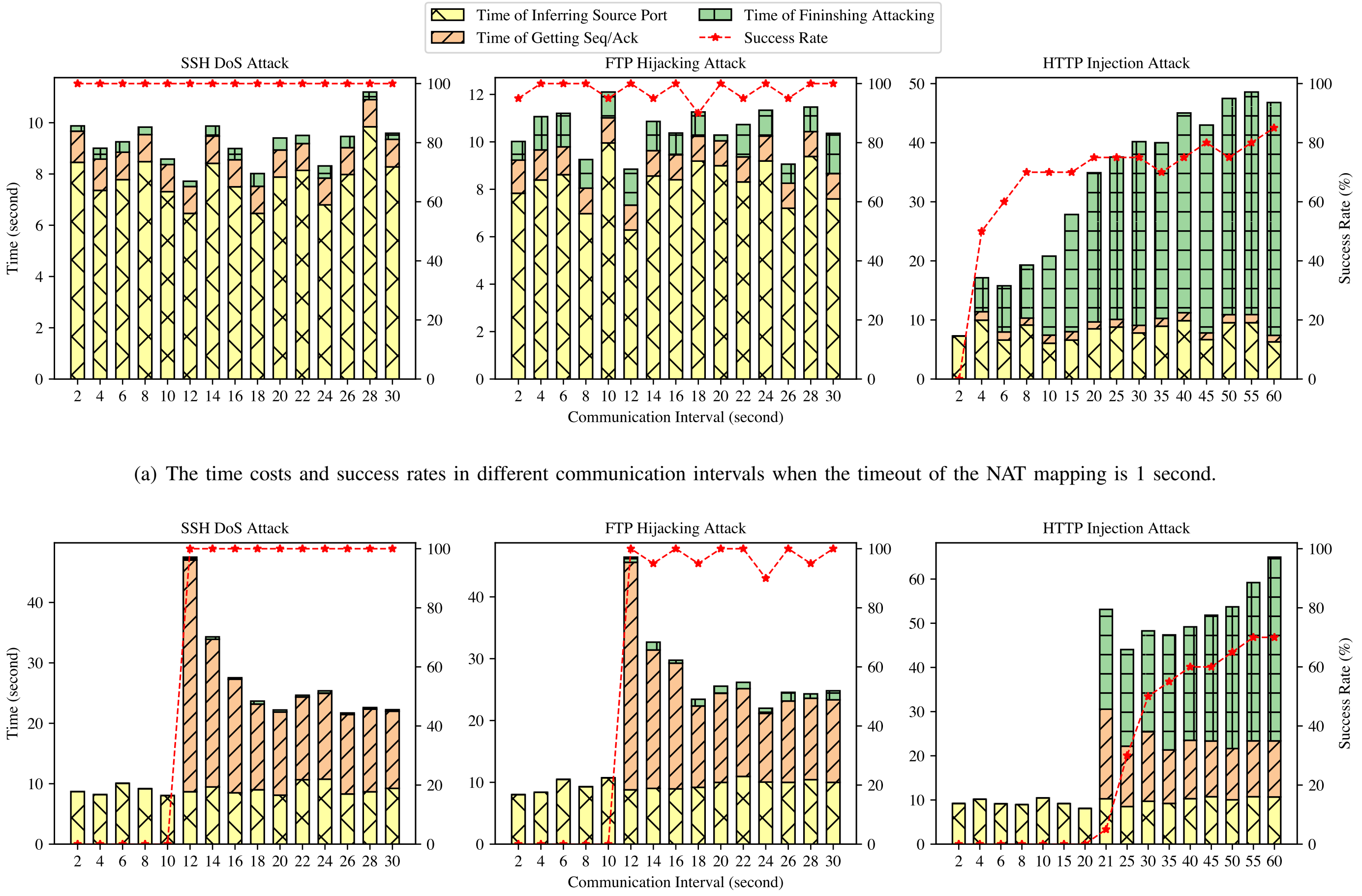
communication interval to 16 seconds with a NAT mapping timeout of 10 seconds. We repeat the experiment 50 times for each bandwidth and record the time cost and the number of successful attacks. The experimental results show that the average time costs (i.e., 29.46 seconds, 28.78 seconds, 29.21 seconds) and success rates (i.e., 96%, 94%, 94%) remain largely unaffected since our attack mainly relies on the time interval left for the attacker to clean the NAT mappings.

Second, we evaluate the three attacks under various communication intervals (e.g., 2 seconds, 4 seconds, 6 seconds, etc.) between the client and server, with a bandwidth of 100KBps. We repeat the experiments 20 times for each communication interval and record the time used in each attack phase and count the successful attacks. The experimental results are shown in Figure 5.

Here, we take the FTP hijacking attack as an example. As shown in Figure 5(a), when the NAT mapping timeout is set to 1 second, if the communication interval is below 1 second, the attack will fail due to the continuously refreshed NAT mappings. When the communication interval is above 1 second, the attack can succeed with a high success rate (97.67%), where the small partial failures are due to the attacks being launched during the login phase of the FTP application. The average time cost is less affected by the communication interval, and it shows a fluctuating trend, which mainly depends on the time to infer the client's source port. Similarly, when the NAT mapping timeout is 10 seconds (as shown in Figure 5(b)), if the communication interval is below 10 seconds, the attack will fail due to the same reason. When the interval is above 10 seconds, the attack can succeed with a high success rate (97.5%). The average time cost shows a downward trend with the increment of communication interval, as the attacker can try fewer times and wait less time for the NAT mappings to be cleaned when the communication interval is longer in an attack.

Distribution of Time Cost and Failures. Besides evaluating the FTP hijacking attack, we also measure the time costs and failure reasons of the other two attacks, i.e., the SSH DoS attack and the HTTP injection attack. For the SSH DoS attack, as shown in Figure 5(a), when the NAT mapping timeout is 1 second, if the communication interval is below 1 second, the attack will fail. However, when the communication interval is larger than 1 second, the attack can always succeed. The average time cost is not very impacted by the communication interval, and it shows a slight fluctuating trend with an average value of 9.24 seconds. The main cost is incurred by inferring the client's source ports. Figure 5(b) shows similar results when the NAT mapping timeout is 10 seconds.

As for the HTTP injection attack, as shown in Figure 5(a), when the NAT mapping timeout is 1 second, if the communication interval is below 2 seconds, the attack will fail due to the continuously refreshed NAT mappings, or the connection will be terminated during the reconstruction of the original NAT mapping as stated in Section V-B. However,



(a) The time costs and success rates in different communication intervals when the timeout of the NAT mapping is 1 second.

(b) The time costs and success rates in different communication intervals when the timeout of the NAT mapping is 10 seconds.

Fig. 5. The time costs and success rates in different communication intervals and different NAT mapping timeouts.

when the communication interval is larger than 2 seconds, with the increase of the communication interval, the time cost and success rate tend to increase. The time cost mainly depends on the waiting period until the client sends a request, as injections before that will not be accepted. In this way, the longer the communication interval is, the more waiting time will cost on average. The failures mainly come from two aspects. First, the original connection may also be terminated. Second, the attacker should inject data before the response of the server when the victim client sends a new request. Similar results can be found in Figure 5(b) when the NAT mapping timeout is 10 seconds. The difference is that the time to get sequence and acknowledgment numbers occupies a larger proportion compared with that of 1 second.

Impacts of NAT Mapping Timeout. We analyze the impact of NAT mapping timeout by comparing the experimental result of the HTTP injection attack when we set the communication interval between the victim client and the server to 60 seconds, reflecting an actual client-server communication scenario in the real world. When setting the NAT mapping timeout to 1 second, we have a time cost of 46.80 seconds and a success rate of 85%. However, we observe a time cost of 64.96 seconds and a success rate of 70% when the NAT mapping timeout is 10 seconds. The reason is that a larger timeout value incurs longer probing overhead. For example, compared to the cases with a 1-second timeout, when the timeout is 10 seconds, the time to obtain sequence and acknowledgment numbers takes up a larger proportion as the attacker must wait for the mappings to be completely cleared (i.e., at least 10 seconds). Under a long timeout, the connection is more likely to be unintentionally terminated, and thus the success rate is reduced. In summary, the increase of NAT mapping timeouts will incur increased time costs and decreased success rates.

Impacts of AP Isolation. AP isolation may influence some phases of our attack. With the policy enabled, the attacker cannot probe potential victim clients in the first phase of the attack, and it cannot reconstruct the original NAT mapping at the router by sending spoofed TCP `ACK` packets to all of the potential victim clients when launching the HTTP injection attack. However, the SSH DoS and FTP hijacking attacks are not affected as the attacker does not need to send packets directly to the victim client. Besides, only less than 10% (9 out of 93) of real-world Wi-Fi networks we observed enforce AP isolation. We also enabled AP isolation on three routers (i.e., TP-Link TL-XDR6020, Linksys E5600, and Xiaomi RA81) in our laboratory and performed the two attacks. The experiment results show that the time cost and success rate are not affected by AP isolation.

B. Comparison with Prior Attacks in Wi-Fi Networks

ARP Poisoning Attack. Compared with our attack, a successful ARP poisoning attack can intercept traffic in both directions, i.e., from the victim client and the router, while our attack can only intercept TCP traffic from the router. However, ARP poisoning attack in wired or wireless LANs has been well-researched since it appeared. Users can install some open-sourced tools [8], [49], [1] to prevent the attack. Besides, some routers (e.g., TP-Link) offer built-in ARP protections. Moreover, AP isolation can also defend against the ARP poisoning attack effectively by preventing communication between clients, while our attack is only partially affected as stated in Section IV-D and Section VI-A. We make a further empirical study in 10 real-world Wi-Fi networks. Three of them have enabled AP isolation and can prevent the ARP poisoning attack. On the other Wi-Fi networks, ARP poisoning can succeed. However, we observe that it fails when we enable protections on routers and client devices (e.g., using the tool developed in [1]). In contrast, our attack can still succeed in these networks, even when these protections are in place.

Eavesdropping Attack. Against WPA2-Personal mode Wi-Fi networks, a malicious attacker who knows the pre-shared key can sniff the frames in the air of other clients. If it wants to decrypt the frames, it needs to capture the 4-way handshake frames when other clients are connecting to the network. Though the attacker can force the victim client to be detached from the current AP by sending fake deauthentication frames [46] and wait for its re-connection, the attack is perceivable by the victim that its device will lose Internet access. However, our attack is stealthier as the attacker only needs to connect to the same network and does not need to make the client disconnect from the existing network to launch the attack. Besides, it's much harder to decrypt the frames encrypted with WPA2-Enterprise mode or WPA3-Personal mode, while our attack can also influence these networks.

Rogue AP Attack. The malicious insider who knows the pre-shared key can also create a rogue clone (evil twin) of the network and entice unsuspecting victims to connect to it, thus intercepting all the traffic [2], [39]. This attack requires broadcasting the same SSID, which can be detected by the network administrator, and some routers also provide protection strategies such as Rogue AP detection [21], [24]. In contrast, our attack is stealthier as there is no specific strategy provided to detect our attack. In addition, a lightweight device compromised by the attacker remotely may not have enough resources to provide the services as a rogue AP. The attacker who is physically in the LAN can set its own device as a rogue AP, but it has to provide a stronger signal than the original AP, and thus the influence is limited to clients in close proximity. Conversely, our attack does not face the signal race that any device in the same Wi-Fi network can launch the attack and potentially influence all clients. Besides, enterprise mode Wi-Fi networks can protect clients from the Rogue AP attack. However, our empirical measurements have revealed that our attack can compromise the traffic of clients within 22 different enterprise mode Wi-Fi networks.

Compared with the prior works, our attack leverages a new side channel vulnerability of the NAT behaviors in routers that can be exploited to hijack TCP connections by off-path attackers, even in enterprise mode networks with AP isolation. Moreover, lots of strategies have been proposed to prevent prior attacks, while our attack is a novel one whose vulnerability has existed in routers for years. Additionally, our attack serves as a valuable supplementary attack in networks equipped with defense measures against existing attacks.

C. Extending the Attack Model

In our extended model, we eliminate the requirement that the attacker and the victim client have to be located in the same Wi-Fi network. Instead, we demonstrate that a remote attacker from the Internet can launch a DoS attack on TCP connections between victim clients behind a vulnerable router and an external server. We require that the attacker can send packets with spoofed source IP addresses, which is a practical assumption considering that approximately a quarter of autonomous systems still do not employ source address validation (SAV), as reported by the Spoofer project [27].

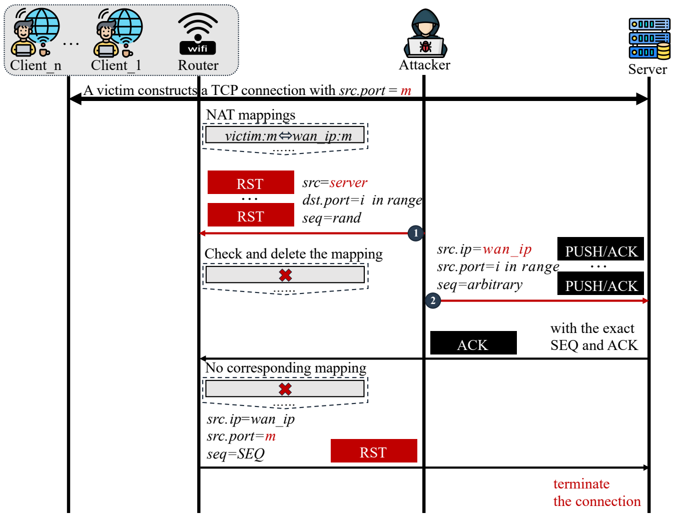


Fig. 6. Remote TCP DoS attack

Assuming that there is a live connection between an outside server and the victim client who resides behind a vulnerable router. Compared to the original attack model, the attacker cannot infer the source port of the victim client anymore using the method before. However, as we show in Figure 6, the attacker can send forged TCP `RST` packets covering the entire space of possible source ports to the public IP address of the vulnerable router. As the routers do not check the sequence number specified in TCP packets strictly, these `RST` packets can easily bypass routers' checks to clean the possible NAT mapping of the victim connection. After the NAT mapping disappears, the attacker can send forged TCP data packets to the server with a spoofed source IP address of the router's public IP, covering the entire space of possible source ports, too. The server will respond to the matched one with an `ACK` packet specified with the current sequence and acknowledgment numbers to the router. However, as there is no corresponding NAT mapping of this `ACK` packet anymore, the router will just send a `RST` packet back to the server with the sequence number received just now. Then the connection will be terminated from the server side. If the client continues communicating with the server, it will receive a TCP `RST` packet back from the server afterward. In this way, the attacker can interfere with TCP communications between the victim and server, causing a DoS attack and affecting user experience.

The attacker needs to detect such victim clients who access the Internet through these vulnerable routers. We find that it is of great convenience for the attacker to identify these vulnerable routers through open search engines [50], [13], which contain a large amount of publicly accessible devices (e.g., routers, web servers, and webcams). For example, millions of TP-Link routers with public IP addresses can be found through FOFA [13]. We estimate that there are tens of millions of vulnerable routers existing in the world which may be influenced by the attack, and we believe this attack is promising and practical and may affect many more users. We leave it as future work to validate the real-world impact in practice owing to reasons such as ethical considerations.

VII. Countermeasures

Responsible Vulnerability Disclosure. We have reported the issue to the affected manufacturers by submitting vulnerability reports and contacting them via email. At the time of writing, we have received positive responses from the OpenWrt community that confirms our findings and has released patches to fix the vulnerability, and seven router vendors (i.e., TP-Link, Huawei, Xiaomi, 360, Mercury, Ubiquiti, and Linksys) that have all acknowledged our reports and are trying to repair their products. In addition, we have been assigned 10 CVE numbers for the vulnerability in different vendors (i.e., TP-Link, Linksys, Mercury, Ruijie, D-Link, Comfast, H3C, OpenWrt, Wavlink, and 360). The other vendors are still investigating the vulnerability. We also provide them with countermeasure suggestions to mitigate the identified attack, and some of them have been adopted by the vendors. As mentioned in Section III, we outline several conditions that characterize a vulnerable router implementation. Intuitively, any breach of these conditions will render the attack ineffective.

Random Port Allocation. The first solution is for the router to use the *random selection* strategy when creating new NAT mappings. In detail, the router can choose a random port from the available port pool and record the port translation when allocating new mappings. With this strategy, the attacker cannot identify whether the port has been used by other internal hosts, and the attack will be foiled. It should be noted that some TCP punch-through schemes (e.g., TCP simultaneous open) may be influenced by *random selection* as they rely on port prediction [58]. Alternatively, clients can utilize some other common-used schemes (e.g., TURN relaying) for NAT punch-through, which will not be affected [16].

Reverse Path Validation. Another effective measure to prevent the attack is to adopt the RFC 3704 recommendation, which suggests using the strict mode to filter out forged packets. In our test, routers from ASUS, Netgear, ZTE, Aruba, Cisco Meraki, and certain models of TP-LINK, Mercury, and Huawei take this recommendation by default, thus defending against our attack. However, this strategy may introduce additional performance overhead and potentially impact the reliability of networking for certain applications (e.g., OpenVPN running on the router may be affected as the reverse path validation may interfere with packet delivery [38]).

TCP Window Tracking. As a middle device between the internal clients and outside servers, the router has to keep the necessary information about connections. However, most routers have disabled TCP window tracking for performance reasons. Nevertheless, we find that a simple TCP `RST` packet can be abused to clear the NAT mapping and be leveraged to launch our attack. In this way, we believe it essential to strictly check the sequence and acknowledgment numbers for received packets. The OpenWrt community has implemented this mitigation as they believe the performance impact should not matter anymore on any currently supported hardware.

VIII. Related Work

Traffic hijacking has been widely studied, and lots of attacks have been proposed. Vulnerabilities that lead to traffic hijacking may exist in protocols at all levels of the TCP/IP protocol stack. For instance, in the same LAN, an attacker can exploit the vulnerability of the ARP protocol to hijack network traffic by sending fake ARP packets and compromising the victim device's ARP cache, which allows the attacker to intercept, modify, or even discard the traffic of victims, thus hijacking the victims' traffic completely [19].

At the IP layer, attackers may leverage the ICMP redirect mechanism to hijack victims' traffic by placing themselves in the man-in-the-middle position [28], [3]. Recently, Feng *et al.* developed a new method to circumvent the ICMP redirect legitimacy checks in Wi-Fi networks and presented an attack to evade the security mechanisms of WPAs [9]. However, the attack targets out-of-date systems (e.g., iOS 1-8, Android before 10.0) except for the latest versions of Linux and FreeBSD. Besides, they also showed that off-path attackers from the Internet could trick public servers into redirecting their traffic to neighboring hosts with forged ICMP redirect messages, thus causing a DoS attack [12].

DNS cache poisoning attacks can also be abused to hijack traffic. In the same LAN, Herzberg *et al.* proposed three methods to circumvent source port randomization, which leverages the port allocation strategies used by NAT devices [18]. Zheng *et al.* developed an attack targeting DNS forwarders (e.g., home routers) by forcing fragmentation using attacker-owned authoritative name servers [59]. Man *et al.* proposed that a purely off-path attacker from the Internet can exploit the side channel in ICMP rate limit or the limited space for storing the next hop exception cache to infer the source ports of DNS requests and poison DNS caches maliciously [30], [31].

To hijack TCP connections so as to inject forged TCP segments into the target connection or terminate it, attackers mainly rely on various side-channel vulnerabilities. Cao *et al.* demonstrated that a global shared variable used in the challenge ACK mechanism could be abused for an off-path attacker to manipulate the victim TCP traffic [6]. Chen *et al.* showed that a timing side channel that exists in half-duplex IEEE 802.11 or Wi-Fi technology [7] and Feng *et al.* discovered a side channel in the mixed IPID assignment [10], [11], which can also be exploited to manipulate TCP traffic by off-path attackers. Tolley *et al.* demonstrated that blind in/on-path attackers could learn the virtual IP of a host behind a VPN and hijack TCP connections supposedly protected by the tunnel [51]. Besides, Schepers *et al.* discovered that modern operating systems fail to manage the security context of their transmit queues securely, thereby allowing a malicious attacker to intercept frames in Wi-Fi networks, thus hijacking TCP connections or intercepting client and web traffic [47].

Fortunately, most of the prior vulnerabilities have already been addressed [35], [6], [11], and the security community has developed corresponding defense measures against these attacks [33], [34], [23]. However, we present a new type of TCP traffic hijacking attack leveraging the vulnerabilities in routers, which can circumvent traditional defenses against TCP traffic hijacking attacks and lead to new challenges for the security communities.

IX. Conclusion

In this paper, we uncover a new off-path TCP hijacking attack in the Wi-Fi networks that leverages vulnerable routers. We find that a malicious insider can abuse the NAT port preservation strategy and insufficient reverse path validation strategy of the router to infer the existence of TCP connections from the LAN to a remote server and then obtain the sequence and acknowledgment numbers by manipulating the state of NAT mappings with forged reset packets due to the vulnerable routers disabling TCP window tracking strategy. We confirm the vulnerability in a wide range of routers from different manufacturers and evaluate the new attack in different scenarios, such as SSH DoS, FTP hijacking, and HTTP injection in various Wi-Fi networks. Finally, we suggest countermeasures, report the vulnerabilities to the affected manufacturers, and have received positive acknowledgments.

Acknowledgment

We thank our shepherd and anonymous reviewers for their thoughtful comments. This work was in part supported by China National Funds for Distinguished Young Scientists under No.61825204, NSFC under No.62132011, and Beijing Outstanding Young Scientist Program under No.BJJWZYJH01201910003011. Ke Xu is the corresponding author.

References

[1] 360-ARP, "360 total security: Free antivirus protection for home and devices," http://www.360totalsecurity.com/en/, Accessed July 2023.

[2] A. M. Alsahlany, A. R. Almusawy, and Z. H. Alfatlawy, "Risk analysis of a fake access point attack against wi-fi network," *International Journal of Scientific & Engineering Research*, vol. 9, pp. 322–326, 2018.

[3] A. Ayer, "Icmp redirect attacks in the wild," https://www.agwa.name/blog/post/icmp_redirect_attacks_in_the_wild, Accessed March 2023.

[4] F. Baker and P. Savola, "Ingress Filtering for Multihomed Networks," RFC 3704, Tech. Rep. 3704, Mar. 2004. [Online]. Available: https://www.rfc-editor.org/info/rfc3704

[5] A. Biggadike, D. Ferullo, G. Wilson, and A. Perrig, "NATBLASTER: Establishing TCP connections between hosts behind NATs," in *Proceedings of ACM SIGCOMM ASIA Workshop*, 2005.

[6] 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, pp. 209–225.

[7] W. Chen and Z. Qian, "Off-path tcp exploit: How wireless routers can jeopardize your secrets," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1581–1598.

[8] A. Chirila, "Arp antispoofer," https://www.softpedia.com/get/Security/Firewall/ARP-AntiSpoofer.shtml, Accessed July 2023.

[9] X. Feng, Q. Li, K. Sun, Y. Yang, and K. Xu, "Man-in-the-middle attacks without rogue ap: When wpas meet icmp redirects," in *2023 IEEE Symposium on Security and Privacy (SP) (SP)*. IEEE Computer Society, 2023.

[10] X. Feng, C. Fu, Q. Li, K. Sun, and K. Xu, "Off-path tcp exploits of the mixed ipid assignment," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, p. 1323–1335.

[11] X. Feng, Q. Li, K. Sun, C. Fu, and K. Xu, "Off-path tcp hijacking attacks via the side channel of downgraded ipid," *IEEE/ACM Transactions on Networking*, pp. 409–422, 2022.

[12] X. Feng, Q. Li, K. Sun, Z. Qian, G. Zhao, X. Kuang, C. Fu, and K. Xu, "Off-Path network traffic manipulation via revitalized ICMP redirect attacks," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 2619–2636.

[13] FOFA, "Fofa search engine," https://en.fofa.info/, Accessed March 2023.

[14] B. Ford, S. Guha, K. Biswas, S. Sivakumar, and P. Srisuresh, "NAT Behavioral Requirements for TCP," RFC 5382, Tech. Rep. 5382, Oct. 2008. [Online]. Available: https://www.rfc-editor.org/info/rfc5382

[15] B. Ford and P. Srisuresh, "Unintended consequences of nat deployments with overlapping address space," Internet Requests for Comments, Internet Engineering Task Force, RFC 5684, February 2010. [Online]. Available: http://www.rfc-editor.org/rfc/rfc5684.txt

[16] B. Ford, P. Srisuresh, and D. Kegel, "Peer-to-peer communication across network address translators," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '05. USENIX Association, 2005, p. 13.

[17] Y. Gilad and A. Herzberg, "Off-path tcp injection attacks," *ACM Trans. Inf. Syst. Secur.*, 2014.

[18] A. Herzberg and H. Shulman, "Security of patched dns," in *Computer Security – ESORICS 2012*. Springer Berlin Heidelberg, 2012, pp. 271–288.

[19] S. Hijazi and M. S. Obaidat, "Address resolution protocol spoofing attacks and security approaches: A survey," *Security and Privacy*, vol. 2, no. 1, pp. 1–9, 2019.

[20] M. Holdrege and P. Srisuresh, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663, Tech. Rep. 2663, Aug. 1999. [Online]. Available: https://www.rfc-editor.org/info/rfc2663

[21] Huawei, "Rogue device detection," https://support.huawei.com/enterprise/en/doc/EDOC1100096321/3eb0a62e/example-for-configuring-rogue-device-detection-and-containment, Accessed July 2023.

[22] M. Kol, A. Klein, and Y. Gilad, "Device tracking via linux's new TCP source port selection algorithm," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[23] M. Lepinski and K. Sriram, "BGPsec Protocol Specification," Internet Requests for Comments, Internet Engineering Task Force, RFC 8205, September 2017. [Online]. Available: http://www.rfc-editor.org/rfc/rfc8205.txt

[24] Linksys, "How to enable rogue ap detection on your linksys wireless-ac access point," https://www.linksys.com/support-article?articleNum=135793, Accessed July 2023.

[25] Linux, "Netfilter conntrack sysfs variables," https://docs.kernel.org/networking/nf_conntrack-sysctl.html, Accessed March 2023.

[26] ——, "rp_filter," https://sysctl-explorer.net/net/ipv4/rp_filter/, Accessed March 2023.

[27] Q. Lone, A. Frik, M. Luckie, M. Korczyński, M. van Eeten, and C. Gañán, "Deployment of source address validation by network operators: A randomized control trial," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 2361–2378.

[28] C. Low, "Icmp attacks illustrated," https://www.sans.org/reading-room/whitepapers/threats/paper/477, Accessed March 2023.

[29] V. Mahajan and S. K. Peddoju, "Deployment of intrusion detection system in cloud: A performance-based study," in *2017 IEEE Trustcom/BigDataSE/ICESS*, 2017, pp. 1103–1108.

[30] K. Man, Z. Qian, Z. Wang, X. Zheng, Y. Huang, and H. Duan, “Dns cache poisoning attack reloaded: Revolutions with side channels,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2020, pp. 1337–1350.

[31] K. Man, X. Zhou, and Z. Qian, “Dns cache poisoning attack: Resurrections with side channels,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2021, pp. 3400–3414.

[32] D. S. Mathy Vanhoef, “Macstealer: Wi-fi client isolation bypass,” https://github.com/vanhoefm/macstealer#id-test-isolation, Accessed July 2023.

[33] J. McCann, S. Deering, and J. Mogul, “Path mtu discovery for ip version 6,” Internet Requests for Comments, Internet Engineering Task Force, RFC 1981, August 1996. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1981.txt

[34] J. Mogul and S. Deering, “Path mtu discovery,” Internet Requests for Comments, Internet Engineering Task Force, RFC 1191, November 1990. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1191.txt

[35] S. Y. Nam, S. Jurayev, S.-S. Kim, K. Choi, and G. S. Choi, “Mitigating arp poisoning-based man-in-the-middle attacks in wired or wireless lan,” *EURASIP Journal on Wireless Communications and Networking*, pp. 1–17, 2012.

[36] Nmap, “The network mapper,” https://nmap.org/, Accessed March 2023.

[37] T. Ohigashi and M. Morii, “A practical message falsification attack on wpa,” *Proc. JWIS*, vol. 54, p. 66, 2009.

[38] OpenVPN, “Concepts-policyrouting-linux,” https://community.openvpn.net/openvpn/wiki/Concepts-PolicyRouting-Linux, Accessed July 2023.

[39] R. Orsi, “Understanding evil twin ap attacks and how to prevent them,” https://www.darkreading.com/attacks-breaches/understanding-evil-twin-ap-attacks-and-how-to-prevent-them, 2018.

[40] pfSense, “pfsense- world’s most trusted open source firewall,” https://www.pfsense.org/, Accessed July 2023.

[41] Ping, “Linux manual page,” https://man7.org/linux/man-pages/man8/ping.8.html, Accessed March 2023.

[42] J. Postel, “Transmission Control Protocol,” RFC 793, Tech. Rep. 793, Sep. 1981. [Online]. Available: https://www.rfc-editor.org/info/rfc793

[43] Z. Qian and Z. M. Mao, “Off-path tcp sequence number inference attack - how firewall middleboxes reduce security,” in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 347–361.

[44] Z. Qian, Z. M. Mao, and Y. Xie, “Collaborative tcp sequence number inference attack: How to crack sequence number under a second,” in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012, p. 593–604.

[45] P. Richter, F. Wohlfart, N. Vallina-Rodriguez, M. Allman, R. Bush, A. Feldmann, C. Kreibich, N. Weaver, and V. Paxson, “A multi-perspective analysis of carrier-grade nat deployment,” in *Proceedings of the 2016 Internet Measurement Conference*. New York, NY, USA: Association for Computing Machinery, 2016, p. 215–229.

[46] D. Schepers, A. Ranganathan, and M. Vanhoef, “On the robustness of wi-fi deauthentication countermeasures,” in *WiSec ’22: 15th ACM Conference on Security and Privacy in Wireless and Mobile Networks, San Antonio, TX, USA, May 16 - 19, 2022*. ACM, 2022, pp. 245–256.

[47] ——, “Framing frames: Bypassing Wi-Fi encryption by manipulating transmit queues,” in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[48] D. Senie and P. Ferguson, “Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing,” RFC 2827, Tech. Rep. 2827, May 2000. [Online]. Available: https://www.rfc-editor.org/info/rfc2827

[49] shARP, https://github.com/europa502/shARP, Accessed July 2023.

[50] SHODAN, “Search engine for the internet of everything,” https://www.shodan.io/, Accessed March 2023.

[51] W. J. Tolley, B. Kujath, M. T. Khan, N. Vallina-Rodriguez, and J. R. Crandall, “Blind In/On-Path attacks and applications to VPNs,” in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2021, pp. 3129–3146.

[52] Traceroute, “Linux manual page,” https://man7.org/linux/man-pages/man8/traceroute.8.html, Accessed March 2023.

[53] M. Vanhoef, “Fragment and forge: Breaking Wi-Fi through frame aggregation and fragmentation,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 161–178.

[54] M. Vanhoef, P. Adhikari, and C. Pöpper, “Protecting wi-fi beacons from outsider forgeries,” in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, 2020, p. 155–160.

[55] M. Vanhoef and F. Piessens, “Key reinstallation attacks: Forcing nonce reuse in wpa2,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1313–1328.

[56] ——, “Release the kraken: new kracks in the 802.11 standard,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 299–314.

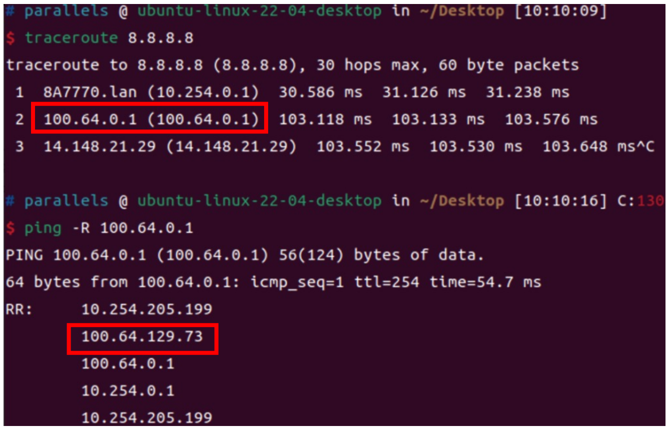
[57] M. Vanhoef and E. Ronen, “Dragonblood: Analyzing the dragonfly handshake of wpa3 and eap-pwd,” in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[58] Z. Yongjun and Z. Shiquan, “Nat hole punching based on simultaneous tcp open,” in *IEEE Computer Society*, USA, 2013, p. 235–238.

[59] X. Zheng, C. Lu, J. Peng, Q. Yang, D. Zhou, B. Liu, K. Man, S. Hao, H. Duan, and Z. Qian, “Poison over troubled forwarders: A cache poisoning attack targeting DNS forwarding devices,” in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 577–593.

APPENDIX

A. *Experimental Results of Probing the Router’s External IP Address*



```
# parallels @ ubuntu-linux-22-04-desktop in ~/Desktop [10:10:09]
$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  8A7770.lan (10.254.0.1)  30.586 ms  31.126 ms  31.238 ms
 2  100.64.0.1 (100.64.0.1)  103.118 ms  103.133 ms  103.576 ms
 3  14.148.21.29 (14.148.21.29)  103.552 ms  103.530 ms  103.648 ms^C

# parallels @ ubuntu-linux-22-04-desktop in ~/Desktop [10:10:16] C:130
$ ping -R 100.64.0.1
PING 100.64.0.1 (100.64.0.1) 56(124) bytes of data.
64 bytes from 100.64.0.1: icmp_seq=1 ttl=254 time=54.7 ms
RR:     10.254.205.199
        100.64.129.73
        100.64.0.1
        10.254.0.1
        10.254.205.199
```

Fig. 7. The snapshot of the probing the external IP address of the router.

Figure 7 show the snapshot of the method to probe the external IP address of the router. The attacker, using a laptop with Ubuntu 22.04, has connected to the Wi-Fi network whose router vendor is Wimaster and whose gateway IP is 10.254.0.1. And it has been assigned with the private IP address of 10.254.205.199. Firstly, it gets the gateways along the way to 8.8.8.8 through Traceroute and finds that the second gateway’s IP is 100.64.0.1, which is a carried-grade IP address [45]. Secondly, it pings the second gateway (100.64.0.1) with the *RECORD_ROUTE* option. The result shows the routes passed, and then the attacker can get the external IP address of the router (i.e., 100.64.129.73).

B. *Full List of Tested Routers*

The detailed information of the 67 tested routers is shown in Table IV.

TABLE IV. ALL TESTED ROUTERS

No.	Router Model	Vendor	OS	Generation	Port Preservation	Reverse-path Validation Disabled	TCP Window Tracking Disabled	TCP Close Timeout (second)	Vulnerable
1	TL-XDR6020	TP-Link	Linux-based	Wi-Fi 6	✔	✔	✔	1	✔
2	TL-R473GP-AC*	TP-Link	Linux-based	-	✔	✔	✔	10	✔
3	TL-R4239GP*	TP-Link	Linux-based	-	✔	✔	✔	1	✔
4	TL-WAR1200L	TP-Link	Linux-based	Wi-Fi 5	✔	✔	✔	1	✔
5	TL-R476G	TP-Link	Linux-based	Wi-Fi 5	✔	✔	✔	1	✔
6	TL-WDR7620	TP-Link	Vxworks-based	Wi-Fi 5	✔	✘	✔	1	✘
7	TL-WR886N	TP-Link	Vxworks-based	Wi-Fi 4	✔	✘	✔	1	✘
8	AX3 Pro	Huawei	EMUI (Linux-based)	Wi-Fi 6	✔	✔	✔	10	✔
9	AR6140E-9G-2AC*	Huawei	VRP (Linux-based)	-	✘	✘	✔	10	✘
10	TC7102	Huawei	EMUI (Linux-based)	Wi-Fi 6	✔	✔	✔	10	✔
11	TC7001	Huawei	EMUI (Linux-based)	Wi-Fi 6	✔	✔	✔	10	✔
12	Q2S	Huawei	EMUI (Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
13	WS5200	Huawei	EMUI (Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
14	T6M	360	360OS(Linux-based)	Wi-Fi 6	✔	✔	✔	10	✔
15	V6G	360	360OS(Linux-based)	Wi-Fi 6	✔	✔	✔	1	✔
16	T5G	360	360OS(Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
17	P1	360	360OS(Linux-based)	Wi-Fi 4	✔	✔	✔	1	✔
18	Magic R100	H3C	Comware(Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
19	Magic R365	H3C	Comware(Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
20	Magic R2+	H3C	Comware(Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
21	W30E	Tenda	Linux-based	Wi-Fi 6	✔	✔	✔	1	✔
22	EM12	Tenda	Linux-based	Wi-Fi 6	✔	✔	✔	1	✔
23	RAX1800Z	China Mobile	AOS(Linux-based)	Wi-Fi 6	✔	✔	✔	10	✔
24	EG105G*	Ruijie	RGOS(Linux-based)	-	✔	✔	✔	1	✔
25	EG105G-V2*	Ruijie	RGOS(Linux-based)	-	✔	✔	✔	1	✔
26	EG210G-P*	Ruijie	RGOS(Linux-based)	-	✔	✔	✔	1	✔
27	NBR*	Ruijie	RGOS(Linux-based)	-	✔	✔	✔	1	✔
28	X32 Pro	Ruijie	RGOS(Linux-based)	Wi-Fi 6	✔	✔	✔	1	✔
29	Redmi RA81	Xiaomi	MiWiFi(Linux-based)	Wi-Fi 6	✔	✔	✔	1	✔
30	Redmi RA67	Xiaomi	MiWiFi(Linux-based)	Wi-Fi 6	✔	✔	✔	1	✔
31	R3L	Xiaomi	MiWiFi(Linux-based)	Wi-Fi 6	✔	✔	✔	10	✔
32	R3G	Xiaomi	MiWiFi(Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
33	CR6609	Xiaomi	MiWiFi(Linux-based)	Wi-Fi 6	✔	✔	✔	10	✔
34	MW300R	Mercury	Vxworks-based	Wi-Fi 4	✔	✘	✔	1	✘
35	X30G	Mercury	Linux-based	Wi-Fi 6	✔	✔	✔	1	✔
36	D121G	Mercury	Vxworks-based	Wi-Fi 5	✔	✘	✔	1	✘
37	YR1900MG	Mercury	Vxworks-based	Wi-Fi 5	✔	✘	✔	1	✘
38	YR1800XG	Mercury	Linux-based	Wi-Fi 6	✔	✔	✔	1	✔
39	RAX20	Netgear	DumaOS(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
40	RAX50	Netgear	DumaOS(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
41	RT-AX57	ASUS	AsusWrt(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
42	RT-AX89X	ASUS	AsusWrt(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
43	E5600	Linksys	Linux-based	Wi-Fi 6	✔	✔	✔	10	✔
44	E9450	Linksys	Linux-based	Wi-Fi 6	✔	✔	✔	10	✔
45	QUANTUM D2G	Wavlink	Linux-based	Wi-Fi 5	✔	✔	✔	10	✔
46	CF-616AC	Comfast	OrangeOS(Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
47	DI-7003GV2*	D-Link	Linux-based	-	✔	✔	✔	1	✔
48	E3630	ZTE	ZXR10ROS(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
49	AX3000	ZTE	ZXR10ROS(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
50	M80*	IP-COM	Linux-based	-	✔	✔	✔	1	✔
51	SK-WR6640X	Skyworth	Linux-based	Wi-Fi 6	✔	✔	✔	10	✔
52	VX3000	Volans	Linux-based	Wi-Fi 6	✔	✔	✔	1	✔
53	VE5200G*	Volans	Linux-based	-	✔	✔	✔	1	✔
54	MG1200AC	Netcore	NOS(Linux-based)	Wi-Fi 5	✔	✔	✔	1	✔
55	NBR1009GPE	Netcore	NOS(Linux-based)	-	✔	✔	✔	1	✔
56	Wimaster-mini*	Wimaster	Linux-based	-	✔	✔	✔	10	✔
57	Wimaster*	Wimaster	Linux-based	-	✔	✔	✔	10	✔
58	IK-Q90	iKuai	iKuaiOS(Linux-based)	Wi-Fi 6	✔	✔	✔	10	✔
59	IK-Enterprise*	iKuai	iKuaiOS(Linux-based)	-	✔	✔	✔	10	✔
60	Instant On AP22	Aruba	ArubaOS(Linux-based)	Wi-Fi 6	✔	✘	✔	10	✘
61	EdgeRouter X*	Ubiquiti	Linux-based	-	✔	✔	✔	10	✔
62	AX1800	JdCloud	Linux-based	Wi-Fi 6	✔	✔	✔	10	✔
63	Cisco Meraki 64*	Cisco Meraki	Linux-based	-	✔	✘	✘	-	✘
64	eero pro	Amazon	Linux-based	Wi-Fi 5	✔	✔	✔	10	✔
65	Google Wi-Fi	Google	ChromeOS(Linux-based)	Wi-Fi 5	✔	✔	✔	10	✔
66	GL-MT3000	GL.iNet	Linux-based	Wi-Fi 6	✔	✔	✔	10	✔
67	pfSense 2.7.0*	pfSense	FreeBSD-based	-	✘	✘	✔	90	✘

✔means that the router is satisfied with the condition, and ✘means that the router is dissatisfied with the condition.
✔means that the router is vulnerable to our attack, and ✘means that the router is immune to our attack.
* means that the model is an enterprise router which does not support Wi-Fi by itself and needs to work together with wireless access points.