SODIUM-24

natjack

A NEW ATTACK CLASS AGAINST NETWORK INFRASTRUCTURE DEVICES

Malcolm Stagg

Independent Researcher

SODIUM-24, LLC

NatJack White Paper

Revision 1.0 (August 2026)

www.sodium24.com

For the Lord gives wisdom; From His mouth come knowledge and understanding;

Proverbs 2:6

I never planned to work on this area of research. Four years ago, in March 2022, by the grace of God I observed an unexpected router behavior, seemingly by accident, and began this research journey.

I greatly appreciate the support of my family and friends as I worked to develop this attack methodology. I especially thank my mother, Vicki Stagg, and brother, Andrew Stagg, for your unconditional support. My sincere thanks to Neil Graves for your help obtaining routers, finding obscure protocol docs, and advice on handling a complex disclosure.

I would like to thank all the vendors and CERT/CC for engaging with this research.

Friends at Summer Moon: I enjoyed hanging out at the shop with y'all and drinking some great coffee while working on a lot of this project. Don't worry, I didn't hack your router!

To God be the glory.

1. EXECUTIVE SUMMARY

NatJack is a newly disclosed network address translation (NAT) table manipulation attack class effective against most virtual and physical network infrastructure performing NAT. Affected systems include Linux Netfilter-based routers which are ubiquitous today. These attacks can be performed to hijack existing TCP connections traversing through the NAT, to intercept and maliciously alter UDP DNS responses, and to perform denial-of-service attacks.

The NatJack attack class affects an entire ecosystem of virtual and physical network infrastructure devices. An initial round of coordinated disclosure took place through CERT/CC case VU#268870 to 13 affected vendors. In-depth testing was performed across 23 products and 32 configurations, resulting in 95 vulnerability reports, relating the described issues to individual product implementations, which were disclosed to affected vendors. The attack class was publicly disclosed at Black Hat USA on August 6, 2026.

All tested routers and network infrastructure devices performing any form of network address translation have been found to be vulnerable to some or all of the NatJack attacks, including many hardware and software devices using completely different codebases not based on Netfilter. Most modern network infrastructure relies on design assumptions that have remained unchallenged for decades since their original development. While these assumptions historically held under cooperative network environments, some no longer withstand adversarial conditions.

The Linux kernel has had the same NAT behavior for many years, and it appears that nearly all Linux-based routers in use today are vulnerable. Additionally, systems such as Docker and Kubernetes typically use Netfilter for their network bridges and are affected by this attack class. Many public cloud services for containerization and virtualization are similarly affected.

This class of attack can potentially be used against workplaces, hotels, cell phone networks, and any other network where a router or CGNAT is shared between multiple users. Local and cloud environments running untrusted containers or virtual machines could also be targeted. In some cases, these attacks may be difficult to detect because time-to-live (TTL) values can be manipulated on attack packets so that they traverse no further than the target NAT. Protocols such as HTTP, DNS, proprietary Telnet-based implementations, and email protocols such as SMTP and IMAP are all potentially vulnerable to a hijacking attack when used without strong encryption.

Perhaps the most impactful attack is a TCP hijacking attack, where an attacker downstream[1] from the NAT overwrites entries in its NAT table, redirecting TCP traffic intended for another downstream machine to the attacker. Hijacking a TCP connection gives the attacker the ability to communicate directly with the target server while impersonating the victim, potentially accessing sensitive data, bypassing authentication schemes, and sending maliciously crafted data. Under certain conditions, this demonstrated behavior allows man-in-the-middle (MITM) attacks to be performed against real-world HTTP connections.

Many of these NAT table attacks have similar impact to an ARP poisoning attack, but bypass all existing mitigations. It is possible to perform attack variations in situations where an ARP attack would fail, including

[1] Downstream refers to the LAN side of the router, where endpoints are assigned IPs from a private RFC 1918 address space. Upstream refers to the WAN side of the router, which may be either a private or public IP space depending on network configuration.

cases where multiple levels of NATs are present and cases where the attacker and victim are positioned in completely different subnets and broadcast domains with only a shared NAT in common.

This is a systemic issue bigger than any one specific implementation and many unique NAT implementations have been found to be vulnerable. This report will focus primarily on Netfilter as one representative open-source implementation. Testing and initial disclosure took place to the following vendors:

- Amazon AWS
- Apple
- Cisco
- Docker
- FreeBSD
- Google / Google Cloud
- Kubernetes
- Linux Kernel
- Microsoft / Azure
- NETGEAR
- OpenBSD
- OpenWRT
- SpaceX Starlink

Patch status updates on tested products will be made available on https://natjack.io.

2. IMPACT OF ATTACKS

Five types of vulnerabilities are described in this white paper, with severity and impact indicated:

2.1. VULNERABILITY 1: DENIAL OF SERVICE

Typical Severity Score[2]: 7.4 (High)
Vector String: CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:C/C:N/I:N/A:H

The attacker can effectively prevent all other users from creating any new connections or accessing any network or internet resources, strongly impacting availability.

2.2. VULNERABILITY 2: VICTIM IP AND PORT INFORMATION DISCLOSURE

Typical Severity Score: 4.7 (Medium)
Vector String: CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:C/C:L/I:N/A:N

Attacker knowledge of the subnet IP and ephemeral port of the victim would not generally be considered serious information disclosure (except in rare cases where *any* knowledge of a connection to a specific target server could be considered sensitive information). However, this knowledge enables other attacks to be performed against the connection. These additional attacks can have significantly greater impact on confidentiality, integrity, and availability.

[2] The severity of each vulnerability may vary somewhat based on individual vendor implementations

2.3. VULNERABILITY 3: UDP DNS HIJACKING

Typical Severity Score: 10.0 (Critical)
Vector String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:L

The attacker can, with some probability, retrieve the full contents of a UDP DNS response intended for the victim, including requested hostnames, impacting confidentiality. After obtaining the DNS response, the attacker can use the included parameters and ports to craft a malicious DNS response and return it to the victim. This could be used to redirect the victim to an attacker-controlled webpage, strongly impacting integrity. The attacker can also cause the victim's DNS requests to begin failing, preventing access to network or internet resources.

2.4. VULNERABILITY 4: TCP/IP HIJACKING (DOWNSTREAM SPOOFING)

Typical Severity Score: 9.6 (Critical)
Vector String: CVSS:3.1/AV:A/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

Hijacking a TCP connection gives the attacker the ability to communicate directly to the target server while impersonating the victim, potentially accessing sensitive data, bypassing authentication schemes, and sending maliciously crafted data.

The attacker can receive valid TCP data from the target server intended for another client, strongly impacting confidentiality. The attacker can also remove any existing TCP connection, strongly impacting availability, and can send arbitrary data to the target server, while impersonating the victim, strongly impacting integrity. Under certain conditions, the demonstrated behavior allows man-in-the-middle (MITM) attacks to be performed against real-world HTTP connections.

2.5. VULNERABILITY 5: TCP/IP HIJACKING (UPSTREAM SPOOFING)

Typical Severity Score: 10.0 (Critical)
Vector String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

This attack variation enables TCP connection hijacking when the attacker and victim are positioned in entirely different subnets and broadcast domains, with only a shared NAT in common, when an external attacker-controlled server is available.

Hijacking a TCP connection gives the attacker the ability to communicate directly to the target server while impersonating the victim, potentially accessing sensitive data, bypassing authentication schemes, and sending maliciously crafted data. For this attack to succeed, knowledge is required of the external ephemeral port used by the victim. However, when an exploit chain is used, the port can be quickly determined and attack complexity becomes much lower.

The attacker can receive valid TCP data from the target server intended for another client, strongly impacting confidentiality. The attacker can also remove any existing TCP connection, strongly impacting availability, and can send arbitrary data to the target server, while impersonating the victim, strongly impacting integrity. Under certain conditions, the demonstrated behavior allows hijacking and spoofing attacks to take place against real-world HTTP connections which use HTTP keepalive.

3. ERRATA AND REVISION HISTORY

Revision 1.0 (August 24, 2026):

- Initial public release.

4. CONTENTS

5. BACKGROUND

5.1. MODERN NETWORK TRUST BOUNDARIES

While early ethernet networks typically contained trusted peers and literally used a shared wire transmitting packets between all connected machines, modern networks have evolved to generally treat connected peers as adversarial rather than cooperative, and have adopted strict layer 2 traffic isolation policies down to the container or virtual machine level (Figure 1). These policies typically include:

- Broadcast domain isolation via VLAN and VRF at the router level
- Host isolation via switch port isolation
- Tenant/workload isolation via virtual switch (vSwitch) isolation

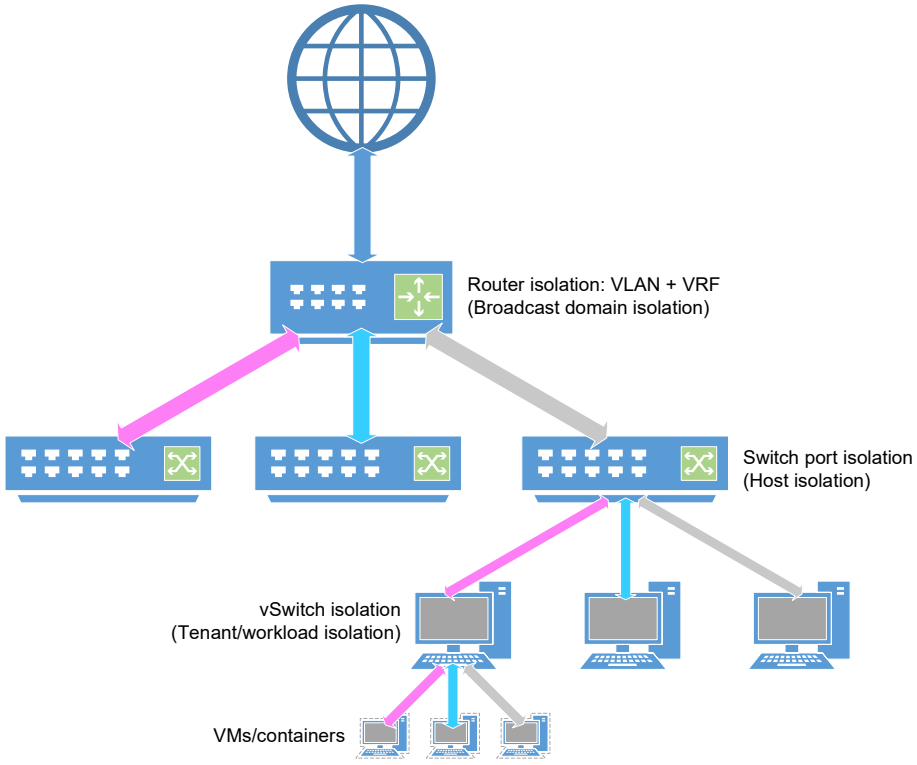

Figure 1: Modern network layer 2 traffic isolation from the broadcast domain level to individual virtual machines on a shared host

5.1.1. ARP SPOOFING MITIGATIONS

Address resolution protocol (ARP) spoofing was an early attack made possible when an untrusted attacker-controlled device was placed on the same local network as a victim device. The attacker could send out spoofed ARP packets to the victim indicating that it was router, and to the router indicating that it was the victim. As a result, all traffic between the router and victim would flow through the attacker, allowing the attacker to act as a man in the middle, observing and manipulating traffic (Figure 2).

Figure 2: Attack setup for an ARP spoofing attack

Even though the attacker and victim had to be in the same broadcast domain for this attack to succeed, modern networks have taken the threat of an ARP spoofing attack seriously enough to apply several mitigation techniques, including:

- Switch port isolation
- Dynamic ARP Inspection (DAI)
- DHCP snooping
- ARP Access Control Lists (ACLs)
- Limiting MAC addresses per port

These mitigation techniques are effective at preventing most common layer 2 attacks.

5.1.2. IP ADDRESS EXHAUSTION

TCP/IP was developed in 1974 by Vint Cerf and Bob Kahn, as an experimental protocol, with a 32-bit (4.3 billion) IP address space, 65536 source/destination ports, and 32-bit sequence numbers to provide reliable data ordering, also preventing blind data injection attacks. Starting in the 1990's, the problem of IP address exhaustion became apparent, with global IP pools now exhausted as of around 2011.

The permanent solution to IP address exhaustion is to add more IPs with IPv6. Unfortunately, adoption of the new standard has been slow, so network address translation (NAT) was proposed as a "temporary" solution to allow internal networks to have a shared external IP. NAT-enabled routers were widely deployed in the late 1990's to early 2000's, and since then NAT has become ubiquitous for more than just routers and internal trusted networks. It is common today that unique workloads and tenants are now positioned behind the same NAT device in corporate networks, cloud infrastructure, virtual machines, and container platforms.

5.1.3. NAT SECURITY BOUNDARY

Although not originally intended to function as a security boundary, NAT today provides a layer 3/layer 4 shared infrastructure layer which bridges the gap between the strong layer 2 traffic isolation in a local network and merged internet traffic flowing from a shared IP address (Figure 3). There is an implicit assumption today that NAT will respect network isolation boundaries in its routing decisions, but unfortunately, design-level assumptions for NAT are largely based on legacy trust boundaries for local networks, and these network isolation boundaries can be fairly easily broken.

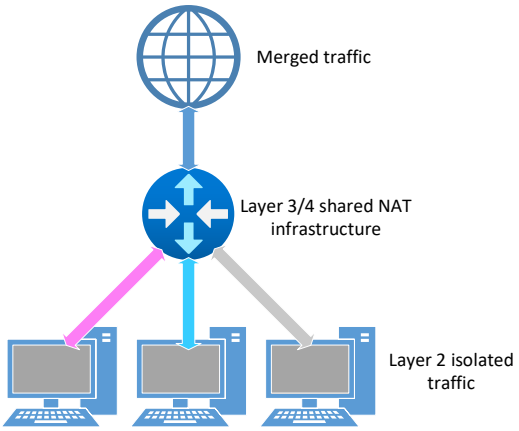

Figure 3: NAT security boundary showing the relation between shared layer 3/4 infrastructure and layer 2 traffic isolation

5.2. NETWORK ADDRESS TRANSLATION

To route data between a local computer on an internal network and a remote server, routers typically perform network address translation (NAT) on traffic sent between the machines[3]. Whenever an outgoing TCP or UDP packet from a computer on the internal network traverses through the router, it will translate the packet's local IP and port to the router's external IP, assigning an unused external port for the connection (Figure 4).

When an incoming packet from the remote server arrives at the router, the router performs any needed packet validations, then forwards the packet back to the computer which initiated the connection on the internal network. To maintain this forwarding, the router must keep a list of active connections, known as the NAT table. Connection events are monitored, with appropriate timeouts, to remove stale entries from the NAT table when they are no longer in use.

Without the benefits provided by NAT, the limited IPv4 address space of approximately 4.3 billion IPs would be quickly exhausted. A NAT device also can add an extra layer of security by effectively acting as a firewall, filtering packets which don't belong to a known connection.

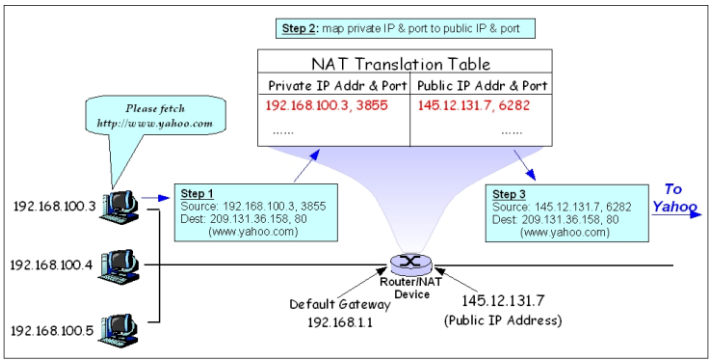

Figure 4: An example NAT translation when retrieving a website[4]

[3] https://datatracker.ietf.org/doc/html/rfc2663#section-3

[4] Yangliy at English Wikibooks, Public domain, via Wikimedia Commons

5.3. LINUX KERNEL NETFILTER

To quote from Netfilter's source code, "We are the man in the middle. All the packets go through us." Netfilter is the Linux kernel's built-in module capable of performing NAT. It uses a connection monitor known as Conntrack to monitor all TCP, UDP, and other connections/streams, adding and removing these connections in the NAT table when needed. Many commercially available routers are Linux-based and use Netfilter to perform NAT.

5.3.1. NAT TABLE SIZE

Netfilter's NAT table is, by default, allocated based on the available system RAM. If the system has at least 4 GB of RAM, a 262144 entry NAT table will be allocated. If the system has 1 GB of RAM, the NAT table will contain 65536 entries. With 32 MB of RAM, it will have only 4096 entries.

5.3.2. PORT ASSIGNMENT BEHAVIOR

The external port assignment behavior of Netfilter is that it will *generally* use port preservation (Table 1). That is, the same port used by the local computer to initialize the connection will also be used as the external port selected by the NAT whenever possible. If that port is unavailable, the NAT will arbitrarily select another available port.

Table 1: NAT table example showing port preservation

Internal IP:Port	External IP:Port	Destination IP:Port
10.0.0.2:**12345**	1.2.3.4:**12345**	Server 1 (5.6.7.8:80)

Netfilter additionally uses an endpoint-independent mapping behavior (Table 2), such that once an internal-to-external port mapping is established, the same port mapping will be used when sending packets from the internal endpoint to *any* destination endpoint. This behavior can be observed in cases where port preservation was not possible. In this example, port 22334 is assigned for communication with server 1 due to a port conflict, then this same port is also used for communication with server 2.

Table 2: NAT table example showing endpoint-independent mapping

Internal IP:Port	External IP:Port	Destination IP:Port
10.0.0.3:12345	1.2.3.4:**22334**	Server 1 (5.6.7.8:80)
10.0.0.3:12345	1.2.3.4:**22334**	Server 2 (8.7.6.5:8080)

5.3.3. FULL TABLE BEHAVIOR

When the NAT table becomes full, Netfilter will attempt to evict entries at random. It will, however, not evict any entries which correspond to an *established* connection. For UDP, which is connectionless, Conntrack looks at whether a packet has been received in both directions over a period of 2 seconds. In this case, the timeout is extended and the entry will not be evicted from the NAT table. An entry in this state is referred to as "assured," as opposed to an "unassured" entry where only an outgoing packet has been seen.

For TCP, a connection which has performed the three-way SYN, SYN+ACK, ACK handshake (described below) is marked as "assured" and will not be evicted until it is detected as being closed by Conntrack. Netfilter makes **NF_CT_EVICTION_RANGE** (defined as 8) attempts to find an entry to evict to make room for a

new entry before giving up. If no eviction can take place to free up room in the NAT table for the new entry, any packets corresponding to the new entry will be dropped.

5.4. TCP CONNECTION HANDLING

The Transmission Control Protocol (TCP) allows a reliable connection to be set up between two endpoints. The connection is established by performing a three-way handshake, where one endpoint will send a SYN (synchronization) packet, the second endpoint replies with a SYN+ACK packet, and the first endpoint responds with an ACK (acknowledge) packet (Figure 5). TCP packets contain a 32-bit sequence number (SEQ) and acknowledge number (ACK), both of which are set during the three-way handshake, then incremented as traffic is sent. For traffic to be successfully received by an endpoint, both SEQ and ACK must be set to valid numbers indicating the position of the packet in the data stream, otherwise the packet will be dropped. This scheme helps to prevent erroneous or malicious data from being injected into a TCP connection.

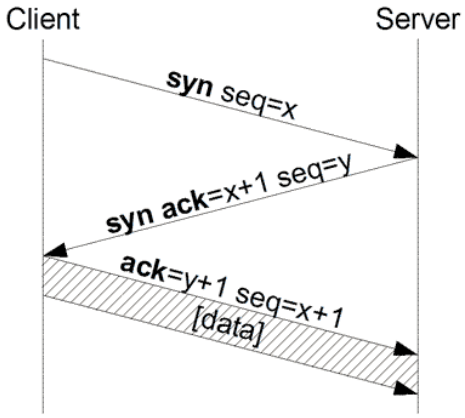

Figure 5: TCP 3-way handshake[5]

5.4.1. PACKET FILTERING

As well as maintaining the SEQ and ACK values, each TCP connection maintains a window for flow control. The window size defines how much data can be sent before an acknowledgement is required. Using window scaling, a window size of up to 2^{30}, or 1 GiB, can be defined.

Conntrack keeps track of the *maximum seen* window size for each connection, and Netfilter uses this window to determine whether a TCP packet's sequence numbers are valid. Netfilter's filtering for acceptable packets is similar but less strict than the TCP connection itself, ensuring that no valid packets are blocked. If a packet is received with SEQ or ACK outside of its window, the packet will be dropped by Netfilter instead of forwarded on to the destination.

[5] Image taken from the Italian Wikipedia, CC BY-SA 3.0, via Wikimedia Commons

5.4.2. TCP STATE MACHINE

Conntrack maintains a state machine for each connection, with default timeouts for each TCP connection state, after which time the connection will be removed from the NAT table, as shown in Table 3.

Table 3: Conntrack default TCP state timeouts

TCP State	Timeout
SYN_SENT (SS)	2 minutes
SYN_RECV (SR)	60 seconds
ESTABLISHED (ES)	**5 days**
FIN_WAIT (FW)	2 minutes
CLOSE_WAIT (CW)	60 seconds
LAST_ACK (LA)	30 seconds
TIME_WAIT (TW)	2 minutes
CLOSE (CL)	10 seconds
SYN_SENT2 (S2)	2 minutes

Two additional states are defined by Conntrack for IGNORED (IG) and INVALID (IV) packets, which are either accepted but have no effect, or are not accepted, respectively. For nonexistent connections, no state (NO) is used. Conntrack processes outgoing packets according to the state machine shown in Table 4. For incoming packets, Conntrack uses the state machine shown in Table 5.

Table 4: Conntrack's outgoing packet state machine, showing the new state when a packet is received in each state

Packet \ State	NO	SS	SR	ES	FW	CW	LA	TW	CL	S2
SYN	SS	IG	IG	IG	IG	IG	IG	SS	SS	S2
SYN+ACK	IV	IV	SR	IV	IV	IV	IV	IV	IV	SR
FIN	IV	IV	FW	FW	LA	LA	LA	TW	CL	IV
ACK	ES	IV	ES	ES	CW	CW	TW	TW	CL	IV
RST	IV	CL	CL	CL	CL	CL	CL	CL	CL	CL

Table 5: Conntrack's incoming packet state machine, showing the new state when a packet is received in each state

Packet \ State	NO	SS	SR	ES	FW	CW	LA	TW	CL	S2
SYN	IV	S2	IV	IV	IV	IV	IV	SS	IV	S2
SYN+ACK	IV	SR	IG	IG	IG	IG	IG	IG	IG	SR
FIN	IV	IV	FW	FW	LA	LA	LA	TW	CL	IV
ACK	IV	IG	SR	ES	CW	CW	TW	TW	CL	IG
RST	IV	CL	CL	CL	CL	CL	CL	CL	CL	CL

5.4.3. CONNECTION CREATION

Generally, Conntrack will start monitoring a new TCP connection when an outgoing SYN is sent. This will cause it to enter the SYN_SENT (SS) state as seen in Table 4. If a SYN+ACK packet is received as a reply, the connection will then enter the SYN_RECV (SR) state as seen in Table 5. Finally, when an outgoing ACK is sent in reply, the connection will enter the ESTABLISHED (ES) state. These three packets occur as part of TCP's three-way SYN/SYN+ACK/ACK handshake.

Looking closely at Table 4, there is another way for a connection to enter Conntrack's ESTABLISHED (ES) state without observing a three-way handshake. This occurs when an outgoing ACK is sent in an untracked[6] connection. This is part of Conntrack's "loose" TCP connection handling, which is enabled by default. One rationale for this behavior is that a valid TCP connection may have existed before the NAT table was constructed, for example in the case where a router is rebooted or the network path has changed. An incoming packet in reply to this ACK will not be filtered by Netfilter, since the windows for filtering have not yet been established at that point.

5.4.4. CONNECTION TERMINATION

As seen in Table 4 and Table 5, Conntrack will move a TCP connection to the CLOSE (CL) state if any incoming or outgoing RST[7] packet is received, while the connection is in any state, as long as that packet passes all the filtering checks based on the tracked TCP connection window. In the CLOSE state, the connection will remain in the NAT table for an additional 10 seconds, by default, before it is removed.

Any TCP packets which do not contain an ACK flag are considered valid as long as the packet's SEQ lies within the tracked window, whether or not the ACK number is valid or even set:

```
if (!(tcph->ack)) {
        /*
         * If there is no ACK, just pretend it was set and OK.
         */
        ack = sack = receiver->td_end;
```

Thus, any incoming or outgoing RST packet will be accepted by Netfilter, causing the connection to enter the CLOSE state and be removed from the NAT table within 10 seconds, as long as the packet has an SEQ lying *somewhere* inside the tracked window.

[6] An untracked connection refers to observed TCP packets which do not belong to any TCP connection currently being tracked by Conntrack, and which may not correspond to any existing NAT table entry.

[7] TCP RST (reset) packets are generally sent due to an unexpected error, where one side of the connection is requesting a forceful close. This differs from FIN (finish) packets which are sent during a graceful close during normal protocol conditions.

5.4.5. RFC 793 HALF-OPEN CONNECTIONS

For outgoing packets (originating downstream from the NAT), a Conntrack behavior can be exploited to allow an RST packet to be processed by Netfilter even if it *does not have* a valid SEQ within the tracked window:

```
if (((test_bit(IPS_SEEN_REPLY_BIT, &ct->status)
       && ct->proto.tcp.last_index == TCP_SYN_SET)
       || (!test_bit(IPS_ASSURED_BIT, &ct->status)
              && ct->proto.tcp.last_index == TCP_ACK_SET))
       && ntohl(th->ack_seq) == ct->proto.tcp.last_end) {
       /* RST sent to invalid SYN or ACK we had let through
        * at a) and c) above:
        *
        * a) SYN was in window then
        * c) we hold a half-open connection.
        *
        * Delete our connection entry.
        * We skip window checking, because packet might ACK
        * segments we ignored. */
       goto in_window;
}
```

As described in the comment in the code above, this case is included to properly handle "half-open" connections, as described in RFC 793, where the endpoints have become desynchronized, or one endpoint has closed the connection without knowledge of the other. The statement checks whether, in an established connection (`IPS_SEEN_REPLY_BIT`), the last processed packet (`ct->proto.tcp.last_index`) was a SYN packet. It then checks whether the ACK number of the RST packet (`th->ack_seq`) matches the SEQ of the previous SYN packet (`ct->proto.tcp.last_end`).

Although this case is likely intended to be hit by an incoming RST packet in reply to an outgoing SYN packet, it can *also* be hit when SYN and RST packets are sent consecutively in the same direction.

The `tcp.last_index` and `tcp.last_end` values can be set as follows in Netfilter's IGNORE state:

```
case TCP_CONNTRACK_IGNORE:
       /* Ignored packets:
        *
        * Our connection entry may be out of sync, so ignore
        * packets which may signal the real connection between
        * the client and the server.
        *
        * a) SYN in ORIGINAL
        * b) SYN/ACK in REPLY
        * c) ACK in reply direction after initial SYN in original.
        *
        * If the ignored packet is invalid, the receiver will send
        * a RST we'll catch below.
        */
       ...
       ct->proto.tcp.last_index = index;
       ct->proto.tcp.last_dir = dir;
       ct->proto.tcp.last_seq = ntohl(th->seq);
       ct->proto.tcp.last_end =
       segment_seq_plus_len(ntohl(th->seq), skb->len, dataoff, th);
       ct->proto.tcp.last_win = ntohs(th->window);
```

Putting all this together, it is possible to bypass the SEQ filtering of outgoing RST packets by performing the following steps while a TCP connection is in the ESTABLISHED state:

- Send an outgoing SYN packet from the downstream endpoint with SEQ=x. The packet will be handled by the IGNORE state, and will cause the `tcp.last_index` and `tcp.last_end` values to be updated.
- Send an outgoing RST packet from the downstream endpoint with ACK=x+1. This packet will be handled by the CLOSE state and will bypass the window validation check.

Sending these two packets consecutively will cause the TCP connection to immediately enter the CLOSE state, whether or not SEQ=x was actually within the tracked window. After a 10 second timeout, the entry is removed.

Note: Patch applied in June 2026 with CVE-2026-63913, but a higher complexity attack is still possible.

5.4.6. RFC 1337 TIME-WAIT ASSASSINATION

RFC 1337 describes an undesirable TCP behavior known as TIME-WAIT Assassination (TWA) where a series of packets can result in the TIME-WAIT state being "assassinated" for a TCP connection, such that it is removed from the TIME-WAIT state prematurely before the TIME-WAIT timeout (2 minutes in the case of Netfilter) has expired. If this were to occur, a new replacement connection could be created while retransmissions from the old connection are still present, possibly causing the endpoints to accept invalid data or lose synchronization with each other.

A similar behavior can actually be exploited in Netfilter to *intentionally* remove a TCP connection from the CLOSE state *without waiting* for the 10 second timeout. A simple TWA primitive was found in Netfilter as follows:

```
case TCP_CONNTRACK_SYN_SENT:
        ...
        if (((ct->proto.tcp.seen[dir].flags
                | ct->proto.tcp.seen[!dir].flags)
                & IP_CT_TCP_FLAG_CLOSE_INIT)
                || (ct->proto.tcp.last_dir == dir
                        && ct->proto.tcp.last_index == TCP_RST_SET)) {
                /* Attempt to reopen a closed/aborted connection.
                 * Delete this connection and look up again. */
                spin_unlock_bh(&ct->lock);

                /* Only repeat if we can actually remove the timer.
                 * Destruction may already be in progress in process
                 * context and we must give it a chance to terminate.
                 */
                if (nf_ct_kill(ct))
                        return -NF_REPEAT;
                return NF_DROP;
        }
```

It is possible to use this primitive behavior to perform TWA very easily:

- Send an RST packet from the downstream endpoint to place the connection in the CLOSE state.
- Send a SYN packet from the downstream endpoint to be handled by the SYN-SENT state.

When these two packets are sent consecutively, the existing connection is immediately removed from Conntrack's connection tracking by the `nf_ct_kill` function, allowing it to be immediately replaced with a new connection without delay. This behavior can, however, only be exploited from the downstream side of the NAT.

5.5. DOMAIN NAME SYSTEM (DNS)

Computers generally translate between a URL and an IP address using Domain Name System (DNS). A DNS request can be made from a client machine to a DNS server by sending a UDP packet to port 53 on the DNS server. The request packet specifies the URL to lookup, a unique identifier, and the information requested (such as an A or AAAA entry for performing IPv4 or IPv6 queries, respectively).

The DNS server will reply to each request with another UDP packet back to the originating port, containing both the unique identifier and the requested response data such as the retrieved IP address. Linux will validate a DNS response to verify that it both originated from the correct DNS server endpoint and that it contains the correct unique identifier.

When an outgoing UDP packet is sent, Netfilter will add an unassured entry to its NAT table so the UDP response will be forwarded back to the correct client machine. The unassured UDP entry will remain in the NAT table, by default, for 30 seconds, but could be prematurely evicted if the NAT table becomes full.

5.6. TERMINOLOGY

The following terms are used throughout this paper while describing components of the attack methodology:

- **Target NAT**: Router or physical/virtual device performing NAT to be attacked.
- **Downstream**: Placed on the LAN-facing side of the target NAT using a private RFC 1918 range.
- **Upstream**: Placed on the WAN or internet-facing side of the target NAT.
- **Target connection/stream**: TCP connection or UDP request/response stream to be attacked.
- **Attack client**: Downstream physical or virtual machine controlled by the attacker.
- **Victim client**: Downstream physical or virtual machine requesting the target connection/stream.
- **Attack server**: Upstream physical or virtual machine controlled by the attacker.
- **Target server**: Upstream physical or virtual machine serving the target connection/stream.
- **Port preservation**: The NAT maps an internal port to the same external port.
- **Endpoint-independent mapping**: Once an internal-to-external port mapping is established, the same port mapping will be used when sending packets from the internal endpoint to *any* destination.
- **Assured entry**: An established connection or stream which will not be evicted from the NAT table.
- **Unassured entry**: An unestablished connection or a stream which can be evicted from the NAT table if it becomes full.
- **Ephemeral port**: A temporary port assigned to a client application for network communication, typically at random.

5.7. RELATED RESEARCH

Although this attack class was developed independently, I would like to acknowledge prior work by Yang et. al. for their contributions to NAT table manipulation attacks. Their group was focused on an attack scenario involving WiFi enabled routers, and they found that many routers enabled non-default, insecure options, which allowed TCP connection hijacking. That research resulted in CVE-2023-30305 to CVE-2023- 30314 for multiple routers. NatJack research has focused primarily on other attack scenarios and what is possible to exploit in a default configuration.

6. VULNERABILITY 1: DENIAL OF SERVICE

A downstream attacker can use this vulnerability to perform a trivial denial-of-service (DoS) attack, preventing the traversal of any new traffic from other downstream machines through the target NAT to the upstream network. This attack is possible because the NAT table is a finite size, and *assured* entries cannot be evicted from the NAT table when it is full.

By sending spoofed TCP three-way handshake packets back and forth, over a large range of source and destination ports, between the attack client and attack server, the attacker can create a huge number of NAT table entries which are, from the perspective of Conntrack, valid and *assured* in the ESTABLISHED state.

If the NAT table is completely full of *assured* entries, no new entries will be added to the NAT table, and all outgoing packets corresponding to these new entries will be dropped (see section 5.3.3). This effectively causes a complete DoS to all downstream machines, where they cannot create any new TCP connections or send UDP packets traversing through the NAT. By default, the timeout of ESTABLISHED TCP connections is defined as 5 days. The NAT device may become unusable for up to 5 days or until it is rebooted.

6.1. ATTACK SETUP

The network setup for this attack is illustrated in Figure 6.

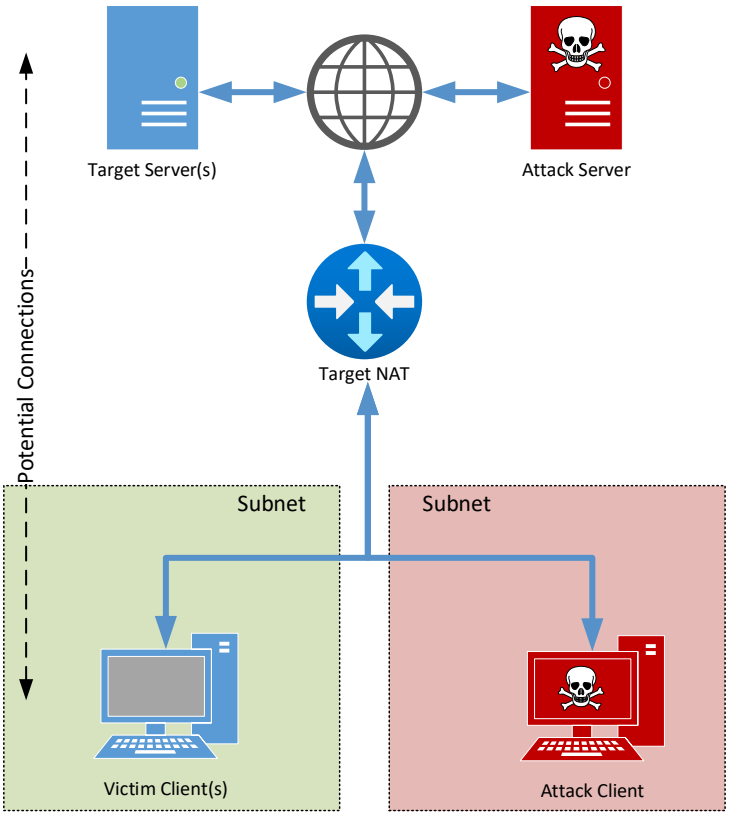

Figure 6: Network setup for the Denial-of-Service attack

6.2. VENDOR/PRODUCT VARIATIONS

Some tested products exhibited the following variations:

- **Unassured connections:** certain products could have their NAT tables filled with UDP streams or SYN floods as well as established TCP connections, also resulting in denial of service (DoS).
- **Maximum entries per client:** certain products enabled a maximum number of entries per client. In some of these products, a port-exhaustion DoS was still possible against an attacker-specified target server.

7. VULNERABILITY 2: VICTIM IP AND PORT INFORMATION DISCLOSURE

A downstream attacker can use this vulnerability to detect the presence of a connection from any downstream client machine, traversing through the NAT, to a known target server. This target server may be, for example, a specific web server where TCP traffic is serviced on port 80. Typically, the attacker would not have knowledge of the ephemeral port used by the victim client to establish the connection to this server. The ephemeral port range can be quite large. On a Linux machine, the range is generally defined as ports 32768-60999, or on a Windows machine ports 49152-65535 (defined in RFC 6335).

Due to a weakness in Netfilter's port assignment algorithm, it is possible for the attacker to automatically detect which ephemeral port was used by the victim client in establishing the connection and leak this port to the attack server. After determining the ephemeral port, the attacker can then exploit the same behavior again and scan an entire subnet to determine the IP address of the victim client which made the connection.

Knowledge of this port enables other attacks including the Vulnerability 5: TCP/IP Hijacking (Upstream Spoofing) attack (section 10), to take place much more quickly and effectively.

7.1. ATTACK SETUP

7.1.1. PORT DISCLOSURE

The network setup for a port disclosure attack is illustrated in Figure 7.

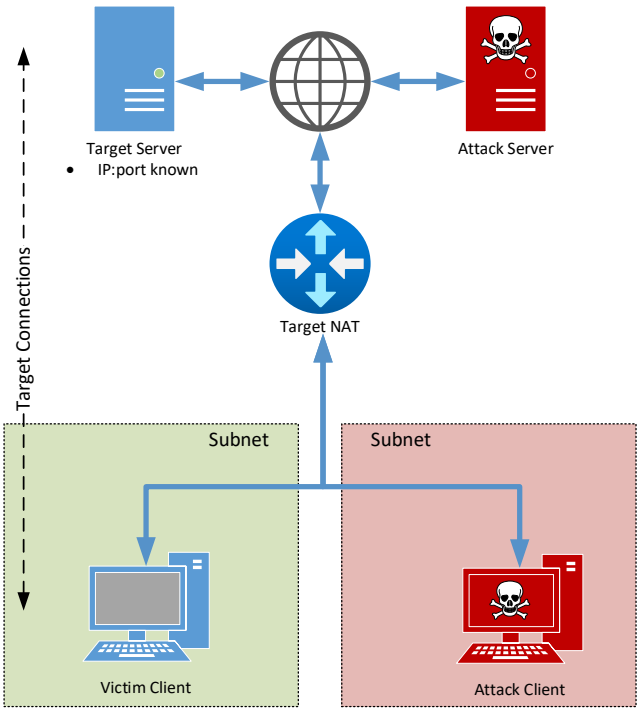

Figure 7: Network setup for the port disclosure attack

7.1.2. IP AND PORT DISCLOSURE

The attack for victim client *IP disclosure* has an additional requirement that the attack client must be capable of spoofing its IP to match the victim client, as illustrated in Figure 8. For this to work, *in most cases* the attack client and victim client machines will need to share the same internal subnet[8].

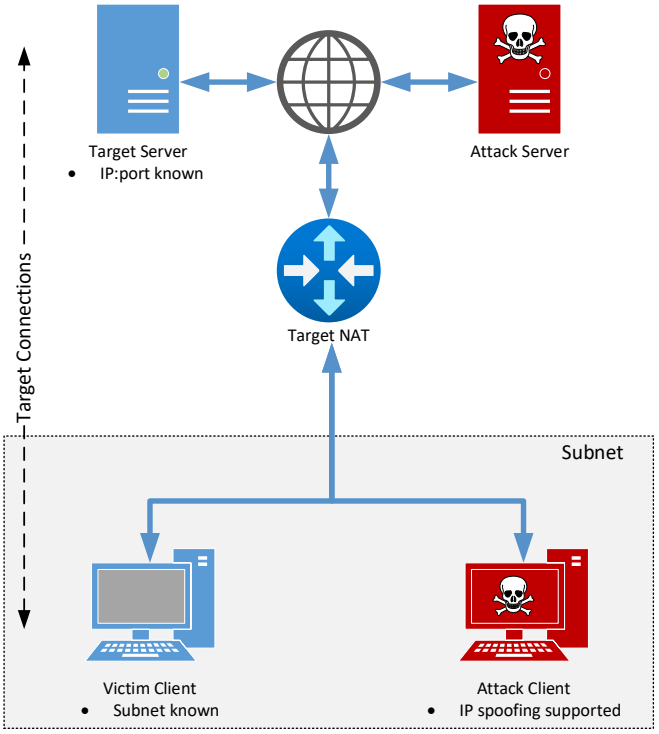

Figure 8: Network setup for the IP and port disclosure attack

7.2. VULNERABILITY DETAILS

7.2.1. NETFILTER PORT ASSIGNMENT ALGORITHM

Netfilter uses the algorithm described below to map an internal port to an external port, while avoiding port conflicts. A conflict can occur if the selected external port is already mapped to the same remote endpoint by another machine.

Case 1: Does the internal IP/internal port combination already exist in the NAT table? If so, attempt to use the same external port unless it will cause a conflict. This is the "endpoint-independent mapping" behavior.

Case 2: Otherwise, attempt to use external port = internal port unless it will cause a conflict. This is the "port preservation" behavior.

Case 3: If all else fails, assign an unused external port arbitrarily.

[8] Typically, when uRPF is properly configured, a router will drop and ignore any spoofed packets originating from a different subnet

7.2.2. PORT CONFLICT DETECTION

This algorithm is vulnerable to information disclosure because the port preservation behavior can be used to scan a target server until a port conflict is found, indicating that the internal port is already being used by a victim client. The endpoint-independent mapping behavior can then be used to leak these scan results to the attacker, allowing the attacker to learn the identity of this port.

The attacker can perform this scan by enumerating all ports from 0 to 65535, using the attack client to send a packet originating from each port to the target server. This could be a TCP SYN packet to leak information about an existing TCP connection, or a UDP packet to leak information about a UDP stream.

If the NAT is *not* using this port for an existing connection, Netfilter will select a port using the default port preservation behavior (Case 2), where the packet is sent with its external source port set to the same port as the internal attacker-chosen port. If, however, the NAT *is* already using this port for an existing connection, the port preservation behavior would cause a conflict, so Netfilter will choose another arbitrary external port for the connection (Case 3).

The attacker can then detect which external port was assigned by exploiting Netfilter's "endpoint-independent mapping" port assignment behavior. If a downstream endpoint establishes a connection to one upstream server, then uses the *same internal port* to establish a connection to a *different upstream server*, Netfilter will always attempt to use the *same external port* for both connections (Case 1).

By sending a packet from the attack client to the upstream attack server using the same internal port, the upstream attack server can detect *whether or not* a port conflict was previously detected by Netfilter. If a conflict is detected, the attacker has effectively learned that the internal port was already in use by a connection from the victim client to the target server, as seen in Table 6.

Table 6: NAT table example showing the leak of the victim client's internal port to the attack server

Internal IP:Port	External IP:Port	Destination IP:Port	Port Assignment Behavior
victim-client:**12345**	external-ip:**12345**	target-server:80	Case 2: Port preservation (default)
attack-client:**12345**	external-ip:**22334**	target-server:80	Case 3: Conflict (arbitrary port)
attack-client:**12345**	external-ip:**22334**	**attack-server**:80	Case 1: Endpoint-independent mapping

7.2.3. SEARCHING FOR THE VICTIM CLIENT IP

Once this internal port has been determined, the attacker can then go a step further and scan an entire subnet to determine which subnet IP belongs to the victim client. The attack client will need to spoof its source IP to match each IP in the subnet. A packet can then be sent from the spoofed IP and previously determined internal port to the target server, followed by an identical packet to the attack server. In the case that *no conflict* is detected by the attack server (i.e. the internal and external ports match), the attacker has successfully determined the IP of the victim client, since this is the *only downstream IP* which will not cause a port conflict when sending a packet to the target endpoint.

7.3. VENDOR/PRODUCT VARIATIONS

Some tested products exhibited the following variations:

- **Endpoint-independent mapping but not port preservation:** The attack generally took significantly longer, but could still often be completed by enumerating all ports the attacker could use to connect to the target server. Any missing ports were likely in use by a victim. The subnet IP search could still be completed.
- **Port preservation but not endpoint-independent mapping:** The attack was still often possible via the attack server sending spoofed packets back to the attack client to detect a port conflict. The victim IP could not be detected in this case.
- **Neither port preservation or endpoint-independent mapping:** The attack generally took significantly longer, but was still often possible via the attack server sending spoofed packets back to the attack client, enumerating ports the attacker could use to connect to the target server. Any missing ports were likely in use by a victim. The victim IP could not be detected in this case.

8. VULNERABILITY 3: UDP DNS RESPONSE HIJACKING

A downstream attacker can use this vulnerability to break intended trust boundaries to intercept a UDP DNS response intended for another downstream client machine, then craft their own response as a replacement. When a downstream client machine sends a UDP packet, traversing through the NAT, to an upstream DNS server, a new *unassured* entry is added to the NAT table. This entry will be checked when a response comes back from the DNS server to the NAT, directing the response back to the correct downstream client machine. It is possible for the attacker, with some probability, to remove and replace this entry with one which instead directs the response to the attack client. The attacker can then retrieve the DNS response and use it to craft a malicious response for the victim client.

8.1. ATTACK SETUP

8.1.1. DOWNSTREAM SPOOFING

The "downstream spoofing" network configuration, as illustrated in Figure 9, is preferred because it allows a spoofed DNS response to be sent immediately from the attack client to the victim client after hijacking. The attack client must be capable of spoofing its IP to match the DNS server and send traffic to the victim client. For this to work, *in most cases* the attack client and victim client machines will need to share the same layer 2 broadcast domain (usually equivalent to layer 3 subnet)[9].

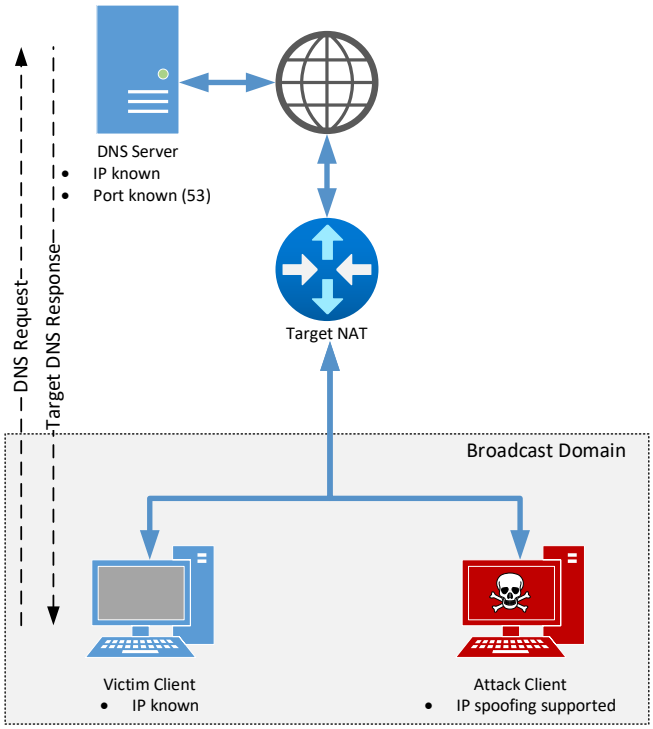

Figure 9: Network setup for the DNS response hijacking attack (downstream spoofing configuration)

[9] This allows the spoofed DNS response to be sent directly to the victim, bypassing uRPF and router filtering.

8.1.2. UPSTREAM SPOOFING

The “upstream spoofing” attack setup, as illustrated in Figure 10, is an alternative network configuration which can be used to send a spoofed DNS response to the victim client from the attack server. This spoofed response will only be delivered after the victim client’s *next retry attempt* when the victim repeats the failed DNS request. The attack server must be capable of spoofing its IP to match the DNS server and send traffic back to the target NAT. For this to work, typically two options exist for the placement of upstream servers:

1) Both attack server and target server share the same internal subnet.
2) Both attack server and target server are located on the internet.

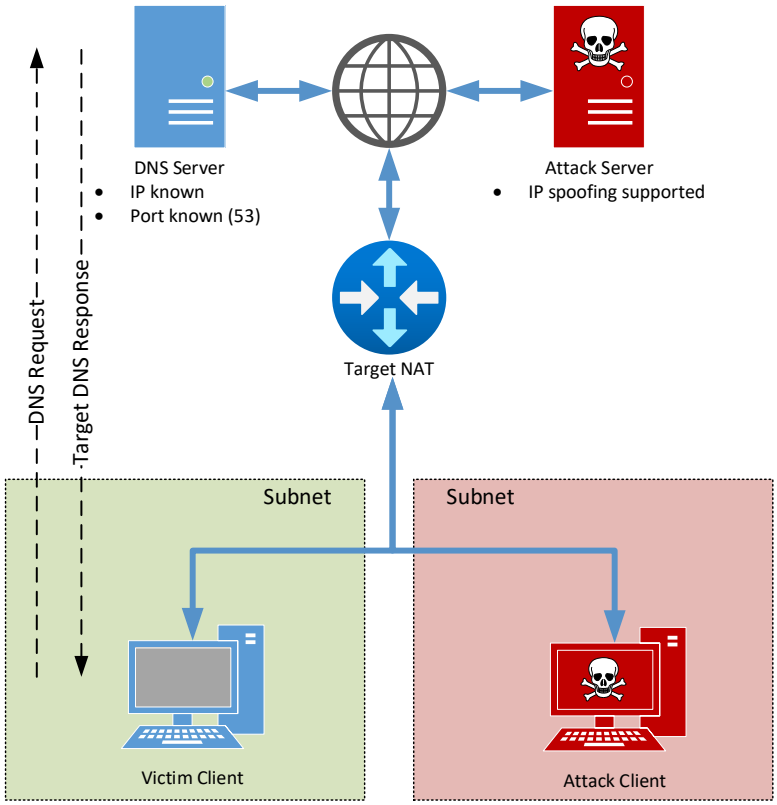

Figure 10: Network setup for the DNS response hijacking attack (upstream spoofing configuration)

8.2. VULNERABILITY DETAILS

8.2.1. REMOVING AN EXISTING ENTRY

Netfilter has a behavior that it will attempt to evict existing unassured entries at random from the NAT table when the NAT table is full so that new entries can be added. An attacker can exploit this behavior to remove existing unassured entries from the NAT table by completely filling up the NAT table with new entries (Figure 11). For example, the attack client can send many UDP packets to a large number of upstream endpoints. This will create many new entries which cause existing unassured entries to be randomly evicted.

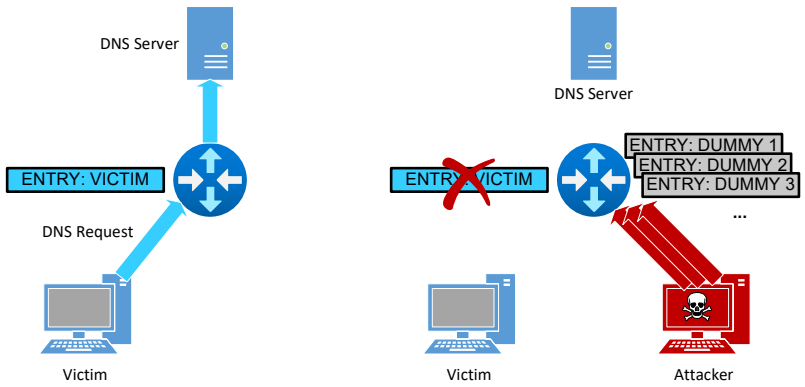

Figure 11: A victim entry is present in the NAT table from an outgoing DNS request (left). The victim's entry is removed by the attacker filling up the NAT table with new dummy entries (right).

8.2.2. REPLACING WITH A NEW ENTRY

Once an existing entry has been evicted, the attack client can create a new entry which is identical to the removed entry *except* that it originates from the attack client instead of the victim client. When a DNS response is returned to the NAT, the packet will be returned to the attack client instead of the victim client (Figure 12). This allows the attacker to hijack the DNS response destined for the victim client. The attacker can create these replacement entries by sending UDP packets to the DNS target server, optionally with a reduced TTL (time-to-live) value so that each outgoing packet is destroyed before reaching the server.

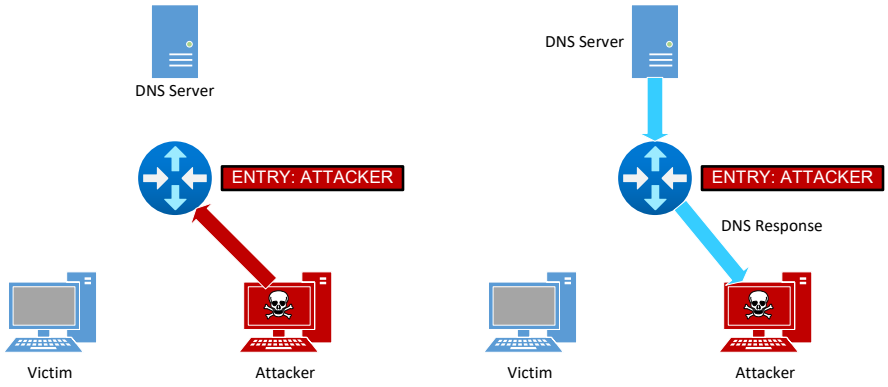

Figure 12: The attacker sends UDP packets to create new NAT table entries (left), causing the DNS server's response to be redirected to the attacker (right)

8.2.3. MALICIOUSLY ALTERING DNS RESPONSES

After obtaining a DNS response and its unique identifier, the attacker not only learns the URL which was requested by the victim client, but also gains the ability to craft their own malicious DNS response for this URL. This malicious response can be delivered to the victim client either by the attack client or attack server spoofing their IP to match that of the DNS server (Figure 13). If delivered from the attack server, the attacker will need to wait for the next DNS request retry from the victim.

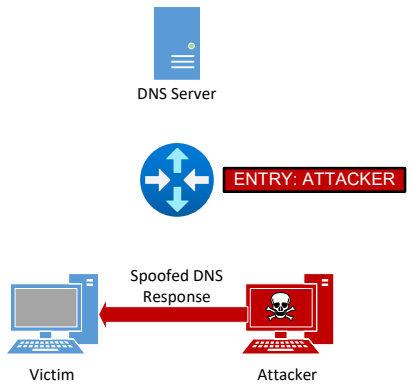

Figure 13: Spoofed replacement response delivered from the attacker to the victim

8.3. VENDOR/PRODUCT VARIATIONS

Some tested products exhibited the following variations:

- **Denial of service:** in nearly all tested devices not using Netfilter, with one or two exceptions, a UDP DNS hijacking attack was not possible. Most of these devices allowed UDP entries to fill up the NAT table until they timeout naturally. While this successfully prevents a hijacking attack, a denial of service (DoS) attack becomes possible instead, as described earlier in section 6.

9. VULNERABILITY 4: TCP/IP HIJACKING (DOWNSTREAM SPOOFING)

Possibly the most serious attack in this attack class is a TCP hijacking attack. A downstream attacker can use this vulnerability to break intended trust boundaries to hijack an existing TCP connection from a downstream victim client machine, traversing through the NAT, to an upstream target server. The attacker can then exchange data with the target server over the hijacked connection, impersonating the victim. After hijacking the connection, the attacker may also be able to perform a man-in-the-middle (MITM) attack to modify data sent from the target server to the victim client. The target server could be located either on a company's internal network or on the internet. Two variations of the TCP hijacking attack are possible, with this first downstream spoofing variation having the greatest impact because of how quickly the attack can be executed. The second upstream spoofing variation (section 10) allows for cross-subnet hijacking.

9.1. ATTACK SETUP

The attack client and victim client are both positioned downstream from the NAT, as illustrated in Figure 14. The attack client must be capable of spoofing its source IP to match that of the victim client. For this to work, *in most cases* the attack client and victim client machines will need to share the same internal subnet[10]. No upstream attack server is needed.

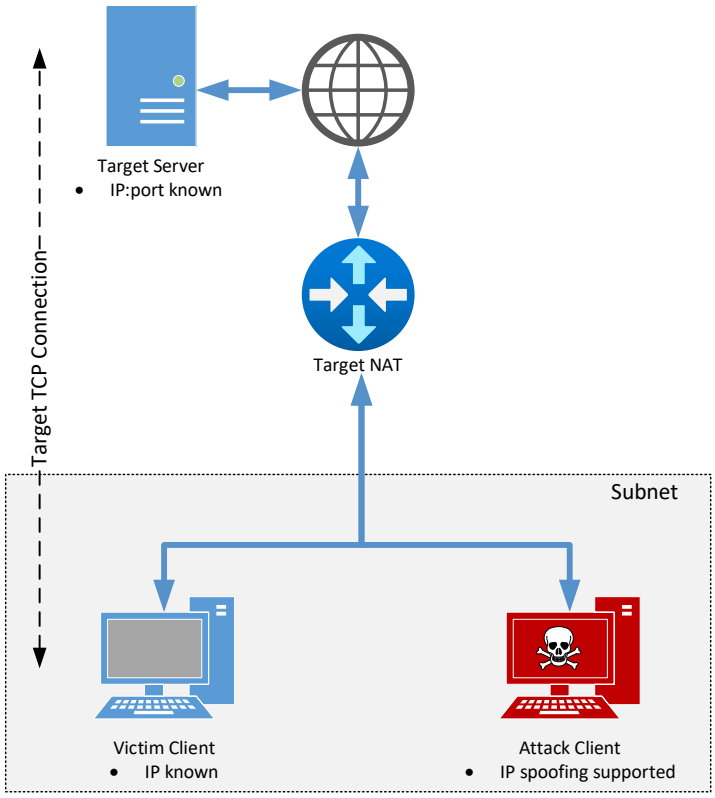

Figure 14: Network setup for the TCP hijacking attack (downstream spoofing)

[10] Typically, when uRPF is properly configured, a router will drop and ignore any spoofed packets originating from a different subnet. Certain products did allow hijacking attacks with no IP spoofing, however.

9.2. VULNERABILITY DETAILS

The TCP hijacking attacks work quite differently from the UDP hijacking attack but follow the same basic pattern: an existing entry from the victim client must first be removed from the NAT table, then replaced with a new entry directed towards the attack client.

9.2.1. REMOVING AN EXISTING ENTRY

Spoofing its source IP to match that of the victim client, the attack client can send three packets through the target NAT to completely and immediately remove an existing TCP connection from the NAT table. Using the previously-described RFC 793 Half-Open Connections behavior (section 5.4.5), the attack client can send a SYN packet with SEQ=x immediately followed by an RST packet with ACK=x+1. At this point, the entry will enter the CLOSE state, to be removed from the NAT table in 10 seconds by default (Figure 15).

The RFC 1337 TIME-WAIT Assassination behavior (section 5.4.6) can then be used to immediately remove the closed connection entry from the NAT table by the attack client sending another SYN packet. This packet will cause the entry to be immediately removed from the NAT table, bypassing the 10 second delay.

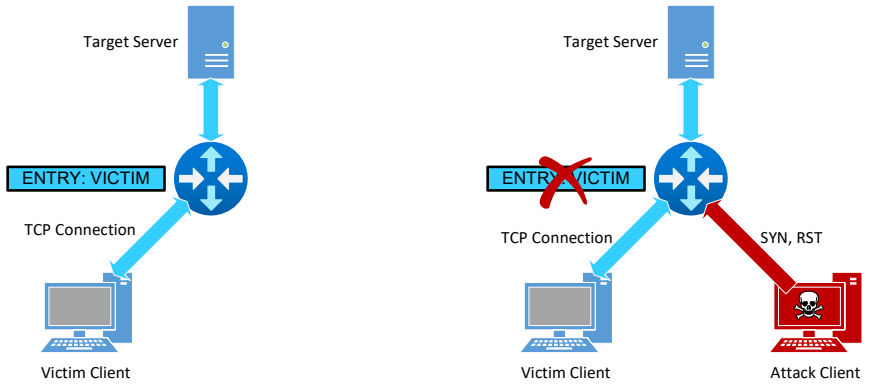

Figure 15: A victim is connected over TCP to the target server with an entry in the NAT table (left). The victim's entry is removed by the attacker sending a spoofed SYN, RST (or SYN, RST, SYN) packet sequence (right).

9.2.2. REPLACING WITH A NEW ENTRY

As described in Connection Creation (section 5.4.3), Conntrack has a "loose" TCP connection handling behavior, which is enabled by default, where an outgoing ACK can be sent over an untracked[11] connection to immediately add the connection to the NAT table, bypassing the need for a valid TCP handshake. To make use of this behavior, an ACK, containing any arbitrary sequence number, can be sent from the attack client to the target server. This will cause a new entry to be added to the NAT table for a connection between the attack client and target server, replacing the connection which was removed (Figure 16).

[11] An untracked connection refers to observed TCP packets which do not belong to any TCP connection currently being tracked by Conntrack, and which may not correspond to any existing NAT table entry.

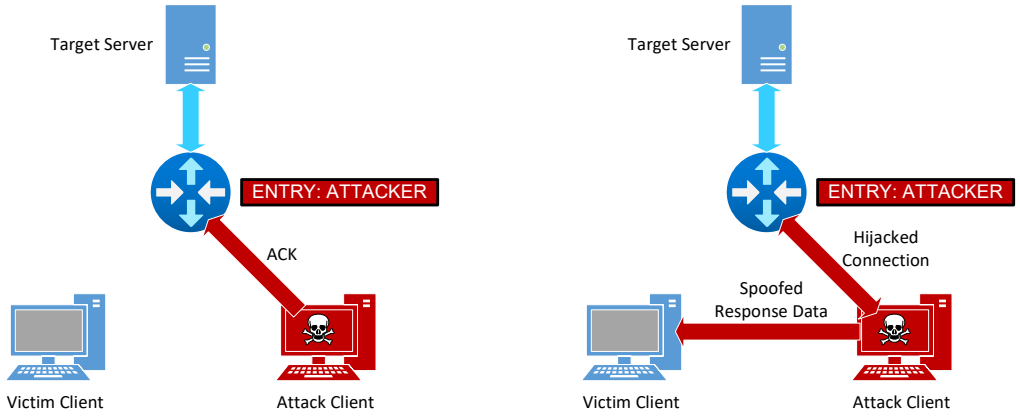

Figure 16: The attacker sends an ACK packet to create a new NAT table entry (left), hijacking the victim's TCP connection (right).

9.2.3. DETERMINING THE VICTIM PORT AND SEQUENCE NUMBERS

Generally, the ephemeral port used by the victim client for the TCP connection will not be known to the attacker. However, since this attack variation can be run extremely quickly, requiring only four packets, it is feasible to run the attack over an entire range of ports. When the correct port is reached, an ACK packet containing an invalid sequence number will be sent over the hijacked connection to the target server. The target server will respond with a challenge ACK packet containing the correct sequence number (SEQ) and acknowledge number (ACK). Upon receiving this packet, the attack client has learned the victim client's ephemeral port, as well as the SEQ and ACK of the TCP connection.

9.3. VENDOR/PRODUCT VARIATIONS

Some tested products exhibited the following variations:

- **Time Wait Assassination (TWA):** only Netfilter and Packet Filter based products exhibited a TWA primitive, allowing entries to be immediately removed with no idle time. Other products may have a similar primitive, but this was not identified.
- **Time wait delay:** different products had varying idle time requirements for entry removal from a closed state, typically ranging from 10 seconds minimum to around 5 minutes maximum. The TCP connection would have to remain idle for this period to be removed and hijacked.
- **Entry removal sequence:** different products required various packet sequences for entry removal. In some products, entries could be removed from a single spoofed RST packet with any sequence numbers. Other products required a specific packet sequence or a flood of spoofed RST packets.
- **No spoofing:** certain products allowed for entry removal with no source IP spoofing, either by an attacker filling up the NAT table, automatically after a longer idle time delay, or both. With these products, the attacker and victim could be positioned in different subnets.
- **Port detection step:** if a packet flood is needed for removal, the higher attack complexity makes a port detection step necessary for executing the attack in practical time.
- **Loose mode:** certain products had "loose mode" disabled, where entries can be added to the NAT table without a valid TCP handshake. In some products, connection hijacking was still possible using another TCP state, such as the SYN-SENT or CLOSED state instead of the ESTABLISHED state.

10. VULNERABILITY 5: TCP/IP HIJACKING (UPSTREAM SPOOFING)

A second variation of a TCP hijacking attack is applicable to the case where the attack client is not capable of spoofing its IP to match the victim client, or the victim client IP is unknown. This applies to scenarios where the attack client and victim client are positioned in different subnets.

A downstream attacker can use this vulnerability to break intended trust boundaries to hijack an existing TCP connection from a downstream victim client machine, traversing through the NAT, to an upstream target server, with the help of an upstream attack server. The attacker can then exchange data with the target server over the hijacked connection, impersonating the victim. After hijacking the connection, in certain specific cases, the attacker may also be able to perform a man-in-the-middle (MITM) style of attack to modify data sent from the target server to the victim client.

10.1. ATTACK SETUP

The "upstream spoofing" attack setup is an alternative network configuration, as illustrated in Figure 17, where the attack client and victim client can be placed *anywhere* downstream from the target NAT. The attack server must be capable of spoofing its IP to match the target server. Typically, two options exist for the placement of upstream servers:

1) Both attack server and target server share the same internal subnet.
2) Both attack server and target server are located on the internet.

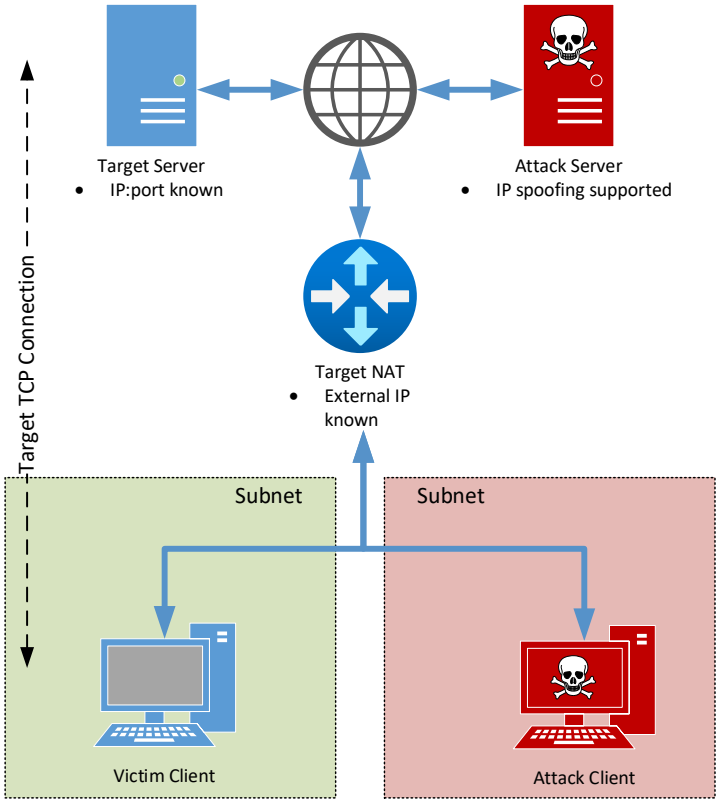

Figure 17: Network setup for the TCP hijacking attack (upstream spoofing)

10.2. VULNERABILITY DETAILS

10.2.1. EXPLOIT CHAIN: DETERMINING THE VICTIM PORT

In order to execute this attack in a reasonable time, the victim client's ephemeral port must be known to the attacker. This can be done, and is demonstrated in the attached PoC's, using Vulnerability 2: Victim IP and Port Information Disclosure (section 7). This exploit chain allows the external ephemeral port to be detected, and the attack to be completed in a reasonable time. Alternatively, the attack could be executed over the entire port range, but this would take a very long time.

10.2.2. REMOVING AN EXISTING ENTRY

An upstream attacker can remove a TCP connection from the NAT table using a method similar to a TCP RST flood attack[12], which is typically only used for DoS. In this case, however, an attacker can take advantage of Netfilter's windowing behavior to force-close a connection in the NAT table without precisely guessing its correct sequence number.

An RST packet containing an incorrect sequence number will be ignored by the victim client endpoint but can be considered valid by Netfilter, as long as the RST packet contains a sequence number within the correct window for the connection. A typical window size is around 64 kB, so an attacker can usually find a valid sequence number within this window after sending tens or hundreds of thousands of RST packets through the NAT. This number, although large, is small enough that it is feasible to execute in a matter of seconds. After the RST flood occurs, both victim client and target server endpoints will still consider the connection to be valid, while the NAT considers the connection to be closed and moves it to the CLOSE state (Figure 18).

After an entry enters the CLOSE state, Netfilter will retain the entry in its NAT table for an additional 10 seconds, by default, after the last packet has been received. After a 10 second idle period occurs, the entry will be completely removed from the NAT table.

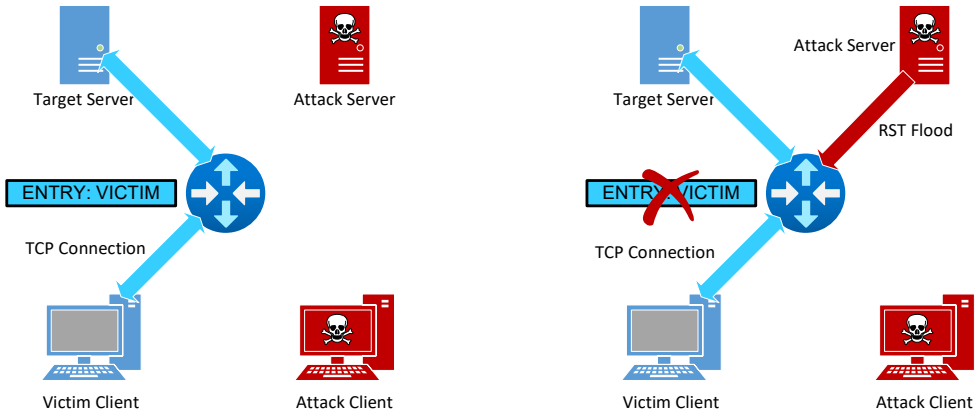

Figure 18: A victim is connected over TCP to the target server with an entry in the NAT table (left). The upstream attack server sends a spoofed RST flood and waits 10 seconds for the victim's entry to be removed (right).

[12] https://www.akamai.com/glossary/what-is-a-tcp-reset-flood-ddos-attack

10.2.3. REPLACING WITH A NEW ENTRY

In this attack variation, two methods can be used to replace the removed NAT table entry with a new entry:

1. The **downstream method** used to add a new entry for the Vulnerability 4: TCP/IP Hijacking (Downstream Spoofing) attack (see section 9.2.2), which makes use of Netfilter's "loose" behavior, can also be used in this attack, because it does not depend on downstream spoofing. This method allows the attacker to immediately obtain sequence numbers.

2. An **upstream method** of adding the NAT table entry is also implemented, which does not rely on Netfilter's "loose" behavior. In this alternative method, a spoofed TCP handshake is performed between the attack client and attack server, where the attack server spoofs its source IP to match that of the target server. After a valid TCP handshake takes place, a new NAT table entry will be created to send traffic between the attack client and target server (Figure 19).

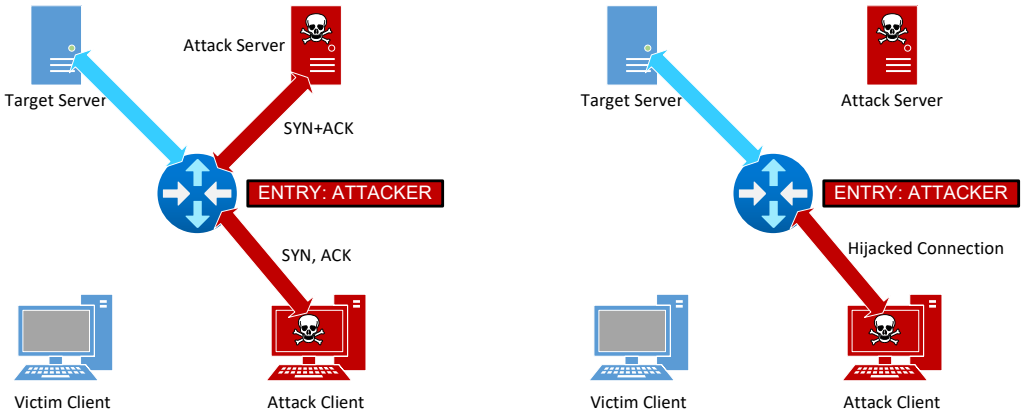

Figure 19: The attack client exchanges a spoofed TCP handshake with the attack server to create a new NAT table entry (left), hijacking the victim's TCP connection (right).

10.2.4. DETERMINING SEQUENCE NUMBERS

Once a NAT table entry has been created with the upstream method described above (section 10.2.3), there is still an issue that the connection uses a specific sequence number (SEQ) and acknowledge number (ACK) which are not known to the attacker. In fact, if the target server attempts to send a packet through the NAT with SEQ and ACK not contained within the current TCP connection window, that packet will be filtered and dropped by Netfilter.

It is possible, however, for an attacker to quickly find a valid SEQ and ACK window for the connection by making use of very large windows. During the spoofed TCP handshake, the connection window can be enlarged as much as possible, by both attack server and client endpoints specifying a window size of 2^{30} (1GiB), or more precisely 65535 with a window scaling factor of 2^{14} (i.e. $65535 * 2^{14}$). To enable traffic from the existing connection to traverse through the NAT, a search must be performed to find the correct SEQ/ACK window. It is possible to enumerate through 16 distinct windows, one of which will contain the correct SEQ and ACK for the connection.

This search takes place by sending an ACK packet from the attack client to the target server, followed by a spoofed ACK packet from the attack server, spoofing its source IP to match the target server, to the attack client. During each packet exchange, the SEQ and ACK are incremented to move to the next window as shown in Figure 20.

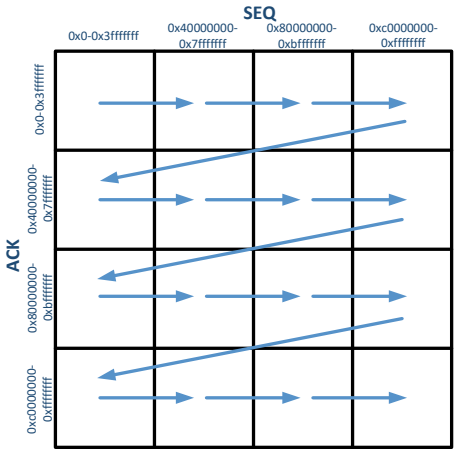

Figure 20: Search for correct SEQ/ACK window during the spoofed packet exchange

When the correct window is reached, the ACK packet sent from the attack client to the target server will reach the target server and cause it to respond with a challenge ACK packet containing the correct SEQ and ACK for the connection. Since these sequence numbers finally fall within the correct window, the challenge ACK packet will traverse through the NAT, reaching the attack client. At this point, the attack client has obtained the correct SEQ and ACK for the connection.

10.3. VENDOR/PRODUCT VARIATIONS

Some tested products exhibited the following variations:

- **Time wait delay:** different products had varying idle time requirements for entry removal from a closed state, typically ranging from 10 seconds minimum to around 5 minutes maximum.
- **Entry removal sequence:** different products required various packet sequences for entry removal. Almost all products required a flood of RST packets, sometimes combined with other packet sequences to execute the attack in reasonable time, such as SYN or ACK packets.
- **Detecting sequence numbers:** certain products implemented less strict packet filtering and did not require the 16-window search process illustrated in section 10.2.4.

11. RECOMMENDED MITIGATIONS FOR CUSTOMERS

These attacks should serve as a new incentive for businesses to prioritize the use of strong encryption such as transport layer security (TLS) in all their systems, even internal servers. Only when TLS is required is a system guaranteed to be secure against any *useful* hijacking[13]. The implementation and use of secure DNS, such as DNSSEC and DNS over HTTPS or DNS over TLS, must also be prioritized. The use of strong encryption will mitigate hijacking attacks, but will not prevent other attacks such as port disclosure and denial-of-service (DoS).

11.1. ENCRYPTION

- Use TLS (including HTTPS) everywhere, even on internal systems. This will completely prevent useful hijacking of connection data.
- Use DNS security such as DNSSEC to prevent spoofing of DNS responses. DNS encryption such as DNS over HTTPS or DNS over TLS is also recommended to prevent hijacking.

11.2. CONTAINERS AND VIRTUALIZATION

- Disable network access of untrusted containers and virtual machines when possible, or set them up in a private internal network.
- Run untrusted containers or virtual machines on their own isolated host machine.
- Avoid running containers as `root`.
- Drop the enabled-by-default `NET_RAW` permissions when running an untrusted container using `--cap-drop=NET_RAW`.

11.3. KUBERNETES

- Run untrusted Kubernetes pods on a different node than trusted pods, in both local and cloud-based deployments.
- Avoid running Kubernetes pods as `root`.
- Drop the enabled-by-default `NET_RAW` permissions using `securityContext.capabilities.drop`.

11.4. CLOUD

- Avoid placing untrusted and trusted workloads behind the same NAT gateway, internet gateway, or cloud NAT.
- Use dedicated IPs for serverless computing rather than shared IPs, for better network isolation from other tenants.

[13] When TLS is enabled, an attacker may still be able to hijack a connection, but no data will be disclosed to the attacker and no data can be injected.

11.5. ROUTERS AND FIREWALLS

- Enable IP source protection (IP Source Guard or similar) to prevent IP spoofing by client devices.
- Ensure untrusted users and devices are in a separate subnet/VLAN from trusted users and devices.
- Set up L3/L4 Access Control List (ACL) rules to specifically prevent untrusted devices from accessing any trusted hosts.
- Limit the maximum number of TCP/UDP connections per client to a reasonable limit, ideally under 10k or so.
- Disable any `nf_conntrack_tcp_loose` or `flags any` connection modes which allow a connection to be added to the NAT table without a valid handshake.
- Disable port preservation and endpoint-independent mapping port assignment behaviors if possible.
- Enable ISN randomization (`modulate state` or similar) when the option is available.

11.6. MONITORING AND DETECTION

Several common indicators of compromise (IoCs) include:

- **NAT table full:** A full, or nearly-full, NAT table.
- **Port range flood:** A flood of TCP or UDP packets from a downstream endpoint with a large range of source ports.
- **IP spoofing:** It may be possible to detect downstream IP spoofing, in some cases, by detecting the same IP address being used by two different physical addresses, or the same physical address sending packets from multiple IP addresses.
- **Sequence of packets:** Certain packet sequences from a downstream endpoint, such as `SYN`, `RST` or `SYN`, `RST`, `SYN`, sent sequentially with a consistent source and destination port.
- **TCP RST flood:** A flood of `RST` packets from an upstream or downstream endpoint where the sequence number is varied.
- **Invalid sequence numbers:** TCP packets containing sequence numbers far outside their current TCP window.
- **Reduced TTL:** Packets may be sent with a reduced time-to-live (TTL) to reach only as far as the NAT device before being destroyed. This might be used by an attacker to prevent a standard endpoint response or to try to evade detection.

12. CVES, PATCHES, AND ADVISORIES

- CVE-2026-56181: Microsoft Windows NAT (affecting Hyper-V in a downstream spoofing configuration).
- CVE-2026-56179: Microsoft Windows NAT (affecting Hyper-V in an upstream spoofing configuration). *Note that this mitigation is disabled by default and must be enabled via a registry key.*
- CVE-2026-63913: Linux Kernel Netfilter (fixing a code flaw and applying a mitigation for the downstream spoofing attack) applied in Linux kernel 7.1 and higher. This is *not a complete fix* but does increase attack complexity.
- FreeBSD *incidental patch* (unrelated to this research) from 2025 for Packet Filter which affects both downstream spoofing and upstream spoofing configurations, applied in FreeBSD 15.0.0 and higher. Code was pulled in from OpenBSD which has had this more secure behavior since 2016. Although

not a complete fix, it makes most attacks *much less practical*. Thanks to sashan (OpenBSD) for pointing out this code.

- Amazon eero: Protecting eero customers from NatJack

AWS Public Statement

AWS investigated and deployed mitigations for reported behavior related to connection state management in services that perform port-level network address translation, NAT gateway and Network Load Balancer (NLB).

This research describes a scenario in which an actor with control of an EC2 instance within a VPC could send crafted TCP reset packets to manipulate NAT flow state, potentially enabling session disruption or interception of TCP and UDP connections traversing the NAT. The described scenario requires the actor to already control an instance within the same VPC as the targeted connection.

AWS has deployed updates across both services that strengthen connection state validation, including enhanced verification of TCP reset packets before modifying flow state. These mitigations are effective across all AWS Regions. Customer applications must follow TCP and UDP best practices, by ensuring that their sockets are closed after the application's idle timeout expires. No further customer action is required to inherit protection. Customers with high connection volumes and long-lived connections should monitor port utilization and may need to allocate additional Elastic IP addresses (NAT gateway) or expand target/subnet capacity (NLB) to accommodate longer port hold times introduced by this hardening [1][2].

The research also describes an information-disclosure technique in which an actor can infer in-use external NAT ports by observing responses to crafted SYN and ACK floods. This technique requires both an instance within the same VPC as the port disclosure sought and a cooperating external server capable of IP spoofing. The disclosed port information is limited to external NAT port numbers in use and does not reveal associated client IP:port pairs. AWS mitigations prevent the session manipulation techniques that would otherwise leverage this information.

We appreciate Malcolm Stagg's (SODIUM-24, LLC) commitment to coordinated disclosure and constructive collaboration throughout this process, and we encourage continued engagement from the security research community.

[1] https://docs.aws.amazon.com/vpc/latest/userguide/nat-gateway-troubleshooting.html
[2] https://docs.aws.amazon.com/elasticloadbalancing/latest/network/load-balancer-troubleshooting.html

13. CONCLUSION

A huge number of physical and virtual network devices are vulnerable to the NatJack attack class. Whenever an untrusted user has the ability to send network traffic originating downstream from the device, trust boundaries may be broken, and trusted users sending traffic through the device are at risk of being a victim of hijacking, denial-of-service (DoS), or both. Most home users will not be seriously impacted, since only trusted users are typically connected to a home router. For businesses and other service providers, the potential impacts of these attacks are much more concerning.

The potential mitigations mentioned in this report should be reviewed and implemented in any affected products. Since many of the suggested attack mitigations have an inherent tradeoff between improving security versus causing a greater likelihood of DoS and breaking protocol compatibility, businesses may need to create and review threat models to identify their greatest risks. Businesses should carefully review their network planning, to ensure, as much as possible, that any untrusted workloads are physically isolated from trusted workloads and internal systems, as well as prioritizing the use of strong encryption in all systems.

NatJack attack monitoring, detection, and prevention is a critically important area of focus and research for companies producing routers, firewalls, VMMs, cloud infrastructure, and network monitoring devices. Future research in this area is pivotal for developing and incorporating more effective mitigation strategies and innovative NAT behaviors which are less vulnerable to attack.

