`GET /large.file HTTP/1.1`: Connection-Based TCP Amplification Attacks

Yepeng Pan
CISPA
yepeng.pan@cispa.de

Lars Richter
CISPA
lars.richter@cispa.de

Christian Rossow
CISPA
rossow@cispa.de

***Abstract*—Amplification Denial-of-Service (DoS) attacks steer high-volumetric traffic to a victim by sending small IP-spoofed requests to UDP-based services. An attacker cannot abuse TCP-based services in the same manner, as TCP is connection-based and requires the attacker to complete a handshake. Hence, previous works only showed that the *connection-less* part of TCP can be exploited for DoS, e.g., by abusing middleboxes or handshakes for stateless reflection attacks.**

This work studies *connection-based* TCP amplification attacks. We first propose a methodology to explore the fundamentals of connection-based TCP amplification attacks—hosts with easily predictable sequence number selection algorithms. This allows attackers to complete IP-spoofed TCP handshakes, opening up the possibility of sending IP-spoofed application-layer (e.g., HTTP) requests to trigger amplified traffic. Our identification revealed over 160k vulnerable HTTP servers in the IPv4 space, out of which 54k servers host "amplifying" ($\geq$ 1 kB large) resources. Using only ≤ 3 sequence number guesses, a single IP-spoofed HTTP request achieves an average amplification factor of 16.77 at an $\approx 80\%$ success rate. Furthermore, we show that an attacker can also spoof cumulative ACKs and additional requests to further increase the impact of the amplification attack.

1. Introduction

Volumetric Distributed Denial of Service (DDoS) attacks continue to threaten infrastructure connected to the Internet. In particular, a plethora of successful *amplification DDoS attacks* rendered core Internet services offline, including PayPal, Spotify, and eBay [1], [2], [3], [4]. In such incidents, attackers multiply their bandwidth by abusing the fact that responses from certain application-layer services are much larger than the corresponding requests. Typically, amplification attacks abuse UDP-based applications without validating the request origin, such as DNS or NTP servers [5], [6], [7], [8], [9].

Existing amplification DDoS attacks are *connectionless*. The aforementioned past incidents abused UDP-based services vulnerable to amplification [9]. Connectionless amplification attacks are simple to launch, as they do not require attackers to prepare any connections on the reflectors they aim to abuse. In fact, the attacker just sends a single IP-spoofed request to the amplifiers, which reflect large responses to the victim. Apart from UDP, connectionless TCP-based amplification attacks are also possible, either by abusing the reflective nature of TCP handshakes [10] or with the help of middleboxes that reply to unsolicited TCP segments carrying certain request triggers [11]. Efforts to find [12] and close [13], [14] amplification vulnerabilities have significantly reduced the threat landscape of such attacks.

It is a long-standing belief that amplification attacks cannot abuse TCP connections. Amplification attacks have the inherent requirement that attackers need to mimic the target by sending IP-spoofed traffic to amplifiers. For connection-based attacks, this requires an attacker to initialize state using the victim's IP address—*without seeing the reflected traffic*. To abuse TCP-based applications as reflectors, attackers need to complete an IP-spoofed TCP handshake. To do so, though, attackers have to guess the correct 32-bit Initial Sequence Number (ISN) that the server chooses as part of the handshake. Though recent advances reduce the complexity of spoofing TCP handshakes for some operating systems [15], spoofing the handshake is still far too costly ($\approx 2^{24}$ spoofed segments) for practical amplification attacks.

In this paper, we show that connection-based TCP amplification DDoS attacks *are*, in fact, practical. We observe that attackers can find and abuse servers that choose a highly predictable ISN during the TCP handshake. This way, an attacker can establish IP-spoofed TCP connections to the vulnerable server, imitating the target's IP address, just like in UDP-based amplification attacks. Once the spoofed connection is established, an attacker can issue IP-spoofed requests based on the target application (e.g., HTTP) to trigger amplified traffic to the victim. Our paper thereby picks up the general idea of Paxson [16] from 2001 and shows how attackers perform practical connection-based TCP amplification attacks in today's networks.

To explore the practicability of connection-based TCP amplification attacks, we take the following steps and contribute in several aspects: (1) We propose a methodology to explore the prerequisites of connection-based TCP amplification attacks, i.e., to identify vulnerable hosts with easily predictable ISNs. In more detail, we perform a network-wide scan and ISN collection to gather 12 ISNs from each HTTP server in the IPv4 space. Since hosts with a secure ISN selection algorithm would pick pseudo-random ISNs, we pair statistical features with unsupervised clustering to

2576-9103/25/$31.00 ©2025 IEEE
DOI 10.1109/ACSAC67867.2025.00096

identify non-random ISN patterns. We then use Markov models to highlight suspicious clusters. From these clusters, we manually identify six predictable ISN pattern types and capture them in so-called *ISN signatures*—a series of heuristics and thresholds manually derived to represent each pattern. When applying the signatures on the ISN sequences of all hosts, we find 160,424 HTTP servers on TCP port 80 alone with easily predictable ISN subject to amplification attacks. Finally, we use a crawler to search for resources hosted by the vulnerable servers and find 54k worthy targets with an HTTP resource that is larger than 1,000 bytes.

(2) We thoroughly evaluate the challenges of connection-based amplification attacks. Unlike UDP-based amplification, connection-based TCP amplification is sensitive to the RST or ICMP packets the victim sends after receiving the reflected traffic, as they could terminate the spoofed connection and stop the amplification attack. Thus, we discuss and evaluate the impact of connection-based TCP amplification attacks targeting both responsive and unresponsive victims. In particular, combining the six known predictable ISN patterns, we design specific attack strategies for vulnerable hosts of different patterns to send a minimum number of spoofed packets in quick succession so an attacker can maximize the amplification factor for both types of victims. Finally, unlike UDP-based amplification attacks, where the attacker simply sends a single request, we show that an attacker can use strategies like sending cumulative ACK packets and additional requests over the spoofed connection to further enhance the impact of connection-based TCP amplification attacks. Overall, we show that, with reasonable efforts (≈ 3 ISN guesses), an attacker can abuse vulnerable hosts in a connection-based TCP amplification attack with $\approx 80\%$ success rate. Using these hosts, an attacker can amplify attack traffic by 16.77x, with a noteworthy number of outliers (14.2% of hosts can be abused to achieve more than 30x amplification).

(3) We group identified vulnerable hosts based on the commonality of hosted resources. For large groups, we identify the vendor of the vulnerable device and report the vulnerability in disclosures (details in Appendix A.8).

2. Background

In this section, we provide background information on TCP, ISNs, TCP spoofing, and amplification attacks.

2.1. Transmission Control Protocol (TCP)

TCP is one of the most commonly used transport layer protocols. Importantly, TCP is connection-oriented.

TCP uses the three-way handshake to establish a connection. Upon receiving the SYN request from a client with a client-chosen Initial Sequence Number (ISN) x, the TCP stack will generate a SYN-ACK packet containing a server-chosen ISN y and an ack number $x + 1$ to acknowledge the client SYN request (see Appendix, Figure 6). Next, the client is supposed to send an ACK packet with sequence number $x + 1$ and ack number $y + 1$ to acknowledge the server-chosen ISN. Upon receiving the correct ACK packet from the client, the server considers the TCP connection to be established, allowing the client to transmit/request data.

In TCP, the congestion window is the sender-side limit on the amount of data the sender can inject into the network before receiving an acknowledgment [17]. Nowadays, there are many congestion control algorithms, including Reno [18], Vegas [19], and CUBIC [20]. We use the congestion control algorithm introduced in RFC 5681 as an example. The initial phase of the TCP congestion control is the slow start. In this phase, the server first sends a small number of segments according to the congestion window. For each received ACK that cumulatively acknowledges new data, the server can increase the congestion window by at most sender maximum segment size (SMSS) bytes, which can effectively extend the window exponentially per round-trip-time (RTT; see Figure 6). Once the congestion window reaches the slow start threshold, the server switches to the congestion avoidance, where the congestion window grows linearly.

As a connection-oriented protocol, TCP guarantees reliable delivery of data. Thus, if payloads sent by the server are not acknowledged by the client, the server will retransmit the data after a timeout.

2.2. Weak ISN Selection Algorithm

The TCP handshake provides weak protection against off-path spoofing attacks, as an attacker cannot establish the connection without knowing the server-chosen ISN. However, such an assumption is only valid if the attacker cannot predict or guess the server ISN. Many major operating systems (OSes) were once subject to predictable ISN selections, e.g., Windows [21], BSD [22], Linux [23]. In particular, one well-known predictable ISN pattern once used by TCP stacks is the "64k" rule [24], [25], where a global ISN counter is incremented by multiples of 64,000. Today, major OSes have updated the ISN generation algorithms, and the RFC suggests better ISN selection algorithms [26]. However, [27] provides anecdotal evidence that predictable ISN selection algorithms still exist.

2.3. Amplification DoS Attacks

In an amplification attack, an attacker spoofs the victim's IP and sends small requests to remote servers. Upon receiving the spoofed request, remote servers send amplified (reflected) traffic to the spoofed victim IP. The ratio between the reflected traffic size and the attacker's request traffic size is the amplification factor that an attacker can abuse.

UDP-based protocols, such as DNS [6], [28], are commonly used in such amplification attacks. Since UDP is connectionless, attackers can easily spoof requests from victim IPs and trigger the server response. In contrast, amplification attacks using TCP-based protocols are harder. Because the TCP handshake requires establishing a connection by sending an ACK packet to acknowledge the server-chosen ISN, an off-path attacker cannot easily spoof a TCP connection. Consequently, existing research focusing on TCP-based

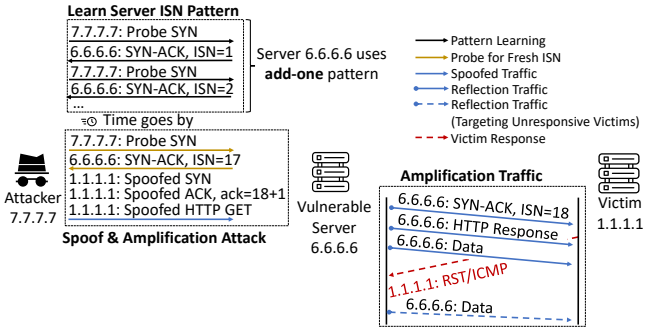

Figure 1: Sketch of a TCP-based amplification attack.

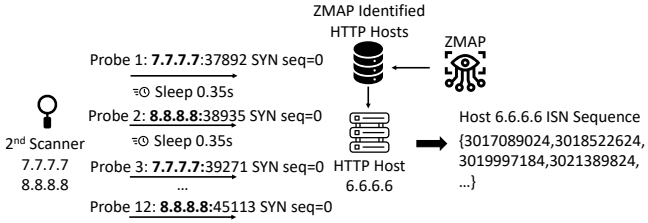

Figure 2: ISN Collection

amplification attacks only used the connectionless part of the TCP protocol [10] or relied on the behavior of certain infrastructures, e.g., middleboxes [11] or CDN [29], [30].

However, weak ISN selection algorithms used by servers make it possible to perform amplification attacks using an *established spoofed* TCP connection. To achieve this, we require that the ISNs used by servers be highly predictable with few guesses. In this work, we systematically examine vulnerable hosts in the IPv4 network and discuss the threat of connection-based TCP amplification attacks.

3. Threat Model and Victim Selection

The threat model of connection-based TCP amplification attacks is similar to its UDP counterpart: an attacker that can send IP-spoofed traffic. As reported by CAIDA [31], 28.2% of IPv4 ASes and 25.5% of IPv6 ASes supported IP spoofing fully or partially in October 2024. *In addition*, we require that the attacker can establish a non-spoofed TCP connection with a vulnerable server (the reflector). By observing the ISNs received in non-spoofed TCP handshakes, the attacker aims to predict which ISNs the vulnerable server will pick for subsequent connections of *any* source IP address. The attacker then uses this knowledge to establish an IP-spoofed TCP connection, impersonating the attack victim. Using such a connection, the attacker can request large resources to force reflectors to send TCP payloads to the victim host.

Figure 1 illustrates our threat model with an example. The attacker probes a few server ISNs with its genuine IP (7.7.7.7), learning a particular ISN pattern; in this case, an easy-to-guess ISN step size of 1. During the spoofing attack, the attacker first uses its genuine IP to probe the server once again and learns the latest server ISN, 17. Quickly thereafter, the attacker attempts to spoof a new TCP connection using the victim IP (1.1.1.1) and the predicted ISN, 18, based on the learned ISN pattern. If successful, the subsequent application-layer request (here: an HTTP GET request) will cause the amplifier to transmit data to the spoofed target.

When the attack traffic reaches the victim, we distinguish between two behaviors that impact the attack effectiveness. *Responsive victims* send a TCP RST or ICMP error messages to signal the amplifier to terminate the connection and hence stop sending traffic eventually. *Unresponsive victims* (e.g., routed but otherwise-silent IP addresses) do not reply to attack traffic or otherwise terminate the spoofed connection. We will distinguish between these two behaviors when assessing the amplification impact (e.g., Section 4.7). Having said this, in most practical settings, attackers can target unresponsive victims, especially when DoSing networks (which contain plenty of unresponsive IP addresses) or firewalled hosts (for which the firewall drops unrelated SYN/ACK segments). In fact, we confirmed in a small-scale scan that out of over 1M IPv4 addresses sampled from the announced BGP prefixes, only 12.7% respond to SYN/ACK segments that do not belong to any connection; hence, 87.3% are unresponsive.

4. Methodology

We follow a two-fold approach to explore the risk of connection-based TCP amplification attacks. First, we aim to identify servers with predictable ISNs, which is fundamental for connection-based TCP amplification. We use a customized scanner to collect ISNs from real servers in the network (Section 4.1). We next use unsupervised clustering to differentiate host clusters with potential "shared predictable ISN patterns" from the others with "secure random ISN patterns" (Section 4.2). Then, we employ Markov models to further filter suspicious clusters with seemingly predictable ISN patterns. (Section 4.3). Finally, we manually examined filtered clusters to summarize ISN patterns and develop "*ISN signatures*" (i.e., a series of heuristics summarized to capture the feature of certain predictable ISN patterns) for each identified pattern and use them to tag hosts with the corresponding predictable pattern type (Section 4.4).

With the knowledge of predictable ISN patterns and vulnerable hosts, we evaluate the feasibility and impact of amplification attacks using these servers. First, to learn the amplification potential of vulnerable hosts, we search for large resources attackers can request from them (Section 4.5). Next, we design a verification methodology to simulate the amplification attack from our own IPs and confirm the availability of identified vulnerable servers (Section 4.6). Finally, we show the methodology to estimate the amplification factor an attacker can achieve (Section 4.7). The ethics discussion in Appendix A.9 details steps we have taken to mitigate the harm of said experiments.

4.1. ISN Probing

To identify vulnerable TCP-based reflectors, we use the two-fold scan approach shown in Figure 2 to collect per-server ISN sequences of HTTP servers. First, targeting the whole IPv4 network, we use Zmap [32] to perform a SYN scan over port 80. We then send 12 SYN packets to each of the identified HTTP servers to collect a sequence of ISNs per server. For each SYN probe, we use the same sequence number but select different source ports and alternate between two source IP addresses. We do so to ensure that the vulnerable server's ISN selection algorithm allows us to predict ISNs across different source addresses—as an attacker needs to first probe the server ISN using its own IP and then predict the ISN for another (the target's) IP.

We model an attacker that waits for the selected server ISN (as obtained from the response to the probe SYN packet) and only then decides the predicted next ISN. Our scanner sends each SYN packet after a fixed delay to align with the attacker's wait time. This can maximize the availability of the learned ISN pattern through scanning and help with the verification. However, it is not helpful to infinitely extend the delay between SYN probes, as this can induce more noise in both verification and scan. As a sweet spot, we first measure the RTT between our host and remote servers identified in the Zmap scan and choose the delay time larger than the RTT of more than 98% of hosts—0.35s. This way, we can minimize noise caused by the delay between probes and maximize the chance that attackers obtain ISNs on time.

Note that our scan does not open an excessive number of unsolicited half-open TCP connections. The half-open connections are closed immediately by a subsequent RST packet when we receive the server's SYN-ACK packet. Because of the delay between probes, we only have one half-open connection open on remote servers in most cases.

4.2. ISN Sequence Clustering

With the collected ISN sequences per host, we now use clustering to identify potentially vulnerable ISN patterns. We base the clustering on the assumption that "secure" ISN generators should include a pseudorandom function over the connection's address 4-tuple (source IP, destination IP, source port, destination port) and a monotonically increasing timer [26]. Because of this pseudorandom function, for SYN requests from different address tuples, a secure host should "randomly" choose ISNs for different clients. In contrast, hosts with predictable ISNs increase a *global* ISN counter that is *shared* across address tuples. Using clustering, we aim to find and group hosts of the latter category, i.e., which do *not* choose ISNs at random. Such potentially predictable ISN sequences will stand out from randomly distributed ISNs. To this end, given the ISN sequence $\{ISN_1, ISN_2, ..., ISN_{11}, ISN_{12}\}$ of a host, we extract the following statistical features.

Wraparounds: A low number of ISN wraparounds indicates that a server uses a monotonically increasing *global* (i.e., shared among all TCP connections) ISN counter. [1] For a secure server, since the ISN is based on a hash value over the address tuple, it is likely to have more wraparounds, as we change the address tuple between two probes. Thus, we count the number of wraparounds (*NWP*) in the raw ISN sequence. We consider each instance of $ISN_{i+1} < ISN_i$ in the ISN sequence as a wraparound.

Fixed Stepsize (GCD): For each host, we calculate the ISN difference sequence $\{\Delta_1, \Delta_2, \cdots, \Delta_{11}\}$, where $\Delta_i = (ISN_{i+1} - ISN_i)\%2^{32}$, and use the greatest common divisor over all ISN differences, i.e., $GCD = gcd(\Delta_1, \Delta_2, \cdots, \Delta_{11})$ in clustering. This aims to identify hosts that use a fixed step size to increase the global ISN counter. In practice, a server may update the global ISN counter with the fixed step size multiple times because of timer updates and noise packets. The *GCD* effectively abstracts from such noise and yields the step size. In addition, servers sharing the same ISN update step size may suggest a shared OS, which can help identify vulnerable models or OSes. The use of *GCD* is inspired by the famous "64k" rule once used by operating systems (see Section 2.2).

Variable Stepsize: Apart from fixed step size, a vulnerable host may use a step size that is also updated periodically. To capture such hosts, we use the following statistics. First, we calculate the Pearson correlation coefficient (*PCC*) of the ISN difference sequence (inspired by [34], [35]), which reflects the linear relationship of the difference sequence. Next, we calculate the difference between neighboring elements in the ISN difference sequence and scale the differences with the *GCD*, $\{\frac{\Delta_2-\Delta_1}{GCD}, \frac{\Delta_3-\Delta_2}{GCD}, ..., \frac{\Delta_{11}-\Delta_{10}}{GCD}\}$. Then, we use the most common difference of difference, *DoD*, and its frequency, *Freq*, in the clustering. Similar to the fixed step size, frequently repeating *DoD*s are also required to predict the server ISN in a few guesses. In addition, by calculating the difference of the ISN difference sequence, we can effectively yield out hosts using a secure (random) ISN selection algorithm. As the ISN values are generated based on a hash function over the address 4-tuple, the difference of ISN differences should be rather random and can be easily distinguished from others.

Repetition Markers: Finally, we include statistics to capture servers that use a (or a few) static ISN(s). Such a host may only select ISNs based on the client SYN request sequence number. In our clustering, we include a flag *RPT* to label if there are repeated raw ISNs in the ISN sequence and compute the number of distinct ISNs *NDST*.

Overall, these features are sufficiently generic to capture hosts with similar statistics, allowing us to single out potentially vulnerable host clusters from the large mass of random ISN generators. We thus represent hosts by their

1. For a particular address tuple, RFC 9293 suggested to use a monotonically increased ISN updated every 4 microseconds, which can ensure the uniqueness of chosen ISNs within the Maximum Segment Lifetime (MSL). This can reduce the probability of sequence number overlap within the MSL, which can cause confusion of duplicate/old segments. With today's bandwidths, the ISN space cycle should be more regular, such as every 274 seconds in Linux [33]. However, given the relatively low frequency of such ISN restarts, they should not significantly affect our short ($< 5s$) probe period—and if so, only once.

feature vectors and then use DBSCAN for the clustering. We chose DBSCAN as it is robust to outliers and can help us yield out hosts using secure ISN generation algorithms as noise [36].

4.3. Cluster Filtering

Despite DBSCAN's capability to remove outliers, it may result in a large number of clusters—too large to further inspect manually. We thus use Markov models to identify truly "suspicious" clusters. Markov models are used to describe and determine the likelihood of a transition between states in a given system. For each host of a cluster, we train and evaluate a Markov model on the host's raw ISN sequence, ISN difference sequence, and ISN difference of difference sequence. If any of the three sequences are predictable, the underlying ISN generation algorithm of the host is likely not random. We consider every individual value of the sequence a single state. Using the raw ISN sequence as an example, the likelihood of a transition from state x to y is determined by how often y directly follows x in the first 11 ISNs of the sequence. We then use the most likely state following the 11th ISN to predict the 12th ISN. If this prediction succeeds for *any* host within a cluster, we consider the cluster as suspicious and proceed with manual examination.

4.4. Tagging Hosts with ISN Signatures

The Markov model highlights potentially vulnerable clusters, from which we manually summarize the ISN patterns. However, clustering may fail to group all hosts of the same ISN pattern type. For example, a widespread pattern is that ISNs are updated based on ISN differences sharing a common *GCD*. Many hosts of this pattern use a *GCD* of 40,960, which our clustering can identify. However, other hosts of this pattern using irregular *GCDs*, e.g., 10 or 32, may not be clustered. That is, though all such hosts share a similar pattern, they fall into different clusters and may even form singletons in the case of host-specific *GCD* statistics.

To generalize the clusters, we manually summarize patterns observed from clusters and map each ISN pattern to an "*ISN signature*". The ISN signature is a series of heuristics and thresholds to capture likely exploitable hosts subject to a particular ISN pattern. For example, one easily predictable ISN pattern is that hosts use a static ISN for all our probes. In this case, we summarize the *static* ISN pattern in an ISN signature that tags a host if it only has one distinct ISN during our scan. To tackle the *GCD* cases mentioned above, we define an ISN signature that tags a host if it only uses a few distinct ISN differences sharing a *GCD* during our scan. The ISN signature ignores the actual value of *GCD* and would tag all hosts subject to the pattern. At the same time, the threshold on the number of distinct ISN differences excludes hosts with a similar pattern but are hard to predict (see example in Appendix A.3). We perform a best effort attempt to define the threshold and heuristics to tag more potentially exploitable hosts. For each tagged host, we pick a set of 3 consecutive ISNs from the 12 collected ISNs and

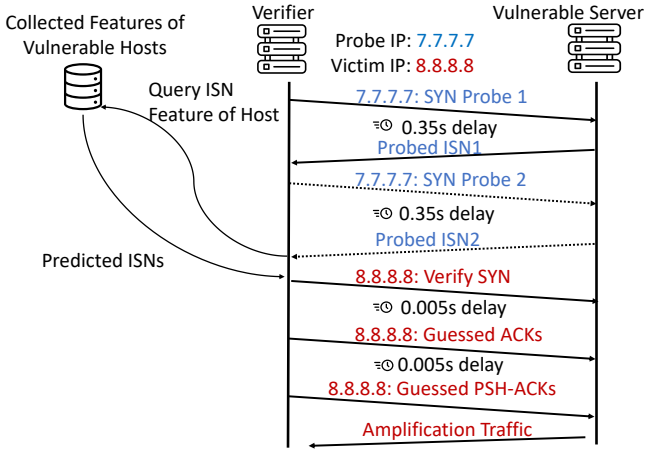

Figure 3: Verification Sketch

use the verification strategy designed for each pattern (more details in Section 5.6.1) to test if we could predict the third ISN with the first two. We track the approximate prediction success rate for tagged hosts to avoid a poor signature choice that would tag too many hosts with hard to predict ISNs.

Once the ISN signatures are developed, we use them to scan the full ISN dataset and tag hosts subject to corresponding ISN patterns.

4.5. Resource Identification

Once an attacker finds reflectors, they want to maximize the impact of the amplification attacks. To this end, an attacker needs to request large resources on the vulnerable host. As we target HTTP, we crawl for HTTP resources. We implement an HTTP crawler using Python's `requests` library [37]. The crawler starts from the root path of the host and recursively explores *src* and *href* links on each page. For performance and ethical considerations, we abort the crawler for each host after 120 seconds. Our crawler uses the HTTP headers in Listing 1. We use the user-agent *'-'* to minimize the HTTP header size during verification. Our crawler follows redirections in principle, but always replaces the location with the server's IP. We also include the Accept-Encoding header. HTTP servers may transmit the encoded content (e.g., using gzip) to the client, which can reduce the traffic sent to the victim in an amplification attack. Thus, we set the Accept-Encoding field to `identity`. If HTTP servers respect the option, we can increase the size of amplification traffic by transmitting unencoded content.

With the resource information, we combine the dataset of vulnerable hosts collected in Section 4.4 with the dataset of host resource size and only keep these vulnerable hosts with "sufficiently" large resources.

4.6. End-to-End Attack Verification

In this section, we introduce a methodology to verify whether hosts that show predictable ISN patterns can indeed be abused in an amplification attack.

4.6.1. Verification Setup. We use a verification setup as shown in Figure 3, which assumes that we learned the server's ISN pattern before (see Section 4.4). The verification uses a host with two public IPs to request a payload via an IP-spoofed connection. We assume that one public IP is controlled by the attacker (the probe IP) and the other is the victim IP to spoof. To ensure that the vulnerable hosts to be verified do not use predictable ISN patterns for specific source IPs, which is inapplicable for real attackers, the two IPs used in the verification do not collide with those for the ISN collection.

We first use the probe IP to send a SYN segment to learn the current server ISNs; optionally, if the ISN pattern requires, we send a second SYN segment to also learn the server's current ISN difference. Next, based on the identified ISN pattern and the collected pattern features, we predict one to three possible next ISN numbers. Following one IP-spoofed SYN segment, for each predicted ISN number, we build and send a sequence of IP-spoofed ACK and PSH-ACK segments; the SYN and ACK segments complete the handshake, while the PSH-ACK segment contains the HTTP GET request learned in Section 4.5. We capture all traffic sent and received from the victim IP for further processing and amplification factor estimation.

Attention has to be given to how much delay we add between the segments. We reuse the same 0.35s delay we used during scanning for the initial SYN segments to improve the usability of the learned ISN pattern. In contrast, we send the subsequent spoofed packets in quick succession (5 ms delay) to assess the maximum amplification against responsive victims. Recall that responsive victims terminate the spoofed connection with ICMP or RST packets after receiving the reflected traffic. Sending all spoofed packets in quick succession shortens the time period between starting the handshake and sending responses in the reflected traffic. Depending on the RTT between the victim and the reflector, the reflector can already dispatch its response before the RST packets can be delivered to it. For example, let the reflector and victim have an RTT of 50 ms; as we only have a 10 ms delay between the spoofed SYN and spoofed request, the victim will still receive the attack payload if the reflector finishes sending the requested payload within 40 ms. The chosen 5 ms delay is meant as a compromise between maximizing the payload that can be sent vs. avoiding packet reordering that might happen without delay.

To gauge the success of the attack, we examine the captured traffic to check if the spoofed connections were successfully established using the guessed ISNs. That is, we validate if the spoofed IP address observes TCP segments without SYN or RST flags that acknowledge the spoofed GET request *and* carry a payload.

4.6.2. Attack Extension. Until now, we have been using connection-based TCP reflectors in the same manner as UDP reflectors. That is, we complete the handshake and send a single HTTP request. However, an attacker could aim to further enhance the amplification factor using two means.

First, an attacker can increase the amplification factor by acknowledging payloads and thereby trigger more segments (reflected traffic). When congestion control (cf. Section 2) is implemented, the server holds back TCP payloads beyond the congestion window until prior segments are acknowledged. The attacker can first learn the hosts' initial congestion windows. After sending the guessed ACKs and PSH-ACK as per Figure 3, the attacker then sends one additional ACK packet for each predicted ISN *plus* the congestion window size. The cumulative ACK covers all segments within a congestion window, which we denote as a "*chunk*".

Second, the attacker can request additional resources via HTTP if the server does not terminate the connection after the first resource is delivered (more details in Section 5.7.1).

4.7. Amplification Factor Estimation

Finally, we analyze the amplification factor an attacker can achieve based on the captured traffic and the type of the targeted victim. To estimate the amplification factor, we need to confirm both the total size of the traffic sent by the attacker to trigger amplified traffic (the cost of the attack) and the total size of reflected traffic. Then the amplification factor is the size of the reflected traffic divided by the attack cost. The cost of the amplification attack is the sum of the size of SYN and RST segments sent from the attacker-controlled host and the spoofed SYN, ACK, and PSH-ACK segments sent from the victim host. These attack preliminaries differ per ISN pattern type and host-specific HTTP request sizes (details in Table 2). For the size of reflected traffic, we discuss the case of responsive and unresponsive victims, respectively.

Unresponsive Victim: For an unresponsive victim, the spoofed TCP connection is not terminated by the victim's RST or ICMP packets, and all packets sent by the reflector can be delivered. Thus, when calculating the size of reflected traffic for unresponsive victims, we simply accumulate the size of all packets received from the vulnerable server. Ideally, an attacker can choose unresponsive targets, e.g., those protected by stateful firewalls. Stateful firewalls can drop the reflected traffic as unsolicited packets, but still expose the target network to volumetric DDoS attacks.

Responsive Victim: When targeting a *responsive* victim, the attack target terminates the spoofed TCP connection, which stops the amplification traffic in principle. However, as discussed in Section 4.6.1, the attack is still effective against responsive victims, although less powerful. Recall that the reflector will transmit data until the termination packets (RST, ICMP errors) from the victim arrive. The connection termination time is thus related to the RTT between the victim and the vulnerable server. To estimate the amplification factor, we choose the median RTT m between our scan host and the servers as a representative RTT—effectively assuming scanned hosts are reflectors and our scanner is the target. Given the received SYN-ACK timestamp t from the vulnerable server, we only consider packets received before $t + m$ as attack traffic.

5. Evaluation

As stated in the methodology, we first perform an IPv4 scan to collect ISNs generated by real servers and then use ML-based clustering, filtering, and manual analysis to identify predictable ISN patterns. Next, we design the ISN signatures to filter vulnerable hosts following different patterns and verify the amplification attack against them according to the patterns. In the following, we discuss our experiment step by step. [2]

5.1. ISN Collection

We explore vulnerable HTTP servers in IPv4. The SYN scan with Zmap identified 55,485,147 hosts listening on port 80. For each host, we send 12 SYN packets to collect 12 server ISNs, using a static delay d between two probes. As discussed in Section 4.1, we first calculate the RTT from our host to the remote HTTP servers. 98.5% of the hosts have an RTT of $< 350ms$, so we set $d = 350ms$. This way, we collect 12 ISNs from 53,002,242 (95.5%) hosts; the remaining hosts are either non-responsive due to IP address churn (between our initial scan and the ISN collection) or face packet loss on our scanner's end.

5.2. ISN Pattern Clustering

Processing and clustering the whole dataset is challenging due to its complexity. We thus "only" use 10 million ISN sequences (18.9% of the entire dataset) in the clustering. Note that this does not significantly affect the overall number of vulnerable hosts. We use clustering only to learn potential patterns, but later use the ISN signatures to identify vulnerable hosts within the *entire* ISN dataset.

In total, the clustering identifies 36,500 clusters, spanning 138,631 hosts assigned to non-singleton clusters. Out of the large number of clusters, 36,239 clusters (covering approximately 60% of the clustered hosts) are small clusters with fewer than 25 hosts. In particular, 34,411 small clusters only have two or three hosts. A closer check shows that hosts in these very small clusters usually have ISN sequences with 12 distinct raw ISN numbers and 11 distinct ISN differences. Instead of vulnerable hosts with predictable ISN patterns, many of them are more likely clustered hosts that use secure ISN selection algorithms but accidentally share the same statistics. In the following, we focus on the 261 clusters with ≥ 25 hosts and use Markov models to filter suspicious clusters for further manual examination. In the end, if there are singular vulnerable hosts or vulnerable hosts in small clusters with known predictable ISN patterns, we can still identify them by tagging the entire raw ISN dataset.

5.3. Cluster Filtering & Pattern Summarizing

Using Markov models, the number of clusters to be examined is further reduced to 54. Through manual examination, we finally identified six predictable ISN patterns (and three inapplicable patterns as described in Appendix A.3).

Type S1 – Static ISN: These hosts use a static ISN number for all our 12 SYN probes. The static ISN is shared across IPs since our scan uses two source IPs.

Type SCx – Continuous static ISN: Hosts of type SCx also use static ISN numbers. However, they only continuously choose the same ISN number for x times and start repeating another different ISN number x times. Depending on the most common repeating duration x of the ISN sequences, we consider them as different types SCx.

Type SA – Alternating static ISN: S1 and SCx hosts continuously select the same static ISN number. Instead, SA hosts alternate the ISN from a static set of ISNs.

Type D1 – Global ISN counter with fluctuating step size: Hosts of type D1 use a global ISN counter shared across address tuples. Though the ISN differences extracted from the ISN sequence of a host are not static, the differences share the same GCD, e.g., 40,960. When normalized with the GCD, the ISN differences used by the server are scoped to a few choices. The variance in the normalized ISN difference can be caused by timing or background traffic noise.

Type D2 – Add-one ISN: Similar to hosts of type D1, D2 type hosts also use a global ISN counter shared across address tuples. However, D2 hosts increase the ISN by zero or one upon receiving a new SYN request. In later verification, it appears that only when the previous SYN requests are reset by a subsequent RST packet does the D2 predictable ISN pattern occur.

Type D3 – Global ISN counter with monotonically increasing step size: Hosts of type D3 also use a global ISN counter shared across address tuples. However, unlike D1 hosts, the ISN difference sequence of a D3 host also follows a monotonic trend, and the ISN difference is updated by a few predictable step sizes.

5.4. Tagging

For recognized predictable ISN patterns, we manually craft ISN signatures to scan the whole raw ISN dataset and filter vulnerable hosts for each predictable ISN pattern. For space reasons, we provide the detailed ISN signature design for each pattern in Appendix A.4. Table 1 summarizes the tagging result over the full ISN dataset. In total, 160,424 hosts (0.3%) are tagged as vulnerable. Out of all tagged hosts, D1 (52.4%), D2 (8.1%), and D3 (16.1%) are common predictable pattern types. That is, the large majority (76.6%) of vulnerable hosts use simple ISN updates that can be captured by differential analysis; another fifth of the hosts carry a static ISN pattern. The remaining predictable patterns sum up to a share of only $< 2\%$.

We cannot rely on a ground truth to measure how complete our methodology is, i.e., if we capture *all* hosts with predictable patterns. Having said this, to approximate the completeness of our signatures, we confirmed that all suspicious clusters, as highlighted by the Markov model, were correctly tagged by our signatures.

2. Project Repo: https://github.com/acsac2025-tcp-amp/acsac2025_tcp_amp. The ISN dataset is available upon request.

Pattern Type	Fraction	# of Hosts
S1	21.5%	34,437
SCx	0.9%	1,512
Sa	1.0%	1,666
D1	52.4%	84,040
D2	8.1%	13,015
D3	16.1%	25,754
Overall	**100%**	**160,424**

TABLE 1: Number and Share of Tagged Hosts per Type.

Pattern	Success Rate	# Vuln. Hosts	Probe	Spoof
S1	97.5%	5,859	S+R	S+A+PA
SC2	69.6%	46	S+R	S+A+PA
SCx (x≥3)	76.9%	13	S+R	S+A+PA
Sa	81.1%	111	2(S+R)	S+2(A+PA)
D1	64.0%	22,743	2(S+R)	S+3(A+PA)
D2	96.1%	8,611	S+R	S+2(A+PA)
D3	90.1%	9,228	2(S+R)	S+3(A+PA)
Overall	79.3%	46,611	-	-

TABLE 2: Verification Result Summary. "Probe" and "Spoof" summarize the number and types of packets we use to probe the server ISNs (with the probe IP) and to spoof the connection (with the victim IP). S=SYN, A=ACK, R=RST, PA=PSH-ACK packet containing the HTTP request.

5.5. Resource Identification

Next, using the crawler from Section 4.5, we search for the large HTTP resources on all tagged hosts. Identifying large resources is important for performing amplification attacks. Note that our HTTP crawler cannot handle JavaScript, so the following results denote a lower bound. Among the identified vulnerable hosts, excluding 280 hosts that became unreachable, our crawler finds 54,043 (33.7%) hosts with an HTTP resource larger than 1,000 bytes. While the other hosts have vulnerable ISN patterns, they did not expose large HTTP resources. In the following, we will verify the 54,043 tagged hosts with large ($\geq 1kB$) resources.

5.6. Attack Verification

In this section, we first introduce the verification strategy per predictable ISN pattern type, including how we probe server ISNs and predict the next ISNs, and then present the verification results.

5.6.1. Verification Strategy per ISN Pattern Type.

S1: We send one SYN packet using the probe IP and wait 0.35s for the SYN-ACK response to learn the server ISN *pisn*. Next, using the victim IP, we prepare one set of SYN, ACK, and PSH-ACK packets to establish the spoofed TCP connection and request for HTTP resource, where the ACK and PSH-ACK packet use the ack number $pisn + 1$.

SCx: We send one probe SYN using the probe IP and learn the server ISN *pisn*. We then send SYN, ACK, and PSH-ACK packets from the victim IP, where the ack number used by ACK and PSH-ACK packets is $pisn + 1$.

SA: We send two SYN packets using the probe IP. We then wait 0.35s after each SYN probe to learn the server ISNs, $pisn_1$ and $pisn_2$. Next, using the victim IP, we send one SYN packet and two sets of ACK and PSH-ACK packets using ack numbers $pisn_1 + 1$ and $pisn_2 + 1$.

D1: We first send two SYN packets from the probe IP to learn the server ISNs $pisn_1$ and $pisn_2$ and the current ISN difference $pdiff = pisn_2 - pisn_1$. From the victim IP, we send one SYN and three sets of ACK and PSH-ACK packets, where the ack numbers are $pisn_2 + pdiff + 1$, $pisn_2 + pdiff + m_1 \times gcd + 1$ and $pisn_2 + pdiff + m_2 \times gcd + 1$ (gcd, m_1, and m_2 are the greatest common divisor and the two most common multiplier fluctuations collected from the scan).

D2: We send one SYN packet from the probe IP to learn the server ISN *pisn*. Next, we send one SYN and two sets of ACK and PSH-ACK packets using the victim IP, where the ack numbers used by the ACK and PSH-ACK packets are $pisn + 1$ and $pisn + 2$.

D3: We send two SYN packets using the probe IP to learn the current server ISNs $pisn_1$ and $pisn_2$ and the ISN difference *pdiff*. Similar to D1 hosts, we also make three ISN guesses. Using the victim IP, we send one SYN packet and three sets of ACK and PSH-ACK packets with ack numbers $pisn_2 + pdiff + m1 + 1$, $pisn_2 + pdiff + m2 + 1$ and $pisn_2 + pdiff + m3 + 1$ ($m1$-$m3$ are the three most common difference of differences observed during the scan).

5.6.2. Verification Success Rate. Using the verification setup discussed in Section 4.6 and the per-pattern verification strategy, we perform the simulated attack. We verified 46,611 out of 54,043 hosts; we had to exclude the remaining cases, as they did no longer respond to SYNs likely due to IP churn. Overall, we successfully requested IP-spoofed resources from 36,973 (79.3%) hosts. Table 2 summarizes the success rate for the different pattern types.

Failure Analysis: Except for hosts of type D1, hosts of other types achieve a relatively high verification success rate. For hosts of type D1, because of more fluctuation in step sizes, there is a higher chance that the actual ISN falls out of the range of the three predicted ISN values. To reason more about the low verification success rate of type D1, we compare the predicted ISN chosen by the verification program with the actual server ISN in the received SYN-ACK packet. It appears that in 2,787 out of 8,198 (34.0%) failure cases, we actually find the correct server ISN in the three ack number attempts. Such a failure can happen for different reasons. First, it is possible that the 0.005 seconds delay we used is insufficient to eliminate all packet reordering, or a slow server may not be able to generate a SYN-ACK response. Also, it is possible that a server may close the half-open TCP connection if the first received ACK packet fails to acknowledge the server's ISN.

5.7. Amplification Factor Estimation

After obtaining the verification success rate, we now estimate the amplification factor that an attacker can achieve for responsive and unresponsive victims, as discussed in Section 4.7. Note that we exclude those hosts that failed in the verification. We do so as we find that across verifications,

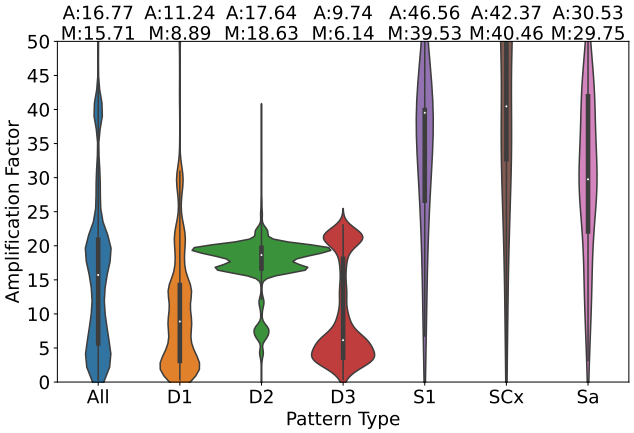

Figure 4: Amplification Factor Per Predictable ISN Pattern. A=average, M=Median. (Unresponsive Victim)

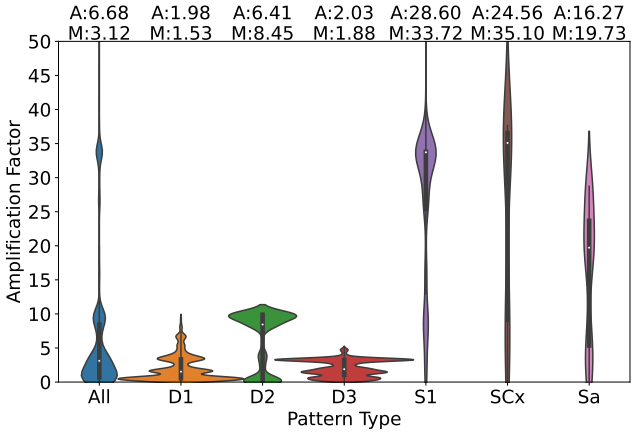

Figure 5: Amplification Factor Per Predictable ISN Pattern. A=average, M=Median. (Responsive Victim with 150ms RTT)

approximately 60% of the failed hosts overlap. Thus, an attacker can choose these more reliable hosts for an attack.

5.7.1. Unresponsive Victims. Figure 4 shows the amplification factor for each predictable ISN pattern type targeting unresponsive victims. When evaluating the amplification factor, we excluded the top 1% of outliers, which include mega-amplifiers potentially caused by abnormal retransmissions. The average amplification factor an attacker can achieve is 16.77, with a median of 15.71.

The amplification factor depends on the attack cost (as also summarized by the two rightmost columns in Table 2), the resource size, the number of retransmissions, and the initial congestion window size. Thus, the amplification strongly varies per ISN pattern type. D1 and D3 hosts have a lower amplification factor than hosts of Sx types. This is because for hosts of D1 and D3 types, we need two SYN probes and three guesses to achieve better success rates, which increases the attack cost. Also, only 52% of D1 hosts and 46% of D3 hosts have resources of more than 5840 bytes (four full-sized 1460 bytes segments), and for those hosts with larger resources, 42% have a small initial congestion window that only allows the server to send ≤ 2 segments. In contrast, 84% of Sx type hosts have resources of more than 5840 bytes, and 85% of these hosts have a large initial congestion window that allows around 10 segments.

Attack Extension: As discussed in Section 4.6.2, when the server's payload is not acknowledged, the server would hold back further TCP payloads, but an attacker can use cumulative ACKs to acknowledge the server payloads and trigger more payloads. To verify the effectiveness of the cumulative ACK strategy, we sampled 700 hosts from the D2 and D3 hosts that host resources sized $> 18kB$. To learn the servers' initial congestion window, we first establish a TCP connection through a normal handshake and send a PSH-ACK packet to request the HTTP resource. We do not send further packets to acknowledge payloads sent by the server and only measure the received payload size. Next, in the verification, 0.2s after sending the HTTP request, we send one additional ACK packet for each predicted ISN *plus* the congestion window size and thereby trigger more segments from the server. As a result, the average amplification factor increases from 15.17 to 20.58 (D2 hosts from 18.65 to 23.66, D3 hosts from 4.94 to 11.81).

Using the same 700 sampled hosts, we now evaluate the strategy that cumulatively acknowledges each chunk *and* then reuses the connection by sending an additional HTTP request after acknowledging the last. This time, 86.8% of D2 hosts completed the first HTTP request and accepted the second request. The average amplification factor for D2 hosts is thus further increased from 23.66 to 29.08. In contrast, many D3 hosts that completed the first HTTP request will close the connection, which is possibly caused by HTTP server implementations that do not keep a TCP connection alive by default. The amplification factor for D3 hosts thus even decreased as the second HTTP request and following ACK packets only increase the attack cost without triggering more reflected traffic. For such hosts, an attacker can acknowledge all chunks except the last (to allow retransmission) instead of sending a second request. This way, the amplification factor for D3 hosts can be increased to 14.57.

5.7.2. Responsive Victims. Recall from Section 3 that attackers can typically select unresponsive victims as those have more favorable amplification. Still, for completeness, we follow the methodology in Section 4.7 to evaluate the amplification factor for responsive victims. From our scan in Section 5.1, we find the median RTT from our host to servers is approximately 150ms. Thus, we assume a responsive victim with a 150ms RTT with the vulnerable server and estimate the amplification factor by excluding the reflected packets dispatched after the connection may already have been terminated. Excluding the top 1% outliers, the average amplification factor for responsive victims with 150ms RTT is 6.68 (median 3.12). Though the factor is smaller than for unresponsive victims, the attack is still effective. In Figure 5,

we summarize the amplification factor per ISN pattern for responsive victims with a 150ms RTT. In Appendix Figure 7, we also summarize the average amplification factor change per ISN pattern for responsive victims under different values of RTTs. Except for hosts of S1 type, the amplification factor is less sensitive to different RTTs within a reasonable range of 1 second. In the first second, most servers can finish transmitting packets allowed by the initial congestion window, which are potentially available for responsive victims. However, retransmissions, which are another important contributing component of the amplification factor, are usually started after 1 second, making them unavailable when targeting responsive victims.

As with the unresponsive victim case, an attacker can use cumulative ACK packets to extend the amplification factor. However, the attacker needs to send the cumulative ACK packets after a shorter delay (e.g., 0.1s) to ensure the new server payloads are sent before the connection is terminated. We perform the experiment again using the same 700 sampled D2 and D3-type hosts and send a single cumulative ACK. Indeed, the average amplification factor can be improved from 5.90 to 9.61 for responsive victims with a 150ms RTT.

6. Completeness Check

Using statistical features and unsupervised clustering, we explored potential predictable ISN patterns and vulnerable hosts. In this section, we aim to check if our method actually covers all predictable ISNs. Unfortunately, we lack a ground truth of vulnerable hosts to verify this hypothesis easily. Having said this, we observed that the vast majority (86.4%) of hosts with confirmed vulnerable ISN patterns (i) exposed ISNs in a small range ($\approx$ 1.4% of the 32-bit ISN space) during our probes *and* (ii) did so in a monotonically increasing way. This can be explained by the mono-increasing time-based ISN counter as suggested by RFC 9293, which, for vulnerable hosts, is shared among clients with different address tuples.

We leverage this observation to cross-check how many of the scanned hosts that exposed small-ranged, mono-increasing ISNs were included in our tagging. Recall that our scan takes $\approx$ 3.85s per host; the collected ISNs for vulnerable hosts shall rarely wrap around, as (i) frequent sequence space overlap may cause acceptance of old segments, and (ii) popular implementations do not drain the ISN space that frequently. With respect to the argument, we refer to the ISN space cycle time of 274s of Linux [33] and use this as a baseline to estimate the ISN range of potentially vulnerable servers using global ISN counters. That is, we inspect those hosts that exposed mono-increasing raw ISN numbers within the range of $60,348,993 = (2^{32}/274) \times 3.85$ ISNs during our scan (the aforementioned $\approx$ 1.4% of the ISN space).

202,769 suspicious hosts used mono-increasing ISNs within this small range. Of those, we identified 138,538 (68.3%) as vulnerable in our methodology. We inspect the others semi-manually. 17k hosts (8.8%) had repetitive ISNs (or ISN differences and ISN difference of differences) within their sequences but were filtered out by our ISN signatures, as exploiting them is likely to cause low success rates when only given one to three ISN guesses. The remaining 46k hosts (22.9%) did not show any repetitions. We picked 20 random hosts to manually analyze their raw ISN sequences, ISN difference sequences, and ISN difference of difference sequences. While we could indeed manually reveal certain patterns (e.g., ISN differences ranging between 94k–112k for one host, or differences reliably in the range of 1M-7M for another host), the fluctuation of the differences was too large to allow for a reliable abuse of these servers. That is, the 46k hosts selected ISN numbers from a small range, which may render them susceptible to guessing when using thousands or even millions of guesses. However, they are impractical for amplification attacks, as the excessive number of required ACK packets increases the attack costs, and hence the amplification factor plummets.

7. Limitations

This work has several limitations. First, the work lacks a baseline of all vulnerable hosts with easily predictable ISN selection algorithms in the IPv4 network. Despite the identified vulnerable hosts, there may still be hosts running unknown, predictable ISN selection algorithms that were not captured in the study.

Second, we manually summarize predictable ISN patterns and corresponding ISN signatures, which reduces the scalability of the methodology. The heuristic and thresholds for ISN signatures are also not error-free, e.g., for type D1. They could be further optimized to remove hosts that cannot be reliably exploited. Optionally, an attacker may use ISN signatures with loose conditions to capture all hosts suspicious to a pattern, even if the host may require many guesses to predict the ISN (see Appendix A.3). Later, the attacker can use repetitive verifications to remove those hosts that cannot be reliably exploited.

Finally, this work only explored one aspect of connection-based amplification attacks, HTTP servers. In principle, other TCP-based applications may also be exploited in a connection-based amplification attack. When exploring HTTP servers, our resource crawling also only explored *src* and *href* in the HTTP response. The above limitations reduce the scope of the study, and the number of hosts that can be abused in a connection-based TCP amplification attack can be even larger than reported.

8. Related Work

Our work relates to prior works in three domains. First, we discuss previous works concerning TCP-based and UDP-based amplification attacks. Next, we introduce works exploring predictable ISN patterns. Finally, we briefly discuss general TCP spoofing and injection attacks.

8.1. TCP/UDP-based Amplification Attacks

Most prior works focusing on amplification attacks exploited UDP-based protocols, e.g., DNS [5], [6], [28], [38],

[39], [40] and NTP [7], [8], including a general overview of vulnerable protocols [9] and attempts to maximize amplification potential [41]. Researchers also explored amplification attacks utilizing routing [42] or application [43] traffic loops. However, all the above research focused on UDP-based applications, as UDP is connectionless, and an attacker can spoof the victim IP and trigger amplified traffic easily. Similar strategies cannot be employed in TCP-based amplification attacks. Since TCP is a connection-oriented protocol, applications and data transmission are only started after the connection is established. Thus, an unpredictable 32-bit server ISN can stop an off-path attacker from efficiently establishing a connection using a spoofed source IP.

Closest to our paper are prior works that revealed the potential of TCP-based amplification attacks. Kührer *et al.* show that the TCP handshake itself offers a mild degree of reflection, as servers would retransmit SYN-ACK packets for a half-open TCP connection until the connection is established or terminated [10]. Similarly, Nguyen *et al.* explored the SYN-ACK and FIN retransmission that can be exploited by an internal attacker in a virtual network exploiting the behavior of the virtual switch [44]. These attacks exploit the connectionless part of the TCP protocol and thus do not require establishing a spoofed TCP connection.

Another type of TCP-based amplification attacks relies on specific network infrastructures. Bock *et al.* showed that censorship middleboxes can be used to perform TCP amplification attacks [11]. As the middlebox is not aware of all routes among the protected server and clients, it tolerates incomplete TCP handshakes and sends blocking pages to the remote client once it observes partial traffic containing the forbidden HTTP requests. A follow-up work focused on patching and identifying such vulnerable devices [45]. Apart from middleboxes, content delivery networks (CDNs) can also be exploited to perform TCP-based amplification attacks. For example, HTTP range requests and CDN back-to-origin policies can cause an imbalance between attacker-to-CDN and the CDN-to-origin traffic [29], [30], [46], [47]. This enables the attacker to use small application-layer requests to trigger large traffic against the CDN infrastructure or an origin. By exploiting middleboxes or CDNs, an attacker can indeed avoid the challenge of setting up spoofed TCP connections. Our work is orthogonal, as we show that IP-spoofed TCP connections can also be abused for DoS, which represents a new angle of amplification.

Lastly, several works exploit TCP-based amplification attacks given very specific attacker setups. Abramov and Herzberg showed that a person-in-the-middle attacker can trigger an ACK storm between a victim server and a client by injecting additional payloads to each side [48]. Sherwood *et al.* found that misbehaving clients/attackers with non-spoofed connections to victim servers can use optimistic ACKs, which are sent before server data segments are received, to overload the link between the attacker and the victim [49]. Attackers in this type of work follow a different threat model and either observe or control TCP connections with the attack target.

8.2. TCP ISN Prediction

Reports on predictable TCP ISN vulnerabilities date back as far as 1985, where Morris found that the 4.2BSD kernel incremented the sequence number by fixed distances [22]. Other popular OSes also once had predictable ISNs, e.g., Windows [21] and Linux [23]. In 1996, RFC 1948 [50] suggested the use of a better ISN generation algorithm, which includes a time counter together with a hash function over the four address tuples, to mitigate predictable ISNs. In 2001, despite these efforts, Zalewski found many common OSes were still feasible targets (given 5000 guesses) for spoofing attacks [51]. Also in 2001, CERT/CC released a vulnerability note that covered eight affected vendors. Nowadays, major OSes mitigated predictable ISNs, while CERT/CC reports that many IoT OSes are still vulnerable [27], which aligns with our findings.

More recently, Zeng *et al.* proposed to use adding-weight chaotic time series for TCP ISN prediction [52]. When tested against Windows 2003, Redhat, and Windows XP, the method shows high prediction accuracy when forecasting the top six digits of the server ISN. Formby *et al.* examined the ISN sequences of smart grid devices from eight vendors in a substation [34]. Using a dataset passively collected over several months, the author explored the general distribution of ISNs over the whole 32-bit space and the relationship between ISN difference and time among consecutive SYN probes. In the end, the authors identified some devices using a small fraction of the ISN space and devices seemingly following the famous ”64k rule”. In 2021, Forescout [53] examined eleven embedded TCP stack implementations and found nine have vulnerable ISN selection algorithms. Except for hosts using weak arithmetic algorithms to increment the ISN, they also found stacks using vulnerable Linear Congruential Generators (LCGs). In our work, given the small number of probes and the pseudo-random nature of LCGs, we cannot reliably identify such vulnerable hosts using our methodology. In Appendix A.7, we still discuss attempts to identify recognizable pseudo-random number generators. Unlike previous works, we propose a approach to learn predictable ISN patterns in the whole IPv4 network and show the practicability of amplification attacks using these vulnerable hosts.

The work that is most related to ours is a network-wide scan for ISN selection algorithms with insufficient algorithm complexity by Espinoza [54]. Using hundreds of ISNs collected from each host, Espinoza employed several methods to estimate the ISN selection algorithm entropy and complexity, including using the NIST statistical test and hierarchical clustering to find non-random ISN patterns, ISN ranges to estimate maximum entropy, and the Pearson correlation coefficient and neural networks to estimate entropy reduction of ISN algorithms. This prior work does not focus on vulnerable hosts with extremely small ISN entropy, which are, however, required for amplification attacks. In total, out of approximately 60 million reachable HTTP hosts, the author found that 11.5% hosts have an entropy level less than 2^{26}, which is susceptible to TCP spoofing

or injection attacks, while the reported number of hosts with extremely low entropy ($\leq 2^6$, a core requirement of amplification) is negligible. In addition, we explored the threat of amplification attacks building on top of weak ISNs, discussing attack strategies beyond the mere identification of vulnerable hosts.

Some ISN predictability measurement methodologies also made it to practical tools. For example, Nmap provides a "difficulty" metric to measure how hard the ISN prediction is. However, as acknowledged by the developers, the algorithm has several weaknesses and may mark easily predictable targets as hard ones [24]. In Appendix A.6, we provide a comparison between Nmap and our methodology.

Though not used in TCP ISN predicting, previous works [35], [55] used ML to identify the changing patterns of IPID counters used by a server. Similar to the TCP ISNs, some OSes use an IPID counter that is globally shared. Both of the two works used supervised classification to classify hosts with different typical anomalous IPID behaviors (e.g., random, global increment, constant, etc.). Instead, we use unsupervised clustering and thus have a chance to identify unknown vulnerable patterns.

8.3. TCP Spoofing and Injection Attacks

In general, plenty of works focus on TCP spoofing and injection attacks. Instead of predicting the ISN, the strategy of these works is to infer the sequence number (for injection) or the ISN (for spoofing) through side channels. Usually, these attacks require collaborative malware on the host or in the same network [56], [57], [58]. Other works exploit the global limits in TCP stack implementations, particular behavior of firewalls, and specific applications [59], [60], [61], [62]. These findings, however, are orthogonal to our threat models, in which attackers search for reliable and efficient methods to predict ISNs. To confirm the sequence number (or ISN) using side channels, an attacker needs to send a large number of probes, which is impractical for amplification attacks.

9. Conclusion

We propose a methodology to identify predictable ISN patterns at the Internet scale. We find that, after almost 40 years of the first report of predictable ISN algorithms, there is still a large number of vulnerable devices using an easy-to-predict ISN selection algorithm in IPv4 alone. We show that such weak ISNs enable attackers to perform practical connection-based TCP amplification attacks. We aim to increase the attention of network operators, especially since we only studied a single protocol (HTTP). Together with other TCP-based applications, an attacker may identify an even larger number of amplifiers.

References

[1] B. Krebs, "DDoS on Dyn Impacts Twitter, Spotify, Reddit," https://krebsonsecurity.com/2016/10/ddos-on-dyn-impacts-twitter-spotify-reddit/, accessed at November 11, 2024.

[2] C. Morales, "NETSCOUT Arbor Confirms 1.7 Tbps DDoS Attack; The Terabit Attack Era Is Upon Us," https://www.netscout.com/blog/asert/netscout-arbor-confirms-17-tbps-ddos-attack-terabit-attack-era, accessed at November 11, 2024.

[3] C. Cimpanu, "AWS said it mitigated a 2.3 Tbps DDoS attack, the largest ever," https://www.zdnet.com/article/aws-said-it-mitigated-a-2-3-tbps-ddos-attack-the-largest-ever/, accessed at November 11, 2024.

[4] A. Dahan, "Business as usual for Azure customers despite 2.4 Tbps DDoS attack," https://azure.microsoft.com/en-us/blog/business-as-usual-for-azure-customers-despite-24-tbps-ddos-attack/, accessed at November 11, 2024.

[5] M. Jonker, A. King, J. Krupp, C. Rossow, A. Sperotto, and A. Dainotti, "Millions of Targets under Attack: a Macroscopic Characterization of the DoS Ecosystem," in *Internet Measurement Conference*, London, United Kingdom, 2017, pp. 100–113.

[6] X. Li, D. Wu, H. Duan, and Q. Li, " DNSBomb: A New Practical-and-Powerful Pulsing DoS Attack Exploiting DNS Queries-and-Responses ," in *2024 IEEE Symposium on Security and Privacy*, Los Alamitos, CA, USA, 2024, pp. 4478–4496.

[7] J. Czyz, M. Kallitsis, M. Gharaibeh, C. Papadopoulos, M. Bailey, and M. Karir, "Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014, pp. 435–448.

[8] L. Rudman and B. Irwin, "Characterization and analysis of NTP amplification based DDoS attacks," in *2015 Information Security for South Africa*, 2015, pp. 1–5.

[9] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse," in *Network and Distributed System Security Symposium*, San Diego, California, USA, 2014.

[10] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Hell of a Handshake: Abusing TCP for Reflective Amplification DDoS Attacks," in *Proceedings of the 8th USENIX Workshop on Offensive Technologies*, August 2014.

[11] K. Bock, A. Alaraj, Y. Fax, K. Hurley, E. Wustrow, and D. Levin, "Weaponizing Middleboxes for TCP Reflected Amplification," in *30th USENIX Security Symposium*, 2021, pp. 3345–3361.

[12] J. Krupp, I. Grishchenko, and C. Rossow, "AmpFuzz: Fuzzing for Amplification DDoS Vulnerabilities," in *31st USENIX Security Symposium*, Boston, MA, 2022, pp. 1043–1060.

[13] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Exit from Hell? Reducing the Impact of Amplification DDoS Attacks," in *23rd USENIX Security Symposium*, San Diego, CA, USA, 2014, pp. 111–125.

[14] Shadowserver, "Shadowserver," https://www.shadowserver.org/, accessed at November 11, 2024.

[15] J. Lell, "Quick Blind TCP Connection Spoofing with SYN Cookies," https://www.jakoblell.com/blog/2013/08/13/quick-blind-tcp-connection-spoofing-with-syn-cookies/, accessed at November 11, 2024.

[16] V. Paxson, "An Analysis of Using Reflectors for Distributed Denial-of-Service Attacks," *SIGCOMM Computer Communication Review*, vol. 31, no. 3, pp. 38–47, Jul. 2001.

[17] E. Blanton, D. V. Paxson, and M. Allman, "TCP Congestion Control," RFC 5681, Tech. Rep. 5681, Sep. 2009. [Online]. Available: https://www.rfc-editor.org/info/rfc5681

[18] A. Gurtov, T. Henderson, S. Floyd, and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 6582, Apr. 2012. [Online]. Available: https://www.rfc-editor.org/info/rfc6582

[19] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," *ACM SIGCOMM Computer Communication Review*, pp. 24–35, 1994.

[20] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks," RFC 8312, Feb. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8312

[21] Windows, "Patch Available to Improve TCP Initial Sequence Number Randomness," 1999, https://learn.microsoft.com/en-us/security-updates/securitybulletins/1999/ms99-046, accessed at November 1, 2024.

[22] R. T. Morris, "A weakness in the 4.2 BSD Unix TCP/IP software," 1985, http://nil.lcs.mit.edu/rtm/papers/117-abstract.html, accessed at November 11, 2024.

[23] Linux, "Linux Kernel 2.2 - Predictable TCP Initial Sequence Number," 1999, https://www.exploit-db.com/exploits/19522, accessed at November 11, 2024.

[24] Nmap, "TCP/IP Fingerprinting Methods Supported by Nmap," https://nmap.org/book/osdetect-methods.html#osdetect-gcd, accessed at November 11, 2024.

[25] B. Harris and R. Hunt, "Tcp/ip security threats and attack methods," *Computer Communications*, vol. 22, no. 10, pp. 885–897, 1999.

[26] W. Eddy, "Transmission Control Protocol (TCP)," RFC 9293, Tech. Rep. 9293, Aug. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9293

[27] C. C. Center, "Multiple TCP/IP implementations may use statistically predictable initial sequence numbers," https://www.kb.cert.org/vuls/id/498440, accessed at November 11, 2024.

[28] J. Bushart and C. Rossow, "DNS Unchained: Amplified Application-Layer DoS Attacks Against DNS Authoritatives," in *Research in Attacks, Intrusions, and Defenses*, 2018, pp. 139–160.

[29] Z. Lin, Z. Lin, X. Liu, J. Chen, R. Guo, C. Chen, and S. Xiao, "CDN Cannon: Exploiting CDN Back-to-Origin Strategies for Amplification Attacks," in *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA, 2024, pp. 5717–5734.

[30] W. Li, K. Shen, R. Guo, B. Liu, J. Zhang, H. Duan, S. Hao, X. Chen, and Y. Wang, "CDN Backfired: Amplification Attacks Based on HTTP Range Requests," in *Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2020, pp. 14–25.

[31] CAIDA, "State of IP Spoofing," Tech. Rep., October 2024, https://spoofer.caida.org/summary.php, accessed at October 30, 2024.

[32] ZMap, "ZMap," https://github.com/zmap/zmap, accessed at November 11, 2024.

[33] Linux, "Linux ISN Space Cycle Time," https://elixir.bootlin.com/linux/v5.15.143/source/net/core/secure_seq.c#L39, accessed at November 11, 2024.

[34] D. Formby, S. S. Jung, J. Copeland, and R. Beyah, "An Empirical Study of TCP Vulnerabilities in Critical Power System Devices," in *Proceedings of the 2nd Workshop on Smart Energy Grid Security*, 2014, pp. 39–44.

[35] H. Schulmann and S. Zhao, "ZPredict: ML-Based IPID Side-channel Measurements," *ACM Transactions on Privacy and Security*, vol. 27, no. 4, 2024.

[36] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[37] requests, "The Python Requests Model," https://pypi.org/project/requests/, accessed at October 30,2024.

[38] G. C. M. Moura, S. Castro, J. Heidemann, and W. Hardaker, "TsuNAME: exploiting misconfiguration and vulnerability to DDoS DNS," in *Proceedings of the 21st ACM Internet Measurement Conference*, 2021, pp. 398–418.

[39] W. Xu, X. Li, C. Lu, B. Liu, H. Duan, J. Zhang, J. Chen, and T. Wan, "TsuKing: Coordinating DNS Resolvers and Queries into Potent DoS Amplifiers," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 311–325.

[40] H. Duan, M. Bearzi, J. Vieli, D. Basin, A. Perrig, S. Liu, and B. Tellenbach, "CAMP: Compositional Amplification Attacks against DNS," in *33rd USENIX Security Symposium*, Philadelphia, PA, 2024, pp. 5769–5786.

[41] S.-J. Moon, Y. Yin, R. A. Sharma, Y. Yuan, J. M. Spring, and V. Sekar, "Accurately Measuring Global Risk of Amplification Attacks using AmpMap," in *30th USENIX Security Symposium*, 2021, pp. 3881–3898.

[42] Y. Nosyk, M. Korczyński, and A. Duda, "Routing Loops as Mega Amplifiers for DNS-Based DDoS Attacks," in *Passive and Active Measurement*, 2022, pp. 629–644.

[43] Y. Pan, A. Ascheman, and C. Rossow, "Loopy Hell(ow): Infinite Traffic Loops at the Application Layer," in *33rd USENIX Security Symposium*, Philadelphia, PA, 2024, pp. 235–252.

[44] S. D. Nguyen, M. Mimura, and H. Tanaka, "Abusing TCP Retransmission for DoS Attack Inside Virtual Network," in *Information Security Applications*, 2018, pp. 199–211.

[45] J. Oortwijn and C. Gañán, "Patch Pilgrimage: Exploring the Landscape of TCP Reflective Attacks and User Patching Expedition," in *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, 2024, pp. 1395–1404.

[46] Z. Lin, Z. Lin, X. Liu, Z. Ying, and C. Chen, "Unveiling the Bandwidth Nightmare: CDN Compression Format Conversion Attacks," in *2024 2nd International Conference on Big Data and Privacy Computing*, 2024, pp. 97–106.

[47] C. Xu, J. Li, and J. Liu, "Yet Another Traffic Black Hole: Amplifying CDN Fetching Traffic with RangeFragAmp Attacks," in *Collaborative Computing: Networking, Applications and Worksharing*, 2021, pp. 439–459.

[48] R. Abramov and A. Herzberg, "Tcp ack storm dos attacks," *Computers & Security*, vol. 33, pp. 12–27, 2013.

[49] R. Sherwood, B. Bhattacharjee, and R. Braud, "Misbehaving TCP receivers can cause internet-wide congestion collapse," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, 2005, pp. 383–392.

[50] S. M. Bellovin, "Defending Against Sequence Number Attacks," RFC 1948, May 1996. [Online]. Available: https://www.rfc-editor.org/info/rfc1948

[51] M. Zalewski, "Strange attractors and TCP/IP sequence number analysis," https://lcamtuf.coredump.cx/oldtcp/tcpseq.html, accessed at November 11, 2024.

[52] F. Zeng, K. Yin, and M. Chen, "Research on TCP Initial Sequence Number prediction method based on adding-weight chaotic time series," in *2008 The 9th International Conference for Young Computer Scientists*, 2008, pp. 1511–1515.

[53] Forescout, "Number:Jack, Weak ISN Generation in Embedded TCP/IP Stacks," https://www.forescout.com/resources/numberjack-weak-isn-generation-in-embedded-tcpip-stacks/, accessed at November 11, 2024.

[54] A. M. Espinoza, "The nature of ephemeral secrets in reverse engineering tasks," 2018, https://digitalrepository.unm.edu/cs_etds/97.

[55] F. Salutari, D. Cicalese, and D. J. Rossi, "A Closer Look at IP-ID Behavior in the Wild," in *Passive and Active Measurement*, 2018, pp. 243–254.

[56] 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, pp. 593–604.

[57] W. Chen and Z. Qian, "Off-Path TCP Exploit: How Wireless Routers Can Jeopardize Your Secrets," in *27th USENIX Security Symposium*, Baltimore, MD, 2018, pp. 1581–1598.

[58] Y. Gilad and A. Herzberg, "Off-Path TCP Injection Attacks," *ACM Transactions on Information and System Security*, vol. 16, no. 4, 2014.

[59] 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, pp. 1323–1335.

[60] 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.

[61] 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*, Austin, TX, 2016, pp. 209–225.

[62] Y. Pan and C. Rossow, " TCP Spoofing: Reliable Payload Transmission Past the Spoofed TCP Handshake ," in *2024 IEEE Symposium on Security and Privacy*, Los Alamitos, CA, USA, 2024, pp. 4497–4515.

[63] Wikipedia, "Commonly used Linear Congruential Generator Parameters - Wiki," https://en.wikipedia.org/wiki/Linear_congruential_generator#Parameters_in_common_use, accessed at March 26, 2025.

[64] lwIP, "LWIP_HOOK_TCP_ISN," https://www.nongnu.org/lwip/2_0_x/group__lwip__opts__hooks.html#ga078d203053911cf3af178392700386a4, accessed at March 26, 2025.

Appendix A.

A.1. TCP Handshake & Slow Start Illustrated

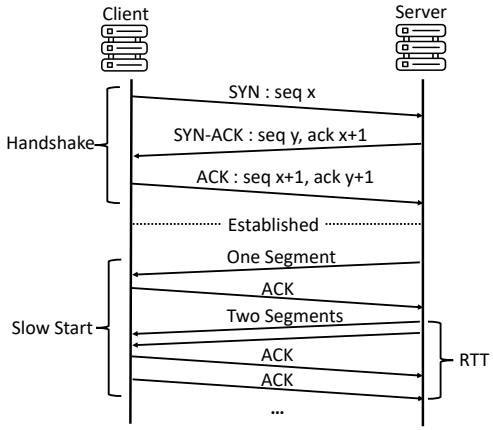

Figure 6: TCP Handshake and Slow Start

A.2. Resource Scan HTTP Request

```
GET / HTTP/1.1
Host: Server IP
User-Agent: -
Accept: */*
Accept-Encoding: identity
```

Listing 1: Crawler HTTP Header

A.3. Inapplicable Patterns

Over-Complex Weak ISN Selection: We also find hosts sharing a similar pattern as those described in Section 5.3 but are likely inapplicable for an amplification attack.

111	93	82	213	45	241
112	188	78	273	220	

TABLE 3: Example Over-Complex Weak ISN Selection

Shown in Table 3 is an example of a normalized (using $GCD = 64000$) ISN difference sequence from one of such hosts. In comparison to hosts of the D1 Type, the host has ISN differences using a wider range of random multipliers together with the shared GCD, and the difference of difference also does not repeat. In terms of a connection-based TCP amplification attack, these hosts cannot be abused, as guessing the correct ISN requires a large number of false ACK attempts, which will greatly increase the amplification attack cost. However, the ISN selection algorithm is still potentially vulnerable and may be exploited by an attacker in a general TCP spoofing attack, where the attacker can send many ACK attempts to spoof a TCP connection.

3958293983	672617874	3958293983	672617874
3958293983	672617874	3958293983	672617874
3958293983	672617874	3958293983	672617874

TABLE 4: Example Source-based Static ISN Selection

Source-based static ISN: In a TCP amplification attack, we require the attacker to predict the server ISN selected for the victim IP, which is not controlled by the attacker. If a server chooses static ISNs based on the source IP of the SYN request, though the pattern appears to be predictable repetitive static ISNs, an attacker cannot use it. However, as the Markov model can observe ISNs collected using both IP addresses, it would highlight those ISN patterns as predictable. Table 4 shows an example from one of the false-positive clusters. It appears that for SYN requests sent from two source IPs, the server used the static ISNs 3958293983 and 672617874, respectively.

$HostA$	2664567320	4210561560	1201209880	··
$HostB$	2127696408	3673690648	664338968	··
$HostC$	2664567320	4210561560	1201209880	··

TABLE 5: Example Middlebox-specific ISN Selection

Middlebox/Firewall Specific ISN Selection: For completeness, we briefly examined the 207 clusters excluded by Markov in Section 5.3. We found that many hosts of these clusters are from similar network ranges, and independent hosts in the same cluster can share the same raw ISN, ISN difference sequences (see example in Table 5). A closer check confirms that the majority of these clusters have over 90% of their hosts originating from 4 major ASes, which

is thus suspicious to be middleboxes deployed by the few ASes. To confirm they are inapplicable for amplification attacks, we sampled clustered hosts from each major AS, and found that they either don't serve any HTTP resource or use a pure IP/port based ISN selection algorithm.

A.4. ISN Signatures

We develop the following signatures for the six predictable ISN patterns summarized from the filtered clusters.

Type S1: Hosts of type S1 choose the same ISN for all our 12 SYN probes during scanning. We tag all hosts with only one distinct ISN as type S1.

Type SCx: Hosts of type SCx would use a repeating ISN for x continuous SYN probes. For those sequences, we calculate the most common repeating duration x of the ISN sequences and label the host with the SCx tag.

Type SA: Hosts of type SA alternatively use multiple repeated ISNs during our scan. Thus, if the ISN sequence contains repeated ISNs and the most common repeating duration of them is one, we tag the host as type SA. Additionally, we exclude a host if there are more than 4 distinct raw ISNs. If the vulnerable host has a low probability of using repeated ISNs, the attacker would need many probing and spoofing packets during verification to successfully spoof the connection. Importantly, as the server does not use the same ISN continuously, we need to further verify that SA type hosts are not inapplicable hosts that select a static ISN based on the source IP. To do this, we verify if any repetitive ISNs are used across probes sent from different source IPs.

Type D1: Hosts of type D1 normally use ISN differences with respect to a shared *GCD*, and ISN differences can fluctuate because of timing or background traffic noise. Thus, we first calculate the frequency of the three most common ISN differences. If the three most common ISN differences are used more than 7 times, we tag the host as type D1. There is still a chance that vulnerable hosts with more different ISN differences can be predictable, especially when the server uses repeated difference of differences (fluctuation size). To capture such hosts, we calculate the difference of difference sequence for a host and tag hosts as type D1 as long as the number of distinct difference of differences is less than 8.

Type D2: Hosts of type D2 usually use a fixed step size of 0 or 1 in the ISN sequence. Thus, we calculate the ISN difference sequence for a host and tag a host as type D2 if, in the majority of cases, the host uses an ISN difference of one or zero.

Type D3: Hosts of type D3 usually use an ISN sequence where the ISN differences are monotonically updated. Hence, for type D3, we first check if the hosts' ISN difference sequence shows a monotonic trend. Next, similar to the requirements for D1 and SA, we exclude hosts with more than 8 distinct difference of difference choices. Hosts satisfying the above condition are then tagged as type D3.

A.4.1. Resolving Tagging Ambiguity. Some vulnerable hosts are ambiguous among different predictable ISN patterns. In the following, we provide more details about the ambiguity resolution.

SCx and Dx: SCx type hosts can be ambiguous with type Dx. For example, type D1 hosts may use a low ISN update frequency, and thus, during our probing, the server could choose the same ISN several times. In this case, we tend to tag the host as type Dx.

Dx types: Hosts of type D2 and D3 can be considered as special cases of type D1. That is, D2 is a special case of type D1 when the common ISN differences are 0 or 1, and D3 is a special case of type D1 when the ISN difference sequence shows a monotonic trend. In both cases, we tend to tag hosts as type D2 or D3 if applicable.

A.5. Responsive Victim Evaluation

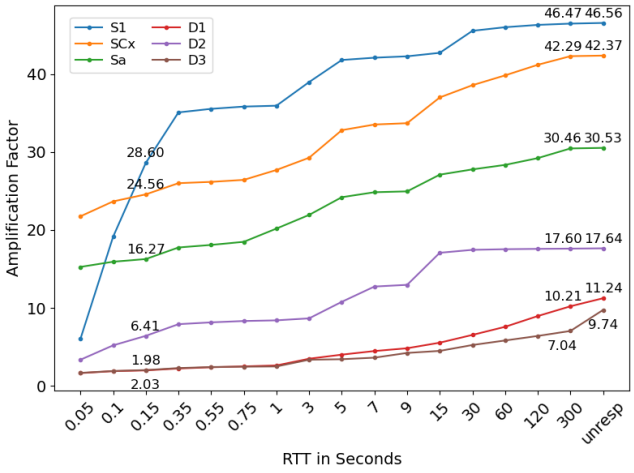

Figure 7: Average Amplification Factor per Predictable ISN Pattern under Different RTTs (Responsive Victim)

A.6. Nmap Comparison

To compare with Nmap, we sample 100 vulnerable hosts from each predictable ISN pattern type and cross-check Nmap's prediction difficulty score against our tagging result. For the Nmap result, as long as it does not mark the host as "Good Luck" level (the most difficult), we consider Nmap to be able to identify the vulnerable host. It turns out the Nmap prediction measure performs well against hosts of type S1 and D2, where it can find almost all tagged vulnerable hosts. For other types, Nmap provides poor results. That is, Nmap fails to identify 63.9% of the D1 hosts, 79.5% of the D3 hosts, and all of the SCx and SA hosts. However, when reading the Nmap documentation about the TCP ISN sequence predictability index, we believe the algorithm should capture more vulnerable hosts, especially those of types D1 and D3. The low identification rate could be due to the probe sending delay variance inflating the difficulty score or because of outlier ISNs.

A.7. Hosts Sharing Vulnerable PRNGs

In practice, TCP/IP stack implementation may also be vulnerable to spoofing and hence amplification attacks if the ISN lacks patterns but is still predictable. One such scenario occurs if a host uses a weak pseudorandom number generator (PRNG) to produce ISNs. If the TCP/IP stack uses a secure PRNG that is seeded properly, the ISN selection should be indistinguishable from a random number sequence. However, if the implementation instead uses a vulnerable PRNG that is either not properly seeded or cannot generate distinct sequences based on different seeds, it is still subject to prediction. Ideally, knowing the used PRNG algorithm, the attacker can reliably predict the next ISN with only one guess. Using the clustering methodology, we cannot identify such vulnerable hosts, as their pseudorandomness does not expose statistical patterns we can grasp.

However, if hosts use vulnerable PRNGs that generate fixed ISN sequences, we might find hosts with raw ISN collisions (i.e., hosts might share the same ISN sub-sequences). Though we only have 12 ISNs collected from each host, there is still a non-negligible probability of finding such a collision as long as the number of vulnerable hosts is large enough and the ISN sequence generated by the PRNG algorithm is short. For hosts with secure ISN generation algorithms, it is unlikely to observe such collisions. Using the set of hosts with all 12 ISNs collected, we extracted all 3-grams (3 subsequent ISNs) from each ISN sequence and searched for 3-gram collisions among all hosts. We remove hosts with ISN collisions while likely not using PRNGs, e.g., those already tagged in Section 5.4 and those show repeat ISNs within the 12 collected ISNs. In total, we identify 11k hosts showing 3-gram ISN overlaps and are thus suspicious of using vulnerable PRNG algorithms. Using the same fingerprinting methodology as described in Appendix A.8, we confirmed that many of these vulnerable hosts are from one home alarm system vendor. We disclosed this vulnerability to the affected vendor.

Note that the number of vulnerable hosts identified with this methodology represents a strict lower bound. The majority of vulnerable hosts sharing the same PRNG may not show overlaps or collisions due to our short ISN sequences. Attackers do not have to follow ethical standards and can thus obtain larger ISN sequences per host. This increases the likelihood of identifying additional PRNG hosts. Furthermore, once the attacker finds hosts with vulnerable PRNGs, the attacker can send an excessive number of SYN packets to learn each state in the PRNG.

Except for finding ISN collisions between collected ISN sequences, we also attempted to compare the collected ISN sequences with well-known LCG algorithms [63]. We feed a host's first raw ISN number of the ISN sequence to the LCG algorithm and generate the next ten ISNs. We then compare the generated ISN sequence with the actual collection and identify hosts that show ISN collisions. This way, we also find 5k hosts seemingly using the LCG parameter choices from glibc and Apple CarbonLib [63].

A.8. Vulnerable Hosts Fingerprinting

Pattern Type	Vendor
S1	TP-Link, NetGear
SCx/Sa	Microsoft-IIS
D1	CANON, Cente, RomPager, KTC RCU
D2	Loxone, Ethernut
D3	Lwip, Axema VAKA, Satel

TABLE 6: Fingerprints per Predictable ISN Pattern Type

As part of our resource identification, we also collected HTTP banners of identified vulnerable hosts. For each predictable ISN pattern type, we group hosts by the HTTP banner and find several suspicious device models. For hosts without HTTP banners, assuming that hosts with the same resource size and path are likely the same type of device, we again group hosts by the resource size and path and manually examine the landing pages of hosts in large groups. In Table 6, we summarize the top vulnerable device vendors for different predictable ISN patterns. Overall, vulnerable hosts identified by us cover a wide range of routers, smart home devices, embedded servers, etc. [3] In parallel with our submission, we disclosed the identified vulnerabilities to the corresponding vendors. We focus our disclosure efforts on vendors, as only they can patch such vulnerabilities, and as end users cannot be reliably reached based on IP addresses. We sent emails to each affected vendor, addressing the vulnerability and providing the vendor with example ISN sequences and landing pages. As of now, we have received feedback from the following affected vendors: (1) Loxone acknowledged our finding and will update the TCP stack in the next release. (2) Axema replied to our email and is now investigating the issue. (3) CANON was already aware of the predictable ISN issue, and the issue is now under investigation. (4) Cente patched the predictable ISN vulnerability but also noted that the devices identified by us will likely not be updated (by users).

A.9. Ethics

Our work involves active scans of the IPv4 network and thus can potentially add load to network devices. We carefully designed our experiment to avoid overloading network devices with the following measures: (1) Our experiments adhere to common best practices outlined in the Zmap paper. On the landing page of the scanning IP addresses, we provided our contact information and procedures to opt out from further scans and experiments. During or after our measurements, we did not receive a request to opt out (while we did receive automated notifications and a few personalized emails asking about the nature of the experiments). The fact that we did not receive any opt-out requests is mainly because we performed similar experiments in the past and adhered to opt-outs from previous scans. (2) At any point

3. The lwip developer is aware of the risk of predictable ISNs and has provided means for developers to replace the basic ISN generation algorithm with secure ones [64].

in time, we refrained from spoofing IP addresses that are not under our control by using a server with two public IP addresses controlled by us. (3) Our scans adhered to strict rate limits. We thus believe that the load added by our experiment is negligible compared to the global scanning noise (by Shodan et al.). The total load of packets we sent per device is comparable to normal device usage. In total, we sent 1 packet with Zmap, 12 SYNs for the ISN collection, then collected the HTTP resources (rate-limited crawls in line with standard HTTP scanning practices), and used at most 11 packets for verification. (4) To slow down the ISN collection, we used a 0.35s delay between SYN probes, which avoids SYN floods against real servers. In fact, as the vast majority of scanned servers have an RTT below 0.35s (to our scanner), and our scanner immediately replies to each SYN-ACK segment with RSTs, our ISN probes did not cause more than one half-open connection on probed servers in general. (5) We also made sure not to have any TCP connections open to servers in parallel, avoiding unnecessary server-side reservations (e.g., for send/receive buffers). (6) Similarly, the small-scale evaluations in which we requested a single resource (and in a very restricted experiment: two resources) from the reflectors did not introduce more load than a normal client requesting a similar resource from the tested device. (7) We refrained from investigating the bandwidth of the identified reflectors. While this indeed would have provided insights into the overall attack capacity of all vulnerable devices, we believe that the (potential) negative side effects of such measurements would outweigh the benefits. (8)We did not attempt to bypass any server-side authentication, though doing so would likely enable attackers to find more devices with substantial resources for amplification attacks. Hence, the resource sizes reported in the paper represent lower bounds.

In hindsight, we noticed that choosing a minimized User-Agent (User-Agent: -) was a missed opportunity to more explicitly expose our identity to other parties—we now changed this for future experiments. Still, we believe that the scan server configuration provided sufficient detail to all scanned parties.