Fragmentation Considered Vulnerable:

Blindly Intercepting and Discarding Fragments

Yossi Gilad[†] *and Amir Herzberg*[‡]
Dept. of Computer Science, Bar Ilan University
[†]*mail@yossigilad.com,* [‡]*amir.herzberg@gmail.com*

Abstract

We show that fragmented IPv4 and IPv6 traffic is vulnerable to DoS, interception and modification attacks by a blind (spoofing-only) attacker. We demonstrated a weak attacker causing over 94% loss rate and intercepting more than 80% of data between peers. All attacks are practical, and validated experimentally on popular industrial and open-source products, with realistic network setups (involving NAT or tunneling). The interception attack requires a zombie behind the same NAT or tunnel-gateway as the victim destination; the other attacks only require a puppet (adversarial applet/script in sandbox).

The complexity of our attacks depends on the predictability of the IP Identifier (ID) field and are simpler for implementations, e.g. Windows, which use globally-incrementing IP IDs. Most of our effort went into extending the attacks for implementations, e.g. Linux, which use per-destination-incrementing IP IDs.

1 Introduction

The Internet Protocol (IP) is the primary protocol that establishes the Internet. It is one of the most basic, widely deployed and used protocols, as well as one of the oldest (dated back to 1974), simplest and most well known and studied. There are many implementations of IP, including many embedded and other systems still using ancient implementations, and new implementations are developed constantly, esp. with the advance of mobile, ubiquitous internetworking. Naturally, over the years there were many abuses of vulnerabilities in the IP specifications (IPv4 [15] and IPv6 [6]). There was also an impressive effort to identify and fix such vulnerabilities; see [8].

It is therefore rather disturbing that in this paper we present significant, exploitable vulnerabilities in both IPv4 and IPv6. The vulnerabilities result from problems in the specifications; the severity of the exploits we found depends on implementation details, e.g., we found Windows platforms to be more vulnerable than Linux systems; however, Linux systems are also vulnerable, in practical scenarios, with a more sophisticated attack.

Our adversary model is a blind (i.e., non eavesdropping) attacker who is capable of spoofing. The vulnerabilities we present allow such attackers to perform devastating Denial of Service attacks on legitimate IP traffic in standard network settings, as well as to intercept (expose) and hijack (modify) traffic in common network topologies, where Network Address port Translators (NAT) devices are used.

Like many or most previous attacks on IP [8], our attacks exploit weaknesses in IP's fragmentation mechanisms. While conceptually simple, fragmentation involves some subtle issues and several related vulnerabilities. Probably the most significant vulnerabilities so far were implementation bugs, which were exploited for very effective DoS attacks: Teardrop [1], Rose [2], and Ping of Death [13] (see survey in [3]). However, these critical implementation bugs were long ago fixed and may still exist only in few relic systems. Furthermore, our work exploits *specification* vulnerabilities, not an implementation bug.

We are aware of only three known *specification* vulnerabilities related to fragmentation. The first uses specially-crafted fragments to bypass filtering by firewalls and intrusion-detection systems; this is discussed and addressed in [19, 14]. The two others are DoS vulnerabilities: fragment cache overflow [10] and fragment mis-association [9]; however, both of these have a very small impact - the maximal loss rate we were able to cause (or found reported) was less than 0.1%.

In contrast, we present highly-effective attacks against fragmented traffic. Like previous attacks, the attacks do not assume MitM capabilities, and can be launched by a mere spoofing (blind) adversary; and like [10, 9], they exploit a *specification* problem and not (just) an implementation bug.

Our attacks are based on predicting the IP-ID value used in packets between the victim source and destination, and then exploiting the predicted IP-ID to cause packet loss, interception or modification (for fragmented packets); Zalewski's note [18] provided an early hint of this attack vector. Indeed, once we can predict the IP-ID, it is easy to cause fragment loss: the attacker sends one tiny spoofed fragment with the expected IP-ID (and the victim source and destination addresses). Once the legitimate fragments arrive, the attacker's fragment will match some of them, resulting in wrong reassembly and packet loss. Modification (of only part of the packet) works similarly, except that the attacker has to carefully construct his fragment so that its content will replace the desired parts of the original payload. In certain (common) scenarios, interception is also easy: when the attacker controls a zombie machine behind the same Network Address Translator (NAT) as the destination, as illustrated in Figure 1, he can intercept a packet by changing the destination port specified in the first fragment. Details of these attacks are in Section 2.

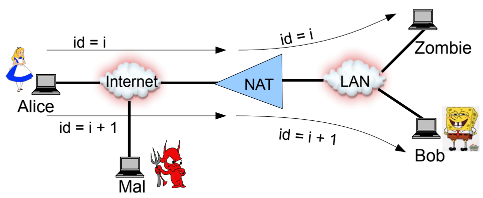

Figure 1: One scenario where obtaining the IP-ID is 'easy'. Alice uses one (per destination) identifier for all hosts behind the NAT. Zombie observes a packet he receives from Alice and obtains the current per-destination identifier that Alice also uses for Bob. Mal is a blind adversary that controls Zombie.

The main remaining challenge is prediction of the IP-ID value. In the popular family of Windows$^{\circledR}$ operating systems, this is not much of a challenge. These systems use a *single, global IP-ID counter* for all destinations, so the attacker can simply predict the IP-ID of a sender by receiving any packet from him (including common responses such as ICMP echo/error or TCP RST/SYN-ACK). Hence, these systems are vulnerable to loss, interception and modification attacks as outlined above (and described in Section 2).

Most of our effort went to efficiently predict the IP-ID for other operating systems, e.g. Linux, which use *per-destination IP-ID counters*. In the (common) NAT scenario of Figure 1, the prediction is still easy (and described in Section 2): since Alice uses the same destination IP address (of the NAT) to send packets to both Zombie and Bob, then Zombie receives the current value of the relevant IP-ID counter whenever he receives a packet from Alice. The harder challenge is to predict the IP-ID when the victim (Bob) is not behind a NAT at all, or when the attacker does not control a zombie behind such NAT. This is what we do in the rest of the paper.

In Section 3 we show how to expose the IP-ID using only a *puppet* PuZo, i.e., code restricted within sandbox (e.g., applet, script), rather than a 'real zombie'. Furthermore, these attacks work not only when Bob is 'behind' a NAT, but also in two other common scenarios: when PuZo runs on Bob's machine, and when Alice and Bob are connected by a tunnel and PuZo is also connected via the tunnel (with Alice), as illustrated in Figure 2.

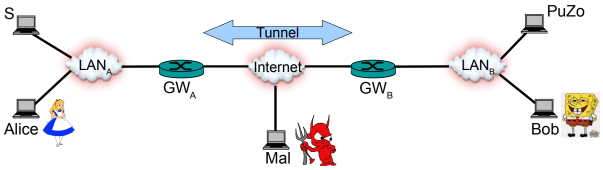

Figure 2: A Tunnel Topology, one of the three topologies for the ID-Exposing Attack (see text). An IP in IP tunnel exists from LAN_A to LAN_B, GW_A uses a per-destination IP identifier counter and PuZo is an end host running adversarial code (possibly a puppet restricted to sandbox - see Subsection 3.3). S is some network server that PuZo may receive data from, and Alice and Bob are honest hosts in LAN_A and LAN_B respectively.

Note that in the tunnel scenario, since Alice uses per-destination IP-ID counters, then packets sent to PuZo will use a separate counter than the one used for packets to Bob. However, we show that PuZo can help the attacker expose the IP-ID used by GW_A to send packets to GW_B, hence communication over the tunnel is vulnerable to the loss, interception and modification attacks as outlined above (and described in Section 2). Of course, if the communication is cryptographically-protected (e.g., IPsec tunneling), then interception and modification may be meaningless.

In Section 3 we assume that there is no legitimate traffic during the ID-exposing process. In Section 4, we present an attack that requires initial knowledge of the IP-ID (e.g., by applying previous attack during period of no traffic), and then 'maintains' this knowledge, while also discarding most of the legitimate traffic. This is a very efficient attack, but since most legitimate traffic is lost, it is mainly useful for DoS. In the full version of this paper [7] we present two improvements to the ID-exposing attack (presented in Section 3) to handle concurrent legitimate traffic; this attack has higher overhead and complexity.

Our attacks apply only to fragmented traffic. Is there (still) lots of fragmented traffic? There is significant effort to avoid fragmentation; in particular, IPv6 supports only source fragmentation (and avoids fragmentation by routers as in IPv4). This follows the seminal work of Kent and Mogul [11] which mainly presented performance and reliability concerns. However, avoiding fragmentation is not trivial. Indeed, fragmentation is still a widely used, esp. for UDP and tunneled traffic; e.g., see [16]. In fact, source fragmentation is used even in IPv6 (and is vulnerable to our attacks).

CONTRIBUTIONS. We identify critical DoS and exposure vulnerabilities which result from specific flaws in the IPv4 and IPv6 specifications. Specifically, with high probability, we can intercept or block fragmented packets, provided the destination is connected via a NAT device or a tunnel, or when the source uses globally-incrementing IP-ID field (e.g., running any Windows OS). These vulnerabilities are very practical, and were validated on widely-deployed systems in realistic environments, e.g., with concurrent legitimate communication. Vendors were informed.

Our work exploits packet losses and delays as a *side-channel*, allowing powerful attacks on confidentiality, integrity and availability. So far, side-channels were mostly used in cryptographic attacks; we show they can be used for critical network security attacks. We believe that there is a significant potential for other vulnerabilities using such side channels.

ORGANIZATION. Section 2 describes different approaches for choosing IP identifiers, considers practical scenarios where an IP identifier can be easily obtained by non eavesdropping adversaries and presents new related attacks. Most notably, we show a scenario of a NAT connected network that allows a spoofing-only adversary to intercept fragments of a packet. Section 3 presents a technique to obtain the IP identifier between two tunnel gateways and Section 4 presents a complementary denial of service attack as well as presents empirical impact measurements on popular IPsec tunnels. Lastly, Section 5 presents our conclusions.

2 Easy Blind Predictable IP-ID Attacks

The IP identifier field (IP-ID) is used to uniquely identify the set of fragments belonging to a specific packet. The IPv4 standard [15] does not specify how to choose the (16-bit) identifier; two methods appear most common[1], *per-destination IP-ID counter*, where the identifier $i(d)$ of destination d is initialized randomly and incremented whenever sending packet to d (e.g., in Linux), or a *global IP-ID counter*, incremented whenever sending packet to any destination (e.g., in Windows). The IPv6 standard [6] uses a (32-bit) identifier field and specifically recommends using a counter to update it, explicitly allowing either a per-destination or a global IP-ID counter.

An attacker can usually learn the value of a global IP-ID counter simply by receiving some packet from the sender. Receiving such packet is often possible, e.g., as a response to a packet sent by attacker (e.g., SYN to public web server), or by causing the client to open a connection to the attacker (e.g., by image embedded in some web page).

It is sometimes also easy to learn the value of a per-destination IP-ID counter; specifically, when the attacker can receive packets sent to the same IP as the destination, e.g., in scenarios as in Figure 1. Here, the recipient, Bob, and another host who is controlled by the attacker, Zombie, are behind a typical many-to-one Network Address Translator (NAT) device [17] and therefore, have the same Internet IP address. If Alice uses per-destination IP-ID counters, then she uses the same counter for packets sent to Zombie and to Bob. Although the NAT device makes some modifications to the IP header, it normally does not change the IP-ID field. Thus, any destination behind the NAT would receive the IP-ID field exactly as sent by Alice. This allows Zombie to obtain the current value of the identifier and report to the attacker, Mal.

In this section we show how to exploit such 'known IP-ID' scenarios for fragment interception and DoS attacks. The next section deals with the more challenging case of per-destination incrementing IP-ID counter, and avoids the zombie requirement.

We next discuss how Mal can use the exposed identifier. One obvious and trivial use is to dramatically improve the (quite weak) results of the mis-association attack of [9], namely send fragments with the expected identifier and some ('random') data, resulting in failure in the defragmentation process. In the next subsection, we show that the exposed identifier also allows the attacker to intercept (expose) information sent to others.

2.1 Fragment Injection and Interception with Known-Identifier

Consider the network topology illustrated in Figure 1, and assume that Mal obtains the identifier of the next (fragmented) packet that Alice would send Bob, and the IP addresses of the NAT and of Alice. The knowledge of the identifier allows Mal to inject his own data into the reassembled packet. Specifically, Mal sends a forged fragment that overwrites the transport header of a fragmented packet (in reassembly). In case of a NAT destination, the packet is forwarded to a host according to the destination port of the transport layer header. Thus, modifying the port would result in a different destination (behind the NAT). Specifically, by changing the destina-

[1] The trivial, intuitively-appealing use of a randomly-chosen identifier, is not recommended, at least for IPv4; by the birthday paradox, in roughly $\lceil 1.2 \cdot 2^8 \rceil = 307$ packets there will be a repetition, which would cripple performance.

tion port to one the NAT assigns Zombie, Mal is able to obtain Alice's fragments. We now explain how this is done.

Replacing Alice's first fragment entirely, thereby rewriting the transport layer header and changing the destination of following fragments, is relatively easy: say that the size of the IP data in Alice's packet fragments is l bytes except the last fragment which may be shorter.

In the initial step of the attack Mal sends to the NAT a forged fragment that matches Alice's packet four reassembly parameters (source address, destination address, transport protocol, IP-ID); we assume he does this step before Alice sends the corresponding packet. The fragment specifies offset $= l + 1$ and MF $= 0$, i.e., it appears as the second and last fragment of a packet; lower offsets are also suitable as long as the forged fragment does not entirely overlap the first fragment of Alice's packet.

Next, Mal waits for Alice's packet fragments to arrive; we present the usual case where they arrive in order (when they arrive in reverse order, the attack is even simpler!). When Alice's first fragment arrives at the cache, then it is immediately reassembled with the existing forged fragment that Mal had sent and they both are removed from the cache. As a result, the rest of the fragments of the packet will be cached. When Mal assumes that all of Alice's fragments arrived at the NAT, he sends a forged 'first' fragment (i.e., offset $= 0$, MF $= 1$) that specifies a new transport layer header, and in it, a new destination port. This fragment is then reassembled with all other fragments of the packet; thereby, changing the destination of the reassembled packet to another host behind the NAT, e.g., to Zombie.

The first fragment (which Mal will not obtain) often includes critical information, e.g., cookies, user name or other credentials. In the full version [7] we present a simple variant of the attack, that allows to capture the first fragment as well.

2.1.1 Empirical Validation

We implemented the fragment interception attack, and tested it on two scenarios using the popular IP tables NAT; the scenarios differed in the number of forged fragments cached at the NAT at the same time (in Linux, depending whether *ipfrag_max_dist* enforces such limitation, see [12]). The network topology for the experiment was as in Figure 1, the LAN consisted of one switch and Mal was connected directly to the 'Internet' interface of the NAT. Our results, illustrated in Figure 3, show that Mal is able to intercept a significant amount of traffic, even with relatively low bandwidth. Furthermore, some traffic not intercepted was denied; only less than 5% of Alice's packets actually reached Bob's application layer in most measurements.

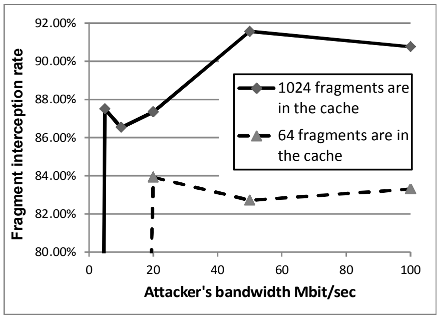

Figure 3: Percent of packets sent to Bob whose first fragment reached Zombie as a function of Mal's bandwidth. All bandwidths except Mal's are 100Mbit/sec, Alice sends to Bob one packet every 5 milliseconds. More than 95% of the traffic did not reach Bob (not shown).

3 IP-ID Exposing Attack

In this section we present an attack that exposes the IP-ID, when the simple techniques mentioned in the previous section fail, i.e., where the sender uses per-destination incrementing identifiers, e.g., using Linux, and where the attacker cannot receive the identifier directly (no Zombie behind NAT). We design a more elaborate attack, that exposes the identifier, for common network scenarios, even in this case.

Our attack requires that the attacker, Mal, would receive some feedback on lost packets which allows him to efficiently learn the identifier. To provide this feedback, the attack requires an adversarial agent PuZo; in contrast to the previous section which required a zombie, PuZo can be merely a *puppet* [4], i.e., malicious code running in sandbox (e.g., script, applet). Furthermore, while the attack works in the NAT scenario as in Figure 1, like the attack in the previous section, the new attack also works in two additional common scenarios: when the puppet runs on Bob's machine, and when Alice and Bob are connected via a tunnel as in Figure 2.

For simplicity, we now describe only the tunneling scenario, and furthermore assume that PuZo is a zombie (not a puppet), and that there is no legitimate traffic during the attack, and no (benign) packet losses. In Subsection 3.3 below, we show how to modify the attack such that PuZo may be just puppet; and in Section 4 we show a simple, effective method to continually expose the ID and drop most packets (once we have the initial IP-ID using attack in this section). In the full version [7] we present further extensions to allow traffic also during ID-exposing, and to deal with benign packet loss.

We note that the attack presented in Section 2 for networks behind a NAT (see Figure 1) will not succeed in the tunnel scenario: PuZo in Figure 2 may be able to obtain a packet from Alice, but, even if he is a zombie, he will not be able to observe the identifier of encapsulated (i.e., Internet) traffic between the two gateways since such traffic carries a different IP header.

We present two versions of the ID-Exposing attack. In Subsection 3.1, we show a naive and rather inefficient attack that requires a total of $O(n)$ packets, where n is the number of possible identifiers ($n = 2^{16}$ for IPv4). In Subsection 3.2, we refine the ID-exposing attack using a meet in the middle technique, sending only $O(\sqrt{n})$ packets; this also makes ID-exposing feasible for IPv6, where the identifier field is 32 bits long.

The ID-exposing attack is based on the following observation: if Mal sends to $\mathsf{GW_B}$ (see Figure 2) a forged fragment with source IP of $\mathsf{GW_A}$ and IP-ID i, then PuZo will not receive legitimate (fragmented) packets sent via the tunnel (i.e., from $\mathsf{GW_A}$ to $\mathsf{GW_B}$), if they use the same IP-ID i. This happens since Mal's fragment will be mis-associated with the legitimate fragments and produce a corrupted packet, which will be discarded at the gateway ($\mathsf{GW_B}$) upon decapsulation.

We assume that PuZo is able to send requests and receive responses from a server, S, at the other side of the tunnel. We assume that S's response packets are long, and fragmented after encapsulation, i.e., between $\mathsf{GW_A}$ and $\mathsf{GW_B}$. Furthermore, we assume that PuZo is able to identify the order by which the packets that he receives were sent (by S), and a packet loss (when it occurs). There are a number of methods for PuZo to conduct the these tasks: sequential IP identifiers, TCP sequence numbers and possibly application layer data. For simplicity, in following discussions we assume that PuZo uses per-destination sequential IP identifiers for this purpose; in Subsection 3.3 we present a more elaborate technique for the case that PuZo is merely a puppet (and therefore, can only use application layer data).

3.1 Naive, inefficient ID-Exposing Attack

Let n be the number of possible identifiers (i.e., 2^{16} for IPv4, 2^{32} for IPv6). The basic technique we present obtains the identifier by the following attack, illustrated in Figure 4, where the adversary, Mal, caches-in one forged fragment at $\mathsf{GW_B}$ that PuZo 'looks for' by receiving $O(n)$ packets from the other end of the network (e.g., from S), until he identifies that one of them was lost (mis-associated with Mal's forged fragment).

The attack begins by an initialization phase (step 1 in Figure 4) where Mal clears existing fragments from the victim gateway ($\mathsf{GW_B}$) fragment cache. During the initiation phase, Mal sends large spoofed fragments, that specify a source different than $\mathsf{GW_A}$. These suppress the current fragments in the table. Under default Linux configuration (kernel version 2.6.39) the initiation phase requires Mal to send 256KB of data. This technique is similar to the fragment cache overflow attack presented in [10]. We denote this step by Clean-Frag-Cache and will also use it later in Subsection 3.2.

When the initiation phase completes, Mal sends one small forged fragment of an encapsulated packet to PuZo's gateway, $\mathsf{GW_B}$ in Figure 2, where it is cached until reassembly (or timeout which is 30 seconds by default for Linux machines). The fragment specifies $\mathsf{GW_A}$ as source (i.e., is spoofed), $\text{offset} = \text{MTU}, \text{MF} = 0, \text{DF} = 0$ with an arbitrary IP-ID value, i (in step 2 of Figure 4, $i = 2222$).

Thereafter, Mal orders PuZo to send a request to a server, S, at other end of the tunnel, for some data, e.g., a large file (step 3 in Figure 4), persuading S to send at least n packets. S's packets, before encapsulation, have sequential IP identifiers (since they are all sent to the same entity, Zombie), $1111, 1112, 1113, \ldots$ in Figure 2. The server's packets reach its gateway, $\mathsf{GW_A}$, where they are encapsulated. Upon encapsulation, the gateway attaches a new IP header that specifies $\mathsf{GW_A}$ and $\mathsf{GW_B}$ as source and destination (i.e., gateway to gateway tunnel), and includes a new IP identifier. Since all the encapsulated packets are sent from $\mathsf{GW_A}$ to $\mathsf{GW_B}$ (i.e., same source and destination), they too have sequential identifiers, $2221, 2222, 2223, \ldots$. We assume that the encapsulated packets sent to PuZo exceed the tunnel MTU and are fragmented after encapsulation (see Figure 4).

Under the assumption that no other traffic takes place, and no loss on network channels, exactly one packet of the first n packets that were sent to PuZo will be lost, since it will be mis-associated with Mal's (forged) fragment. PuZo receives all other packets, which specify sequential IP identifiers (that differ from the identifiers of encapsulated packets). Thus, PuZo is able to detect x, the IP identifier of the packet that was lost.

Let $\tilde{x}$ be the IP identifier of the last packet that PuZo had received. PuZo computes $\tilde{i} = \tilde{x} - x \pmod{n}$, the total number of packets that where sent from the server to PuZo after the lost packet. PuZo then sends $\tilde{i}$ back to Mal (step 4 in Figure 4) who concludes that the next identifier is $i + \tilde{i} + 1 \pmod{n}$.

3.2 Meet in the Middle, ID-Exposing Attack

We now revise the attack presented in the previous Subsection to generate only $O(\sqrt{n})$ packets. The revised version of the ID-exposing attack is composed of two phases: first, a meet in the middle attack that narrows down the number of possible IP identifiers to $\sqrt{n}$. Sec-

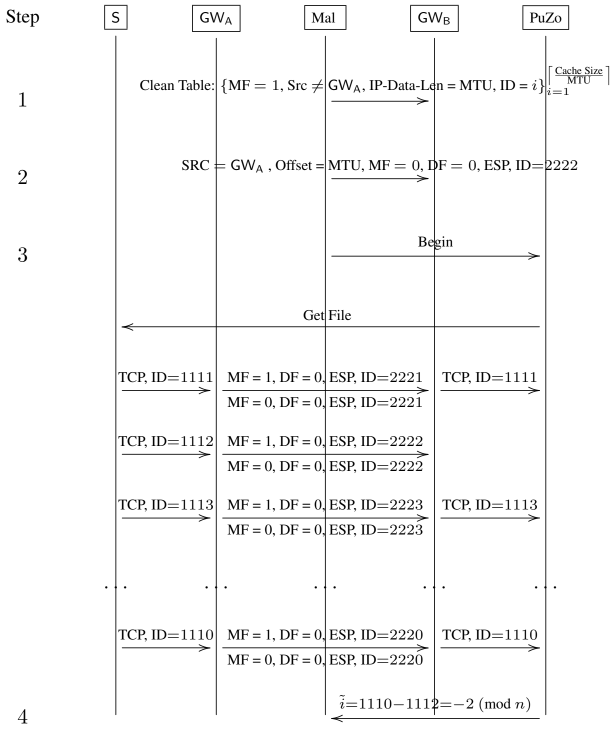
Step
S
GW_A
Mal
GW_B
PuZo
1
Clean Table: {MF = 1, Src ≠ GW_A, IP-Data-Len = MTU, ID = i} i=1 ⌈Cache Size / MTU⌉
2
SRC = GW_A , Offset = MTU, MF = 0, DF = 0, ESP, ID=2222
3
Begin
Get File
TCP, ID=1111
MF = 1, DF = 0, ESP, ID=2221
MF = 0, DF = 0, ESP, ID=2221
TCP, ID=1111
TCP, ID=1112
MF = 1, DF = 0, ESP, ID=2222
MF = 0, DF = 0, ESP, ID=2222
TCP, ID=1113
MF = 1, DF = 0, ESP, ID=2223
MF = 0, DF = 0, ESP, ID=2223
TCP, ID=1113
TCP, ID=1110
MF = 1, DF = 0, ESP, ID=2220
MF = 0, DF = 0, ESP, ID=2220
TCP, ID=1110
ĩ=1110−1112=−2 (mod n)
4

Figure 4: ID-exposing attack in $O(n)$ packets. Notice that since S sends n packets to PuZo, the identifier (counter) wraps around. Assume no traffic is sent between other entities and no network loss/corruption while the attack takes place (see our technical report [7] for extensions to handle these issues). MTU marks the maximal packet length that Mal can send to $\mathsf{GW_B}$ in a single packet. ESP is an example of a tunneling protocol (i.e., IPsec).

ond, an exhaustive search over the remaining possible identifiers to discover the correct one.

In the meet in the middle phase, illustrated in Figure 5, Mal sends $\sqrt{n}$ fragments. Each fragment specifies an identifier that is $\sqrt{n}$ apart from the previous one [2]. For simplicity, we start at identifier 0, i.e., send identifiers $0, \sqrt{n}, \ldots, (\sqrt{n}-1)\sqrt{n}$. In this version of the attack, the server S only sends PuZo $\sqrt{n}$ packets. As in the previous version, one of these packets would specify an IP identifier that was also sent by Mal and therefore, will not reach PuZo. PuZo will then compute $\tilde{i}$ and send it to Mal. Since Mal had sent $\sqrt{n}$ forged fragments (i.e., $\sqrt{n}$ 'traps'), at the end of the meet in the middle phase he obtains a list of $\sqrt{n}$ possible identifiers. Algorithm 1 (in Appendix A) summarizes Mal's logic above.

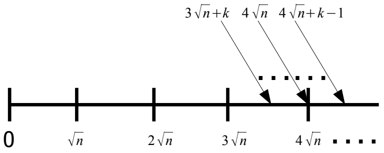

Figure 5: Meet in the Middle ID-Exposing Attack. Mal sends $\sqrt{n}$ forged fragments that are $\sqrt{n}$ apart; thereby, laying $\sqrt{n}$ 'traps'. In this example, when the attack begins the current identifier value $\mathsf{GW_A}$ uses for packets to $\mathsf{GW_B}$ is $3\sqrt{n}+k$ for some $k<\sqrt{n}$. During the meet in the middle phase the identifier is incremented $\sqrt{n}$ times, once for every packet sent to PuZo. When $\mathsf{GW_A}$ encapsulates a packet for PuZo with an identifier $4\sqrt{n}$, a multiple of $\sqrt{n}$, it will be lost and PuZo will detect this event.

Next begins the exhaustive search phase, where Mal searches for the correct identifier in a divide and conquer methodology. Namely, at round r, Mal sends $\frac{\sqrt{n}}{2^r}$ forged fragments that match $\frac{\sqrt{n}}{2^r}$ possible identifiers. These fragments are saved at the recipient's fragments cache ($\mathsf{GW_B}$ in Figure 2). Thereafter, PuZo sends a request to S who sends (at least) one packet in response, which is fragmented after encapsulation. If the packet does not reach PuZo, then the current identifier is one of those sent by Mal at this round of the search. Otherwise, it is one of the other identifiers. Mal continues this process, that eliminates half the possible identifiers in each round, until he obtains the current identifier. Algorithm 2 summarizes the exhaustive search phase.

Before the meet in the middle phase and any round of the exhaustive search phase, Mal cleans the victim gateway's ($\mathsf{GW_B}$) cache, as described in the initiation phase of the previous version of the attack (see Section 3.1).

3.2.1 Empirical Validation

We tested the meet in the middle ID-exposing technique on two widely used Linux IPsec gateways that use fragmentation to cope with oversized encapsulated packets. See discussion of the implementations and setup in Section 4.2. We conducted 100 iterations of the ID-exposing attack for IPv4, and obtained the current identifier in each iteration within less than 20 seconds.

3.3 IP-ID Exposing using Puppet

We now briefly outline how the attack can be modified, s.t. PuZo may be a puppet, i.e., code running within sandbox [4]. The challenge is that a puppet PuZo needs to identify the offset of a lost packet, but can only use

[2] In recent IPv4 Linux stacks, Mal may only be able to cache in up to *ip_frag_max_dist* different forged fragments at a time (see [12]), under default settings this value is 64. In this case the difference between two sequential identifiers that Mal sends is $\frac{2^{16}}{64}=1024$, which is also the number of packets S sends to PuZo. No such limitation exists for IPv6.

TCP socket services, and in particular PuZo cannot read data from TCP and IP headers.

The solution is quite simple: first, Mal sends his forged fragment or fragments to $\mathsf{GW_A}$, S's gateway (cf. to PuZo's gateway). Second, PuZo opens multiple *separate* connections to S, and on each of these connections he sends a long request, such that the request itself will be fragmented after encapsulation. A request that has the same IP-ID as of Mal's fragments will be discarded and therefore, PuZo will not receive a response in that connection. Say that the requests that PuZo sends are numbered $1, \ldots, x$, and $\tilde{x}$ is the index of the lost request, PuZo's feedback to Mal is $\tilde{i} = x - \tilde{x}$, and the attack continues as before.

4 Continual Deny and Expose Attack

Section 3 introduced a method to obtain the current identifier attached to encapsulated traffic by a (Linux) tunnel gateway, $\mathsf{GW_A}$ (see Figure 2). Trivially, given the identifier, it is possible to execute a fragment mis-association attack (see [9]) to deny fragmented traffic from $\mathsf{GW_A}$ to $\mathsf{GW_B}$, the two tunnel gateways. Most notably, this includes fragmented encapsulated traffic from $\mathsf{LAN_A}$ to $\mathsf{LAN_B}$. However, it is difficult for Mal to maintain long term synchronization with the current value of the identifier since it is incremented for every packet sent from $\mathsf{GW_A}$ to $\mathsf{GW_B}$.

In this section we present an attack that causes $\mathsf{GW_B}$ to discard fragmented traffic sent from $\mathsf{GW_A}$ given an initial value for the corresponding IP identifier; we assume that $\mathsf{GW_A}$ uses a per destination counter, e.g., a Linux machine. We also assume the topology in Figure 2, and the existence of a zombie/puppet PuZo behind $\mathsf{GW_B}$; for simplicity, our description will assume zombie. We present empirical measurements of our attack on two extensively deployed (commercial and open source) Linux based IPsec implementations that fragment encapsulated traffic.

4.1 Attack Process

Mal begins the continual deny and expose attack after the ID-exposing process (see Section 3) where he obtained i, the IP identifier of the next packet sent by $\mathsf{GW_A}$ to PuZo's gateway, $\mathsf{GW_B}$. We use the identifier's counter property to maintain a small interval of possible IP-ID values at any given time.

During the attack, Mal always keeps two sequences of consecutive identifiers cached at $\mathsf{GW_B}$, between these sequences a small gap of identifiers that Mal did not send, see Figure 6. Packets sent from Alice to Bob, two hosts in Figure 2, that arrive fragmented at $\mathsf{GW_B}$ for decapsulation would reach Bob only if they specify an IP identifier not within one of Mal's sequences. The small gap between the two sequences allows PuZo to monitor the progress of the identifier value by testing whether he can receive (fragmented) packets via the tunnel. Furthermore, the devision into two sequences provides Mal early notifications to update his cached fragments. Namely, Mal has time to send a new sequence of fragments until the second sequence, already cached at $\mathsf{GW_B}$, becomes obsolete. These two characteristics allow even attackers with relatively small bandwidths to cause high loss rates for fragmented traffic, as we validate by experiments described in the next subsection.

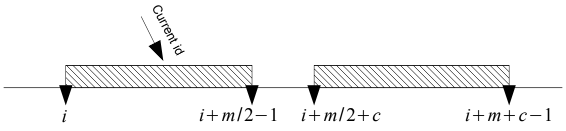

Figure 6: Abstract view of $\mathsf{GW_B}$'s cache during the attack. Mal keeps two sequences of forged fragments cached in. A fragmented packet with IP-ID j may only arrive at its destination if: (1) j is within the gap; or (2) $j < i \lor j \geq i + m + c$. In the former case Mal sends the next sequence (which replaces the first one), in the latter case, synchronization with the current identifier is lost.

The attack includes an initialization phase, where Mal sends PuZo's gateway, $\mathsf{GW_B}$, the first two sequences of forged fragments. Let m be the maximal number of fragmented packets from a single source address that may be cached simultaneously at $\mathsf{GW_B}$. m is usually either 64, in case a specific limitation exists (via *ipfrag_max_dist*, see [12]) or several thousand fragments otherwise.

Each of the two sequences that Mal sends is composed of $\frac{m}{2}$ (spoofed) fragments with consecutive identifiers. The fragments in the first and second sequences specify the identifiers within the intervals $[i, i + \frac{m}{2} - 1]$, $[i + \frac{m}{2} + c, i + m + c - 1]$ respectively, where c is some small positive integer. Namely, between the two sequences there is a small gap of c identifiers that were not sent by Mal, see Figure 6. Thereafter, Mal updates: $i \leftarrow i + m + 2c$, and orders PuZo to begin his role in the attack. This completes the initiation phase and begins the attack itself.

When PuZo receives Mal's message, he begins sending requests to a server, S, at the other end of the tunnel (see Figure 2), who sends in response a single or few long packets which will be fragmented after encapsulation. This process is similar to the one described in the ID-exposing attack in Section 3. PuZo sends such requests periodically every τ seconds (see Algorithm 3). The length of the gap, c, must be sufficiently large to allow PuZo to receive a packet given the request interval τ. However, c should be small such that only few legitimate packets will pass through during the ‘in-gap’ period.

When one of S’s packets reaches PuZo, he concludes that the current identifier is within the gap between the two sequences, and notifies Mal, who then sends the next sequence of identifiers (see Algorithm 3); i.e., identifiers in the interval $[i, i + \frac{m}{2} - 1]$, and updates the current index: $i \leftarrow i + \frac{m}{2} + c$. The new sequence of fragments that Mal sends supersedes the first of the two sequences that he had previously sent and are currently cached at $\mathsf{GW_B}$ since the LRU cache management paradigm (see [5]).

When the identifier is within the gap, legitimate traffic is not disrupted. Thus, PuZo makes frequent requests and generates response traffic from S to himself (without waiting between requests). Each response packet advances the identifier counter until eventually it is after the gap. At this time the identifier within the tunnel equals to that of a forged fragment within the second sequence cached by Mal in $\mathsf{GW_B}$. PuZo identifies this event when a response packet is lost, i.e., assumes that mis-association had occurred and goes back to sending a request every τ seconds [3]. Algorithms 3, 4 summarize Mal and PuZo’s logic above.

4.2 Empirical Evaluation

We tested the continual deny and expose attack on two of the most popular, well-known IPsec implementations: an open-source implementation and a commercial implementation [4]. Both implementations cope with oversize encapsulated packets by IP fragmentation, and are vulnerable to our attacks.

We performed the continual deny and expose attack on both implementations and measured its impact on TCP and UDP traffic as a function of Mal’s bandwidth. Figure 7 illustrates the loss rate of packets sent from Alice to Bob (see network topology in Figure 2). For most attacker bandwidths, TCP data communication was completely blocked; thus, Figure 7 omits these results. The figure shows that with reasonable bandwidths, Mal is able to cause significant packet loss. Not surprisingly, the lesser the number of forged fragments that Mal can cache in, the more (legitimate) packets that reach Bob and the greater the bandwidth Mal needs in order to perform the attack (compare consecutive and dashed lines). The dashed lines in Figure 7 also indicate that for particularly low bandwidths Mal cannot keep the attack ‘alive’, and quickly looses synchronization with the current IP identifier value within the tunnel.

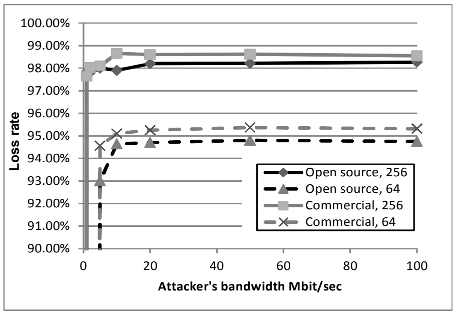

Figure 7: The loss rate of packets sent from Alice to Bob as a function of Mal’s bandwidth. All bandwidths except Mal’s are 100Mbit/sec. Each measurement is the average of 5 iterations of a 5 minute attack. Consecutive lines mark a scenario where Mal is able to cache in (at $\mathsf{GW_B}$) 256 forged fragments at once, while the dashed lines mark a scenario where Mal is only able to cache in 64 such fragments (the default Linux limitation). The length of the gap between two consecutive sequences that Mal sends (i.e., c), is 4. PuZo receives a packet from S every 10 milliseconds, and Alice sends Bob a packet every 5 milliseconds.

5 Conclusions

We presented critical attacks against both IPv4 and IPv6. The attacks can be deployed by a blind (spoofing) attacker, and result in packet interception, modification and loss. Our main conclusion is the need to improve the specifications and validation of common networking protocols such as IP, following (and motivating) [8]. As a more immediate conclusion, we believe implementations of IPv4, IPv6, IPsec and other tunnels, should be carefully tested against the vulnerabilities described within and in particular modify their ‘IP-ID choosing’ paradigm. Furthermore, it is advisable that erratas be issued for the relevant specifications, esp. considering that recently, with the adoption of mobile TCP/IP devices, there may be many new implementations. Finally, since many implementations may be impractical to fix in timely fashion, appropriate defenses should be added to firewalls and IDS/IPS devices.

Acknowledgments

This work is based on preliminary research by Roman Yakirevich, who identified (other) problems in handling of fragmentation in TCP and IPsec implementations; many thanks to Roman for his assistance, also in helping to test our attacks. Amit Klein and Charlie Kaufman gave us invaluable comments and encouragement; many thanks! Thanks also to the anonymous referees for their comments and suggestions.

[3] PuZo also quits the ‘frequent request’ mode when he assumes that he had caused S to send c or more packets since sending the ‘responded’ request.

[4] Product names are hidden to allow vendors time to patch, these details are available from authors.

References

[1] Teardrop DoS Attack, CERT advisory. http://www.cert.org/advisories/CA-1997-28.html, 1997.

[2] The Rose Attack Explained. http://digital.net/~egandalf/Rose_Frag_Attack_Explained.htm, 1997.

[3] Jason Anderson. An analysis of fragmentation attacks. Unpublished manuscript, available at http://www.ouah.org/fragma.html, March 2001.

[4] Spyros Antonatos, Periklis Akritidis, Vinh the Lam, and Kostas G. Anagnostakis. Puppetnets: Misusing web browsers as a distributed attack infrastructure. *ACM Trans. Inf. Syst. Secur*, 12(2), 2008.

[5] Christian Benvenuti. *Understanding Linux Network Internals*. O'Reilly & Associates, Inc., 2006.

[6] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard), December 1998.

[7] Yossi Gilad and Amir Herzberg. Fragmentation Considered Vulnerable - Technical Report. http://www.cs.biu.ac.il/~herzbea/security/TR/11_1.pdf, 2011.

[8] Fernando Gont. Security Assessment of the Internet Protocol version 4. Internet draft, April 2011.

[9] J. Heffner, M. Mathis, and B. Chandler. IPv4 Reassembly Errors at High Data Rates. RFC 4963 (Informational), July 2007.

[10] Charlie Kaufman, Radia Perlman, and Bill Sommerfeld. DoS protection for UDP-based protocols. In Vijay Atluri and Peng Liu, editors, *Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS-03)*, pages 2–7, New York, October 27–30 2003. ACM Press.

[11] Christopher A. Kent and Jeffrey C. Mogul. Fragmentation considered harmful. Research Report 87/3, Western Research Lab, December 1987. An abbreviated version was published in the proceedings of the ACM SIGCOMM, 390–401, 1987.

[12] Kernel.org. Linux Kernel Documentation. http://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt, 2011.

[13] Ed M. Kenney. Ping o' death. http://www.insecure.org/sploits/ping-o-death.html, 1996.

[14] I. Miller. Protection Against a Variant of the Tiny Fragment Attack (RFC 1858). RFC 3128 (Informational), June 2001.

[15] J. Postel. Internet Protocol. RFC 791 (Standard), September 1981.

[16] Colleen Shannon, David Moore, and K. C. Claffy. Beyond folklore: observations on fragmented traffic. *IEEE/ACM Transactions on Networking*, 10(6):709–720, December 2002.

[17] P. Srisuresh and K. Egevang. Traditional IP Network Address Translator (Traditional NAT). RFC 3022 (Informational), January 2001.

[18] Michal Zalewski. A new TCP/IP blind data injection technique? BugTraq mailing list post, http://lcamtuf.coredump.cx/ipfrag.txt, 2003.

[19] G. Ziemba, D. Reed, and P. Traina. Security Considerations for IP Fragment Filtering. RFC 1858 (Informational), October 1995.

A Attack Algorithms

```
Input: $n$ - the number of possible IP identifiers.
Output: $A$ - an array of $\sqrt{n}$ possible values of the
        current identifier.
Clean-Frag-Cache();
for $i = 0$ to $\sqrt{n} - 1$ do
    Send-Forged-Fragment($i\sqrt{n}$);
end
/* PuZo requests $\sqrt{n}$ packets from S
   and identifies the number of
   packets sent after a loss ($\tilde{i}$)  */
Order-PuZo('ID-Exposing, meet in the middle');
$\tilde{i} \leftarrow$ Get-PuZo-Response();
for $i = 0$ to $\sqrt{n} - 1$ do
    $A[i] \leftarrow i\sqrt{n} + \tilde{i} + 1 \pmod{n}$;
end
return $A$;
```

Algorithm 1: Meet in the Middle Phase of the ID-exposing Attack (Section 3.2): send $O(\sqrt{n})$ fragments, obtain a list of $\sqrt{n}$ possible current identifiers of encapsulated packets sent from S's network (LAN_A in Figure 2) to PuZo's network (LAN_B in Figure 2).

```
Input: $A$ - an array of $\sqrt{n}$ possible values of the
       current identifier.
Output: the next identifier.
$b \leftarrow 0, e \leftarrow \sqrt{n} - 1$;
while $b \neq e$ do
    Clean-Victim-Frag-Cache();
    for $i = b$ to $\lfloor \frac{e}{2} \rfloor$ do
        Send-Forged-Fragment($A[i]$);
    end
    Order-PuZo('ID-Exposing, exhaustive search');
    packetReceived $\leftarrow$ Get-PuZo-Response();
    if packetReceived = True then
        $e \leftarrow \lfloor \frac{e}{2} \rfloor$;
    end
    else
        $b \leftarrow \lfloor \frac{e}{2} \rfloor$;
    end
    /* Account for the packet S sent
       PuZo.                          */
    Inc($A$);
end
return id $\leftarrow A[b]$;
```

Algorithm 2: Exhaustive Search Phase of ID-exposing Attack (Section 3.2): given a list of possible identifiers conducts an exhaustive search to obtain the correct one. The function 'Inc' increases by 1 every element of its array parameter.

```
Input: $i$ - next identifier specified in a packet sent
       from $\text{GW}_\text{A}$ to $\text{GW}_\text{B}$. $m$ - maximal number of
       IP fragments that specify $\text{GW}_\text{A}$'s source
       address and can be kept simultaneously in
       $\text{GW}_\text{B}$'s cache. $c$ - small integer that
       represents the identifier gap size between
       sequences that Mal sends.
for $j = 1$ to $2$ do
    Send-Forged-Sequence($i$, $\frac{m}{2}$);
    $i \leftarrow i + \frac{m}{2} + c$;
end
Order-PuZo('DoS');
while True do
    Wait-For-PuZo-Notification();
    Send-Forged-Sequence($i$, $\frac{m}{2}$);
    $i \leftarrow i + \frac{m}{2} + c$;
end
```

Algorithm 3: Mal's attack logic for 'continual deny and expose' as presented in Section 4.1. Denies fragmented encapsulated traffic from reaching its destination, a host behind PuZo's gateway.

```
Input: $\tau$ - time in seconds between requests.
Wait-For-DoS-Request();
while True do
    repeat
        Send-Request();
        Sleep($\tau$);
    until Response-Arrives;
    Notify-Mal();
    repeat
        Send-Request();
    until Response-Does-Not-Arrive;
end
```

Algorithm 4: PuZo's attack logic for 'continual deny and expose' as presented in Section 4.1. PuZo requests for packets and notifies Mal when a packet (response) is received after each series of packet loss.