# Overlapping data in network protocols: bridging OS and NIDS reassembly gap

Lucas Aubard[1], Johan Mazel[2], Gilles Guette[3], and Pierre Chifflier[2]

[1] Inria, Rennes, France `lucas.aubard@inria.fr`
[2] ANSSI, Paris,France `firstname.lastname@ssi.gouv.fr`
[3] IMT Atlantique, Cesson Sévigné, France `gilles.guette@imt-atlantique.fr`

**Abstract.** IPv4, IPv6, and TCP have a common mechanism allowing one to split an original data packet into several chunks. Such chunked packets may have overlapping data portions and, OS network stack implementations may reassemble these overlaps differently. A Network Intrusion Detection System (NIDS) that tries to reassemble a given flow data has to use the same reassembly policy as the monitored host OS; otherwise, the NIDS or the host may be subject to attack. In this paper, we provide several contributions that enable us to analyze NIDS resistance to overlapping data chunks-based attacks. First, we extend state-of-the-art *insertion* and *evasion* attack characterizations to address their limitations in an overlap-based context. Second, we propose a new way to model overlap types using Allen's interval algebra, a spatio-temporal reasoning. This new modeling allows us to formalize overlap test cases, which ensures exhaustiveness in overlap coverage and eases the reasoning about and use of reassembly policies. Third, we analyze the reassembly behavior of several OSes and NIDSes when processing the modeled overlap test cases. We show that 1) OS reassembly policies evolve over time and 2) all the tested NIDSes are (still) vulnerable to overlap-based evasion and insertion attacks.

**Keywords:** Intrusion detection system · IP · TCP · Overlapping data

## 1 Introduction

Some Internet protocols use chunking[4] mechanism. It was introduced to answer a potential discrepancy between media link capacity at a node's entry and a node's exit along the path between a sender and a receiver. When chunking occurs, the receiver must reassemble all the chunks to retrieve the initial data packet. However, the chunking mechanism can lead to overlaps. The most common case is a chunk retransmission with the same data that starts and finishes at the same byte offsets. Nevertheless, other types of overlaps exist, i.e., partial overlaps, and the data can be different on the overlapping portion. IPv4 and TCP RFC

---

[4] We use the term chunking as a generic way to refer to "splitting an original data chunk into several". Thus, it both refers to the "fragmentation" mechanism for IPv4 and IPv6 and the "segmentation" mechanism for TCP.

specifications [8, 15] neither forbid data overlaps nor specify the behavior an implementation must adopt (e.g., prefer data from the older chunk). IPv6 RFC specification initially did not forbid overlaps in the first drafts, but since 2017, it has banned them [13].

Network packet analysis is one of the possible techniques commonly used to detect intrusions. Some widely deployed Network Intrusion Detection Systems (NIDS) are signature-based, meaning that they match suspicious patterns or signatures of known attacks on the reassembled flow data. Therefore, NIDSes must reassemble consistently the network traffic with the monitored hosts to detect attacks. Ptacek and Newsham [28] introduced a set of IP and TCP ambiguities that may lead to NIDS misassemblies with supervised hosts and thus, NIDS circumvention. The ambiguities exist because 1) NIDSes receive a copy of the network traffic from and to the hosts, and 2) NIDSes and monitored hosts are distinct machines. Thus, NIDSes cannot easily determine how a host processes a specific packet when data overlap occurs.

Figure 1 is an illustration of the data overlap issue. The reassembled data here differs depending on which chunk the reassembly policy favors. Someone with bad intentions may exploit the multiple reassembly possibilities to hide a malicious payload. If the NIDS reassembles with reassembly strategy 1 while the host reassembles with strategy 2, the former cannot see the malicious payload and raise any security alert.
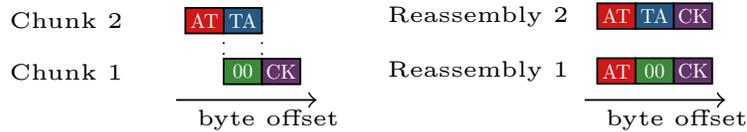


**Fig. 1.** Data overlap ambiguity illustration.

Several works showed that reassemblies depend on the IPv4 [20, 25, 28, 31], IPv6 [10, 14, 22] and TCP [20, 26, 28, 31] implementations. Since attackers may use the overlapping ambiguity to bypass their security functionalities, well-known NIDSes like Suricata [6] and Snort [29] introduced a feature allowing users to associate each supervised host (through IP address) to a specific reassembly policy [4,5,7]. Other NIDSes like Zeek (formerly Bro [27]) have chosen a different approach: implementing only one reassembly policy but allowing users to enable overlap-related alerts. The set of implemented IPv4 and TCP reassembly policies in Suricata and Snort are based on the works of Novak and Sturges [25, 26] published in 2005 and 2007. These works are the latest that tested OS policies for IPv4 and TCP. Thus, we identify two main problems. The first one is that the OSes Novak and Sturges tested have since been updated, and their protocol implementation may have changed. The second problem arises from the manual approach the related works [10, 14, 25, 26, 28, 31] used to design their test cases. There is thus no certainty on test case coverage exhaustiveness. Reassembly

policies implemented within Suricata and Snort may be based on out-of-date and/or partial policy descriptions, giving a wrong sentiment of security regarding overlap-based attacks. So, the knowledge of modern OS reassembly policies must be updated (see Section 4.1), and the exhaustive coverage of overlap test cases must be ensured (see Sections 3.1 and 5).

After Ptacek and Newsham's seminal work [28], an entire research area has focused on finding any sequence of packets (i.e., not only based on the overlap ambiguity) that protocol implementations process differently. Research has especially intensified from the growing deployment of censorship systems (CS) since the technique has been found successful in their circumvention [11, 12, 20, 21, 33, 34]. The methods used to find such sequences have moved from manual [20, 21, 33] (in which authors have to discover the ambiguities all by themselves) to semi-automatic ones using fuzzing [11, 37, 38] or symbolic execution [34, 35]. Until 2020, some works [20, 21, 33, 34] reported the (more and more) relative success of overlapping chunk-based attacks. We argue that the recent works using semi-automatic approaches did not perform extensive (i.e., complete) testing using overlap-based strategies, mainly because fuzzing and symbolic execution methods are good at finding novel chunk sequence examples but not at exhaustive testing.

To our knowledge, no work has verified that NIDSes' overlap reassembly policies are consistent with OSes' since Ptacek and Newsham unveiled the issue in 1998. We fill that gap in this paper. The question that will guide us throughout is: *Do NIDSes reassemble overlap test cases differently as the OSes, and therefore, is it possible to use data overlaps to attack a NIDS or the hosts it supervises?* The contributions are the following:

– In Section 2, we extend *insertion* and *evasion* definitions to overlap-based attack context. This enables us to cover all the related attack scenarii and to characterize the requirements for the overlapping data portion.
– In Section 3.1, we model chunk sequences with Allen's spatio-temporal reasoning [9] and, thus, ensure overlap coverage exhaustiveness (in contrast to 10 over 13 related-works, see Section 5).
– In Section 4, we describe IPv4, IPv6, and TCP reassembly policies of a large range of OSes, including recent ones (e.g., Windows 11, the Linux-based Debian 12, FreeBSD 14.1, OpenBSD 7.6, Solaris 11.4) and of three widely deployed NIDSes (i.e., Snort, Suricata and Zeek). We find that:

  i) OS reassembly policies evolve over time. In particular, Windows and Linux-based OSes have changed their IPv4 reassembly policies (regarding state-of-the-art), while TCP policies have barely been modified.
  ii) Snort, Suricata, and Zeek reassemble IPv4, IPv6, and TCP chunks in a partially consistent way with OSes. This opens the way to insertion and evasion attacks. A CVE [23] was assigned to some of the disclosed problems.

## 2    Problem Definition and Threat Model

This section first defines insertions and evasions in the specific context of overlap-based attacks. It then details the attacker capabilities needed to exploit OS and NIDS reassembly discrepancies.

### 2.1    Problem Definition

An attacker may use overlap data ambiguity to exploit NIDS and host reassembly divergence. We extend Ptacek and Newsham [28] and Wang et al. [34] *insertion* and *evasion* packet-based attack definitions to fit the context of IP and TCP chunk overlaps. The main shortcomings that we address are:

- Ptacek and Newsham consider a data chunk as being either *totally* accepted or dropped but not *partially* accepted. With the overlap from Figure 1, the "ATTACK"/"AT00CK" reassembly divergence is impossible.
- Wang et al. do not consider on purpose malicious (or "filtered") payload insertion inside the NIDS flow data, nor do they consider IP-based attacks.

We treat the NIDSes and the host OSes as black boxes in the following. We consider the IP and TCP data stream pushed to the upper layer; thus, our definitions apply to IP and TCP overlap-based attacks.

Let $P = \{$IPv4, IPv6, TCP$\}$ be the Internet protocol set with a chunking mechanism we target. Let $C^p$ be the set of all possible chunks for protocol $p \in P$.

**Definition 1 ($p$ protocol data buffer synchronization).** *Given a chunk sequence $c_f...c_l \in C^p$, with $c_f$ (resp. $c_l$) the first (resp. last) sent chunk, we say that the NIDS and the supervised host have their $p$ data buffers synchronized if the next upper-layer data streams are the same.*

The $p$'s data buffer desynchronization requires chunks linked with a particular overlap type that the NIDS and the host reassemble differently (see Section 4) and carefully crafted overlapping data (see Section 2.2). Such data is either an *evasion* or an *insertion* payload depending on the target (i.e., the host or the NIDS). *Evasion and insertion attacks aim to desynchronize NIDS and host $p$ protocol data buffers.*

**Definition 2 (Evasion in a data overlap-based context).** *An evasion attack consists of* some malicious payload *that is not visible in the data analyzed by the NIDS while it is visible on the* supervised host *'s reassembled payload.*

**Definition 3 (Insertion in a data overlap-based context).** *Symmetrically, an insertion attack consists of* some malicious payload *that is visible in the flow data analyzed by the* NIDS *while it is not on the host's reassembled payload.*

| Attack type | Implem. | Target | Reassembled data | Attack scenario | Works [28] | [34] | Us |
|---|---|---|---|---|---|---|---|
| Evasion | NIDS | | - | E1 | ✓ | ✓ | ✓ |
| | Sup. host | ✗ | "ATTACK" | | | | |
| | NIDS | | "AT00CK" | E2 | | ✓ | ✓ |
| | Sup. host | ✗ | "ATTACK" | | | | |
| Insertion | NIDS | ✗ | "ATTACK" | I1 | ✓ | | ✓ |
| | Sup. host | | - | | | | |
| | NIDS | ✗ | "ATTACK" | I2 | | | ✓ |
| | Sup. host | | "AT00CK" | | | | |

**Table 1.** Attack types illustrated with Figure 1 reassembly cases and based on Definitions 2 and 3. - means the implementation *ignores* the flow chunk data.
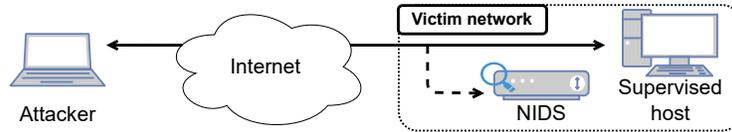


**Fig. 2.** The considered threat model.

**Illustration** We use the overlap chunk sequence introduced in Figure 1 to illustrate insertion and evasion attack types in Table 1. As we can see, the non-targeted host either reassembles differently (with a benign payload, for example) or completely ignores the chunk sequence. Data buffer desynchronization is one of the most dreaded risks for a NIDS since it eventually allows an attacker to bypass all its security mechanisms. See related CVE 2019-18625, 2019-18792, or 2021-37592, whose scores are high or critical. On the one hand, attackers can use evasions to circumvent the NIDS inspection function. An alert pattern can thus reach the supervised host without the NIDS noticing it, as [28, 34] did. An evasion can also impact other NIDS functionalities, such as file or TLS certificate extractions, since a unique (overlapped) bit is sufficient to corrupt them. On the other hand, the insertion attack can alter the NIDS's normal behavior, for instance, by exploiting a known NIDS vulnerability (e.g., 2019-12175, 2023-7242, or 2024-47522 CVEs). Or it can raise false positive alerts, wasting analysts' time.

## 2.2 Threat Model

The threat model we consider consists of an attacker, a NIDS, and a supervised host, as illustrated in Figure 2. The NIDS gets a copy of all the network packets the host receives and sends. We also suppose the NIDS is configured such that the supervised host's IP address is associated with the corresponding reassembly policy[5], if the NIDS offers such a feature. The attacker should be able to:

---

[5] Based on the current knowledge, we consider that hosts with more recent OS versions than the ones tested by Novak and Sturges in [25, 26] (e.g., Debian 12) should have

- identify supervised host and NIDS reassembly policies, enabling them to choose a good (i.e., differently reassembled) overlap case candidate.
- craft IP header fields and payload to perform IP fragment-based attack.
- craft TCP header fields and payload to perform TCP segment-based attack.

**Gaining supervised host and NIDS reassembly policy knowledge** The attacker may want to learn about the monitored host and NIDS reassembly policies to increase the chances of a successful attack. A good approximation to learn about the host reassembly policy is to determine the host OS. Several tools exist to perform OS fingerprinting. Active ones, such as Nmap [2] or Hershel(+) [30], use specifically crafted packet sequences or retransmission times to identify a host OS, while passive tools, such as p0f [36] or nPrintML [18], analyze packet header fields in existing communications. Based on this knowledge and considering the hypothesis that the NIDS is correctly configured, the attacker can determine the NIDS reassembly policy. Finally, they can craft an overlap chunk sequence based on the results reported in Section 4.2 and their objectives.

**Crafting the data chunks** According to the selected overlap case, the attacker chunks the original malicious packet into several pieces. They must appropriately manipulate the header fields *Fragment Offset* and *More Fragments* (resp. *Sequence Number*) to perform an IP (resp. a TCP) chunk-based attack. But, the other header fields of the crafted chunks must be consistent with the carried data (e.g., correct IP or TCP checksum, correct length). The original payload must be in plaintext[6], and choosing the malicious, (i.e., "ATTACK" in Table 1) payload depends on what the attacker wishes to do. However, there are some constraints on the overlapping data portion which is *not* visible in the non-target flow data, especially concerning the syntactic and semantic correctness of the upper-layer protocols. This correctness depends on the performed attack scenario, i.e., E1, E2, I1, or I2, as described in Table 1.

*E1 and I1-related constraints* The non-targeted implementation ignores the overlap chunk sequence; thus, no data is pushed to the upper protocol data stream. It means that the overlapping data portion does not matter and, ultimately, *any data* that fits the required length is, in fact, possible. The overlapping data portions can even be the same.

*E2 and I2-related constraints* The NIDS and the supervised host both push data to the upper-layer protocol. For the "AT00CK" reconstruction to be harmless, it must syntactically and semantically conform with all the upper protocols. In particular, as for any upper-layer checksum: 1) if the checksum is contained

---

the latest OS family representative reassembly policy (e.g., *linux* because Linux 2.4 was the latest tested version) associated in the NIDS configuration file.

[6] The attacker may however take advantage of the overlap ambiguity during encryption initialization to corrupt the NIDS processing of TLS certificates or ciphersuites for example.

within the overlapping portion, then it should be correctly adjusted to fit the "AT00CK" reassembled payload; otherwise, 2) the overlapping data should be adapted to fit the checksum[7]. Any upper-layer syntax or semantic direspect may cause unwanted side effects (e.g., NIDS alerts, NIDS or supervised host failure) that could affect the attacker's stealthiness.

## 3 Testing method

This section describes the overall method used to obtain OS and NIDS reassembly policies, which are then compared in order to find insertion and evasion opportunities. First, it introduces the algebra used to model overlapping chunk sequences. Then, the section describes all the test case characteristics. Finally, it details the hosts we use to test NIDSes and OSes.

### 3.1 Chunk sequence modeling

In the present subsection, we document how a spatio-temporal reasoning algebra can be adapted to model overlapping chunks.

**Spatio-temporal reasoning** Spatio-temporal reasoning is particularly well suited to model packets of protocols that allow chunking and, thus, overlapping. Indeed, we can associate byte offset with one spatial dimension and arrival time with one temporal dimension. Allen's interval algebra [9] is such a spatio-temporal algebra. It consists of 13 different relations, which are described within the first two columns of Table 2. As we can see, there are four non-overlapping Allen relations, i.e., $M$, $Mi$, $B$, and $Bi$, and nine overlapping ones, i.e., $Eq$, $O$, $Oi$, $S$, $Si$, $D$, $Di$, $F$, and $Fi$. The rightmost column transposes the relation meaning in terms of Internet packet sequence.

While these relations describe the relative byte-wise and time-wise position of two chunks, they do not give any information regarding the chunk contents. As a result, one cannot deduce from an overlapping relation whether the overlapping portion contains the same or different data.

**Overlap test case modeling** We use Allen's interval relations (or Allen relations for short) to ensure the exhaustiveness of overlap test cases. In the following, a test case is always time-wisely described. In other words, if $c_1 \mathcal{R} c_2$ with $c_1 \in C^p$ and $c_2 \in C^p$, then $t_1 < t_2$ (i.e., $c_1$ arrives before $c_2$).

There are nine overlapping Allen relations; thus, we consider that *exhaustiveness in terms of overlap coverage is reached by testing these nine overlap cases*. Thanks to the modeling, we can now prove that Novak and Sturges [25, 26] and Atlasis [10] manually found and tested all the possible overlapping test cases. See Table 8 for the other work transposition into Allen formalism.

---

[7] Only two dedicated octets are required to make a payload fit any internet checksum because it is computed with 2-octet words [8]. See Appendix C "Deceiving TCP checksum" of Feng et al. work [16] for a payload crafting example.

| Relation $\mathcal{R}$ | Relation $\mathcal{R}$ inverse | Meaning |
|---|---|---|
| X *M* Y $\frac{\quad Y \quad}{\quad X \quad}$ | X *Mi* Y $\frac{Y \quad\quad}{\quad\quad X}$ | Meet: in-order (resp. out-of-order) contiguous chunks |
| X *B* Y $\frac{\quad\quad Y}{X \quad\quad}$ | X *Bi* Y $\frac{Y \quad\quad}{\quad\quad X}$ | Before: data hole between one chunk ending byte and the other chunk's payload starting byte |
| X *Eq* Y $\frac{\quad Y \quad}{\quad X \quad}$ | - | Equal: complete data overlap with the chunks starting and finishing at the same byte offsets. Data retransmissions are *Eq* overlaps. |
| X *O* Y $\frac{\quad Y \quad}{\quad X \quad}$ | X *Oi* Y $\frac{\quad Y \quad}{\quad X \quad}$ | Overlap: partial data overlap |
| X *S* Y $\frac{\quad Y \quad}{\quad X \quad}$ | X *Si* Y $\frac{\quad Y \quad}{\quad X \quad}$ | Start: partial data overlap |
| X *D* Y $\frac{\quad Y \quad}{\quad X \quad}$ | X *Di* Y $\frac{\quad Y \quad}{\quad X \quad}$ | During: partial data overlap |
| X *F* Y $\frac{\quad Y \quad}{\quad\quad X}$ | X *Fi* Y $\frac{\quad Y \quad}{\quad\quad X}$ | Finish: partial data overlap |

**Table 2.** Allen's interval algebra relations and the corresponding meaning in terms of Internet packet sequences.

| Characteristic | Description | Example |
|---|---|---|
| Overlap type | Allen relation(s) | ***O*** |
| Chunk payload | AABBCCDD → DDCCBBAA ensuring checksum validity | ↓ |
| Reassembly trigerring | ① *IP*: the rightmost finishing and lastly sent fragment has the *More Fragments* (MF) bit unset  ② *TCP*: extra segment at the byte-wise beginning of the test case segments |  |
| Mode | ⓐ *single*: overlaps tested individually  ⓑ *multiple*: overlaps tested altogether |  |
| Upper-layer service | *IP*: ICMP or ICMPv6 Echo  *TCP*: TCP Echo | |

**Table 3.** The IP and TCP test case characteristics.

## 3.2   Test case characteristics

Table 3 summarizes all the overlap test case characteristics. The chunk payloads of a test case are chosen so that 1) they align with the IP header field's unit, which is 8-byte, 2) no matter which overlapping data is preferred, the higher layer checksum is valid, and, of course, 3) the preferred chunk can be distinguished from the other. Novak introduced these payload patterns in [25]. We also use

Novak and Sturges's trick [25, 26], which ensures that all the chunks have been received before the chunk sequence reconstruction occurs, with ①  and ②. Finally, overlaps can be tested *singly* within nine separate chunk sequences ⓐ as [26, 28] did ("individual overlap tests" in  [26]). Or, differently, *multiple* overlaps can be tested altogether within a unique chunk sequence, resulting in one reassembly ⓑ (Novak and Sturges name it "model overlap tests" in  [26]). This last mode has been the most tested in the related works that targeted the OSes [14, 25, 26, 31]. We specifically use Novak and Sturges' *multiple* chunk sequence. The third column of Table 3 illustrates the $O$ relation's test for some introduced characteristics, with simplified chunk payloads to reduce figure size.

### 3.3   OS and NIDS host targets

We perform OS testing through a classical Base-Target architecture. The targeted OSes are varying Vagrant/Virtualbox-based boxes. The testing scripts are all launched from the Base box. As for the NIDSes, they are tested within Docker if an official image exists; otherwise, the tests are performed locally. To deduce NIDS reassembly policies, we alert on 1) the chunk payload patterns introduced in Table 3 and 2) the upper-layer service (i.e., ICMP for IPv4, ICMPv6 for IPv6, and port 7 for TCP). The following IPv4 entry rules are, for example, used to match on the "AABBCCDD" pattern:

- Suricata and Snort: `alert icmp [192.168.0.1] any -> any any(msg:` `"AABBCCDD detected"; content:"AABBCCDD"; sid:1; rev:7;)`
- Zeek[8]: `signature ipv4-AABBCCDD { ip-proto == icmp src-ip ==` `192.168.0.1 payload /.*AABBCCDD.*/ event "AABBCCDD` `detected"}`

In both cases, Network Interface Controller (NIC) offloading is disabled so as not to interfere with the targeted implementation reassembly. See Section 6.1 for more details on OS and NIDS reassembly interferences.

## 4   Results

In this section, we first describe IPv4, IPv6, and TCP reassembly policies of some OSes. The tested OS versions cover a large spectrum for the last 10 years. We then verify NIDS reassembly consistency with these OSes. The versions we target are:

- *OS*: Windows 10 (21h2) and 11 (23h2), Debian 9 (Linux 4.9) and 12 (Linux 6.1), FreeBSD 10.2, 12.1 and 14.1, OpenBSD[9] 6.0, 6.9 and 7.6 and, Solaris 11.2 to 11.4 (SunOS 5.11).
- *NIDS*: Suricata v7.0.4, Snort v3.1.83, and Zeek v6.2.0.

---

[8] Since Zeek's preferred pattern-matching method is scripting, we verified that test case reassemblies are the same across the two matching methods.

[9] The tested FreeBSD and OpenBSD OS versions reassemble the same way the overlap test cases; therefore, we only report FreeBSD policies.

| OS kernel | Protocol version | Test case | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Testing mode | Overlapping relation | | | | | | | | |
| | | | F | Fi | S | Si | O | Oi | D | Di | Eq |
| Windows 21h2, 23h2 | v4 | multiple | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| | | single | n | ∅ | n | o | ∅ | ∅ | n | o | n |
| | v6 | multiple | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| | | single | n | ∅ | n | o | ∅ | ∅ | n | o | n |
| Linux 4.9, 6.1 | v4 | multiple | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| | | single | n | ∅ | n | o | ∅ | ∅ | n | o | n |
| | v6 | multiple | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| | | single | n | ∅ | n | o | ∅ | ∅ | n | o | n |
| SunOS 5.11 | v4 | multiple | n | o | o | o | o | o | n | o | o |
| | | single | n | ∅ | n | o | o | o | n | o | n |
| | v6 | multiple | n | o | o | o | o | o | n | o | o |
| | | single | n | ∅ | n | o | o | o | n | o | n |
| FreeBSD 10.2, 12.1, 14.2 | v4 | multiple | n | o | o | o | o | n | n | o | o |
| | | single | n | ∅ | n | o | o | n | n | o | n |
| | v6 | multiple | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ |
| | | single | n | ∅ | n | o | ∅ | ∅ | n | o | n |

**Table 4.** OS IP reassembly policies. o (resp. n) means that oldest (resp. newest) chunk data is prefered and ∅ means that the OS ignores the overlap. Bold blue means that testing modes are reassembled differently. IPv4 (resp. IPv6) cell backgrounds encodes in consistency with [25] (resp. [14]).

## 4.1   OS reassembly policies

This sub-section details IPv4, IPv6, and TCP reassembly policies for recent OSes in both *multiple* and *single* testing modes. When relevant, we also compare our findings with the latest related work: IPv4 *multiple* mode testing from [25] findings, IPv6 *multiple* mode from [14], and TCP *multiple* and *single* modes from [26].

**IP protocols** Table 4 reports OS IPv4 and IPv6 test case reassembly policies.

*IPv4* Windows and Linux policies have evolved for the *multiple* testing mode since Novak [25]. Both OSes now ignore overlap chunks. The other OSes have not changed their policies. However, the newly tested mode, namely *single*, shows different reassemblies for all the OSes when compared to the *multiple* mode. In total, 6 out of 9 overlapping relations are reassembled differently for Windows and Linux-based OSes, while it accounts for 3 out of 9 for the remaining OSes. We hypothesize that the context introduced by the adjacent chunks used inside *multiple* mode causes the discrepancies between the two testing modes. We thus argue that the *single* mode reassemblies should be used to obtain context-agnostic reassemblies inside the NIDSes. In this testing mode, all the OSes reassemble $F$, $S$, $Si$, $D$, $Di$, and $Eq$ relations the same way, never ignoring the

| OS kernel | Testing mode | Test case | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Overlapping relation | | | | | | | | |
| | | F | Fi | S | Si | O | Oi | D | Di | Eq |
| Windows 21h2, 23h2 | any | o | o | o | o | o | o | o | o | o |
| Linux 4.9, 6.1 | multiple | n | o | o | o | o | n | n | o | o |
| | single | n | o | n | o | o | n | n | o | o |
| SunOS 5.11 | multiple | n | o | n | o | n | o | n | o | n |
| | single | n | o | n | o | n | o | n | o | o |
| FreeBSD 10.2, 12.1, 14.2 | any | n | o | o | o | o | n | n | o | o |

**Table 5.** OS TCP reassembly policies. o (resp. n) means that oldest (resp. newest) chunk data is prefered. Bold blue means that testing modes are reassembled differently. Cell backgrounds encodes in consistency with [26].

test cases. $Fi$ relation is never reassembled, possibly due to the fragment with the $MF$ bit unset's drop. Finally, $O$ and $Oi$ are the only overlap test cases that show different reassemblies depending on the OS.

*IPv6* FreeBSD OSes do not reassemble $O$ and $Oi$ Allen relations, which differs from the observed IPv4 behavior. Except for FreeBSD, all OSes reassemble IPv4 and IPv6 overlapping fragments the same way, and thus, the previous paragraph descriptions also apply to IPv6 fragments. The Windows and Linux-based OSes ignore all the overlapping relations in a *multiple* test mode, which is inconsistent with Di Paolo's [14] findings for $O$, $Oi$, and $Eq$ relations. Since the OS versions that Di Paolo tested are very close to ours, we hypothesize that the lack of tested relation set exhaustiveness for that mode impacts the extracted reassembly.

**TCP protocol** Table 5 describes OS TCP reassembly policies and compares findings with state-of-the-art ones [26]. Windows and FreeBSD reassemble similarly the overlaps across testing modes. The former OS always reassembles with the oldest segment data. These policies are consistent with [26] description. The latest Linux-based OSes reassemble $S$ relation differently depending on the testing mode, favoring old (resp. new) data for *multiple* (resp. *single*) mode. SunOS 5.11 also reassembles the $Eq$ overlap differently, which is consistent with [26] findings.

**Takeaways** IPv4, IPv6, and TCP reassembly policies are more complex than described in the state-of-the-art as overlap reassemblies change depending on the test mode. IP policies have evolved; for example, the Windows and Linux families now show the same reassemblies. TCP reassembly policies have been unchanged since 2007, except Linux's. Finally, OSes continue to reassemble some overlap test cases differently by favoring old or new data or ignoring the chunks.

| Protocol | Implementation | Overlapping relation | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | *F* | *Fi* | *S* | *Si* | *O* | *Oi* | *D* | *Di* | *Eq* |
| IPv4 | **FreeBSD** | **n** | ∅ | **n** | **o** | **o** | **n** | **n** | **o** | **n** |
| | Suricata-*bsd* | n | o | n | o | o | o | n | o | n |
| | Snort-*bsd* | n | ∅ | n | o | o | n | n | o | n |
| | Zeek | n | o | n | o | o | o | n | o | n |
| IPv6 | **FreeBSD** | **n** | ∅ | **n** | **o** | ∅ | ∅ | **n** | **o** | **n** |
| | Suricata-*bsd* | n | o | n | o | o | o | n | o | n |
| | Snort-*bsd* | n | ∅ | n | o | o | n | n | o | n |
| | Zeek | o | o | o | o | o | o | o | o | o |
| TCP | **FreeBSD** | **n** | **o** | **o** | **o** | **o** | **n** | **n** | **o** | **o** |
| | Snort-*bsd* | n | o | o | o | o | n | n | o | o |
| | Suricata-*bsd* | n | o | o | o | o | n | n | o | o |
| | Zeek | o | o | o | o | o | o | o | o | o |

**Table 6.** NIDS reassembly consistency with FreeBSD 10.2, 12.1, 14.1 in a *single* testing mode. o (resp. n) means that oldest (resp. newest) chunk data is prefered and ∅ means the OS ignores the test case. Green (resp. red) means that NIDS reassembly is the same as (resp. different from) FreeBSD.

## 4.2   NIDS/OS reassembly consistency

This section compares the NIDS reassembly policies we observed with the ones of OSes (see Section 4.1) for IPv4, IPv6, and TCP protocols and the *single* mode. Because of space issues, we first check reassembly discrepancies between NIDSes and one OS family, namely FreeBSD. We choose this OS because it offers the most insertion and evasion attack[10] opportunities when configured inside Snort and Suricata using the *bsd* policy. We finish by summarizing results for all the tested OSes and providing metrics on NIDS attack opportunities. Full results can be found in `https://gitlab.inria.fr/laubard/dimva_2025_artifacts`.

**Consistency with FreeBSD OSes**  Table 6 gathers IP and TCP NIDS reassembly policies in the *single* mode, which is the easiest an attacker can exploit.

*IP* NIDSes are all vulnerable to overlap-based attacks with at least two overlap types, except for Snort with IPv4 chunks. Zeek and Suricata can be subject to IPv4 insertion attack with *Fi* relation and IPv4 evasion or insertion with just *Oi*. Moreover, with only two consistent IPv6 test case reassemblies with the FreeBSD, several more relations can be used to perform insertion or evasion attacks on Zeek. Despite being based on the same *bsd* policy description as [25], Suricata reassembles IPv4 overlaps differently. Snort is, interestingly, perfectly consistent with the OS for IPv4 protocol even though no previous work had described reassemblies for the *single* test mode. Additionally, IPv4 and

---

[10] Based on Section 2.1 definitions, if the NIDS and the host reassemble differently a test case by not ignoring it (i.e., E2 and I2 from Table 1), the attacker can either perfom an insertion or an evasion. The other attack cases E1 and I1 are straightforward.

| Protocol | NIDS | Test case inconsistencies | | Tested OS kernels with possible attack | |
|----------|------|:---:|:---:|:---:|:---:|
| | | | | **Evasion** | **Insertion** |
| IPv4 | Suricata | 8 | 22% | F | W, L, S, F |
| | Snort | 4 | 11% | | W, L |
| | Zeek | 9 | 25% | F | W, L, S, F |
| IPv6 | Suricata | 9 | 25% | | W, L, S, F |
| | Snort | 6 | 17% | | W, L, F |
| | Zeek | 28 | 78% | W, L, S, F | W, L, S, F |
| TCP | Suricata | 1 | 3% | S | S |
| | Snort | 1 | 3% | S | S |
| | Zeek | 11 | 31% | L, S, F | L, S, F |

**Table 7.** NIDS inconsistencies with OS reassemblies and corresponding attack opportunities for a *single* testing mode. W, L, S, and F respectively correspond to the tested Windows, Linux, SunOS and, FreeBSD/OpenBSD kernels.

IPv6 fragments are reassembled similarly by Snort and Suricata. Zeek notably reassembles all the overlaps by favoring the oldest IPv6 fragment data.

*TCP* Snort and Suricata policies are consistent with the tested FreeBSD OSes for all the overlapping test cases. The NIDSes, thus, consistently implemented the reassembly policies that Novak and Sturges described [26]. Zeek, which does not have such a reassembly policy configuration capability, reassembles $F$, $Oi$, and $D$ inconsistently, as it always reassembles with the oldest segment's data. An attacker can use these overlaps to perform insertion and evasion.

**Consistency with all OSes** Snort reassembles perfectly consistently IPv4 test cases with the BSD and Sun-based OSes and IPv6 test cases with SunOS. We can find at least one IP test case that is reassembled differently for the remaining OSes, i.e., the Linux and Windows ones. Neither Suricata nor Zeek consistently reassembles all IP overlapping test cases with any of the characterized OSes. Therefore, at least an insertion or an evasion attack can target these NIDS-OS couples. Suppose that Snort or Suricata TCP reassembly policy is correctly associated with the host; there is no possible overlap-based attack except in one case: TCP *solaris* policies and SunOS-based OSes with the $Eq$ test case. On the contrary, Zeek reassembles segments consistently with Windows OSes but does not with the remaining OSes. See `https://gitlab.inria.fr/laubard/dimva_2025_artifacts` for more details.

Table 7 gives more general consistency metrics and related attack opportunities. In particular, Zeek and Suricata reassemble IPv4 overlaps inconsistently for about 20% of them. Snort performs better with 11% inconsistent test cases. Snort and Suricata globally perform better than Zeek for IPv6, with the same or fewer OSes that can be targeted with an insertion or evasion attack. Zeek, which has different IPv4 and IPv6 reassembly policies, performs worse for version 6 (28 inconsistencies) than for version 4 (9 inconsistencies). TCP enables fewer attack

| Author | Work | Year | Protocol | Testing mode | Tested Allen relations | Target type |
|---|---|---|---|---|---|---|
| Ptacek et al. | [28] | 1998 | IPv4/TCP | *single* | *Fi, D* | NIDS/OS |
| Shankar et al. | [31] | 2003 | IPv4 | *multiple* | *O, Oi, Eq* | OS |
| | | | TCP | *multiple* | *O, D* | |
| Novak et al. | [25] | 2005 | IPv4 | *multiple* | all | OS |
| | [26] | 2007 | TCP | *multiple* *single* | | |
| Atlasis | [10] | 2012 | IPv6 | *Na* | all | OS |
| Khattak et al. | [20] | 2013 | IPv4/TCP | *single* | all | CS |
| Wang et al. | [33] | 2017 | IPv4/TCP | *single* | *Eq* | CS |
| Lin et al. | [22] | 2024 | IPv6 | *single* | *Eq* | NIDS/OS |
| Bock et al. | [11] | 2019 | IPv4/TCP | *Na* | Unknown | CS |
| Wang et al. | [34] | 2020 | TCP | *Na* | *F, D, Oi* | CS/NIDS |
| | [35] | 2021 | TCP | *Na* | - | OS |
| Zhang et al. | [37] | 2022 | IPv4/TCP | *Na* | *Eq*/Unknown | NIDS |
| Di Paolo et al. | [14] | 2023 | IPv6 | *multiple* | *O, Oi, Eq* | OS |
| **Us** | **-** | **-** | **IPv4/IPv6/ TCP** | ***multiple single*** | **all** | **NIDS/OS** |

**Table 8.** Summary regarding overlap-based works. "Unknown" means that there is partial or no information on the covered relations for the work tool's run.

opportunities, with only one overlap that Suricata and Snort reassemble incorrectly. Zeek exhibits TCP-based evasion or insertion attacks for 3 OSes out of 4. Since one inconsistency is enough to perform an insertion and/or an evasion, the OSes (i.e., columns 4 and 5 in Table 7) that can be targeted are of importance.

**Takeaways** As expected, Snort and Suricata (which allow policy configuration) perform overall better than Zeek (which uses a unique policy). Snort can protect itself completely against IPv4 and IPv6 evasion attacks (for the tested OSes) and almost entirely against TCP segment-based attacks. Because OS reassembly policies evolve and are more complex than initially thought, NIDSes must (continuously) verify the consistency of the implemented policies.

## 5   Related Works

This section presents the related works that analyzed and described IP and TCP implementation reassembly policies. To ease the comparison with these works, we transpose the covered test cases into Allen's formalism in Table 8. We categorize the works that tested implementation reassembly policies in two families according to the test case generation approach.

## 5.1   Manually generated overlap cases

In 1998, Ptacek and Newsham [28] first showed that OSes could behave differently when reconstructing overlapping IPv4 and TCP chunks. The reassembly ambiguity it poses for NIDSes opened a new research axis, and several works [10, 25, 26, 31] tried to unveil the IPv4, IPv6, and TCP reassembly policies of OSes. Novak and Sturges's [25, 26] works reached exhaustivity for the first time regarding the tested overlap types. More recently, Lin et al. [22] showed that some OS and NIDS do not comply with RFC [13] as regards IPv6 data overlaps (which states that implementations should discard the entire fragment sequence in the presence of any overlap type). However, since they tested a unique overlap type, they may have overestimated OS compliance.

In parallel, other works tried to evade censorship systems (CS) with different elusive packet sequence strategies. In 2013, Khattak et al. [20] described the IPv4 and TCP reassembly policies of the Great Firewall of China (GFW) based on the complete set of overlap relations. Wang et al. [33], which manually designed a unique IPv4 and TCP overlap test case, showed that some middleboxes could interfere with the (original) overlapping fragment sequence by dropping or reassembling it, eliminating, thus, any ambiguity. These works, however, do not consider that the server OSes may have different reassembly policies.

Overall, only 3 of the 8 works were exhaustive in regards to the covered overlap relations. The lack of a unified overlap formalization may explain why most recent works target fewer overlap types than before, as shown in Table 8. Finally, none of the works conducted a complete reassembly discrepancy analysis for NIDS/OS couples.

## 5.2   Semi-automatically generated overlap cases

Other works focused on chunks overlaps from a different perspective. Bock et al. [11] used a genetic algorithm named Geneva to find packet sequences differently processed between CS and hosts. Theoretically, this algorithm can perform evasion attacks with overlapping IPv4 or TCP chunks as it can modify the corresponding header fields and payload. However, it did not find any such chunk sequences. Zhang et al. [37] derived Geneva to find novel packet sequences that bypass Suricata or Snort. Their tool, StateDiver, found one (quite complex) successful technique using IPv4 fragmentation and TCP segmentation. We suppose that Geneva and StateDiver failed to find more overlap-based techniques because 1) successful evasion attempts drove the tools away from this strategy and/or 2) some tested DPIs and hosts may have had the same reassembly policy. One cannot be sure that all the overlap cases were tested exhaustively.

Di Paolo et al. [14] also used a fuzzing-like approach to verify OS compliance with the IPv6 specification [13] when processing overlapping fragments. They derived the Shankar and Paxson model [31] by permuting and duplicating the chunks, and they found that none of the tested OSes (which were Linux, Windows, or BSD-based) conform.

Wang et al. introduced SymTCP and Themis tools [34, 35], which both use symbolic execution to find TCP packet sequences that are processed differently between TCP implementations. SymTCP successfully found a data overlapping strategy to evade Zeek version 2.6. However, while this method could theoretically cover all overlap types, SymTCP cannot find exhaustive overlapping test cases because of its incapability to model retroactive behaviors on data buffers (as rightly explained in [34] section IX). Themis could not find any TCP-based attack strategy based on data overlap ambiguity because all the tested implementations were Linux-based. These implementations may, therefore, have the same TCP reassembly policy. In theory, the Themis tool can show discrepancies in reassembly policies if there are any. However, if none are found, one cannot easily retrieve the reassembly policy. The authors also highlight that adapting the tool to any OS may require quite important efforts.

## 6   Discussions and future works

This section discusses the exploitability of the results described in Section 4. It also debates NIDS countermeasures regarding overlap data ambiguity and gives recommendations.

### 6.1   Overlap-based attack usability

**Relation differences** An attacker that would like to use overlaps to perform an insertion or an evasion attack may struggle differently depending on the relation. If the goal is an evasion by making the NIDS misassemble the transport header, then the attacker would benefit more from $S$, $Si$, or $Eq$ overlaps because they make the chunks start at the same byte offset. If these relations are not used, the attacker may need to add a small chunk on the left, which may be considered as "weird" chunks by NIDSes, especially for IP fragments. Zeek, for instance, considers fragments under 64 bytes as too small, producing a "weird" logging entry. Differently, suppose the attacker aims to make the NIDS hash calculation fail for a given file. Any overlap relation is helpful in that case because one bit flip on the overlapping data portion is enough to change the hash.

**Context importance** The overlap relation reassembly may change depending on the testing mode, as mentioned in Section 4. IP testing exhibits many reassembly differences between the modes. We hypothesize this is partly due to the increasing importance of the fragment which has the *More Fragments* bit unset with *single*. If this fragment is dropped, the reassembly conditions are not met, and the test case is ignored. Differently, the TCP testing mode has much less impact on overlap reassemblies, making the results more adaptable to a larger diversity of segment sequences. It should not be forgotten, however, that in both testing modes, an extra segment was added before (byte-wise) and sent after (time-wise) the overlap segments. In future works, we plan to extend overlap cases' *testing context* (e.g., adding an extra non-overlapping fragment

after the overlapping chunks). This should give a more complete picture of OS reassembly policies.

**Chunk sequence alteration before reaching NIDS or supervised host**
Offloaded stacks on NIC might impact the OS and NIDS policies described in Section 4. NIDS developers advise configuring NIDS instances so nothing alters the supervised traffic, such as NIC offloading. These recommendations, however, do not guarantee that such alteration does not occur on the supervised hosts themselves. We thus plan to analyze NIC's impact in future works.

[21,33] show that some middleboxes drop or reassemble IPv4 fragments, but what middlebox causes this is unclear (e.g., routers, end host's firewall). We also plan to test these middlebox reassembly policies to clarify this point.

Finally, due to well-known and unwanted transport issues, chunks may be delayed or dropped, changing the original overlap relation(s) between the chunks.

### 6.2   Reassembly policy configurability

Suricata and Snort allow one to configure reassembly policies according to the supervised host OS, while Zeek does not. We analyze and compare configurable and non-configurable reassembly policy costs in the following.

**Configurability cost**  Making a NIDS configurable regarding various implementation reassembly policies necessitates several steps. As OS network protocol implementations may evolve over time, checking whether their policies have changed regularly is necessary. NIDSes should be able to easily modify and add reassembly policies as well as extend the mapping between OS versions and reassembly policies. NIDS reassembly policies must be carefully tested to ensure the NIDS reassembles consistently with OSes and that no bug was introduced. Finally, NIDS users must correctly configure their NIDS instance to associate IP addresses with reassembly policies. This configuration task is challenging as an organization's IT infrastructure may rapidly evolve and comprise hundreds (or many more) of supervised hosts. Moreover, IP addresses may be non-static, increasing the human cost of such a configuration even more. This configuration could be painlessly automated through passive OS fingerprinting [18,36] or active fingerprinting [30,31]. As several OSes may be behind an IP address, NIDSes should consider changing the IP address-based reassembly to a flow-based reassembly. We plan to investigate these challenges in future works.

**Non-configurability cost**  A NIDS that does not make the reassembly policy configurable must propose another countermeasure to face overlap-based attacks. For example, an alert-based solution is possible and would consist in raising an alert whenever an inconsistent data overlap is detected (Zeek, Suricata and Snort implement such a countermeasure). Several approaches may be adopted depending on whether a chunk sequence with overlapping data is inherently

considered malicious[11]. If so, there may not be the need for extra information logging, but if not, such a NIDS must log the beginning and finishing byte offsets as well as data on overlapping portions for further analysis. In any case, reassembled data from these chunk sequences must not be used (e.g., for TLS certificate extraction) because the NIDS would not know the monitored host reassembled payload.

### 6.3   Recommendations for OSes and NIDSes

Overlap ambiguity is a long-standing problem, as Ptacek and Newsham [28] initially reported 25 years ago. OSes have changed their reassembly policies over time. However, they still exhibit reassembly diversity. We hypothesize that this diversity is partially caused by the lack of recommendations inside IPv4 and TCP RFCs [8, 15]. We recommend that the OSes implement the same policy (e.g., always use original data, ignoring overlapping fragments) so that ambiguities (slowly) disappear with new releases. Until then, NIDSes with configurable policies must propose multiple reassembly policies and, continuously testing their consistencies. The NIDSes must especially implement single mode rassemblies because they best describe OS behaviors independently of the testing context.

## 7   Ethical considerations

### 7.1   Responsible Disclosure

We contacted Suricata, Snort, and Zeek developers about NIDS inconsistencies with respect to the latest Windows, Linux, SunOS, and FreeBSD/OpenBSD overlap reassembly policies. We gave the NIDSes some months to fix the reported issues before submitting the paper. Snort did not respond to the solicitations, and Zeek acknowledged the results. The CVE-2024-32867 [23] was assigned to the Suricata *bsd*-related misassemblies. We also notified Suricata that we found a display bug during the TCP tests. In particular, some overlapping chunk payloads appeared twice in the *payload* field of the *eve.json* file (the main logging file). This, however, does not impact the TCP buffer with which the pattern matching is done. This bug is now fixed.

### 7.2   Censorship Systems

Improving NIDS security and performance has the side effect of improving censorship systems (CS). Different techniques may be used to elude CS, such as using a VPN [24], encapsulating or mimicking a non-censored protocol traffic [1,3],

---

[11] John and Olovsson's work [19] analyzed the data consistency of some *Eq* IPv4 fragment overlaps in 2008. But, to our knowledge, no work has systematically analyzed whether overlapping chunks with inconsistent data are observed in the wild, and if so, inferred the beningness of such chunk sequences. There are, however, benigm reasons for complete or partial overlaps to occur (for example, see [17, 32]).

inserting a packet that desynchronizes host and censorship-related stateful network traffic analysis tools [11, 12, 20, 21, 33, 34]. The data overlapping strategy falls into the latest technique. Some works showed that it was possible to use data overlaps to circumvent CS at least until 2017 [20, 33], but then, works reported the strategy's unusability [11, 34]. Thus, our results should not affect the censorship elusion techniques currently used. Nonetheless, even if this strategy is in use to circumvent censorship systems, we consider that improving defense capabilities outweigh the negative impacts on censorship elusion techniques.

## 8   Conclusion

In this paper, we adapt well-known *evasion* and *insertion* attack types, refining some specific characteristics related to the overlapping ambiguity. We propose to use Allen's interval algebra-based modeling to describe chunk sequences and ensure the enumeration exhaustiveness of overlap types. This enables us to test OS reassemblies completely regarding overlapping pairs of IPv4 and IPv6 fragments as well as TCP segments. The results show that OS reassembly policies have evolved since the last testing campaigns. Overall, we demonstrate that 9 (resp. 6) out of 12 IP or TCP reassembly policies are *inconsistent* with the tested OSes for Suricata (resp. Snort). Zeek only reassembles consistently with Windows OSes the TCP overlaps. This exposes these NIDSes to insertion and evasion attacks. NIDSes with configurable reassembly policies are less subject to attacks, especially segment-based ones, since TCP policies have changed little. The CVE [23] was assigned to the Suricata *bsd*-related misassemblies we uncovered. Finally, we intend to extend the test context (e.g., multi-chunk overlaps) to completely capture OS reassembly policy complexity as test cases are reassembled differently across testing modes.

## Acknowledgments

## References

1. Meek pluggable transport., `https://support.torproject.org/glossary/meek/`
2. Nmap, `https://nmap.org/`
3. Obfs4 pluggable transport., `https://support.torproject.org/glossary/obfs4/`
4. Snort ip reassembly policies., `https://snort.org/faq/readme-frag3`
5. Snort tcp reassembly policies., `https://snort.org/faq/readme-stream5`
6. Suricata, `https://suricata.io/`
7. Suricata reassembly policies., `https://docs.suricata.io/en/suricata-7.0.4/configuration/suricata-yaml.html#host-os-policy`
8. Internet Protocol. RFC 791 (1981). `https://doi.org/10.17487/RFC0791`, `https://www.rfc-editor.org/info/rfc791`

9. Allen, J.F.: Maintaining knowledge about temporal intervals. CACM (1983)
10. Atlasis, A.: Attacking ipv6 implementation using fragmentation. Black Hat (2012)
11. Bock, K., Hughey, G., Qiang, X., Levin, D.: Geneva: Evolving censorship evasion strategies. In: ACM CCS (2019)
12. Bock, K., Naval, G., Reese, K., Levin, D.: Even censors have a backup: Examining china's double https censorship middleboxes. In: ACM SIGCOMM (2021)
13. Deering, S., Hinden, R.: RFC 8200: Internet protocol, version 6 (ipv6) specification (2017)
14. Di Paolo, E., Bassetti, E., Spognardi, A.: A new model for testing ipv6 fragment handling. In: ESORICS (2023)
15. Eddy, W.: Transmission Control Protocol (TCP). RFC 9293 (2022). `https://doi.org/10.17487/RFC9293`, `https://www.rfc-editor.org/info/rfc9293`
16. Feng, X., Li, Q., Sun, K., Xu, K., Liu, B., Zheng, X., Yang, Q., Duan, H., Qian, Z.: PMTUD is not panacea: Revisiting IP fragmentation attacks against TCP. In: NDSS (2022)
17. Floyd, S., Mahdavi, J., Mathis, M., Podolsky, M.: RFC2883: An extension to the selective acknowledgement (SACK) option for TCP (2000)
18. Holland, J., Schmitt, P., Feamster, N., Mittal, P.: New directions in automated traffic analysis. In: ACM CCS (2021)
19. John, W., Olovsson, T.: Detection of malicious traffic on back-bone links via packet header analysis. CWIS (2008)
20. Khattak, S., Javed, M., Anderson, P.D., Paxson, V.: Towards illuminating a censorship monitor's model to facilitate evasion. In: FOCI (2013)
21. Li, F., Razaghpanah, A., Kakhki, A.M., Niaki, A.A., Choffnes, D., Gill, P., Mislove, A.: lib• erate,(n) a library for exposing (traffic-classification) rules and avoiding them efficiently. In: ACM IMC (2017)
22. Lin, B., Zhang, L., Guo, Y., Zhang, H., Fang, Y.: Research on security protection evasion mechanism based on ipv6 fragment headers. In: IEEE LCN (2024)
23. NIST: CVE-2024-32867
24. Nobori, D., Shinjo, Y.: VPN Gate: A Volunteer-Organized Public VPN Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls. In: NSDI (2014)
25. Novak, J.: Target-based fragmentation reassembly (2005)
26. Novak, J., Sturges, S.: Target-based tcp stream reassembly (2007)
27. Paxson, V.: Bro: a system for detecting network intruders in real-time. Elsevier Computer networks
28. Ptacek, T., Newsham, T.: Insertion, evasion, and denial of service: Eluding network intrusion detection. Tech. rep., Secure Networks, Inc (1998)
29. Roesch, M., et al.: Snort: Lightweight intrusion detection for networks. (1999)
30. Shamsi, Z., Loguinov, D.: Unsupervised clustering under temporal feature volatility in network stack fingerprinting. In: ACM SIGMETRICS (2016)
31. Shankar, U., Paxson, V.: Active mapping: Resisting nids evasion without altering traffic. In: SP (2003)
32. Touch, J.: Rfc 6864: updated specification of the ipv4 id field (2013)
33. Wang, Z., Cao, Y., Qian, Z., Song, C., Krishnamurthy, S.: Your state is not mine: A closer look at evading stateful internet censorship. In: ACM IMC (2017)
34. Wang, Z., Zhu, S.: SymTCP: eluding stateful deep packet inspection with automated discrepancy discovery. In: NDSS (2020)
35. Wang, Z., Zhu, S., Man, K., Zhu, P., Hao, Y., Qian, Z., Krishnamurthy, S.V., La Porta, T., De Lucia, M.J.: Themis: Ambiguity-aware network intrusion detection based on symbolic model comparison. In: MTD (2021)

36. Zalewski, M.: p0f (2014), `https://lcamtuf.coredump.cx/p0f3/`
37. Zhang, Z., Yuan, B., Yang, K., Zou, D., Jin, H.: Statediver: Testing deep packet inspection systems with state-discrepancy guidance. In: ACSAC (2022)
38. Zou, Y.H., Bai, J.J., Zhou, J., Tan, J., Qin, C., Hu, S.M.: {TCP-Fuzz}: Detecting memory and semantic bugs in {TCP} stacks with fuzzing. In: ATC (2021)