Overlapping IPv4, IPv6, and TCP data: exploring errors, test case context, and multiple overlaps inside network stacks and NIDSes with PYROLYSE

Lucas Aubard
Inria
Rennes, France
lucas.aubard@inria.fr

Johan Mazel
ANSSI
Paris, France
johan.mazel@ssi.gouv.fr

Gilles Guette
IMT Atlantique
Cesson Sévigné, France
gilles.guette@imt-atlantique.fr

Pierre Chifflier
ANSSI
Paris, France
pierre.chifflier@ssi.gouv.fr

***Abstract*—IP fragmentation and TCP segmentation allow for splitting large data packets into smaller ones, e.g., for transmission across network links of limited capacity. These mechanisms permit complete or partial overlaps with different data on the overlapping portions. IPv4, IPv6, and TCP reassembly policies, i.e., the data chunk preferences that depend on the overlap types, differ across protocol implementations. This leads to vulnerabilities, as NIDSes may interpret the packet differently from the monitored host OSes. Some NIDSes, such as Suricata or Snort, can be configured so that their policies are consistent with the monitored OSes.**

The first contribution of the paper is PYROLYSE, an audit tool that exhaustively tests and describes the reassembly policies of various IP and TCP implementation types. This tool ensures that implementations reassemble overlapping chunk sequences without errors. The second contribution is the analysis of PYROLYSE artifacts. We first show that the reassembly policies are much more diverse than previously thought. Indeed, by testing all the overlap possibilities for $n \leq 3$ test case chunks and different testing scenarios, we observe 15 different behaviors out of 23 tested implementations depending on the protocol. Second, we report eight errors impacting one OS, two NIDSes, and two embedded stacks, which can lead to security issues such as NIDS pattern-matching bypass or DoS attacks. A CVE [1] was assigned to a NIDS error. Finally, we show that implemented IP and TCP policies obtained through chunk pair testing are usually inconsistent with the observed triplet reassemblies. Therefore, contrary to what they currently do, NIDSes or other network traffic analysis tools should not apply $n = 2$ pair policies when the number of overlapping chunks exceeds two.

***Index Terms*—IP, TCP, intrusion detection, evasion**

I. INTRODUCTION

A host often needs to send more data than the medium or an underlying protocol can send at once. IP fragmentation and TCP segmentation address this generic networking problem by chunking the original (upper-layer) data into several packets. When chunking is used, the receiver must reassemble all the chunks to reconstruct the original data packet. However, this chunking mechanism can result in overlaps. The most frequent scenario is when a chunk is retransmitted with identical data, starting and ending at the same byte offsets. Nevertheless, partial overlaps can also occur, and the data in the overlapping sections may differ. The specifications for IPv4 and TCP (as outlined in their respective RFCs [2], [3]) do not prohibit data overlaps or dictate how implementations should handle them (for example, whether to prioritize data from the older chunk). IPv6 RFC specification initially did not forbid overlaps in the first drafts, but has banned them since 2017 [4].

Network Intrusion Detection Systems (NIDSes) match suspicious patterns or signatures of known attacks on the reassembled flow data. In 1998, Ptacek and Newsham [5] introduced insertion and evasion attacks, whose goal is to desynchronize the NIDS reassembly state from the monitored hosts. Such desynchronizations thus allows an attacker to fool the NIDS pattern-matching functionality. Overlapping chunks with incoherent data portions is one of the insertion/evasion strategies the authors described in the paper. Indeed, they noticed variations in reassembly strategies for IPv4 and TCP data across NIDSes and Operating Systems (OSes). In response, NIDSes proposed two countermeasures: raise alerts whenever there is an overlap or associate host IP addresses with the reassembly policies they implement.

Later studies confirmed and enriched Ptacek and Newsham's findings. Notably, Novak and Sturges [6], [7] found that OSes reassembled in seven (resp. six) different ways the overlapping IPv4 fragments (resp. TCP segments). Suricata [8] and Snort [9] NIDSes provide reassembly policy configurability [10]–[12] which is based on these works. However, IPv4 and TCP reassembly policies of OSes have evolved, and thus, Snort and Suricata can be subject to insertion and evasion attacks when supervising hosts with recent OS versions [13]. In addition to NIDS midpoint stacks, some works [14]–[17] tested the circumvention of several censorship systems (CSes) with IPv4 or TCP overlapping chunks with some success. Apart from OSes and midpoint stacks, such as NIDSes and CSes, no other IP and TCP implementation reassembly policies were tested.

Novak and Sturges [7] and we [13] previously observed different reassemblies for the same overlap type depending on whether the overlaps were tested altogether (within a unique chunk sequence) or separately. The IP and TCP testing contexts are thus important; however, the authors did not explore every context aspect (e.g., different *More Fragments* bit unsetting strategies when multiple rightmost fragments over-

lap). Furthermore, they described implementation reassembly policies by exhaustively testing pairs ($n = 2$) of overlapping chunks. What happens if more than $n = 2$ chunks overlap?

In this paper, we tackle the following questions: *How diverse and correct are IPv4, IPv6, and TCP implementations when exploring test case context and up to $n = 3$ overlaps?* Corollary, *can the NIDSes deduce the monitored hosts' reassembly of any n overlapping chunk sequence from their $n = 2$-based policies?* The paper's contributions are introduced as follows:

- In Section III, we propose a new testing tool named PYROLYSE, short for Protocol bYte stReam OverLapping ambiguitY reaSsembly tEsting. This generic and easily extensible tool enables one to test a diverse range of IPv4, IPv6, and TCP implementation reassembly policies regarding to overlapping chunk sequences. In its current implementation, PYROLYSE ensures exhaustiveness for test cases of up to $n = 3$ overlapping chunks and implements 42 IP and 11 TCP testing scenarii. It uses a reassembly output model for $n \leq 3$ chunks to describe real-world implementations[1].
- In Section IV, we describe the $n = 3$-based reassembly policies of some OSes, NIDSes, embedded/IoT, unikernel, NIC, and DPDK-compatible stacks. We uncover a wide diversity of reassembly policies, as almost every stack has its own policy.
- In Section V, we report the reassembly errors we encountered while testing the protocol stacks. We found that one OS (OpenBSD), two NIDSes (Suricata and Snort), and two custom stacks (lwIP and mirage-tcpip) reassemble some overlap cases with errors. A CVE (now patched) was attributed to the Suricata IP-related error, and OpenBSD fixed the two reported bugs.
- In Section VI, we show that $n = 3$ test case reassemblies cannot easily be deduced from $n = 2$-based policies. To explore if the NIDSes have any chance to reassemble consistently when $n > 2$, we implement some custom algorithms that try to reassemble $n = 3$ test cases using $n = 2$ ones. We establish that protocol implementation behavior is only deductible for 25 stacks out of 61.

II. Background

A. Insertion and evasion attacks

In 1998, Ptacek and Newsham [5] introduced a set of IP and TCP ambiguities (i.e., a network packet that is differently process by two stacks) that may lead to the NIDS stack desynchronization with the monitored hosts. Since the traffic monitoring machine and the monitored hosts are distinct, the NIDS receives a copy of the monitored host traffic and, thus, has no easy way to know the current host reassembly state.

[1]In this study, we focus on $n \leq 3$ because we primarily want to verify if $n = 2$-based implementation reassembly policies are relevant for any n to improve NIDSes. Since we find inconsistencies between $n = 2$ and $n = 3$ policies (Section VI) for most implementations, we don't need to test $n > 3$. However, testing $n > 3$ may uncover other reassembly errors than those described in Section V.

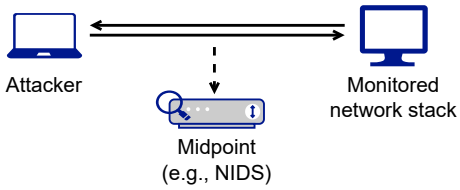


Fig. 1. The considered threat model to perform insertion and evasion attacks.

Figure 1 illustrates this generic insertion and evasion attack-related threat model. By carefully crafting some packets, an attacker can use a reassembly ambiguity to attack the NIDS or the host it monitores by *inserting* or *evading* malicious data in one of the two data flows [5], [13], [17]. The attack is an insertion if the target is the NIDS and an evasion if the target is the monitored host [13]. Some vulnerabilities [18]–[20] related to the NIDS stack desynchronization were recently attributed high or critical scores, meaning that these kinds of attacks are still a security concern for NIDSes. CSes operate roughly in the same way as NIDSes, and some works described new insertion and evasion techniques throughout the years [14]–[17], [21]–[24].

B. Data overlapping ambiguity

One of the IP and TCP ambiguities outlined by Ptacek and Newsham [5] consists of chunking an original data packet into several pieces and making some data portions overlap. Indeed, they observed that the OSes and NIDSes they tested reassembled differently the few overlapping chunk sequences under analysis, which was still recently observed [13]. An attacker can use the NIDS misassembly to insert or evade some malicious data. The rightmost column of Table I illustrates the overlap ambiguity. Evasion occurs if the monitored and monitoring flow data are respectively reassembled as "ATTACK"/"AT00CK" or "ATTACK"/$\varnothing$[2]. Conversely, insertion occurs if the NIDS reassemble with the malicious data.

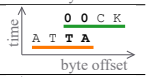
C. Stack reassembly policies

1) $n = 2$-based reassembly policies: Suricata and Snort NIDSes implemented various reassembly policies to address the overlap reassembly ambiguity with the host they monitor. The users have the option to associate monitored host IP addresses with a reassembly policy. The implemented policies are based on Novak and Sturges' works [6], [7] published in 2005 and 2007. The OS reassembly policies, which were obtained by exhaustively testing all the possible overlap types for $n = 2$ chunks, have since evolved [13]. Both works observed different reassemblies when overlaps were tested individually (within multiple chunk sequences) or altogether (within one chunk sequence). This aspect is named *testing mode* in [13].

This last work also introduced Allen's interval algebra, a spatio-temporal reasoning, to model overlapping chunk sequences for $n = 2$. The algebra comprises nine overlapping

[2]$\varnothing$ means that the overlapping chunk sequence is ignored.

TABLE I
ALLEN'S INTERVAL ALGEBRA RELATIONS.

Descr.	Relation $\mathcal{R}$	Relation $\mathcal{R}$ inverse	Example for $\mathcal{R}$
Meet	X M Y	X Mi Y	
Before	X B Y	X Bi Y	
Equal	X Eq Y	-	
Overlap	X O Y	X Oi Y	
Start	X S Y	X Si Y	
During	X D Y	X Di Y	
Finish	X F Y	X Fi Y	

and four non-overlapping relations, as described in Table I. They described the reassembly policies timewisely. An implementation may favor the oldest or newest data chunk or completely ignore the chunks for a given overlap relation.

2) Beyond $n = 2$-based reassembly policies: Atlasis [25] described IPv6 overlap reassembly policies and used combinations of three fragments for the testing. The author introduced a test case generation strategy that yielded one or two distinct overlapping relations inside every fragment sequence. Triple overlaps, i.e., a similar data portion overlaps three chunks as in Figure 4, are not addressed. Therefore, this strategy is not exhaustive for $n = 3$ chunks.

In contrast, Di Paolo et al. [26] used a fuzzing-like approach to check OS conformity with the IPv6 specification [4] when processing overlapping data fragments. They derived the Shankar and Paxson model [27] by permuting and duplicating the chunks and found that none of the tested OSes (Linux, Windows, or BSD-based) comply. In their second and most exhaustive testing scenario, they performed the permutation and four duplications of the original fragment sequence, which is composed of six fragments. In total, 30 fragments overlap with O, Oi, and Eq relations. However, the six other overlapping relations were not addressed, and the testing exhaustivity was not reached for any n. Oprea [28] re-used Di Paolo's model to uncover IPv6 evasion and insertion opportunities on Suricata. If configured with Extreme Performance Tuning guidelines [29] and running on a Linux Ubuntu 22.04 host, he showed that exploitable attacks can target Suricata when monitoring Windows 10, OpenBSD 7.4, or FreeBSD 14.0 hosts.

As Appendix A summarizes, none of the related works provide a platform that 1) exhaustively tests any n chunk sequence, 2) tests a wide range of IPv4, IPv6, and TCP network stack implementations, and 3) describes the obtained reassembly policies. Suricata and Snort NIDSes have implicitly considered that $n = 2$-based reassembly policies could be extended beyond $n = 2$ by actually applying them. However, no work has ever checked the consistency of $n = 2$ policies with more overlapping chunks.

III. PYROLYSE DESIGN

PYROLYSE, available at https://github.com/ANSSI-FR/pyrolyse, aims to audit various protocol implementation types when processing overlapping IPv4, IPv6, and TCP chunks. The targeted implementations are treated as opaque boxes to facilitate the testing setup. Figure 2 illustrates the tool's testing pipeline. First, PYROLYSE generates exhaustive overlap test cases, thanks to Allen's spatio-temporal reasoning. Second, it tests the IP and TCP reassemblies of various protocol implementations through a virtualizable environment (when possible) for reproducibility. Third, it extracts and describes the implementations' reassembly policies with a formalism easing their use. Fourth, PYROLYSE analyses the implementation policies, for example, by checking $n = 2$-based policy consistency with the $n = 3$ observed reassemblies.

A. Overlap test cases generation

The test case generation step starts with the chunk sequence generation. The chunk sequence exhaustiveness is ensured with the Allen relations, as introduced in Section II-C1. More precisely, PYROLYSE generates all the combinations of $\binom{n}{2}$ relations. Using SparQ [30], PYROLYSE both checks if an Allen relation sequence is coherent, and, if so, computes the starting and finishing byte offsets and the time position of every chunk. If the sequence is incoherent, the test case is invalid and thus discarded. Then, PYROLYSE eventually adds real-world context (e.g., contiguous chunks located before the overlapping chunks) to the coherent chunk sequence with the testing scenario. Finally, it populates the chunks with carefully chosen payloads to respect the upper-layer checksum correctness.

1) Chunk sequence: The test case's chunks offset and time characteristics are derived from the Allen relation sequence.

a) Allen relations: The Allen's interval algebra allows one to reason both in terms of space and time. In our case, the space is actually the byte offset. The 13 Allen's relations are used to generate *test cases*. Since a relation links two chunks, we define a *test case for n chunks* as being *the $\binom{n}{2}$ Allen's relations that link the n chunks*. The relations are described in Table I. As we can see, four are non-overlapping relations: M, Mi, B, and Bi. Thus, some test cases show no overlap at all. We use the SparQ tool [30] to ensure the coherence of the $\binom{n}{2}$ Allen relations sequence.

- $\boldsymbol{n = 2}$: $\binom{2}{2} = 1$ Allen's relation describes the link between two chunks, and the 13 relations are coherent test cases for $n = 2$ chunks, as discussed in Section II-C.
- $\boldsymbol{n = 3}$: Since $\binom{3}{2} = 3$, three Allen relations describe all the $n = 3$ chunk test cases. We represent the triplet of chunks from Figure 3 with the notation (p_{01}, p_{02}, p_{12}) = (O, Si, Mi), where indexes refer to the chunk pairs'

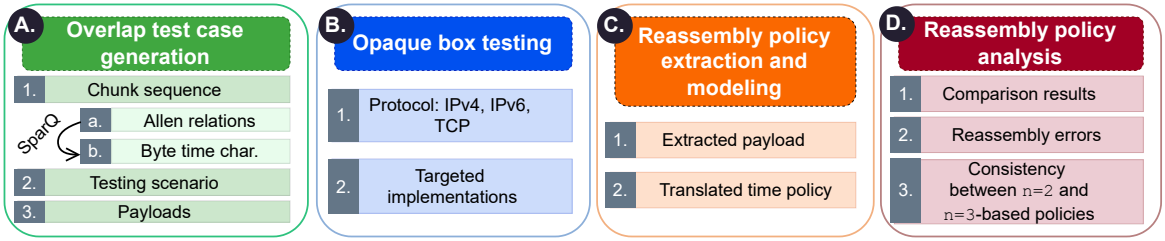


Fig. 2. PYROLYSE testing pipeline.

time position. We find 409 coherent and unique triplet test cases.

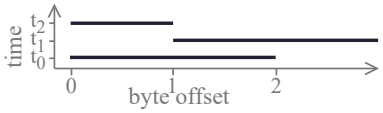


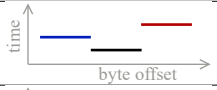
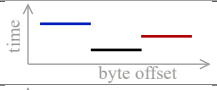
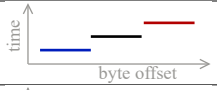
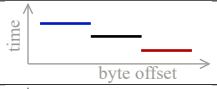
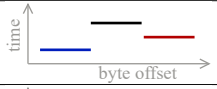
Fig. 3. The (O, Si, Mi) triplet of chunks.

b) Byte time characteristics: SparQ *quantify* operation translates the coherent Allen relation sequence into a time and data offset example of the n chunks.

2) Testing scenarii: As mentioned in Section II-C, some related works [7], [13], [27] added an extra segment before (bytewise) the test case chunks and sent it after (timewise) them, to delay the targeted implementation reassembly. Novak and Sturges [7] and we [13] also observed different reassemblies for $n = 2$ depending on whether the overlaps were tested altogether or separately. Therefore, the testing scenario, i.e., the context surrounding the original test case chunks described with the $\binom{n}{2}$ Allen relations, significantly impacts the test case reassembly and, ultimately, the obtained implementation reassembly policy. The scenarii introduced in this part aim to capture as much of the implementation policies' complexity as possible. We differentiate two context types: one that is *protocol-agnostic* and another that is *protocol-dependent*.

a) Protocol-agnostic context: An extra chunk may be added before and/or after (in terms of byte offset) the test case's chunks and sent before and/or after (in terms of time) them. Such extra chunks are non-overlapping and contiguous with the test case's chunks. Their addition may delay the reassembly or modify its behavior. We refer to the (optional) added extra chunk(s) as the *protocol-agnostic* context. The related scenarii are illustrated in Table II for *any* test case (here represented as a black line). In addition to the original test cases, two chunks are introduced: one before, in terms of byte offset, the original test case, named *Start* and one after, named *End*. *Start* and *End* may timewise precede or follow the original test case or not exist at all. We uniquely identify 11 permutation cases by carefully crafting the scenarii names. If both extra chunks precede or follow the original test case, the scenario name order defines which chunk was sent before. Furthermore, "e"/"s" is short for "End"/"Start" and "p"/"f" for "precedes"/"follows". The scenario $s^{ef,sf}$ thus means that the original test case chunks are sent first, then *End* is sent, and finally, *Start* is sent.

TABLE II
PROTOCOL-AGNOSTIC SCENARII. BLUE LINE REPRESENTS *Start* CHUNK, RED LINE REPRESENTS *End* CHUNK AND BLACK LINE IS A PLACEHOLDER FOR THE n CHUNKS THAT COMPOSE THE TEST CASE.

Name/description	Illustration
s^{c}: continuous	
s^{sp}: Start precedes	
s^{sf}: Start follows	
s^{ep}: End precedes	
s^{ef}: End follows	
$s^{sf,ef}$: Start follows and End follows	
$s^{ef,sf}$: End follows and Start follows	
$s^{sp,ef}$: Start precedes and End follows	
$s^{ep,sf}$: End precedes and Start follows	
$s^{sp,ep}$: Start precedes and End precedes	
$s^{ep,sp}$: End precedes and Start precedes	

b) Protocol-dependent context: Protocol semantics include peculiarities that may influence the final reassembly in the presence of data overlaps. Specifically, we identify one peculiarity for IP protocols that comes from the More Fragment

(MF) bit in the header field. The MF bit should be unset for the last, i.e., rightmost, fragment according to the RFC [2], [4]. However, it is ambiguous if several rightmost fragments overlap in the context of overlapping chunks. For example, if two or more fragments finish at the same byte offset but start at a different one, which fragment(s) must have bit MF unset? The final fragment reassembly may change depending on the chosen strategy regarding MF bit unsetting. In the $n \leq 3$ testing context, we introduce two sets of strategies: one in which we consider one (i.e., oldest, newest, middle) or multiple (i.e., oldest/newest, oldest/middle, middle/newest, all) *rightmost finishing* chunks and, another in which we do the same for the *rightmost starting* chunks. We name the scenarii with the first letters of every sub-strategy. For instance, of means that the *oldest* rightmost *finishing* fragment has the MF bit unset.

c) Association between protocol-agnostic and protocol-dependent parts: Test scenarii comprise a protocol-agnostic part and an optional protocol-dependent part. We thus suffix test scenario names with the protocol-dependent part when necessary and possible: e.g., s^c_{nf} for IP protocols or s^c for TCP.

The IP-related MF bit ambiguity only applies for scenarii whose protocol-agnostic context does not contain *End* extra chunk (i.e., s^c, s^{sp}, and s^{sf}). Indeed, for those with *End* chunk, this extra chunk has bit MF unset. Since TCP does not have a protocol-dependent part[3], the TCP scenarii correspond to the one described in Table II. In total, there are 42 IP and 11 TCP scenarios, representing 10,362 and 4,642 test cases, respectively. Note that some IP test cases are the same in some IP-dependent scenarii (e.g., if a test case has one unique rightmost finishing fragment, this tested fragment sequence is the same in any s_{af}, s_{of}, and s_{nf}-related scenario).

3) Chunk payload population with checksum-impactless patterns: We use ICMP Echo messages (resp. TCP Echo service) on top of IP to obtain IPv4 and IPv6 (resp. TCP) reassembly policies, as discussed in Section III-B. The IP fragments must convey the ICMP header, which contains a *checksum* field. The ICMP checksum is computed with the 2-byte one's complement of the one's complement of the ICMP header and data. Besides using its header and data, ICMPv6 also uses an IP pseudo-header to compute its checksum. This pseudo-header is constructed with the source and destination IP addresses, the protocol, and the ICMPv6 length.

We wish to distinguish between the test case overlapping portions and that the checksum is valid no matter the implementation reassembly[4]. We thus introduce *unique 8-byte-long patterns that do not impact the IP upper-layer checksum*. The first six bytes of a pattern correspond to its unique ID (three bytes for the chunk ID and three bytes for the pattern offset), while the last two bytes are the IP upper-layer checksum correction bytes. See Appendix B for more details.

[3]We actually tested to acknowledge the target's data 1) as soon as the target sends a response and 2) only once all the test case's segments have been sent. Since we did not observe any reassembly change across the tested targets, we do not consider any TCP-dependent part.

[4]For a given test case, we expect to observe varying reconstructed payload lengths for IP scenarii that test overlaps on the rightmost finishing or starting fragment(s) (i.e., the scenarii with different MF bit unsetting strategies). More generally, these patterns are introduced to prevent the checksum from impacting the implementations' reassembly. For example, this property enabled the discovery of the OpenBSD early response error described in Section V-A3.

B. Opaque box testing

PYROLYSE is an opaque box IPv4, IPv6, and TCP protocol testing framework.

1) Upper-layer services: Test case reassemblies are obtained with the ICMP/ICMPv6 Echo service for IPv4/IPv6 and the TCP Echo service through port 7 for TCP. For the IP testing, the fragment sequences of all $n \leq 3$ test cases are first stored within PCAP files, which are then replayed with the Tcpreplay tool [31] on the targeted implementations. TCP, which is a stateful protocol, requires initiating a new connection for every test case before sending the related segment sequence. PYROLYSE implements this TCP logic and enables a host to communicate with a target from the connection initialization phase to the termination, with a FIN handshake or an RST packet. The tool acknowledges any echoed data that the targeted implementation sends. Finally, tcpdump [32] captures IP and TCP host communications.

2) Implementation targets: PYROLYSE supports testing network and midpoint stacks, such as NIDSes.

a) Network stacks: Network stack testing is performed on Base/Target machine pairs. All the scripts are launched from the Base machines. The targeted stacks cannot be tested similarly; we thus use three different setups:

i. The OSes are Vagrant/Virtualbox boxes publicly available from the Vagrant Cloud [33]. The vagrant provisions enable ICMP Echo messages and the TCP Echo service if necessary.
ii. We add embedded/IoT, unikernel, and DPDK-compatible stacks inside Debian boxes since these stacks are not readily available on the Vagrant Cloud. In the target Vagrant provisions, the stack is compiled and bound to a TUN/TAP interface. A TCP Echo server is deployed on every stack to perform the corresponding tests.
iii. The tested NICs use proprietary hardware and network stacks that provide full TCP stack offloading. They are thus physically tested.

In all the cases, the testing outputs are PCAP files. Note that on OS, embedded/IoT, unikernel, and DPDK-compatible hosts, NIC offloading is disabled not to alter the test cases reaching the targeted machine, as we previously did [13].

b) NIDS midpoint stacks: The NIDS official Docker container is used if it exists; otherwise, the NIDS is tested locally after compilation. Signature files are created for all the NIDSes, in which one signature entry is associated with precisely one of the patterns, and the matching buffer is the upper-layer service. The matching direction is from client to server. PYROLYSE generates PCAP files for IP testing but does not synthesize TCP test cases with session initialization and termination. The PCAP files obtained from the TCP testing

TABLE III
DESCRIPTION OF THE TIME POLICIES USED TO DESCRIBE IMPLEMENTATION REASSEMBLY. n IS THE OVERLAPPING CHUNK NUMBER.

n	Overlap type	Notation	Time policy	Policy description
2	Single pair	TP_{pairs}	old	old data is kept
			new	new data is kept
			ignores	test case chunks are ignored
3	Residual pair	$TP_{rp,t}$	none	not applicable because the considered chunk pair does not overlap
			old, new, ignores	same as above
			partialIgnore	only the pair overlapping data is ignored
	Triple	$TP_{t,t}$	none	not applicable because there is no triple overlap
			old, new, ignores	same as above
			middle	data in the middle chunk is kept

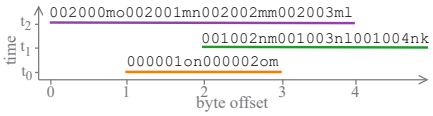


Fig. 4. The triplet (O, D, Oi) representation populated with the IPv4 checksum-impactless patterns.

of one network stack are used to test the NIDSes. The NIDS testing outputs are log files.

C. Reassembly policy extraction and modeling

To obtain reassembly policies, PYROLYSE first extracts the test case reconstructed payload from the PCAP or log files, depending on the targeted implementation. Then, it translates the implementation reassemblies into a time policy, in which the test case reassemblies are described regarding the temporal aspect of the preferred chunk data. Table III introduces this formalism to ease the analysis and use of the reassembly policies. The $n = 2$ pair time policies from TP_{pairs} introduced in [13] is extended with the $n = 3$ triplet time policy set $TP_{triplets}$, that is equal to $TP_{t,t} \times TP^3_{rp,t}$.

Figure 4 illustrates the (O, D, Oi) triplet test case. If an OS reassembles with 002000mo 000001on 000002om 001003nl 001004nk payload, the time policy is thus: $tp_t = old$ and ($tp_{rp_{01},t}$, $tp_{rp_{02},t}$, $tp_{rp_{12},t}$) = ($none$, old, old). $tp_{rp_{01},t}$ is $none$ since the entire overlap of the orange and green chunks does not overflow the boundaries of the triple overlap.

D. Reassembly policy analysis

PYROLYSE performs three analysis types:

- The first compares reassembly policies across testing scenarii and protocol implementations to identify what part of the testing context leads to different reassemblies and to qualify and quantify implementation policy diversity.
- The second one identifies the reassembly errors. Section III-A3 testing payload patterns allow us to find patterns in the test case reassembled payloads that are not correctly located.
- The last analysis consists of checking time policy consistencies between $n = 2$ and $n = 3$ test cases with different methods (that will be introduced in the corresponding Section VI) to determine whether implementation reassembly policies are generalizable to any n.

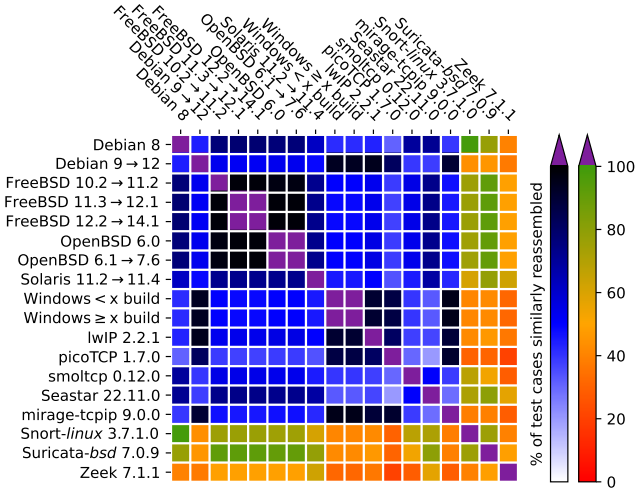
IV. USE CASE 1: EXHAUSTIVE REASSEMBLY POLICIES FOR $n \leq 3$ TEST CASES AND CONTEXT CHUNKS

The first use case for PYROLYSE is obtaining an IPv4, IPv6, or TCP protocol implementation reassembly policy and describing it with the Section III-C formalism. In this section, we give a high-level overview regarding the policies of the following implementations: *OSes* (Windows, the Linux-based Debian, FreeBSD, NetBSD, OpenBSD, Solaris), *embedded/IoT* (lwIP, uIP, picoTCP, smoltcp), *unikernel* (mirage-tcpip), *DPDK-compatible* (Seastar), and *NICs* (Chelsio TOE T520-CR, Xilinx Onload 9.0.1) stacks and *NIDSes* (Suricata, Snort, Zeek). For all the targeted protocol implementations, we test the latest stable version. In addition, we test a range of versions from 10 years to the latest for the general-purpose OSes. All the tested OS versions that exhibit the very same IP and TCP reassembly policies are counted as a unique implementation in the following. See Appendix C for more details on the targeted implementations. Full reassembly policies can be found at [34]. This section first describes the implementations' reassembly policy diversity. Then, it characterizes the impact of the multiple testing scenarii on the test case reassembly.

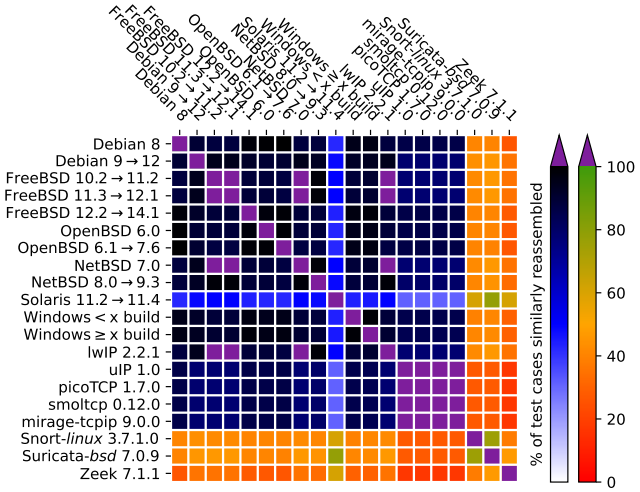
A. Reassembly diversity across implementations

Overall, we observe 15 IPv4, 14 IPv6, and 14 TCP distinct behaviors for 18 to 23 protocol implementations. Figure 5 illustrates this reassembly diversity. Each cell reports the similarity between pairs of protocol implementations. For IPv4 and IPv6 (resp. TCP), the reassemblies of the 10,362 (resp. 4,642) test cases are compared. Similarity scores are computed as the percentage of test cases that exhibit the same time policy. As the purple and dark squares reveal, IPv6 and TCP tend to be reassembled more similarly across implementations than IPv4.

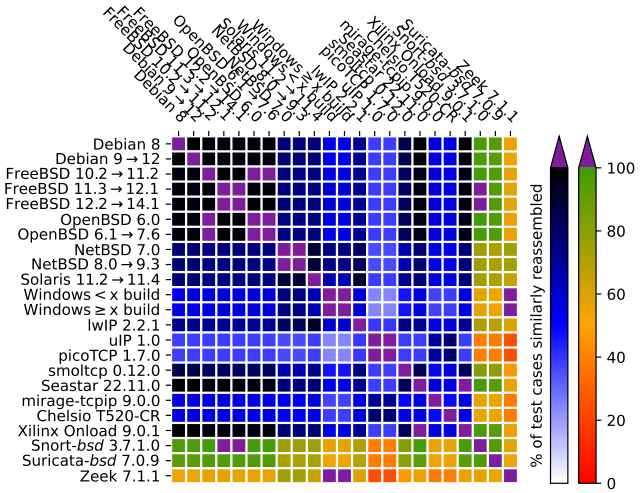
1) Reassembly evolution across OS versions: Five out of the six tested OS families have modified at least one of their IPv4, IPv6, or TCP reassembly policies within the last 10 years. In particular, the Linux-based Debian OSes changed the IPv4, IPv6, and TCP policies between Debian 8 and Debian 9. The most important changes impacted the IPv4 policy since only 44% of the test cases are reassembled the same between these two Debian versions. IPv6 is similarly reassembled for 89% of the test cases and 99% for TCP. Interestingly, none of the TCP $n = 2$ test cases are reassembled differently between versions 8 and 9. All the FreeBSD policies have also evolved. For IPv4 and TCP (resp. IPv6), the change occurred between versions 11.2 and 11.3 (resp. 12.1 and 12.2). The new policies share 98% IPv4, 88% IPv6, and 98% TCP similarities with the previous ones. Finally, Windows, OpenBSD, and NetBSD have



(a) IPv4



(b) IPv6



(c) TCP

Fig. 5. IPv4, IPv6, and TCP reassembly similarities across tested implementations. Purple cells encodes perfect reassembly similarities. Green, orange, and red cells encodes similarities between the NIDSes with their default reassembly policies and the other implementations. **x build** corresponds to the 2022/10/25 build of the Windows OSes.

changed their IPv6 policies slightly, with less than a 2% difference. The versions introducing the change are the 2022/10/25 Windows builds[5], OpenBSD 6.1, and an unidentified NetBSD version between 7.0 and 8.0[6]. The latest Windows OSes now exhibit the same reassembly behavior when processing IPv4 and IPv6 overlapping or non-overlapping fragments, which is also true for Solaris and the latest Debian OSes.

2) Reassembly consistency between NIDSes and network stacks: Suricata, Snort, and Zeek NIDSes reassemblies were specifically tested to measure their consistency with the stacks they are likely to monitor. Snort and Suricata (which allow users to configure the reassembly policies) were tested with their default policies, which are always *bsd*, except for Snort and IP protocols where the *linux* policy is used[7].

Only Zeek reassembles all the TCP test cases consistently with Windows OSes, always favoring the oldest data segment. Snort and Suricata never reassemble entirely consistently the IP test cases with any OS, including with the OS family their default policy tries to reproduce. Since the *linux* policy has evolved, Snort IPv4 inconsistencies even reach 66% with the latest Debian OSes. IPv6 test cases, that Snort and Suricata both reassemble in the same way as IPv4, are never reassembled with more than 77% similarities for Suricata and 60% for Snort across OSes. Finally, Snort and Suricata reassemble TCP test cases with 0 to 4%, 2 to 4%, and 25 to 29% inconsistencies with FreeBSD, OpenBSD, and NetBSD OSes, respectively. *Every reported NIDS inconsistency can lead to an insertion or evasion attack.*

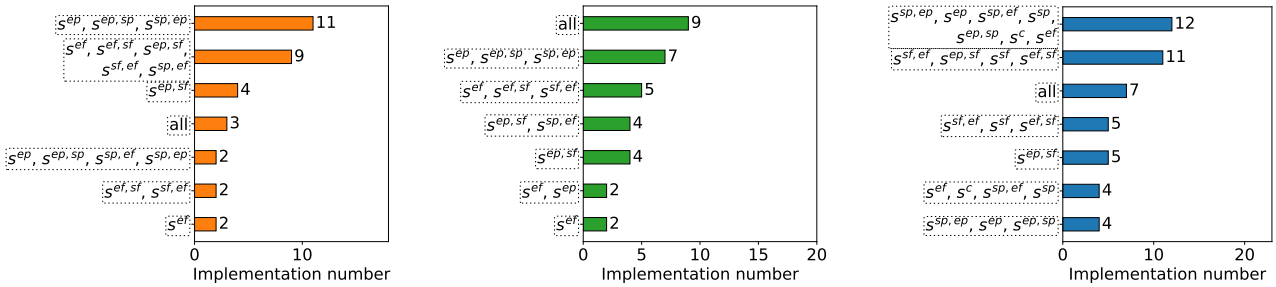
3) Interesting similarities and dissimilarities: IPv6 implementations reassemble the test cases quite similarly. This can be explained for two reasons. First, most of the embedded stacks do not reassemble IPv6 fragments at all. Second, all the tested OSes but Solaris *ignore* about 80% of the test cases. The strong IPv4 similarity between the latest Windows and Debian OSes is also due to the many ignored test cases. This observation is valid for both OSes as they both exhibit identical IPv4 and IPv6 policies. Surprisingly, FreeBSD and OpenBSD OSes, which had the same TCP policy until FreeBSD version 11.2, have a TCP reassembly policy closer to Debian OSes than NetBSD. Corollary, NetBSD and Solaris OSes also have close TCP reassembly policies. BSD OSes, thus, do not have a unified TCP reassembly policy anymore. Furthermore, mirage-tcpip, picoTCP, and uIP do not reassemble data segments that overlap with already received ones. Most of the time, the last two stacks and Chelsio T520-CR[8] do not reassemble out-of-order segments either. Finally, Seastar and Xilinx Onload reassemble TCP test cases similarly, and they have TCP policies close to Debian, FreeBSD, and OpenBSD OSes.

[5]We tested this evolution for Windows Server 2019, Windows 10 (22H2), and Windows 11 (21H2) boxes.

[6]We did not find Vagrant boxes between these two NetBSD versions.

[7]As mentioned in [13], the lack of automated configuration and the potentially high number of monitored machines make it very costly to configure a NIDS in a real-world deployment. Therefore, we suppose the situation closest to reality uses the default NIDS reassembly policies.

[8]The Chelsio team replied that this behavior concerns T5 ASICs and that T6 ASICs should resolve the problem, but we did not test them.



(a) IPv4 (b) IPv6 (c) TCP

Fig. 6. Top 7 protocol-agnostic scenario groups that are similarly reassembled across implementations. The groups are sorted by the implementation number.

B. Reassembly diversity across testing scenarii

For a given implementation, PYROLYSE groups the testing scenarii showing the same time policies.

1) Protocol-agnostic scenarii: The first notable result is that $s^{ep,sp}$/$s^{sp,ep}$ and $s^{sf,ef}$/$s^{ef,sf}$ pairs are always in the same groups for any IPv4, IPv6, or TCP implementation. We cannot find other such pairs for the protocol-agnostic scenarii, and therefore, all the other introduced scenarii enrich and extend the reassembly policies. Figure 6 reports the top 7 scenario groups across implementations and protocols.

a) IP: We observe quite a lot of diversity across implementations with 14 IPv4 scenario groups for a total of eight (protocol-agnostic) scenarii in Figure 6a. Eleven implementations reassemble all scenarii with End preceding the test case chunks similarly, except $s^{ep,sf}$. This means that the hole left by $Start$ impacts these implementation reassemblies, as the top two scenario groups reveal. Three implementations, namely Zeek, lwIP, and picoTCP, reassemble all their testing scenarii identically. Zeek always favors old data chunks, no matter the context. PicoTCP always ignores the test cases that contain only IPv4 fragments[9]. The defragmentation process may thus not be present in this implementation.

IPv6 implementations also behave in several ways, as 13 scenario groups are observed. Nine implementations reassemble test cases similarly across testing scenarii as shown in Figure 6b. Zeek, once again, favors old data, and four (picoTCP, uIP, mirage-tcpip, and smoltcp) ignore all test cases. The four others (FreeBSD 10.2 to 12.1, NetBSD 7.0, and lwIP) only reconstruct test cases that exhibit no overlap at all or overlaps, but, in this case, any newest overlapping fragment is entirely included in the oldest ones. We observe the same most common scenario groups as IPv4, which contain scenarii with the End preceding chunk. However, as for IPv4, some implementations have unique scenario groups; thus, we cannot draw any general conclusions.

b) TCP: Eleven scenarii in total are tested. We observe less diversity than for IP protocols since the implementations only show seven groups of testing scenarii, as shown in Figure 6c. The first two top groups show that a $Start$ segment that follows the test case sequence (thus introducing a data hole on the left) often makes the reassembly distinct from the other scenarii. This behavior is not surprising since TCP is a stateful protocol, and thus, data is pushed to the upper layer as long as there is no data hole on the left of the received segments. However, five implementations behave differently between s^{sf}, $s^{ef,sf}$, and $s^{sf,ef}$ on the one hand and $s^{ep,sf}$ on the other. As observed with IPv4, introducing multiple (at least two) data holes that are filled throughout the test case segment sequence impacts some implementations. Finally, Windows OSes, FreeBSD 11.3 to 14.1, Zeek, Snort, and mirage-tcpip exhibit a unique scenario group, meaning that an implementation reassembles all the test cases the same across the scenarii, with Windows OSes and Zeek notably always preferring the oldest chunk data.

2) Protocol-dependent scenarii: Eighty-five groups of IPv4 protocol-dependent scenarii are observed across the tested implementations. Apart from different versions of the same implementation, no stack has precisely the same groups of scenarii. The s^{c}, s^{sf}, and s^{sp}-like scenarii are never reassembled similarly for any implementation. This is expected for s^{sf}, but it is slightly more surprising for the s^{c} and s^{sp} scenarii because the reassembly conditions[10] can here be met earlier, i.e., before all the test case chunks have been received. For the s^{c}-like scenarii, some test case chunks are actually complete datagrams, which explain the reassembly difference with s^{sp}. Regarding the discrepancy between unsetting the rightmost *starting* and the rightmost *finishing* chunk(s), we observe some similarities for the s^{sf}-like scenarii for Zeek, Seastar, OpenBSD, FreeBSD, lwIP, and mirage stacks. However, for every OS, there are always at least one couple of scenarii, e.g., s^{sf}_{of}/s^{sf}_{os} for OpenBSD, that are reassembled differently. More surprisingly, except Zeek and Seastar, implementations sometimes reassemble in a different way s_{af} (resp. s_{as}) and s_{of} (resp. s_{os}) test cases. It probably means these implementations drop the oldest fragments if multiple fragments have the MF bit unset.

Slightly fewer, i.e., 83, IPv6 scenario groups are ob-

[9]Some s^{c} test cases actually are structured in such a way that a unique chunk is enough to reassemble a complete payload.

[10]For a given IPID, IP reassembly conditions are met whenever the corresponding fragment sequence exhibits no data hole and a fragment with the MF bit unset has been received.

TABLE IV
REASSEMBLY ERROR SUMMARY UNCOVERED BY PYROLYSE. ! MEANS THAT THE ERROR IS PARTIALLY FIXED.

Type	Impacted protocol	Impacted implem.	Possible security issue	CIA triad	Fixed?
Reassembly logic	IPv4/IPv6	Suricata	pattern-matching bypass		!
		Snort			✗
	IPv4	OpenBSD	payload shortening	I	✓
	TCP	Suricata	pattern-matching bypass		!
Connection termination	TCP	OpenBSD	resource exhaustion/ DoS	A	✓
		lwIP			✗
		mirage-tcpip			✗
Log display	TCP	Suricata	security investigation hindrance		✓

served. Even if four implementations have the same protocol-dependent scenario group (since they ignore all the test cases), the IPv6 policy and, with them, the scenario groups of five OS families have evolved. The other IPv6 implementations behave the same as IPv4.

3) Impact of testing scenarii for configurable NIDSes: As we've just discussed, the testing scenario considerably impacts the implementation reassembly, and the effect differs depending on the implementation. Since the diverse testing scenarii introduced in this document have never been addressed in any related work, it is unsurprising that the configurable NIDSes Suricata and Snort do not behave consistently with most OSes. Indeed, Suricata does not exhibit the same groups of IP and TCP scenarii as BSD OSes, nor does Snort with Debian OSes for IP or OpenBSD and NetBSD for TCP. The only exception is Snort with FreeBSD, which exhibits the same TCP scenario groups. The diverse observed implementation behaviors across scenarii must be transcribed inside the NIDSes if they intend to reassemble consistently with them.

Takeaway

The 42 IP and 11 TCP scenarii and $n \leq 3$ test cases uncovered diverse implementation reassembly policies (fully available at [34]). They have increased from 5 IP and 6 TCP policies [6], [7], [25] to 15 IPv4, 14 IPv6, and 14 TCP stack behaviors. PYROLYSE provides an automated way to extract such reassembly time policies and ease testing the reassembly consistency of security equipment (such as NIDSes) with the network stack implementations.

V. USE CASE 2: REASSEMBLY ERRORS

This section details the reassembly errors PYROLYSE discovered. After manual investigation, we identify three categories of errors: reassembly logic, connection termination, and log display. One OS (OpenBSD), two NIDSes (Suricata and Snort), and two other stacks (lwIP and mirage-tcpip) are impacted. In total, we reported eight bugs to the implementation developers; half are entirely or partially fixed. Table IV summarizes the encountered errors.

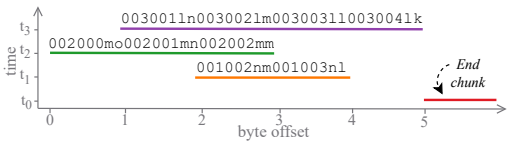


Fig. 7. The (Oi, O, D) test case representation in a s^{ep} testing context.

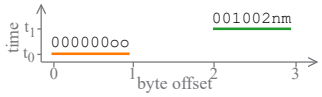


Fig. 8. The (B) test case representation in a s^{c} testing context.

A. Reassembly logic

The reassembly logic category groups the errors that directly affect the reconstructed payloads. The observed bugs lead to data holes or data located after a data hole in the reassembled payload or an early response from the targeted implementation. The hole-related issues impacted Suricata and Snort NIDSes. This error is critical since it damages the NIDS's ability to perform its key functionality, namely, malicious pattern-matching. Suricata partially fixed the problem for IP and TCP, but Snort did not. The early response was observed while testing the OpenBSD IPv4 stack.

1) Data holes in the reassembled payload: Snort and Suricata allow users to configure the reassembly policy according to the monitored OS IP addresses. The reassembly error impacts each NIDS's reassembly policies differently.

a) Suricata: The error is present in 93 to 801 test cases, impacts about half of the IP scenarii, and concerns all Suricata reassembly policies (including the default one *bsd*). One impacted test case is the triplet illustrated in Figure 7. With the *default* policy, the reconstructed datagram is 002000mo 002001mn 001002nm 001003nl **.....**. Since Suricata raises no alert related to the 0030041k pattern, it is blind between offsets 4 and 5, making the pattern-matching functionality inoperable. After some investigation, we concluded that an incorrect reconstructed packet length calculation and an early exit along the fragment queue processing caused the behavior. This reassembly error was assigned the CVE 2024-32867 [1] and is partially fixed from Suricata version 7.0.5.

b) Snort: The error concerns all reassembly policies (including the default one *linux*) for 26 to 211 triplet test cases. The behavior is observed for some triplet test cases containing a third fragment that overlaps the previous two, such as (Oi, O, D) of Figure 7. Snort's reassembly is 002000mo 002001mn 002002mm 001003nl **.....** with the *bsd* policy. The NIDS uses a variable $slide$ to track the amount of data to remove from the third fragment. However, after the p_{12} overlap resolution (i.e., the choice to ignore or to reassemble with old or new data), the $slide$'s value is overwritten by the p_{02} overlap resolution, which subsequently impacts the final datagram reconstruction.

2) Reassembly with data located after a hole: For any TCP scenario and test cases *with a hole*, Suricata reassembles with some data located after (in terms of byte offset) the data

hole. Since a test case with a hole is characterized with one (resp. two or three) Allen's before-like relation, i.e., B or Bi, for $n = 2$ (resp. $n = 3$), there are two over 13 (resp. 72 over 409) of a scenario's test cases that are impacted, which represent a total of 814 test cases. Figure 8 illustrates one impacted test case. The observed Suricata reassembly is 000000oo 001002nm from the *payload_printable* field of the eve.log file. Suricata also raised an alert for both 000000oo and 001002nm patterns. The NIDS can then be fooled if a malicious data segment fills the data hole from 1 to 2. This issue is partially fixed from Suricata version 7.0.7.

3) Early response: OpenBSD responded to the fragment sequence for 14 triplet cases tested with s^{ef}, $s^{sp,ef}$, and $s^{sf,ef}$ scenarii, whereas it did not receive all the fragments. More precisely, OpenBSD responded before processing the fragment with the MF bit unset. It resulted in the truncation of the original datagram, i.e., the reassembled datagram length was less than it should have been. This is how PYROLYSE was able to discover the error. The latter was introduced in OpenBSD's code while implementing DoS protection in pf[11] fragment handling on 2018/09/04. The code avoided traversing "the list of fragment entries to check whether the pf(4) reassembly is complete" by using a data hole counter. The bug is now patched [35].

B. Connection termination

A similar error was observed during the TCP testing for three tested implementations: OpenBSD, mirage-tcpip, and lwIP. This bug always led to the non-termination of some test case-related TCP connections. The impacted test cases were, however, different across the implementations.

On the one hand, OpenBSD did not reset the TCP connection if there had previously been a data hole during that connection. Several sequence numbers were tested to reset it without success. This reassembly error was observed in all the testing scenarii and is now patched [36].

On the other hand, mirage-tcpip 9.0.0 and lwIP 2.2.1 echoed some data but not the maximum possible amount for the test cases in question. We believe they considered the reset packet sequence number invalid because it was located after all the data was sent (including the overlapping segments). Since they did not echo the maximum possible amount of data, the initiator SND.NXT internal variable was probably not the same as the RCV.NXT value in mirage-tcpip and lwIP.

C. Log display

Suricata reassemblies were obtained through log files using two methods: the signatures and the reconstructed payload-related log field. Suricata's log payload field contained duplicate payload patterns for about 72% to 84% of the TCP test cases, depending on the scenario. This manifested the same way as a reassembly logic error, since some patterns from the reconstructed payload were not correctly located. However, after an unsuccessful attempt to raise an alert on the duplicated reconstructed payload log field, we classified the bug as a log display error. After some investigation, we discovered that the signatures and the payload field do not use the same TCP reassembly buffers, which explains the divergent behaviors across extraction payload methods. The bug is not critical because we do not observe the pattern duplication in the buffer used for the pattern-matching. However, it can confuse the NIDS security investigators, who may think of a false positive alert and, ultimately, misclassify a malicious security event. Suricata version 7.0.7 patched this reassembly error.

[11]Pf is the OpenBSD solution for packet filtering. It reassembles overlapping IPv4 fragments.

TABLE V
TCP TEST CASE INCONSISTENCY NUMBER ACROSS COUPLE LATEST OS/NIDS REASSEMBLY POLICY, FOR THE s^{sf} TESTING SCENARIO.

OS	NIDS policy	Number of inconsistent test cases	
		$n = 2$ (/13)	$n = 3$ (/409)
Windows 11	Suricata-*first*	0	12
	Snort-*first*	0	0
Debian 12	Suricata-*linux*	0	21
	Snort-*linux*	0	15
Solaris 11.4	Suricata-*solaris*	1	40
	Snort-*solaris*	1	36
OpenBSD 7.6	Suricata-*bsd*	0	30
	Snort-*bsd*	0	42

Takeaway

PYROLYSE found eight reassembly errors while testing the 23 targeted implementations with the $n \leq 3$-related 42 IP and 11 TCP testing scenarii. Three reassembly logic bugs impact Suricata and Snort NIDSes and are critical because they can lead to a malicious pattern-matching functionality bypass. Developers wholly or partially fixed half of them.

VI. USE CASE 3: REASSEMBLY INCONSISTENCY BETWEEN $n = 2$ AND $n = 3$ TEST CASES AND CUSTOM REASSEMBLY ALGORITHM TESTING

This section aims to link the observed $n = 2$ and $n = 3$ implementation reassemblies. The configurable Suricata and Snort NIDSes currently apply $n = 2$ pair policies to any n overlapping chunk sequence. When an overlapping chunk arrives, it is compared to already received ones, one by one, and the pair policies are applied. First, the section shows that the current NIDS approach is incorrect since one scenario does not exhibit any inconsistency between some OSes and NIDSes for $n = 2$, but many for $n = 3$. Therefore, $n = 2$ policies cannot be used alone to reassemble any n sequence inside the NIDSes. Second, it identifies some root causes for the observed inconsistencies. Finally, considering these root causes, it explores the extension of $n = 2$ pair time policies (as described in Section III-C) with manually implemented reassembly algorithms, as a way for the NIDSes to reassemble any n sequence.

A. Suricata and Snort's inconsistencies with OSes for $n = 3$ overlapping segments

Snort and Suricata users can associate host IP addresses with a set of implemented reassembly policies. As a preliminary approach to verify if $n = 3$ test case reassemblies can

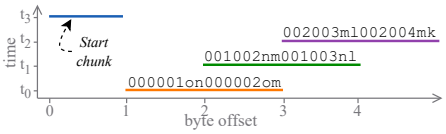


Fig. 9. The (O, M, O) test case representation in a s^{sf} testing context.

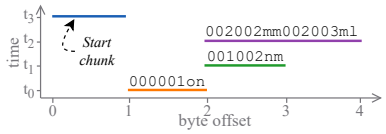


Fig. 10. The (M, M, S) test case representation in a s^{sf} testing context.

be deduced from $n = 2$-based policies, we compare the reassembly consistency of Suricata and Snort with some OSes. Since these NIDS-implemented policies are based on Novak and Sturges's work [7], we use our related TCP scenario s^{sf}. Table V shows that both Suricata and Snort reassemble all $n = 2$ test cases similarly with the latest Windows, Debian, and OpenBSD OSes. However, the two NIDSes reassemble differently with these OSes some $n = 3$ test cases, except Snort with Windows. Therefore, the reassembly consistency between a NIDS and an OS for $n = 2$ chunk sequences does not necessarily imply any kind of reassembly consistency for $n \geq 3$ chunk sequences. Overall, the implicit direct application of $n = 2$-based implementation policies when $n \geq 3$ inside the NIDSes is incorrect (except for Snort and Windows).

B. Inconsistency between n=2 and n=3 reassemblies: observations and root cause example

We here first report an observation from the $n \leq 3$ observed policies regarding to $n = 2$ and $n = 3$ inconsistencies. Then, we detail one root cause for some of these inconsistencies.

1) Observations from the $n \leq 3$ observed policies: As discussed in Section IV-A1, some OSes updated their reassembly policies within the last 10 years. These updates impact $n = 3$ test case reassemblies without necessarily affecting the $n = 2$ ones, as reported for Linux's TCP stack. For example, Debian 8 and Debian 9 to 12 reassemble the (O) pair favoring the oldest data in a s^{sf} context. Yet, Debian 8 and Debian 9 to 12 reconstruct differently the (O, M, O) test case, which is illustrated in Figure 9, i.e., 000001on 000002om 001003nl 002004mk vs 000001on 000002om 002003ml 002004mk. This separated evolution proves that $n = 3$ implementation policies cannot be derived — at least solely — from the $n = 2$ testing.

2) Impact analysis of the chunk merging mechanism: One interesting function involved in the Linux 6.1 (i.e., our Debian 12's kernel) TCP reassembly process is $try\_to\_coalesce()$ [37]. This function seems to merge data segments, as the source code function description mentions "before queuing skb[12], try to merge them to reduce overall memory use and queue lengths, if cost is small".

A protocol implementation that merges chunks, as Linux 6.1 seems to do, reassembles differently from what is expected for some triplet test cases. Figure 10 illustrates such a test case.

Let's consider the Debian 12 TCP reassembly policy in which the S overlap is reassembled favoring the newest data while O is with the oldest.

- Expected reassembly (i.e., without merging) from $n = 2$ policy: the green and purple segments overlaps with S relation, newest data is preferred. The resulting reassembled data is 000001on 002002mm 002003ml.
- Non-expected reassembly (i.e., with merging) from $n = 2$ policy: the orange and green segments are contiguous, they are thus merged. Orange-green and purple segments' corresponding Allen relation is now O, which is reassembled favoring the oldest segment data. The resulting reassembly is 000001on 001002nm 002003ml.

This reassembly mechanism may modify the chunk configuration, the associated Allen relations, and therefore, the applied time policies.

C. Reassembling $n = 3$ test cases with custom algorithms

One way for the NIDSes with configurable policies to reassemble consistently the $n \leq 3$ triplet test cases with the stacks is to apply the $n \leq 3$ time policies in the same direct way as the current $n = 2$-based policies. This can be done through 1) multiple conditional branches or 2) a time policy hashmap look-up. However, 1) is very tedious and error-prone for $n \leq 3$ due to the large test case number and the implementation policy diversity (as described in Section IV). Doing 2) requires resolving the overlaps once all the chunks have been received and computing the corresponding Allen relation sequence. Furthermore, these solutions do not ensure the NIDSes are consistent for any n. Considering the $n = 2$ and $n = 3$ inconsistency causes introduced in the last part, another approach for the NIDSes would be to extend reassembly policies using custom-implemented algorithms that leverage observed reassembly mechanisms such as the one described in Section VI-B2. If these algorithms are consistent with $n = 3$ cases, they might also be correct for $n > 3$. Additional tests would, however, need to be performed to verify this aspect. We implement some algorithms inside PYROLYSE and attempt to reproduce the $n = 3$ implementation behaviors. First, we describe these algorithm principles and then analyze the consistency of the reassembly between the algorithms and the implementations.

1) Principle: As previously discussed, protocol implementation reassembly policies based on $n = 2$ testing are not easily extensible to any n. We hypothesize that addressing more testing scenarii than state-of-the-art and $n = 3$ test cases brings more reassembly algorithm parts into play. It gives a more complete but also more complex picture of reassembly policies. We characterize reassembly algorithms using three main components:

(i) the function(s) or code that we consider to directly impact the overlap resolution, i.e., the choice to ignore or to reassemble with old or new data. Suppose some

[12]Short for socket buffer.

TABLE VI
REASSEMBLY CONSISTENCY OF THE 3,376 IP AND 4,642 TCP ($n \leq 3$) PROTOCOL-AGNOSTIC TEST CASES ACROSS IMPLEMENTATIONS.

Protocol	Consistency between...			
	...$n = 2$ and $n = 3$ residual pairs inside implementations (baseline)		...implementations and custom reassembly algorithms $n = 3$ time policies	
	#	Implem. names	#	Implem. names
IPv4	1/18	picoTCP	6/18	Windows OSes, lwIP, picoTCP, Seastar, mirage-tcpip
IPv6	4/20	picoTCP, uIP, mirage-tcpip, smoltcp	10/20	Windows OSes, FreeBSD 10.2 to 11.2, FreeBSD 11.3 to 12.1, NetBSD 7.0, lwIP, picoTCP, mirage-tcpip, uIP, smoltcp
TCP	2/23	Windows OSes	9/23	Windows, OpenBSD, and FreeBSD OSes, smoltcp, Seastar

overlaps are not addressed in the code (i.e., no action is explicitly taken for the overlap resolution). In that case, it likely means that the overlap is undetected, and consequently, the implementation would overwrite data on the overlapping portions. Both cases lie in this item: they belong to *the overlap resolution component*.

(ii) the mechanisms that indirectly impact the overlap resolution in the scope of our testing. Such mechanisms may, for instance, be present for performance reasons (e.g., chunk merging [37]) or due to protocol specificities (e.g., MF bit information tracking for IP). They belong to the *reassembly mechanisms with side effect component*.

(iii) the mechanisms that have no direct or indirect impact on the reconstructed payload in the scope of our testing. Such mechanisms belong to the *reassembly mechanisms with no side effect component*.

The intuition is that the implementations perform the overlap resolution (i) by pairs of two chunks. Thus, $n = 2$ testing alone can capture (i) behavior. However, other (ii)-like mechanisms may also impact the final test case reassembly, as seen in Section VI-B2. These mechanisms explain, at least to some extent, the Suricata and Snort's inconsistent reassemblies for $n = 3$ reported in Section VI-A. We thus hypothesize that by considering $n = 2$ reassembly policies coupled with some indirectly implicated reassembly mechanisms, the NIDSes may be able to reassemble consistently with the implementations. We implemented custom reassembly algorithms to verify this point. The algorithms are further detailed in Appendix D.

2) Triplet test case consistency analysis: The consistency performance of custom reassembly algorithms is exposed in Table VI. The second column lists implementations whose time policy consistency of residual pairs inside triplets (i.e., we ignore triple overlap as defined in Section III-C) is consistent with $n = 2$-based time policies. The rightmost column reports the implementations in which at least one reassembly algorithm successfully reproduces its $n = 3$ behavior. The second-column consistency check method is more limited than the third-column since it does not consider the triple overlap (if there is any). It is thus unusable in practice, but we add it as a baseline. The implementations that exhibit the best consistency between $n = 2$ pairs and residual pairs are some of those that only showed one scenario group in Section IV-B1. They either always favor the oldest chunks or ignore all test cases. The third column shows that the custom algorithms explain more implementation triplet policies than the baseline (while also trying to predict triple overlaps). These implementation behaviors for $n \leq 3$ can thus all be abstracted and are easily explained. For example, Seastar merges any overlapping or non-overlapping IPv4 fragments and TCP segments after the overlap resolution. Another example is NetBSD 7.0, lwIP, FreeBSD 10.2 to 12.1, which drop the newest fragment when they ignore an IPv6 $n = 2$ test case (which explains the related observation in Section IV-B1). See the related mechanism explanation in Appendix D2. However, more than 10 stack policies cannot wholly be inferred for each protocol.

Takeaway

Most implementations show inconsistencies between $n = 2$ pair and $n = 3$ triplet test cases. This is mainly due to some reassembly mechanisms (for example, introduced for performance reasons) that have a side effect on the applied policy. Consequently, NIDSes must not use $n = 2$-based time policies alone to reassemble any n overlapping chunk sequences. PYROLYSE implements some custom reassembly algorithms composed of sets of observed reassembly mechanisms. They achieve reassembly consistency for only 25 of the 61 protocol stacks.

VII. DISCUSSION

A. Implementation reassembly policy compliance with RFC

IPv4 and TCP RFC [2], [3] do not specify the behavior implementations must adopt in the presence of fragment or segment data overlaps. However, in the TCP receive window definition, the specification says that "[...] segments overlapping the range RCV.NXT to RCV.NXT + RCV.WND - 1 carry acceptable data [...]. Segments containing sequence numbers entirely outside this range are considered duplicates or injection attacks and discarded". Consequently, even if the specification is unclear, we believe that accepting overlapping data segments is the most RFC-compliant behavior. PicoTCP, uIP, and mirage-tcpip do not reassemble overlapping segments.

Differently, IPv6 [4] states that implementations must drop the entire fragment flow if any of the bytes overlap. As reported in Section IV-A, picoTCP, uIP, mirage-tcpip, and smoltcp stacks ignored all the test cases; they are thus perfectly compliant with the standard. Even if the other IPv6 implementations but Solaris *ignore* about 80% of the test cases (see Section IV), we still observe overlapping test cases that are reassembled, which is non-compliant with the RFC but consistent with the related works [25], [26], [38].

B. NIDS challenges to address more than n = 2 overlapping chunks

Addressing various testing scenarii and exhaustive sequences of three overlapping chunks showed that implementation policies are much more diverse than previously thought.

Pair ($n = 2$) and triplet ($n = 3$) test case reassemblies are often inconsistent because reassembly mechanisms have side effects on triplet test case structure and thus reassembly. Therefore, $n = 2$-based policies cannot be used alone as they currently are inside configurable NIDSes. The generic reassembly algorithms we implemented abstract stack behaviors, but only achieve consistency with 25 of the 61 protocol stacks. The remaining algorithm inconsistencies may be due to some other non-implemented reassembly mechanisms and/or unwanted introduced behaviors that are very implementation-specific. One could investigate the former by implementing and testing mechanisms that have not yet been addressed. For 44 out of the 62 protocol stacks, the custom algorithms improve $n = 2$ and $n = 3$ consistency and explainability compared to the baseline method. This is promising but not satisfying from a NIDS perspective, since a unique inconsistency can lead to the NIDS bypass. Moreover, while $n > 3$ is not tested in this paper, we strongly suspect that similar stack reassembly inconsistencies between $n = 2$ and $n > 3$ would be observed if tested. Consequently, we recommend that the NIDSes that still wish to offer policy configurability use the $n = 2$ pair time policies in any case and raise an alert as soon as processing a $n > 2$ overlapping chunk sequence. The currently implemented $n = 2$ time policies inside the NIDSes should be enriched with all the testing scenarii addressed in this paper, since they reproduce real-world context. Furthermore, the research community should revisit overlapping chunk occurrence and structure in the wild since the last work on this topic was from 2008 [39]. This would help NIDS developers assess if alerts on $n > 2$ overlapping chunk sequences would produce false positive alerts.

VIII. Ethical considerations

A. Responsible Disclosure

Reassembly errors: Every reassembly error described in Section V was reported to the impacted implementation developers at least one month before this paper submission:

- OpenBSD fixed the IPv4 and TCP issues.
- Suricata completely fixed the TCP log display and partially addressed the reassembly logic-related errors. We informed Suricata developers that some bugs could still be observed with their fix, but did not receive any answer. The CVE-2024-32867 [1] was assigned to the IP error.
- mirage-tcpip is currently analyzing the reassembly error.
- Snort and lwIP did not respond to our disclosure.

The detailed process is described in Appendix E.

Implementation reassembly policies: To our knowledge, only Suricata and Snort NIDSes have configurable reassembly policies. Both IDSes based them on Novak and Sturges's works [6], [7], which date back to 2005 and 2007. Since this paper contains both an extension of prior OS reassembly policies [6], [7], [13] and new aspects (e.g., testing scenarii), we gradually communicated Suricata and Snort's results so they can address them progressively. We sent the first report and related artifacts (i.e., implementation policy descriptions) to Suricata and Snort in January 2024 and the last one in March 2025. We sent the same reports to Zeek, another widely deployed NIDS that previously offered the policy configurability feature. We did not receive any response.

B. Censorship Systems

Improving NIDS security and performance has the side effect of enhancing censorship systems. Some works showed that data overlaps could circumvent CSes until 2017 [14], [15]. But subsequent works noticed that this approach became ineffective [16], [17]. 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 outweighs the negative impacts on censorship elusion techniques.

IX. Conclusion

In this paper, we described PYROLYSE, which we used to test the behavior of a wide range of IPv4, IPv6, and TCP protocol implementations when processing overlapping chunk sequences in various testing scenarii. The tool identified eight reassembly errors during the testing campaign, including reassembly logic, connection termination, and NIDS log display errors. One was assigned a CVE [1], and five are completely or partially fixed. The obtained OS, embedded/IoT, unikernel, NIC, or DPDK-compatible stack reassembly policies with PYROLYSE revealed many distinct behaviors, as 15 IPv4, 14 IPv6, and 14 TCP policies were observed over the 23 tested implementations. Suricata, Snort, and Zeek NIDSes reassemble inconsistently with the other tested protocol stacks, except the TCP tests for Zeek with Windows OSes and Snort with FreeBSD 11.3 to 14.1. Since we did not find a way to explain the observed $n = 3$ overlapping test case reassemblies for all the protocol implementations, we state that the obtained policies cannot be extended from $n = 2$ pairs to any n overlapping IP fragment or TCP segment sequences. We provide precise related recommendations for the NIDSes.

Acknowledgment

This work was supported by a grant from the French National Cybersecurity Agency (ANSSI). We thank Olivier Levillain and Gregory Blanc for their help in chunk sequence modeling, and Zhengyu Zu for his work in OS TCP testing. We also thank CERT-FR that guided us through the responsible disclosure process as well as stack and NIDS developers for their constructive answers and corrections: Jason Ish and Victor Julien for Suricata, Alexander Bluhm for OpenBSD, Nicolas Tsiftes for uIP, Anil Madhavapeddy for mirage-tcpip, Thibaut Vandervelden for smoltcp, Tom Reu for Chelsio, and Michael Tuexen, Timo Voelker, Gleb Smirnoff, Rodney W. Grimes, and Olivier Cochard for FreeBSD. We are grateful to Kosek et al. [40] for their help with lwIP and uIP testing. Finally, we thank the reviewers for their insightful comments and our shepherd, Andrea Oliveri, for his guidance.

REFERENCES

[1] NIST. (2024) CVE-2024-32867. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-32867

[2] "Internet Protocol," RFC 791, 1981. [Online]. Available: https://www.rfc-editor.org/info/rfc791

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

[4] S. Deering and R. Hinden, "RFC 8200: Internet protocol, version 6 (ipv6) specification," 2017.

[5] T. Ptacek and T. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," Secure Networks, Inc, Tech. Rep., 1998.

[6] J. Novak, *Target-based fragmentation reassembly*, 2005.

[7] J. Novak and S. Sturges, *Target-based tcp stream reassembly*, 2007.

[8] Suricata. [Online]. Available: https://suricata.io/

[9] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks." 1999.

[10] Suricata reassembly policies. [Online]. Available: https://docs.suricata.io/en/suricata-7.0.4/configuration/suricata-yaml.html#host-os-policy

[11] Snort ip reassembly policies. [Online]. Available: https://snort.org/faq/readme-frag3

[12] Snort tcp reassembly policies. [Online]. Available: https://snort.org/faq/readme-stream5

[13] L. Aubard, J. Mazel, G. Guette, and P. Chifflier, "Overlapping data in network protocols: bridging OS and NIDS reassembly gap," in *DIMVA*, 2025.

[14] S. Khattak, M. Javed, P. D. Anderson, and V. Paxson, "Towards illuminating a censorship monitor's model to facilitate evasion," in *FOCI*, 2013.

[15] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. Krishnamurthy, "Your state is not mine: A closer look at evading stateful internet censorship," in *ACM IMC*, 2017.

[16] K. Bock, G. Hughey, X. Qiang, and D. Levin, "Geneva: Evolving censorship evasion strategies," in *ACM CCS*, 2019.

[17] Z. Wang and S. Zhu, "SymTCP: eluding stateful deep packet inspection with automated discrepancy discovery," in *NDSS*, 2020.

[18] NIST. (2021) CVE-2021-37592. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-37592

[19] ——. (2024) CVE-2024-55629. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-55629

[20] ——. (2024) CVE-2024-37151. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-37151

[21] F. Li, A. Razaghpanah, A. M. Kakhki, A. A. Niaki, D. Choffnes, P. Gill, and A. Mislove, "lib.erate(n) a library for exposing (traffic-classification) rules and avoiding them efficiently," in *ACM IMC*, 2017.

[22] Z. Wang, S. Zhu, K. Man, P. Zhu, Y. Hao, Z. Qian, S.-V. Krishnamurthy, T. La Porta, and M.-J. De Lucia, "Themis: Ambiguity-aware network intrusion detection based on symbolic model comparison," in *MTD*, 2021.

[23] K. Bock, G. Hughey, L.-H. Merino, T. Arya, D. Liscinsky, R. Pogosian, and D. Levin, "Come as you are: Helping unmodified clients bypass censorship with server-side evasion," in *ACM SIGCOMM*, 2020.

[24] K. Bock, G. Naval, K. Reese, and D. Levin, "Even censors have a backup: Examining china's double https censorship middleboxes," in *ACM SIGCOMM*, 2021.

[25] A. Atlasis, "Attacking ipv6 implementation using fragmentation," *Black Hat*, 2012.

[26] E. Di Paolo, E. Bassetti, and A. Spognardi, "A New Model for Testing IPv6 Fragment Handling," in *ESORICS*, 2023.

[27] U. Shankar and V. Paxson, "Active mapping: Resisting nids evasion without altering traffic," in *SP*, 2003.

[28] I.-C. Oprea, "Lost in reassembly: Exploiting ip fragmentation in computer networks," Master's thesis.

[29] P. Manev and A. Herz, "Suricata Extreme Performance Tuning: SepTun Mark III," Presented at Suricon 2024. [Online]. Available: https://suricon.net/wp-content/uploads/2024/12/SuriCon2024-Peter-Manev_Andreas-Herz_Suricata-Extreme-Performance-Tuning-SepTun-Mark-III.pdf

[30] Sparq tool. [Online]. Available: https://github.com/dwolter/SparQ

[31] Tcpreplay tool. [Online]. Available: https://tcpreplay.appneta.com/

[32] tcpdump tool. [Online]. Available: https://www.tcpdump.org

[33] Vagrant cloud. [Online]. Available: https://app.vagrantup.com/boxes/search

[34] "PYROLYSE tool and paper artifacts," 2025. [Online]. Available: https://github.com/ANSSI-FR/pyrolyse

[35] A. Bluhm, "OpenBSD patch related to pf's IP reassembly error." [Online]. Available: https://github.com/openbsd/src/commit/9915416fe885bd67bf19209c5fe2f9bbbb7191ab

[36] ——, "OpenBSD patch related to pf's TCP reassembly error." [Online]. Available: https://github.com/openbsd/src/commit/12e4c257ea8089c7f56eb0c2c4493a0458273f4f

[37] "The try_to_coalesce() function source code." [Online]. Available: https://elixir.bootlin.com/linux/v6.1/source/net/ipv4/tcp_input.c#L4653

[38] B. Lin, L. Zhang, Y. Guo, H. Zhang, and Y. Fang, "Research on security protection evasion mechanism based on ipv6 fragment headers," in *IEEE LCN*, 2024.

[39] W. John and T. Olovsson, "Detection of malicious traffic on back-bone links via packet header analysis," *CWIS*, 2008.

[40] M. Kosek, L. Blöcher, J. Rüth, T. Zimmermann, and O. Hohlfeld, "MUST, SHOULD, DON'T CARE: TCP Conformance in the Wild," in *International Conference on Passive and Active Network Measurement*. Springer, 2020, pp. 122–138.

[41] D. Song, "Fragroute tool," 2002. [Online]. Available: https://www.monkey.org/~dugsong/fragroute/

[42] "INTANG tool," 2017. [Online]. Available: https://github.com/seclab-ucr/INTANG

[43] "Geneva tool," 2019. [Online]. Available: https://github.com/Kkevsterrr/geneva

[44] "SymTCP tool," 2020. [Online]. Available: https://github.com/seclab-ucr/SymTCP

[45] "Themis tool," 2021. [Online]. Available: https://github.com/seclab-ucr/Themis

[46] "Ipv6-fragmentation tool," 2023. [Online]. Available: https://github.com/netsecuritylab/ipv6-fragmentation

[47] "Ipv6-fragmentation tool," 2023. [Online]. Available: https://github.com/linbin89/FragEva6-Guard

[48] S. Krishnan, "RFC 5722: Handling of Overlapping IPv6 Fragments," 2009.

[49] "Oprea's tool," 2025. [Online]. Available: https://github.com/OpreaCristian2002/reassembly-policies-5g-using-ipv6

[50] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain validation++ for mitm-resilient pki," in *CCS*, 2018.

[51] X. Zheng, C. Lu, J. Peng, Q. Yang, D. Zhou, B. Liu, K. Man, S. Hao, H. Duan, and Z. Qian, "Poison over troubled forwarders: A cache poisoning attack targeting {DNS} forwarding devices," in *USENIX Security*, 2020.

[52] X. Feng, Q. Li, K. Sun, K. Xu, B. Liu, X. Zheng, Q. Yang, H. Duan, and Z. Qian, "PMTUD is not panacea: Revisiting IP fragmentation attacks against TCP," in *NDSS*, 2022.

APPENDIX

A. Related works comparison

Table VII compares the related works and present paper contributions.

B. Additional information regarding the checksum-impactless patterns

For the IP testing, the IP fragments must convey the ICMP header, which contains a *checksum* field. The ICMP checksum is computed with the 2-byte one's complement of the one's complement of the ICMP header and data. Besides using its header and data, ICMPv6 also uses an IP pseudo-header to compute its checksum. This pseudo-header is constructed with the source and destination IP addresses, the protocol, and the ICMPv6 length.

a) Requirements: First, the overlapping data portions must be different to distinguish which chunk data is preferred. Second, the patterns must be 8-byte-long since this corresponds to the IP *Fragment Offset* unit. Third, the IP upper-layer checksum must be correct no matter the final test case

TABLE VII

SUMMARY REGARDING OVERLAP-BASED WORKS. S^{IP} (RESP. S^{TCP}) CORRESPONDS TO THE 42 IP (RESP. 11 TCP) SCENARII INTRODUCED IN SECTION III-A2. * MEANS THAT THE SOURCE CODE IS PRESENT INSIDE THE PAPER. (1) MEANS SEVERAL CHUNKS COMPOSED THE TEST CASES BUT THE CHUNKS OVERLAP ONLY PAIRWISE (IT REFERS TO THE *multiple* MODE IN [13]). (2) MEANS THE METHOD/TOOL SHOULD BE EXHAUSTIVE IN THEORY, BUT IS NOT BECAUSE IT WOULD EXHAUST RESOURCES OR REQUIRE SIGNIFICANT CHANGES TO THE TESTING METHOD/TOOL. (3) MEANS THE AUTHORS INSTRUMENTED THE NETWORK TARGET (I.E., NON-OPAQUE-BOX APPROACH); THUS, IT REQUIRES SIGNIFICANT EFFORT TO ADAPT THE TOOL TO OTHER NETWORK TARGETS.

Author	Work	Year	Protocol	Testing scenario	n overlaps	Test case exhaustivity	Tool characteristics: available	Tool characteristics: target extensible	Tool characteristics: n extensible	Reassembly policy format	Target stack: midpoint	Target stack: network
Ptaceck and Newsham	[5]	1998	IPv4/TCP	s^c	2	✗	✓ [41]	✗	✗	natural language	NIDS	OS
Shankar and Paxson	[27]	2003	IPv4	s^c	$2^{(1)}$	✗	✗	✗	✗	natural language		OS/router/ printer
			TCP	s^{sf}	2	✗						
Novak and Sturges	[6]	2005	IPv4	s^c	2	✓	*	✗	✗	natural language		OS
	[7]	2007	TCP	s^{sf}								
Atlasis	[25]	2012	IPv6	s^c_{nf} s^c_{mnf}	3	✗	*	✗	✗	figures		OS
Khattak et al.	[14]	2013	IPv4/TCP	?	2	✓	✗	✗	✗	natural language	CS	
Wang et al.	[15]	2017	IPv4	s^{sf}_{af}	2	✗	✓ [42]	✓	✗	natural language	CS	
			TCP	s^{sf} s^c	2	✗						
Bock et al.	[16]	2019	IPv4/TCP	-	-	✗(2)	✓ [43]	✓	-	Geneva's genetic building blocks + natural language	CS	
	[23]	2020										
	[24]	2021										
Wang et al.	[17]	2020	TCP	-	2/3	✗(2)	✓ [44]	✓(3)	✓	natural language	CS/NIDS	OS(3)
	[22]	2021	TCP	-	3	✗(2)	✓ [45]	✓(3)	✓	natural language		OS(3)
Di Paolo et al.	[26]	2023	IPv6	s^c_{af}	$2^{(1)}$	✗	✓ [46]	✓	✗	nb. of target responses		OS
Lin et al.	[38]	2024	IPv6	s^c	$2^{(1)}$	✗	✓ [47]	✓	✗	compliance with RFC 5722 [48]	NIDS/ firewall	OS
Oprea	[28]	2025	IPv6	s^c_{af}	$2^{(1)}$	✗	✓ [49]	✓	✗	nb. of exploitable overlap. sequences	NIDS	OS
Us	[13]	2025	IPv4/IPv6	s^c_{nf}	2	✓	✗	✗	✗	Allen relations + pair time policies	NIDS	OS
			TCP	s^{sf}								
Us	**-**	**-**	**IPv4/IPv6**	S^{IP}	**2/3**	✓	✓ [34]	✓	✓	**Allen relation sequence + pair/triplet time policies**	**NIDS**	**OS/IoT/ unikernel/NIC/ DPDK-comp.**
			TCP	S^{TCP}								

reassembly (i.e., no matter the chosen overlapping portion pattern and no matter the reassembled ICMP Echo Request payload length). As a result, we need *unique 8-byte-long patterns that do no impact on the IP upper-layer checksum.*

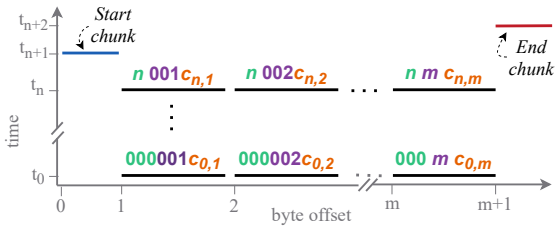


Fig. 11. The checksum-impactless patterns, in a $s^{sf,ef}$ testing scenario. The first three bytes (green parts) correspond to the chunk ID. The following three bytes (purple parts) correspond to the pattern offset. The last two bytes (orange parts) are the checksum correction bytes.

b) Design: Figure 11 illustrates the set of chunk patterns that are generated inside PYROLYSE. The first six bytes of a pattern correspond to its unique ID while the last two bytes are the IP upper-layer checksum correction bytes. Since the checksum is a 2-byte word sum, we can manipulate the last two bytes so that any pattern ones' complement contribution to the upper-layer checksum is null (as already outlined for UDP and TCP upper-layers in related works [50]–[52]). Because the ICMPv6 length is also used to compute the checksum, the last IPv4 and IPv6 patterns' two bytes are different for the same pattern ID.

C. Additional information regarding the targeted implementations

The Table VIII describes the protocol implementation that are targeted in the present paper.

D. Custom reassembly algorithm test

1) The pipeline: Figure 12 describes the overall pipeline used for the test of the implemented reassembly algorithms. The pipeline is illustrated for a specific testing scenario, since pair and triplet time policies may change across scenarii. In other words, the process is repeated for every scenario.

Let's take the $s^{sf,ef}$ and Debian 12 OS as an example for the IPv4 protocol. First (cf ① on Figure 12), we run Debian 12 test in order to retrieve the $n = 2$ time policies. We get

TABLE VIII
PROTOCOL IMPLEMENTATION TARGET DESCRIPTIONS.

Implementation			Addressed protocol	Comments
Type	**Name**	**Version**		
OS	Windows	<2022/10/25 build	IPv4/IPv6/TCP	-
		≥2022/10/25 build		
	Debian	8	IPv4/IPv6/TCP	-
		9 → 12		
	FreeBSD	10.2 → 11.2	IPv4/IPv6/TCP	The parameters related to IP reassembly are dynamically computed based on the FreeBSD host's RAM. In RAM scenarii lower than 1GB, we observe IP reassembly inconsistencies across the runs. We do not oberve any reassembly difference with pf, ipfw, and ipfilter firewalls enabled/disabled.
		11.3 → 12.1		
		12.2 → 14.1		
	OpenBSD	6.0	IPv4/IPv6/TCP	We observe reassembly differences with pf firewall enabled/disabled. The reassemblies provided in the paper correspond to OpenBSD reassemblies *with* pf enabled, since it is enabled by default. OpenBSD's reassemblies without pf enabled can be found at [34].
		6.1 → 7.6		
	NetBSD	7.0	IPv6/TCP	IPv4 test cases with fragment sizes lower than 40 bytes do not receive any answer. We thus hypothesize the used patterns are not well suited to test NetBSD IPv4 policy. We do not observe such behavior with IPv6 or TCP. We do not observe any reassembly difference with npf firewall enabled/disabled.
		8.0 → 9.3	IPv6/TCP	
	Solaris	11.2 → 11.4	IPv4/IPv6/TCP	-
IoT/ embedded	lwIP	2.2.1	IPv4/IPv6/TCP	-
	picoTCP	1.7.0	IPv4/IPv6/TCP	-
	uIP	1.0	IPv6/TCP	IPv4 test cases are reassembled inconsistently across the tool runs. We thus could not extract the implementation policies.
	smoltcp	0.12.0	IPv4/IPv6/TCP	We increased the default value of the REASSEMBLY_BUFFER_COUNT parameter from 1 to 2000. The default value leads to run inconsistencies for IPv4 tests.
Unikernel	mirage-tcpip	9.0.0	IPv4/IPv6/TCP	-
DPDK-compatible	Seastar	22.11.0	IPv4/TCP	Seastar does not implement IPv6.
NIC	Chelsio TOE	T520-CR	TCP	There is no IPv4 and IPv6 fragment offloading; therefore, fragments are forwarded as is to the OS kernel.
	Xilinx Onload	9.0.1	TCP	
NIDS	Suricata	7.0.9	IPv4/IPv6/TCP	These NIDSes have configurable reassembly policies. Sections IV and VI-C report NIDS reassemblies with the *default* policies. Sections V and VI-A report NIDS reassemblies with the *configured* policies.
	Snort	3.7.1.0	IPv4/IPv6/TCP	
	Zeek	7.1.1	IPv4/IPv6/TCP	This NIDS does not provide the configurable reassembly policies.

an association between individual Allen relations (or pair test cases, since $n = 2$ test cases are composed of one unique Allen relation) and pair time policies, e.g., Eq is associated to Old, Fi to Old, F to $Ignore$. Second (cf ② on Figure 12), on one hand, we test *individually* the implemented reassembly algorithms with $n = 3$ test cases, based on Debian 12's $n = 2$ time policies; on the other hand, we test Debian 12 with $n = 3$ test cases. Third (cf ③ on Figure 12), this finally enable us

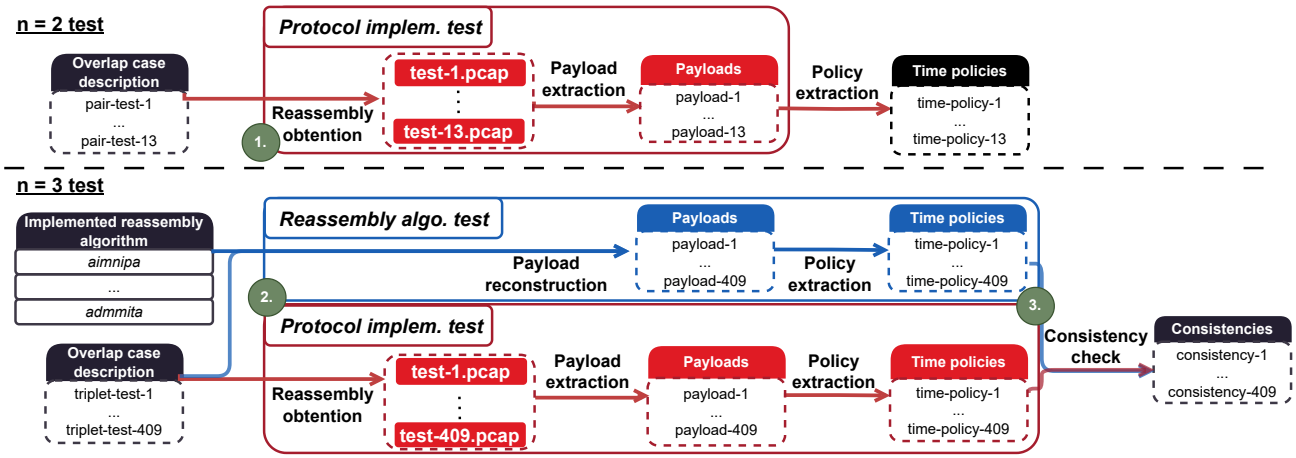


Fig. 12. The pipeline for the reassembly algorithm test. This process is repeated for every (protocol, scenario, implementation) tuple. Black items are input or output elements.

to get the number of test case reassembly inconsistencies for every implemented algorithm by comparing their reassemblies with Debian 12.

2) The observed reassembly mechanims with side effect: This part describes the reassembly mechanisms with side effect we observed within protocol implementation source codes. This manual investigation (temporarily) diverges from the opaque box approach described in Section III.

a) Chunk merging for {no, meet, any} relations: When there is no data hole between chunks, an implementation can merge them, for performance reason for example, as we've seen with Linux 6.1 in Section VI-B2. An implementation may want to merge only in some very specific situations. That is why we implemented three merging strategies:

- no chunk merging, i.e., *no* strategy.
- chunk merging only for perfectly contiguous chunks, in other words M or Mi relations, i.e., *meet* strategy.
- chunk merging for any Allen relation (except B and Bi relations), i.e., *any* strategy.

b) Chunk characteristic {immediate, delayed} alteration during overlap resolution: When a new chunk arrives and is compared with queued chunks before its (possible) insertion, the alteration of the newly arrived chunk's characteristics, i.e., beginning and ending data offsets, are modified either *immediately* or *delayedly*. More precisely, if the chunk overlaps with several ones, either 1) its beginning and ending offsets are adjusted after every overlap comparison with already inserted chunks or 2) its characteristics stay unchanged for the remaining comparisons until its (possible) insertion.

c) Interpretation of the ignore *time policy as {drop of all the triplet's chunks, drop of the pair's chunks, drop of the oldest pair chunk, drop of the newest pair chunk}:* An implementation can *ignore* a test case without any hole by not answering back. Because we treat the tested implementations as opaque boxes, we do not know exactly what caused the implementation not to respond to the test case. For example, IPv6 specification [4] says that all the fragments that corresponds to the same original datagram should be silently discarded. In the $n = 3$ testing context, if the time policy of at least one of the relations describing a triplet test case is *ignored* (based on $n = 2$ testing), this could either mean that:

1. all the test case chunks are dropped and thus, the entire triplet test case is *ignored*. In this case, we consider that no matter if p_{01}, p_{02} or p_{12}'s time policy is *ignore*, the information is stored and, therefore, all the following chunks are dropped. In other words, the flow is not reset (i.e., the implementation keeps the information that a pair has been ignored).
2. only the corresponding pair's chunks are dropped. The flow is then reset. Therefore, if we consider a triplet where p_{01} is ignored and a chunk 2 with data starting at byte offset 0 and finishing the rightmost, the triplet test case can ultimately be reassembled with chunk 2's data.

Finally, an implementation that *ignores* the pair p_{ij} in a triplet test case can also:

3. drop the oldest, i.e., i^{th}, chunk.
4. drop the newest, i.e., j^{th}, chunk.

3) List of reassembly algorithms: This part list the IP and TCP reassembly algorithms we implemented and tested. In our case, a reassembly algorithm is basically a set of reassembly mechanism with side effet described in Appendix D2.

a) IP protocols: We implemented the 12 reassembly algorithms reported in Table IX.

We did not implement the *oldest chunk pair drop* ignoring strategy because of its development effort. And, we do not associate the *delayed* chunk characteristic alteration with *all chunk pair drop* and *newest chunk pair drop* because such association does not always make sense.

TABLE IX
IMPLEMENTED IP REASSEMBLY ALGORITHMS.

Algorithm	**Reassembly mechanism**		
	characteristic alteration	**relation merging**	**drop *ignore* interpretation**
aimnipa	immediate	no	all chunk pair
aimnipn	immediate	no	newest chunk pair
aimnita	immediate	no	all chunk triplet
admnita	delayed	no	all chunk triplet
aimaipa	immediate	any	all chunk pair
aimaipn	immediate	any	newest chunk pair
aimaita	immediate	any	all chunk triplet
admaita	delayed	any	all chunk triplet
aimmipa	immediate	meet	all chunk pair
aimmipn	immediate	meet	newest chunk pair
aimmita	immediate	meet	all chunk triplet
admmita	delayed	meet	all chunk triplet

TABLE X
IMPLEMENTED TCP REASSEMBLY ALGORITHMS.

Algorithm	**Reassembly mechanism**	
	characteristic alteration	**relation merging**
aimn	immediate	no
admn	delayed	no
aima	immediate	any
adma	delayed	any
aimm	immediate	meet
admm	delayed	meet

b) TCP protocol: We implemented the six reassembly algorithms reported in Table X.

We did not implement the mechanism related to the $Ignore$ pair time policy interpretation, since we did not observe in practice such time policy for TCP.

E. Detailed responsible disclosure process

Table XI give details regarding the responsible process.

TABLE XI
RESPONSIBLE DISCLOSURE SUMMARY. PROVIDED ARTIFACTS ARE COMPOSED OF A TECHNICAL REPORT AND THE RELATED PCAP OR LOG MATERIALS.

Disclosure type	**Date**	**Recipient**	**Description**	**Comments**
Reassembly errors/ bugs	06/11/2023	Suricata	IP reassembly logic error	Partially fixed, credited with CVE [1]
	06/11/2023	Snort	IP reassembly logic error	Not fixed
	06/11/2023	Suricata	TCP log display error	Partially fixed
	06/12/2024	Suricata	TCP reassembly logic error	Partially fixed
	24/01/2025	OpenBSD	IP reassembly logic error	Fixed [35]
	28/01/2025	OpenBSD	TCP connection termination error	Fixed [36]
	27/03/2025	lwIP	TCP connection termination error	Not fixed
	27/03/2025	mirage-tcpip	TCP connection termination error	Not fixed
($n = 2$) pair policy update and extension	29/01/2024	Suricata/Snort/ Zeek	Latest OS and NIDS pair reassembly policies obtained with the IP s^c_{nf} and TCP s^{sf} scenarii. It reported the NIDS reassembly inconsistencies with the latest OSes.	Suricata fixed its default, i.e., bsd, IP reassembly policy [1]
	26/08/2024	Suricata/Snort/ Zeek	OS pair reassembly policies obtained with the S^{IP} and S^{TCP} testing scenarii. It reported the reassembly diversity across testing scenarii.	No answer
($n = 3$) triplet policy description	26/02/2025	Suricata/Snort/ Zeek	Latest OS triplet reassembly policies obtained with the S^{IP} and S^{TCP} testing scenarii. It reported the reassembly inconsistency between ($n = 2$) pairs and ($n = 3$) triplets test cases. The custom reassembly algorithms were described and tested.	No answer
	31/03/2025	Suricata/Snort/ Zeek	Triplet reassembly policies of older OS versions, NIDS, and all the other network stacks. Policies are obtained with the S^{IP} and S^{TCP} testing scenarii. It reported the reassembly inconsistency between ($n = 2$) pairs and ($n = 3$) triplets test cases. The custom reassembly algorithms were described and tested.	No answer