COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT

UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : *Informatique*

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

Recouvrements de données dans les protocoles IPv4, IPv6 et TCP : exploration des réassemblages de divers piles réseaux et supervision réseau

Par **Lucas AUBARD**

Thèse présentée et soutenue à Rennes, le 12 janvier 2026
Unité de recherche : IRISA

Rapporteurs avant soutenance

Georgios SMARAGDAKIS	Full professor à l'Université Technique de Delft, Pays-Bas
Hervé DEBAR	Professeur à Télécom SudParis, France

Composition du jury

Président :	Yassine HADJADJ-AOUL	Professeur à l'Université de Rennes, France
Rapporteurs :	Georgios SMARAGDAKIS	Full professor à l'Université Technique de Delft, Pays-Bas
	Hervé DEBAR	Professeur à Télécom SudParis, France
Examinateurs :	Vincent NICOMETTE	Professeur à l'INSA de Toulouse, France
	Johanna AMANN	Open Source Architect chez Corelight, Royaume-Uni
Dir. de thèse :	Gilles GUETTE	Professeur à l'IMT Atlantique, France
Co-encadrants de thèse :	Johan MAZEL	Inspecteur chez ANSSI, France
	Pierre CHIFFLIER	Inspecteur chez ANSSI, France

Remerciements

Mon marathon "thèse" s'est achevé le 12/01/2026, et il est certain que sans l'équipe de qualité qui m'a entourée ces quelques années, le résultat final aurait été bien différent.

Tout d'abord, je vous suis pronfondémment reconnaissant, Johan MAZEL, Gilles GUETTE et Pierre CHIFFLIER, pour votre confiance, votre disponibilité et votre expertise. Je vous remercie pour toutes les réponses que vous avez pu apportés aux différents défis de cette thèse (e.g., divulgation responsable, publications, outillage) et qui se seront révélées avoir été très pertinentes. À vos côtés, j'ai beaucoup appris, et je suis prêt à appliquer ces nouvelles connaissances dans mes prochaines aventures professionnelles. Je remercie également mon encadrant de mobilité, Kensuke FUKUDA, pour l'opportunité et son accueil chaleureux.

Ensuite, je remercie les membres de l'équipe "PIRAT\');" (anciennement "CIDRE") d'avoir été si accueillant et de faire régner une ambiance conviviale. J'ai beaucoup apprécié les midis jeux, les parties de babyfoot et autres "mercredis pizza"! Je veux spécialement vous remercier, Maxime LANVIN, Matthieu BATY et Manuel POISSON, pour vos relectures attentives et constructives.

Je tiens à vous remercier particulièrement, Héloïse, Audrey, Gabriel, Sébastien et Kévin, pour votre soutien indispensable, dans les bons et les moins bons moments de la thèse. Nos discussions m'ont permit d'appréhender certains aspects avec du recul et de surmonter le "mur" de la deuxième année. Merci à mon groupe de musique SOUL qui m'a aussi permit de m'évader (de manière non protocolaire, cette fois-ci) chaque semaine.

Enfin, je remercie ma famille de m'avoir motivé et supporté tout au long de mes études. Un merci spécial à mon frère Martin pour avoir démystifier le doctorat dans la famille et pour ses précieux conseils. Et merci à Arthur pour ton soutien et tes encouragements au quotidien.

Résumé substantiel en français

L'ère de l'information a débuté au milieu du XX^{ème} siècle et se caractérise par le passage des industries traditionnelles à une économie centrée sur les technologies de l'information. Internet a largement favorisé l'avènement de cette ère. Les protocoles de communication IP et TCP ont été, dès le début, des protocoles clés qui ont soutenu les évolutions du trafic Internet. Les systèmes d'information (SI) ont rapidement été abusé par différents acteurs (universitaires, étatiques, criminels) et, la protection des SI est devenue en même temps qu'un enjeu prioritaire, un secteur économique (cybersécurité) à part entière.

Parmi les moyens existants permettant de se prémunir contre les cyberattaques, la supervision réseau est une technique largement employée pour la surveillance de réseaux de plusieurs dizaines, centaines ou même milliers de machines (par ex., les réseaux d'entreprises ou d'administrations). Elle permet de détecter les attaques (détection d'intrusion), et même de réagir si cette supervision dispose d'une composante préventive (prévention des intrusions). Les systèmes de détection ou de prévention d'intrusion (IDS ou IPS) utilisent différentes approches: l'approche par signatures et l'approche par anomalies. La première utilise une base de connaissance composée de signatures d'attaques s'étant déjà produites. La seconde utilise, la plupart du temps, des techniques d'apprentissage automatique pour apprendre les comportements légitimes des utilisateurs du SI à protéger pour, dans un second temps, détecter les comportements anormaux. Ces IDS traitent différents types de données, comme par exemple les paquets réseaux, issus des communications locales ou distantes, ou encore les données systèmes des machines des utilisateurs. Les IDS utilisant du trafic réseau sont appelés systèmes de détection réseau, ou NIDS, et ceux traitant les données systèmes, HIDS.

En 1998, Ptaceck et Newsham ont décrit deux classes d'attaque pouvant cibler les NIDS: les attaques par insertion et les attaques par évasion [1]. Elles exploitent toutes deux les ambiguïtés protocolaires qui surviennent lorsque la RFC d'un protocol (i.e., sa spécification) n'est pas claire ou n'est pas respectée. Dans de tels cas, l'hôte qui supervise (NIDS) et celui qui est supervisé divergent dans leur manière de traiter un paquet réseau, ce qui entraîne la désynchronisation des flux de données générés par les deux machines. L'une de ces ambiguïtés utilise les recouvrements de données permis

par la fragmentation IP et la segmentation TCP.

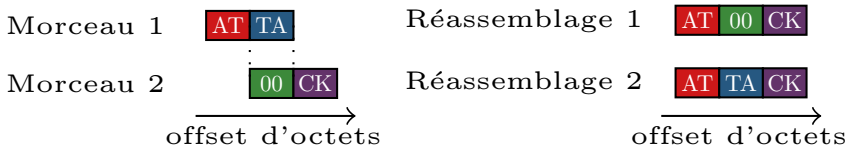


Figure 1: Illustration de l'ambiguïté de recouvrement de données.

La figure 2 est une illustration de l'ambiguïté de recouvrement de données. Les données réassemblées ici diffèrent selon le morceau de données privilégié, ou politique de réassemblage. Un attaquant peut exploiter les divers réassemblages possible pour dissimuler une charge utile malveillante. Si le NIDS reconstruit les données selon la stratégie 1 alors que l'hôte supervisé réassemble selon la stratégie 2, le premier ne peut pas détecter les données malveillantes et donc, ne peut pas lever d'alerte de sécurité. Ptaceck et Newsham ont démontré que les NIDS testés pouvaient être victimes d'attaques basées sur le recouvrement de données, car les piles IPv4 et TCP des systèmes d'exploitation (OS) testés réassemblent différemment les morceaux recouvrants [1]. Depuis ces travaux, les NIDS ont proposé deux contre-mesures : lever une alerte à chaque recouvrement de données ou appliquer des politiques de réassemblage différenciées en fonction de l'hôte supervisé.

Cette thèse a pour but l'amélioration des systèmes de détection d'intrusion réseaux (NIDS) face aux attaques par insertion et par évasion utilisant les ambiguïtés de recouvrement IP et TCP. Nous menons donc une analyse multidimensionnelle concernant l'ambiguïté de recouvrement de données dans les protocoles IPv4, IPv6 et TCP.

Caractéristiques des recouvrements IPv4 dans des captures Internet

Selon qu'un recouvrement de donnée est intrinsèquement considéré comme malveillant ou non, les contre-mesures appliquées au sein d'un NIDS peuvent différer. Certains travaux ont rapporté et mesuré la présence de recouvrements IPv4 et TCP jusqu'en 2008 [2]–[4]. D'autres travaux [5], [6] décourageaient l'utilisation de la fragmentation IPv4 en raison de son abus potentiel (par ex., évasion ou insertion de NIDS ou de parefeux, déni de service) ou de son impact sur les performances de la communication. Dans le chapitre 3, nous vérifions si la fragmentation IPv4 est toujours présente sur Internet et nous fournissons des mesures actualisées des recouvrements IPv4, à partir de traces capturées au niveau d'une dorsale (ou "backbone link", en anglais).

RQ.1 – Quelles sont les caractéristiques des fragments IPv4 dans la nature, et dans quelle mesure les recouvrements de fragments se produisent-ils ?

Notre analyse de traces Internet issues du jeu de données MAWI [7] et capturées de 2007 à 2024 montre que la fragmentation IPv4 représente entre quelques dixièmes de pourcents et trois pourcents des paquets, chaque année. Nous notons que certains épisodes, tels que l'évolution de services (par ex., UDP/DNSSEC) ou l'introduction de nouveaux services (par ex., UDP/QUIC), s'appuient, plus ou moins temporairement, sur la fragmentation IPv4. Des PMTU variant de 1 396 à 1 500 octets (signe de PMTU liés à des réseaux mobiles et à des tunnels VPN) expliquent également que la fragmentation est observée au fil des années. Même si quelques hôtes concentrent la majeure partie du trafic fragmenté, entre 493 et 25 567 hôtes ont envoyé du trafic IPv4 qui s'est retrouvé être fragmenté. Dans les traces que nous avons analysé, 0.3% (resp. de 3.0 à 5.2%) des séries de fragments contiennent des fragments qui se recouvrent partiellement (resp. complètement). Selon la période, nous avons également mesuré qu'entre 0.1 et 0.4% des hôtes envoient des fragments susceptibles de se recouvrir au niveau de l'hôte destinataire, car ils réutilisent le même `IPID` dans les 5 secondes pour différents datagrammes IPv4 originaux. De tels hotes ne se conforment pas aux RFC [8], [9], car ils ne limitent pas leur débit ou utilisent un `IPID` constant pour des datagrammes non atomiques (c'est-à-dire, des datagrammes pouvant être fragmentés sur le chemin liant les deux hôtes).

Les NIDS ne doivent donc pas ignorer les fragments IPv4 lors de la reconstruction des flux de données des hôtes surveillés. Il est également possible de lever une alerte de sécurité lorsqu'un recouvrement partiel ou complet de fragments se produit, mais cela peut entrainer un nombre non-négligeable de faux positifs.

Politiques de réassemblage IPv4, IPv6 et TCP de divers piles réseaux, et attaque de NIDS par évasion et insertion

Dans le chapitre 2, nous étendons le modèle d'attaque par insertion et évasion introduits par les travaux précédents afin qu'il corresponde mieux à l'exploitation de l'ambiguïté de recouvrement IP et TCP. Les attaques par évasion (où la cible est un hôte surveillé par le NIDS) et par insertion (où la cible est le NIDS) visent à désynchroniser le flux de données du NIDS et de l'hôte supervisé. Nous identifions quatre scénarios d'attaque d'évasion ou d'insertion, qui contraint les portions de données recouvrantes. Si le NIDS

ou l'hôte surveillé ignore les paquets qui se recouvrent, le recouvrement peut, en fait, contenir n'importe quelle donnée. Sinon, cette partie recouvrante doit être correcte tant sur le plan sémantique que syntaxique. Enfin, nous expliquons que lorsqu'un protocole à état est utilisé au dessus des fragments IP ou des segments TCP, la désynchronisation du NIDS et du flux de données de l'hôte surveillé peut persister, même après l'attaque par recouvrement.

Dans le chapitre 4, nous décrivons les politiques de réassemblages de multiples piles réseaux. Nous évaluons également dans quelle mesure les NIDS Suricata, Snort et Zeek, qui sont trois NIDS libres et reconnus, peuvent faire l'objet d'attaques par recouvrements IP et TCP. Pour ce faire, nous introduisons un nouveau modèle de test *complet* des piles réseaux qui permet i) d'atteindre l'exhaustivité en terme de recouvrements, et ce, pour n'importe quel nombre n de morceaux recouvrants, avec l'algèbre des intervalles d'Allen [10], ii) de contextualiser les cas de test recouvrants avec des scénarii de test (42 pour IP et 11 pour TCP), iii) de remplir les données IP et TCP des cas de test avec des patrons spécifiques, iv) de décrire les politiques des piles via un formalisme temporel. Nous testons et décrivons les politiques d'un grand nombre de piles réseaux (OS, Internet des Objets, unikernel, DPDK et cartes réseaux) et des NIDS Suricata, Snort et Zeek pour les paires et les triplets (i.e., $n \leq 3$) de morceaux recouvrants. L'outils PYROLYSE développé durant cette thèse et les artefacts relatifs sont publiquement accessibles [11], [12].

RQ.2 – Dans quelle mesure les réassemblages des piles IPv4, IPv6 et TCP varient et sont corrects lors de l'exploration exhaustive de plusieurs (jusqu'à $n = 3$ ici) morceaux recouvrants et de divers contextes de test ?

Les politiques de réassemblage obtenues sont bien plus diverses que ce que les travaux de l'état de l'art avaient précédemment décrit. Nous avons, en effet, découvert 12 comportements différents pour 15 piles IPv4 testées, 14 pour 17 piles IPv6 et 14 pour 20 piles TCP, quand l'état de l'art obtenaient 5 politiques IP et 6 politiques TCP différentes. Tester de manière exhaustive l'ensemble des recouvrements possible pour deux ou trois morceaux recouvrants (i.e., $n \leq 3$) avec des contextes de test variés a permis de montrer que presque chaque implémentation dispose de sa propre politique de réassemblage pour les recouvrements de données. En outre, non seulement certaines de ces piles ont leur propre politique, mais elles réassemblent certains cas de test recouvrants avec des erreurs. De fait, nous avons découvert huit erreurs de réassemblage que nous avons rapporté aux éditeurs concernés (i.e., Suricata, Snort, OpenBSD,

lwIP et mirage-tcpip). Une CVE [13] a été assigné à une des erreurs impactant Suricata et, au moment de l'écriture du présent manuscrit, la moitié de ces erreurs ont été partiellement ou complètement corrigé.

RQ.3 – Les NIDS Suricata, Snort et Zeek peuvent-ils faire l'objet d'attaques par recouvrement de données lorsqu'ils surpervisent des hôtes avec des OS multi-usages (par ex., Windows, Linux, FreeBSD) ?

Lorsque les configurations par défaut de Suricata, Snort et Zeek sont utilisées, seulement Snort et Zeek sont parfaitement cohérents avec l'un des OS testés, et ce, pour TCP. Snort est cohérent avec la dernière version de FreeBSD et Zeek avec toutes les versions de Windows testées. Quand les politiques de Snort et Suricata sont configurées, cependant, ces NIDS présentent moins d'incohérences que Zeek, qui appliquent une politique de réassemblage unique. Par exemple, lorsque Snort et Suricata (à partir de la version 7.0.5) sont testés dans les conditions de l'état de l'art, ils sont complètement protégés contre les attaques par évasion IPv4 et IPv6 et presque complètement contre les attaques par insertion et évasion TCP. Étant donné que chaque incohérence rend possible une attaque par insertion ou par évasion de données, nous avons rapporté chacune des nouvelles politiques de réassemblage basées sur les recouvrements de paires et de triplets de morceaux ($n \leq 3$) aux trois éditeurs de NIDS.

Table of Contents

Résumé substantiel en français 5

Introduction 15

Context . . . 15
Research questions . . . 17
RQ.1 – What are the characteristics of IPv4 fragments in the wild, and to what extent does data overlap occur? . . . 17
RQ.2 – How diverse and correct are IPv4, IPv6, and TCP implementation reassemblies when exploring test case context and up to $n = 3$ overlaps? . . . 17
RQ.3 – Can the NIDSes Suricata, Snort, and Zeek be subject to overlapping-based attacks when monitoring hosts with general purpose OSes? . . . 18
Thesis organization and contributions . . . 18

1 Background and State-of-the-art 21

1.1 IPv4, IPv6, and TCP Internet protocols: the chunking mechanism, reassembly, and overlapping data . . . 22
1.1.1 IPv4 . . . 22
1.1.2 IPv6 . . . 27
1.1.3 TCP . . . 28
1.1.4 RFC specification summary in regards to overlapping data . . . 31
1.2 Overlapping chunks in the wild . . . 32
1.2.1 IPv4 . . . 32
1.2.2 IPv6 and TCP . . . 34
1.2.3 Limitations . . . 35
1.3 Evasion and insertion attacks . . . 36
1.3.1 Ptacek and Newsham model . . . 37
1.3.2 Wang et al. model . . . 37
1.3.3 Limitations . . . 38
1.4 Stack reassembly policies in the presence of overlaps . . . 39
1.4.1 Manually generated overlap cases . . . 39

1.4.2 Semi-automatically generated overlap cases . . . 42
1.4.3 Limitations . . . 44
1.5 Implemented countermeasures inside the NIDSes . . . 44
1.5.1 Snort . . . 45
1.5.2 Suricata . . . 45
1.5.3 Zeek . . . 47
1.5.4 Limitations . . . 47

2 Overlap-based attack characterization **49**
2.1 Problem formulation . . . 49
2.1.1 Definitions . . . 50
2.1.2 Illustration . . . 50
2.2 Threat Model . . . 51
2.2.1 Gaining monitored host and NIDS reassembly policy knowledge . . . 52
2.2.2 Crafting the data chunks . . . 53
2.3 Desynchronization persistence across protocol data buffers after the overlap attack 54
2.3.1 p protocol data buffer desynchronization persistence . . . 54
2.3.2 Connection desynchronization of an upper protocol . . . 56

3 IPv4 fragmentation characteristics from Internet traffic **57**
3.1 Method . . . 57
3.1.1 MAWI dataset . . . 58
3.1.2 Analyzing tools . . . 58
3.1.3 Series construction methods . . . 59
3.2 IPv4 fragment characteristics . . . 60
3.2.1 Overview . . . 60
3.2.2 Series types . . . 61
3.2.3 Size analysis . . . 62
3.2.4 Upper protocol analysis . . . 65
3.2.5 Source IP host diversity . . . 67
3.3 Analysis of series with misassociation evidence . . . 69
3.3.1 RFC compliance . . . 69
3.3.2 Header field analysis . . . 71
3.4 Conclusion . . . 73

4 IPv4, IPv6, and TCP stack reassembly policy analysis **75**
4.1 Method . . . 76
4.1.1 Overlap test case model . . . 76
4.1.2 PYROLYSE tool . . . 86
4.1.3 Comparison with related methods . . . 89
4.2 Tested stacks . . . 89
4.2.1 Network stacks . . . 91
4.2.2 Midpoint stacks . . . 93
4.3 Detailed NIDS insertion and evasion opportunities for $n = 2$ . . . 93
4.3.1 OS reassembly policies . . . 93
4.3.2 NIDS/OS reassembly consistency . . . 96
4.4 Stack reassembly characterization for n $\leq$ 3 . . . 99
4.4.1 Reassembly diversity . . . 99
4.4.2 Reassembly errors . . . 111
4.4.3 Inconsistencies between $n = 2$ and $n = 3$ reassemblies . . . 115
4.5 Discussion and limits . . . 121
4.5.1 Reassembly compliance with RFCs . . . 121
4.5.2 From theory to practice: test cases' usability in real-world attack scenarii . . . 122
4.5.3 Reassembly policy configurability cost . . . 125
4.5.4 Configurable NIDS challenges to address $n > 2$ overlapping chunks . . . 127
4.6 Conclusion . . . 128

Conclusion **129**
Contribution summary . . . 129
IPv4 fragment overlaps in real Internet traffic . . . 129
IPv4, IPv6, and TCP policies and related NIDS evasions and insertions . . . 130
Perspectives and Future works . . . 132
Measure IPv6 and TCP overlaps in Internet traffic . . . 132
Extend the targeted stacks and protocols inside PYROLYSE . . . 133
Extend the configurable NIDSes . . . 133
Improve the security of network stacks with respect to data overlaps . . . 134

Glossary **136**

Bibliography **139**

Appendices **151**

A Tested implementation details 151

B Detailed responsible disclosure process 151

C Custom reassembly algorithm test 154

C.1 The pipeline 154

C.2 The observed reassembly mechanims with side effect 155

C.3 List of reassembly algorithms 156

Table of Figures **159**

Introduction

Contents

Context **15**
Research questions **17**
RQ.1 – What are the characteristics of IPv4 fragments in the wild, and to what extent does data overlap occur? 17
RQ.2 – How diverse and correct are IPv4, IPv6, and TCP implementation reassemblies when exploring test case context and up to $n = 3$ overlaps? 17
RQ.3 – Can the NIDSes Suricata, Snort, and Zeek be subject to overlapping-based attacks when monitoring hosts with general purpose OSes? 18
Thesis organization and contributions **18**

Context

The information age started in the mid-20th century and is characterized by a shift from traditional industries to an economy centered on information technology [14]. The Internet has widely favored this age happening. Many Internet-compatible devices and communication protocols have been built to permit information to circulate worldwide. The past decades have come up with a growing Internet access [15], [16], some usage changes (from blog reading to video streaming [17], [18], Large Language Models-based chatbots [19], IoT [20]), and equipment improvement (throughput [18], computation, size of hardware components [21]). IP and TCP have notably been key protocols from the beginning and they have supported the Internet traffic evolutions.

It did not take long for the Internet systems to be abused (e.g., Creeper in 1971 [22], ANIMAL in 1975 [23], Morris worm in 1988 [24]), to such an extent that defending them has become a priority for companies [25] and governments [26], [27] in today's world. Computer security, or cybersecurity, has developed and become highly specialized. One of the cybersecurity generalists and top-ranked academic conference IEEE S&P [28] listed 30(!) sub-topics of interest in 2025: "intrusion detection and prevention" and "protocol security" are two of them. In 1994, Mukherjee et al. [29] explained

that the intrusion detection's goal is to "identify, preferably in real time, unauthorized use, misuse, and abuse of computer systems by both system insiders and external penetrators". Depending on the intrusion detection system (IDS), the detection relies on analyzing the user activity artifacts, such as system logs or network traffic. IDSes can be signature-based, if they use known malicious attack patterns, or anomaly-based, if they detect attacks from a user activity's deviation compared with the normal. Nowadays, IDSes are most commonly deployed at the company or institution level. These IDSes defend side-by-side with other security equipment that have complementary purposes, such as firewalls (block unauthorized network traffic), antivirus (analyze host-level files), EDR (Endpoint Detection and Response), or SIEM (aggregate and correlate the logs to generate security events).

In 1998, Ptaceck and Newsham [1] described the insertion and evasion attacks that can target Network IDSes, or NIDSes (which thus process network traffic). These attacks exploit protocol ambiguities that arise when the RFC specification is unclear (because the expected behaviour is not explicitely specified) or not respected. In such cases, the monitored and monitoring (i.e., the NIDS) hosts deviate in how they process a network packet, leading to the desynchronization of the two host data buffers. One such protocol ambiguity is the overlap of chunk data that IP fragmentation and TCP segmentation permit. In such cases, the ambiguity originates from unclear IPv4 and TCP specifications.

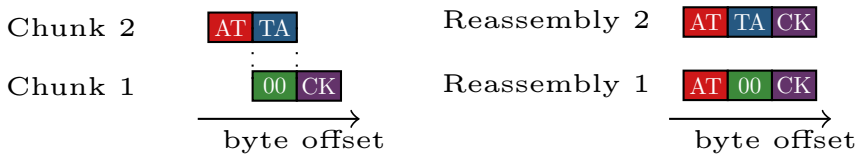


Figure 2: Data overlap ambiguity illustration.

Figure 2 is an illustration of the data overlap ambiguity. The reassembled data here differ depending on which chunk the reassembly policy favors. Someone with bad intentions may exploit the multiple reassembly possibilities to hide a malicious payload. If the NIDS reassembles with reassembly strategy 1 while the host reassembles with strategy 2, the former cannot see the malicious payload and raise any security alert. Ptaceck and Newsham showed that the tested NIDSes could be subject to overlap-based attacks, as OSes reassemble the IPv4 and TCP overlap chunk sequences differently. Since then, the NIDSes proposed two countermeasures: raise an alert whenever an overlap occurs or apply differentiated reassembly policies according to the monitored host.

In this thesis, we conduct a multidimensional analysis regarding the data overlapping ambiguity of IPv4, IPv6, and TCP. The first dimension is the occurrence and structure analysis of IPv4 fragment overlaps in the wild. The second tackles the theoretical attacking model, which we extend from the previous works (term definitions, overlap test case generation, or reassembly policy formalism). The third dimension addresses the IPv4, IPv6, and TCP reassembly policies of network and NIDS stacks, as well as the related attack opportunities.

Research questions

This thesis will tackle the following research questions.

RQ.1 – What are the characteristics of IPv4 fragments in the wild, and to what extent does data overlap occur?

IPv4, IPv6, and TCP semantically permit chunk data overlaps. Only IPv6 forbids them in its specification, and, while IPv4 and TCP RFCs allow the overlaps, they do not mention which data the implementations must reassemble with. As mentionned, an attacker can bypass a NIDS by using the data overlap ambiguity. To what extent do we observe IP and TCP chunks in the wild? If such chunks are not observed, NIDSes just have to raise an alert whenever they process a chunk. However, if they are observed in a significant amount, do data overlaps occur in the chunk sequences? Are such sequences inherently malicious, or can some benign reasons explain their presence on the Internet? Assessing whether a chunk sequence with overlapping data is inherently considered malicious is important because it can directly influence the NIDSes' countermeasures. In this thesis, we will analyze the characteristics of IPv4 chunk sequences from some MAWI traces.

RQ.2 – How diverse and correct are IPv4, IPv6, and TCP implementation reassemblies when exploring test case context and up to $n = 3$ overlaps?

IP and TCP are important protocols, as most communications are IP-based (version 4 or 6) and TCP and UDP got their fair share regarding the transport OSI layer-related traffic. They thus fit a lot of usages, and depending on them, the protocol implementation may vary. For example, IoT or embedded devices have hardware constraints that may

require a lightweight IP or TCP implementation, in the sense that the editors may choose not to implement some security or specified features. Other implementations, such as Suricata or Snort NIDSes, propose that users configure the IP and TCP reassembly from a list of implemented policies. In this thesis, we will analyze to what extent the varying implementations of these protocols behave differently when processing packet sequences comprising at most $n = 3$ overlapping chunks. Since network events may modify or alter the original overlap chunk sequence structure before reaching the target host, we will also study several testing contexts to simulate such events.

RQ.3 – Can the NIDSes Suricata, Snort, and Zeek be subject to overlapping-based attacks when monitoring hosts with general purpose OSes?

A Network Intrusion Detection System (NIDS) reconstructs the network packets from and to the monitored hosts. The NIDS' main goal is to raise an alert whenever it encounters a known malicious or suspicious traffic pattern. Thus, a NIDS must reassemble the network traffic with the monitored hosts as accurately as possible. If the NIDS does not take the same decision as the monitored host regarding a data overlap, an attacker can exploit this decision discrepancy to attack the victim host.

One main goal of this thesis is to improve the NIDSes regarding overlap-based attacks. The Suricata and Snort NIDSes are configurable, allowing users to associate some implemented reassembly policies with the monitored hosts. In this thesis, we will assess this countermeasure's effectiveness by analyzing the reassembly consistency of Suricata [30] (configured and not configured), Snort [31] (configured and not configured), and Zeek [32] NIDSes with some recent versions of general purpose OS hosts.

Thesis organization and contributions

Chapter1 aims to bring the reader's knowledge at the state-of-the-art with regards overlapping data attacks with IPv4, IPv6, and TCP protocols. The chapter briefly outlines IP fragmentation and TCP segmentation, on top of which overlapping data attacks are built. After focusing on works that measured data overlaps in the wild, it details the evasion and insertion attack classes. Then, it addresses related works that analyzed IP and TCP implementation behaviors when processing data overlaps. Finally, it presents the Network Intrusion Detection Systems (NIDS) countermeasures to defend against

overlap-based attacks.

Chapter 2 adapts insertion and evasion attacks to the overlap-specific context. It first extends the related works' insertion and evasion definitions. It then details the attacker's capabilities by outlining the considered threat model. It finally provides some insights regarding the overlapping data's syntactic and semantic correctness.

Chapter 3 answers the first research question by analyzing the IPv4 fragment characteristics of some Internet traffic. After giving some general characteristics (fragment distribution, upper layer service, fragment size), it focuses on the overlaps.

Chapter 4 answers the second and third research questions by studying the IPv4, IPv6, and TCP reassembly policies of various network stack types and some NIDSes. It provides multiple theoretical contributions regarding overlap test case modeling, exhaustiveness, context as well as stack reassembly policy formalism. These contributions are implemented inside PYROLYSE, which we used to test the protocol implementations and which is publicly available [11]. The chapter analyses the evasion and insertion opportunities of Suricata, Snort, and Zeek NIDSes with $n = 2$ overlapping chunks. It then discusses the diversity, errors, and generalization of stack policies based on $n = 3$ chunks.

Chapter 5 concludes by outlining and summarizing the most significant results of this thesis. It discusses future research directions based on the contributions.

During this thesis, we published two full papers in international conferences:

- L. Aubard, J. Mazel, G. Guette, and P. Chifflier, "Overlapping data in network protocols: bridging OS and NIDS reassembly gap", in *DIMVA*, 2025 [33], which received the "Distinguished paper" award.
- L. Aubard, J. Mazel, G. Guette, and P. Chifflier, "Overlapping IPv4, IPv6, and TCP data: exploring errors, test case context, and multiple overlaps inside network stacks and NIDSes with PYROLYSE", in *RAID*, 2025 [34].

Chapter 1

Background and State-of-the-art

Contents

1.1 IPv4, IPv6, and TCP Internet protocols: the chunking mechanism, reassembly, and overlapping data . . . 22
1.1.1 IPv4 . . . 22
1.1.2 IPv6 . . . 27
1.1.3 TCP . . . 28
1.1.4 RFC specification summary in regards to overlapping data . . . 31
1.2 Overlapping chunks in the wild . . . 32
1.2.1 IPv4 . . . 32
1.2.2 IPv6 and TCP . . . 34
1.2.3 Limitations . . . 35
1.3 Evasion and insertion attacks . . . 36
1.3.1 Ptacek and Newsham model . . . 37
1.3.2 Wang et al. model . . . 37
1.3.3 Limitations . . . 38
1.4 Stack reassembly policies in the presence of overlaps . . . 39
1.4.1 Manually generated overlap cases . . . 39
1.4.2 Semi-automatically generated overlap cases . . . 42
1.4.3 Limitations . . . 44
1.5 Implemented countermeasures inside the NIDSes . . . 44
1.5.1 Snort . . . 45
1.5.2 Suricata . . . 45
1.5.3 Zeek . . . 47
1.5.4 Limitations . . . 47

This chapter introduces some knowledge necessary for the reader to understand the content of the following chapters. First, it explains the concepts of IPv4 and IPv6 fragmentation, as well as TCP segmentation. Second, it compiles and analyzes the works that studied IP and TCP chunk overlaps in the wild. Third, the chapter describes the insertion and evasion classes of attacks, which are used to attack some security equip-

ment, such as NIDSes. Fourth, it gives the current state of knowledge regarding the overlap-based reassembly policies of IP and TCP stacks. Finally, it describes the currently implemented countermeasures inside the NIDSes regarding overlap-based attacks.

1.1 IPv4, IPv6, and TCP Internet protocols: the chunking mechanism, reassembly, and overlapping data

IPv4, IPv6, and TCP protocols provide one chunking mechanism, which refers to IPv4 and IPv6 fragmentation and TCP segmentation. It consists of splitting an original data packet into several data chunks and was introduced to answer a potential discrepancy between the transit links medium or underlying protocol capacities. When chunking occurs, the receiver must reassemble all the chunks to retrieve the initial data packet. But, the chunking mechanism can lead to *overlaps*. The most common case is a chunk retransmission with the same data starting and finishing at the same byte offset. However, other types of overlaps exist, i.e., partial overlaps, and the *data can be different* on the overlapping portion. Fragmentation and segmentation have been at the core of IPv4 and TCP from the beginning and have, thus, been necessary for the protocols' proper functioning. The following parts detail some protocol specificities related to their chunking mechanism.

1.1.1 IPv4

The Internet protocol version 4 (IPv4) was initially defined in 1980 in RFC 760 [35] and updated with RFC 791 [8]. In the OSI model, it belongs to the network layer; hence, its goal is to transport the network data packets between distant machines through the worldwide interconnected networks, or the Internet. In 2025, IPv4 is still the most used IP version [36].

1.1.1.1 Header

Figure 1.1 represents the IPv4 header, typically 20-octet long since options are rarely used.

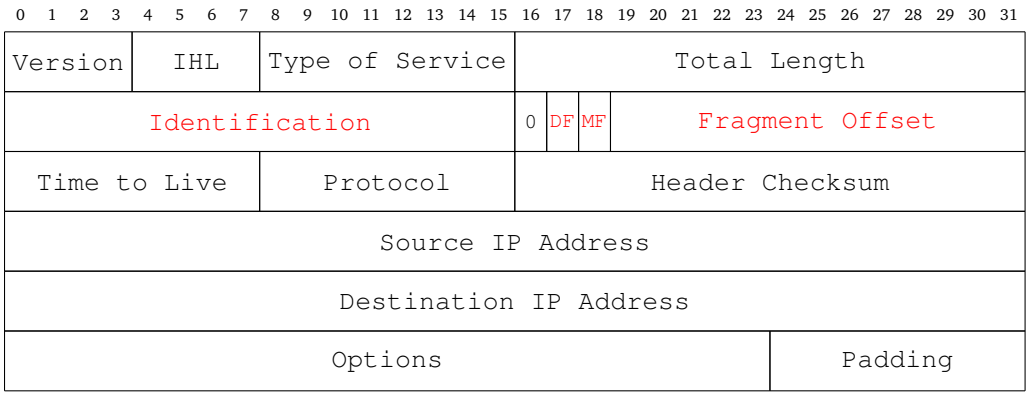


Figure 1.1: The IPv4 header. Red fields are used for fragmentation.

Several fields of the IPv4 header are dedicated to the fragmentation process.

- `Identification` (`IPID`) distinguishes the chunks, or fragments, of one datagram from the fragments of others. For the related three-tuple ⟨`srcIP`, `dstIP`, `next proto`⟩, the `IPID` should not be reused as long as "the datagram will be active in the internet system"[1] [8].
- `More Fragment` (`MF`) flag indicates whether the fragment is the bytewise last (or rightmost). Any fragment on the left has the `MF` bit sets.
- `Fragment offset` indicates the relative starting position of the fragment's data on the overall datagram data. Its unit is eight octets.
- `Don't Fragment` (`DF`) flag informs whether an IPv4 packet can be fragmented.

We refer to a datagram as an IPv4 packet with both `MF` and `offset` fields unset. A fragment has either `MF` set or a non-null `offset`. Note that the fragmentation indirectly impacts the `Header Checksum` and `Total Length` fields since they need to be recomputed to be consistent with the data the fragment carries.

1.1.1.2 Typical on-path fragmentation

IPv4 fragmentation mostly occurs on-path from the packet's sender to the receiver. Routers or VPN gateways are common on-path machines performing fragmentation.

One typical fragmentation scenario consists of two machines communicating as illustrated in Figure 1.2. A source host sends an IPv4 packet of x bytes. Routers on the

[1] It corresponds to the Maximum Datagram Lifetime (MDL) in RFC 6864, which provides 120 seconds as an MDL example. But, it is not a hard specification; IPv4 implementations should set the MDL according to their needs, and the MDL should be comprised between the receiver's reassembly timeout and sender's Maximum Segment Lifetime (MSL) [9], [37].

path are tasked to deliver the packet to the destination host.

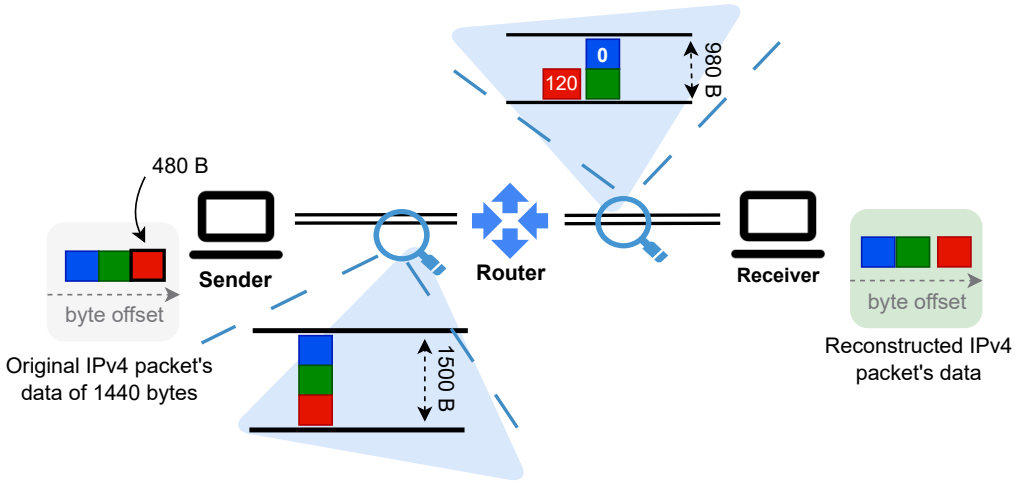


Figure 1.2: IPv4 fragmentation illustration. The link capacities correspond to the link MTUs, i.e., IPv4 header and data.

Fragmentation process At some point, one router on the path cannot send this x-byte packet because its exit link has only y-byte capacity, with $y < x$. To reach the destination host, the datagram needs to be fragmented. The RFC 791 specification [8] states that any fragment with bit `MF` set must carry y data bytes, and the last must carry the remaining rightmost data. In Figure 1.2, the original $1,440$-byte datagram is chunked into two new packets, or fragments. The blue-green one carries the leftmost $1,000$ bytes, has `MF` bit set, and an `offset` of 0. The red one carries the rightmost remaining 480 bytes, has `MF` bit unset, and an `offset` of $960/8 = 120$. The two fragments have the same `IPID`.

De-fragmentation process Based on the `IPID` fields, the destination host can associate the fragments altogether to reconstruct the original datagram. When 1) the fragment with bit `MF` unset is received and 2) there is no hole between the fragments' data, the host pushes the reconstructed data to the next protocol buffer.

Additional information A datagram with the `Don't Fragment` (`DF`) bit set must not be fragmented on its way. Thus, if the original datagram from Figure 1.2 has the `DF` bit set, the router must discard the packet and send an ICMP "Fragmentation Needed" message back to the sender, informing the datagram did not reach the receiver and providing

the router's exit Maximum Transmission Unit (MTU)[2]. On the contrary, if the `DF` bit is unset, the on-path fragmentation is transparent for the sender, i.e., fragmentation is performed without notice.

1.1.1.3 Some network events cause overlaps

Many reasons can cause packet duplication, i.e., complete overlap, in networks: network congestion, load balancing, misconfigured devices, etc. For example, in the case of on-path load balancing, the router could duplicate the blue-green fragment, resulting in data overlap with two fragments that start and finish at the same `offset` and with the same data. Shannon et al. [2] reported that 8.5% of the captured fragment sequences contained at least one fragment duplicated in 2001. Therefore, complete data overlaps often happen on the Internet because of various recurring network events.

1.1.1.4 The data can be different on the overlap portion

RFC 4963, 6274, and 6864 [9], [38], [39] describe that fragment loss can result in overlapping data. As we can see in Figure 1.3, we consider the same original 1,440-byte IPv4 packet as previously, and name it p_1. Again, discrepant entry and exit MTUs at one router lead to packet fragmentation. However, the two fragments get routed differently. One router the red fragment reaches is congested (i.e., its fragment queue is full). The fragment is, therefore, dropped. The other fragment reaches the destination host, but the original datagram p_1 cannot be reconstructed since the fragment with `MF` bit unset has not arrived. The blue-green fragment waits until the arrival of other fragments with the same `IPID` or gets dropped if it times out.

[2]The MTU value corresponds to the maximum size of an IP packet, i.e., header and data, a machine can send at once through the corresponding network interface.

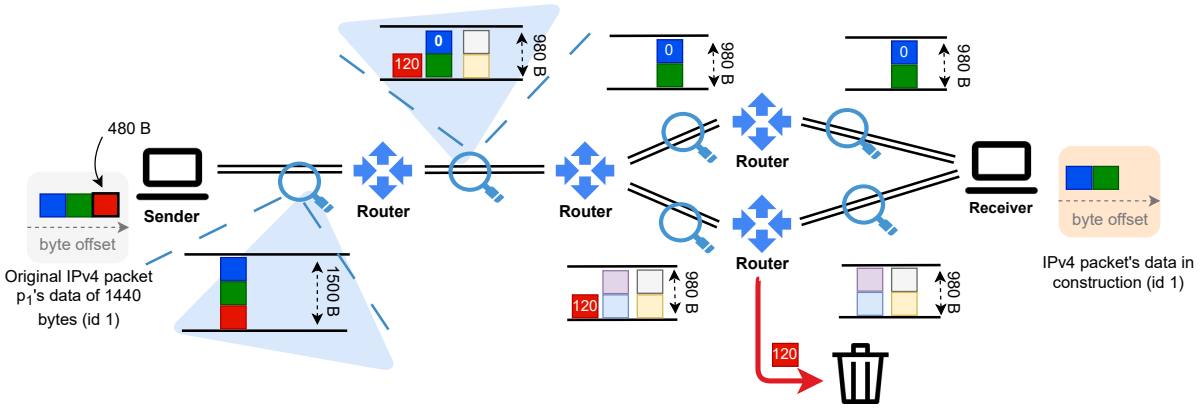


Figure 1.3: IPv4 fragmentation with loss.
The link capacities correspond to the link MTUs, i.e., IPv4 header and data.

In Figure 1.4, the source host sends an IPv4 datagram p_2 with the same `IPID` as the previously fragmented packet p_1. This new datagram p_2 is 1,190-byte long and carries different data. It gets fragmented before reaching the destination host. Considering that the previously received blue-green fragment has not yet timed out[3], the host can either reconstruct favoring the oldest data (from blue-green fragment) or the newest data (from purple-yellow fragment).

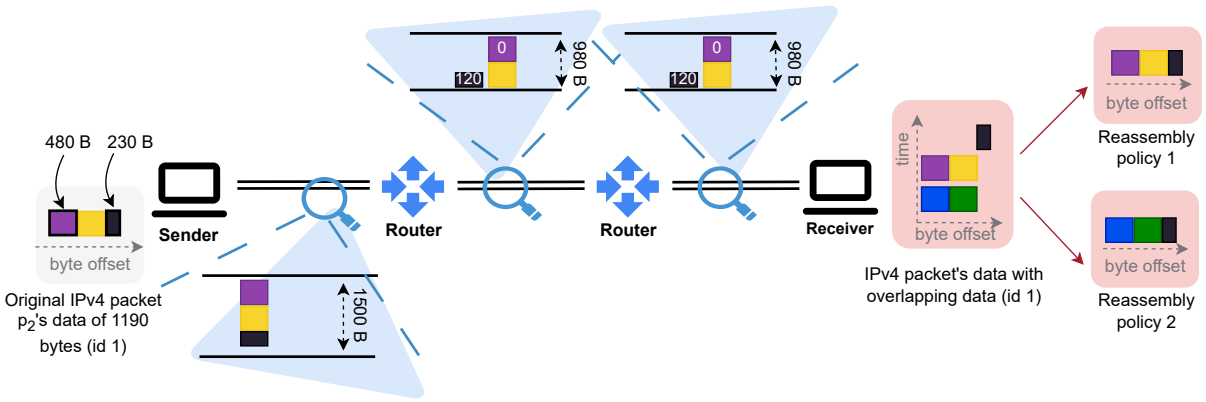


Figure 1.4: IPv4 fragmentation with overlap.
The link capacities correspond to the link MTUs, i.e., IPv4 header and data.

[3]Here, the sender thus reuses the `IPID` for another original datagram, i.e., p_1, in less than the receiver's reassembly timeout.

1.1.1.5 RFC specification regarding to overlapping data

As we've just illustrated, IPv4 semantics allow complete or partial data overlaps. The IPv4 specification [8] does not forbid overlaps, but it does not state how IPv4 implementations should resolve an overlap, e.g., to reassemble with data from the oldest fragment, or drop the overlapping fragment. The RFC, however, runs through one processing example in which the newest data is preferred.

1.1.2 IPv6

Internet Protocol Version 6 (IPv6) is IPv4's successor. Its adoption is slow, since the standardization began in 1998 [40], but in 2013, only 1% of the overall Internet traffic was IPv6-based [36], [41], and, in 2025, IPv6 has not yet overtaken IPv4 in traffic volume [36]. In 2017, the five Regional Internet registries (RIRs), responsible for allocating the public IP addresses, reached their last IPv4 /8 prefix they can attribute. These past few years, the RIRs have actively searched for inactive spaces to re-allocate them, but the RIR ARIN waiting list's time to get such addresses was about 130 days in 2020, for example. A parallel market has thus emerged, proposing organisations to buy or lease IPv4 addresses [42]. Therefore, one main goal for this major update is to tackle this IPv4 address space exhaustion problem, as there can only be 2^{32} = 4 294 967 296 IPv4 public addresses. IPv6 solves the problem by affecting 16 octets (instead of 4 for IPv4) in its header to a machine address; thus, there can be $3.4 * 10^{38}$ addresses in total. The new IP version also introduces extension headers, easing packet routing.

1.1.2.1 Header

The handling of fragmentation changes a little bit in IP version 6. A fragment-specific extension header follows the IPv6 header if the packet is an IPv6 fragment. `MF` bit, `fragment offset`, and `identification` fields compose this extension header. The IPv6 `IPID` field is 32-bit long, significantly longer than IPv4's related 16 bits.

1.1.2.2 Processing

Unlike IPv4, IPv6 fragmentation can only happen at the source host, which explains the removal of the `DF` bit. If a router cannot send an IPv6 datagram because of MTU issue, it sends back an ICMPv6 "Packet Too Big" message to the source. However, RFC highly

recommends that IPv6 implementations implement the Path MTU discovery (PMTUD)[4] mechanism to minimize the probability of such an event occurrence. The source host fragments the IPv6 data if the PMTU is lower than the upper-layer data and header size. IPv6 overlaps may occur for various reasons, including duplication or a fragment loss followed by the transmission of another IPv6 fragmented datagram with the same `IPID`, as described for IPv4 in Section 1.1.1.4.

1.1.2.3 RFC specifications in regards to overlapping data

IPv6 semantics allow data overlaps, and initially, IPv6 specifications [40], [45] did not forbid overlaps nor did they explain how an overlap should be resolved (e.g., to reassemble with data from the oldest fragment). A later standard related to overlapping specific issues [46] stated that in the presence of an overlap, all the fragments that correspond to the same original datagram must silently be discarded. The latest IPv6 RFC standard [47] has confirmed this statement.

1.1.3 TCP

The transmission control protocol (TCP) was initially defined in 1980 in RFC 761 [48] and updated with RFCs 793 [8] and 9293 [49]. In the OSI model, it belongs to the transport layer. It particularly provides reliable (through packet loss and error detection), connection-oriented, and multiplexed (through concurrent upper-layer service use) communications between two hosts. Alongside UDP, they are the two main layer four protocols.

1.1.3.1 Header

Figure 1.5 represents the TCP header, which is 20-octet long without the options.

[4]The PMTU is the minimum on-path MTU from the sender to the receiver. The PMTUD [43], [44] is a technique that aims to find such a PMTU value.

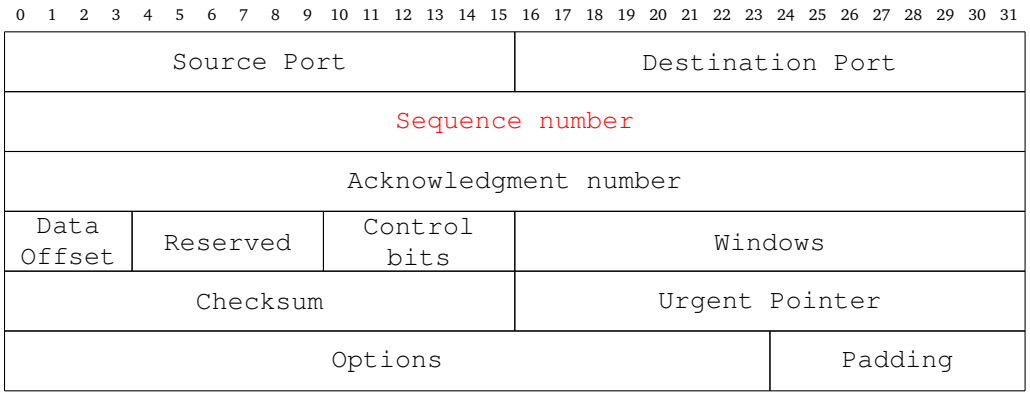


Figure 1.5: The TCP header.

Both the `sequence number` field, which indicates the relative starting position of the segment's data on the overall data flow, and the length of the encapsulated payload allow the end host to reassemble the segments correctly.

1.1.3.2 Segmentation processing

As for IPv6 fragmentation, TCP segmentation only happens at the source host. A host A determines the maximum size of the segments it sends to B with the Maximum Segment Size (MSS) TCP option that the host B must (now [49]) provide. MSS is supposed to avoid IP fragmentation and maximize host connection performance in the meantime. The MSS a host provides may correspond to the minimum IP MTU (536 for IPv4 and 1,280 for IPv6), the result of a PMTUD or PLPMTUD[5], or the MTU on directly connected links. If no MSS is provided, the default segment sizes should be the minimum IP MTU.

The Figure 1.6 illustrates a simple segmentation scenario. An upper protocol that needs to send more data than the receiver host's MSS experience data segmentation. In other words, its data will be placed in multiple TCP segments, and the receiver will have to reassemble these segments.

[5]As PMTUD, PLPMTU aims at finding the PMTU. It is specified in RFC 8899 [50] and stands for Packetization Layer Path MTU Discovery for Datagram Transports.

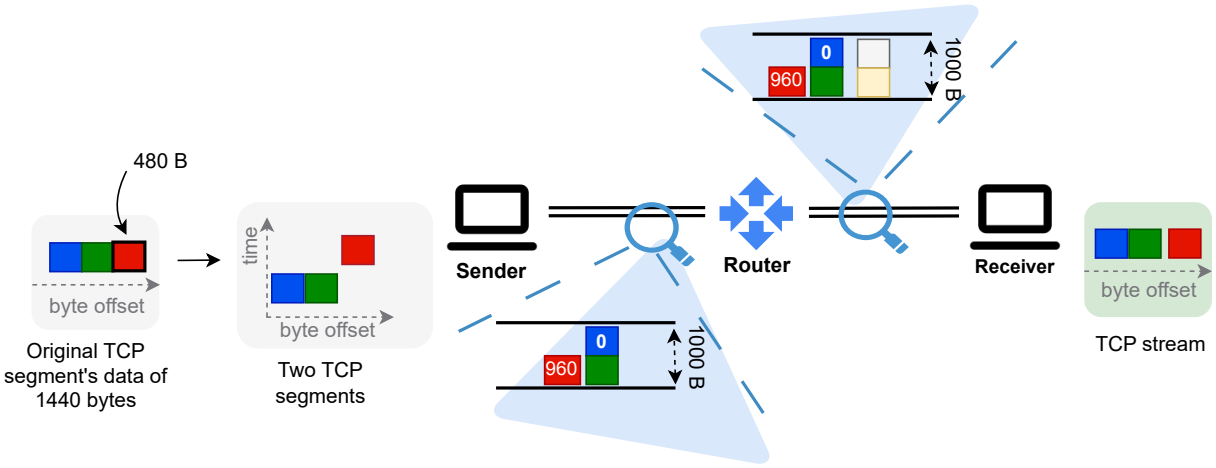


Figure 1.6: TCP segmentation illustration. The link capacities correspond to the link MTUs, i.e., IPv4 header and data. The TCP headers are 20 byte-long here.

1.1.3.3 Retransmission case

The most common overlap case in TCP is a retransmission. Segments may take different paths and arrive disordered at the receiver. The receiver's data stream contains (temporarily) a data hole from sequence numbers b to e". The receiver may think that the missing segment $seg_{b \to e}$ was lost and inform the sender with an ACK packet, whose acknowledgement number corresponds to the beginning of the data hole. The sender can subsequently re-send the segment $seg_{b \to e}$. However, if the first $seg_{b \to e}$ was actually delayed because of network reasons (and not lost), there is a complete data overlap at the receiver side.

1.1.3.4 Some cases leading to partial overlaps

RFC 9293 [49] describes a case that can lead to other overlap types that are not just sheer retransmissions. It requires multiple events in conjunction: 1) packet loss, 2) sending host reboot, 3) TCP connection reuse, and 4) TCP sequence number reuse. Finally, all that needs to happen within one MSL[6] (2 minutes [49]).

Another overlap case may arise when the MTU and related MSS increase or decrease during the connection. If we consider the same situation as in the retransmission case from Section 1.1.3.3 but with a decreased receiver MSS, the sender must chunk the

[6]The MSL, short fort Maximum Segment Life, is defined in [49] as "the time a TCP segment can exist" in the Internet.

initial segment. This results in partial overlaps at the receiver side.

1.1.3.5 RFC specifications in regards to overlapping data

As described, TCP semantics allow segments to wholly or partially overlap. The TCP specifications [49], [51] do not forbid overlaps but do not state how implementations should resolve an overlap. The RFCs, however, run through one processing example in which the newest data is preferred.

1.1.4 RFC specification summary in regards to overlapping data

Table 1.1 summarizes what permits the protocol RFCs regarding the data overlaps.

Protocol	RFC	Semantics	Recommandation
IPv4	791 [8]	possible	authorize
IPv6	8200 [47]	possible	forbid
TCP	9293 [49]	possible	authorize

Table 1.1: Latest RFC summary regarding data overlaps.

Section takeaways

IPv4 and IPv6 fragmentation and TCP segmentation were introduced if an application needs to send more data than the underlying protocol or the egress link capacity can simultaneously carry. Fragmentation and segmentation chunk the original application data into several pieces to address these data size constraints. However, chunking can lead to data overlaps for benign reasons (e.g., complete overlaps with duplication or partial overlaps with `IPID` reuse), and the data on the overlapping portion can be different. IPv4 and TCP specifications allow chunks with overlapping data, without stating which overlapping data must be kept, while IPv6 implementations must discard such fragment sequences.

1.2 Overlapping chunks in the wild

In this section, we detail the works that measured some characteristics of IP and TCP chunks in Internet traffic, with a special focus on fragment and segment overlaps. First, we analyze IPv4, and then IPv6 and TCP. Finally, we discuss the works' limitations.

1.2.1 IPv4

In this subsection, we first define the term "fragment series", which Shannon et al. originally introduced in [2]. Then, we detail their IPv4 fragmentation and overlap analysis.

1.2.1.1 Fragment series

In 2001, Shannon et al. [2] performed an in-depth study regarding IPv4 fragment characteristics inside some traces captured in March of the very same year. The authors did not reconstruct the original datagrams; they rather analyzed the IP fragments' headers and the first bytes of the first (i.e., leftmost) fragments. They grouped the fragments inside what they called "fragment series", which they defined as "the ordered list (as monitored on the network) of fragments whose source is a single original datagram". Concretely, a fragment series comprises all the fragments with the four-tuple ⟨`srcIP`, `dstIP`, `IPID`, `next proto`⟩ observed within a certain time window (600 seconds here). It is unclear if they chose this fragment aggregation method to ease the analysis, save some computation cost, or address the so-called "cold start" problem. A following work [3] that studied IPv4 fragment series also used this terminology and time window grouping strategy.

Shannon et al. also introduced several series types to characterize the fragment series, which are represented in Figure 1.7. A series is:

- *complete* if there is no data hole.
- *duplicate* if there is a full data overlap between fragments.
- *overlap* if there is a partial data overlap between fragments.
- *correct* if there is no data hole and no data overlap.

Note that a series is always of multiple types, e.g., complete and duplicate.

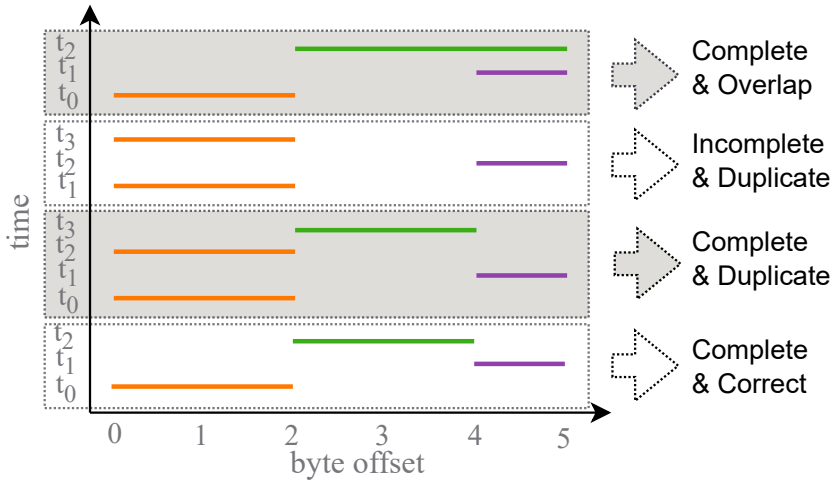


Figure 1.7: Fragment series types.

1.2.1.2 General IPv4 fragment characteristics

Shannon et al. [2] analyzed the fragment serie characteristics in seven traces captured in March 2001, at three Inter Exchange Points (IXPs) and one WAN link. UDP, IPENC, IPsec, TCP, ICMP, GRE, IP in IP are the most fragmented `next proto`. When performing UDP and TCP service analysis, they observed that top UDP services are L2TP (0.43%), REALAUDIO-UDP (0.33%), and SQUID ICP (0.17%) and top TCP services are SMTP (54%), FTPDATA (32%), and WWW (5%). They particularly measured that at least 16% fragments are tunneled traffic, which is evidence of the tunnel extra bytes addition.

In 2007, John and Tafvelin [52] also studied some IPv4 fragment statistics at one Swedish backbone link. Every day between the 7th and the 26th of April 2006, four 20-minute long traffic traces were captured. They noted a stop in IPv4 fragmentation rising trend from 0.67% in [2] to 0.06% here. They also observed very contrasted results between inbound and outbound traffic (97% fragments are UDP and 3% are TCP for incoming while 19% fragments are UDP, 18% are TCP and 63% are ESP for outgoing).

Finally, Gilad and Herzberg [6] measured 0.15% of IPv4 fragmented traffic from the 1.05TB CAIDA 2012 traces [53]. Moreover, 1% of UDP, 1% of GRE and 1% of IPsec IPv4 packets were IPv4 fragments, and 15% of UDP-IKE negotiations were fragmented. While primarily presenting new techniques (interception and DoS) abusing IP fragmentation in their paper, the authors recommended to avoid fragmentation as much as possible.

1.2.1.3 Characteristics of fragment series with overlaps

Shannon et al. [2] measured 7.5% complete series with only duplicates and 1.1% complete series with duplicates and overlaps[7] in the traffic they analyzed. The authors, however, did not analyze these series types in more detail.

In 2008, John and Olovsson [3] especially looked for several anomalies in the fragment headers and quantified them. The analyzed dataset was the same as in [52] work. The authors quantified the number of UDP, TCP, and ICMP problematic series with some suspect characteristics (e.g., series with a short first fragment, single packet series). Overlapping data-wise, they observed 37 *overlap* (i.e., with partial data overlap) and 1,864 duplicate (i.e., with full data overlap) series for a total of 20,8 million series. They found that most overlap series are incomplete, i.e., with a data hole, and that the (8 to 48-byte) overlapping portions fit the data hole length. The authors verified that the overlapping fragments within duplicate series are not sheer retransmissions or network duplicates. Since they did not have access to the fragment payloads, they analyzed the transport header checksums when possible. This checksum consistency analysis could, therefore, only be performed when the fragments that overlapped had a null offset. They found that "many" overlaps in duplicate series are sheer retransmissions, and only 104 series with inconsistent checksums are complete.

1.2.2 IPv6 and TCP

Gilad and Herzberg [6] provided some IPv6 fragment characteristics. For example, they measured that 0.8%, 1%, and 1% of the UDP, GRE, and IPsec traffic were fragmented in IPv6, respectively. They noted that IPv4 and IPv6 fragmentation occurred in the same orders of magnitude, i.e., 0.15% against 0.11% in the CAIDA 2012 traces [53]. Up to our knowledge, no work has ever documented data overlaps in the wild in IPv6 fragments series.

TCP is a stateful protocol, meaning that after connection establishment, data can be transmitted continuously (i.e., in multiple data segments) from both sides, as explained in Section 1.1.3. Thus, TCP segmentation is not optional and is used extensively. The only work we found that analyzed some TCP overlaps is Vutukuru et al. [4] in 2008. In this study, the authors described a robust and light method to address partial TCP overlaps in the context of TCP traffic normalization. They analyzed five traces captured

[7] No result is given for overlap series only.

at five distinct points, representing a total of 663 million TCP packets. Depending on the trace, Vutukuru et al. measured packet overlaps from 0.04 to 0.58%; *20 to 30% of them are partial overlaps*. According to the authors, these partial overlaps are retransmissions but with sequence numbers and payload sizes that are misaligned with previous segments.

1.2.3 Limitations

In this subsection, we describe the limitations of related works for IPv4, as well as IPv6 and TCP.

IPv4

The methods that Shannon et al. [2] and John and Olovsson [3] used to group the fragment series are not representative of how the end-hosts reconstruct the original datagrams, since they use reassembly timeouts and fragment queues. Moreover, while Shannon et al.'s 600-second time window seems very high (five times the 120-second MDL example from RFC 6864 [9]), the 1-second time frame used by John and Olovsson seems very short. In the first (resp. second) case, the number of duplicate and overlap series may have been under-estimated (resp. over-estimated). Furthermore, RFC 6864 [9] clarified certain aspects related to fragmentation in 2013. A host that sends datagrams to a given receiver at high rates has a chance to cause fragment misassociations, or overlaps, that can ultimately result in data corruption on the receiver's side. Therefore, RFC 6864 explicitly specified that non-atomic datagrams, i.e., datagrams that can be fragmented on-path, must rate limit themselves to ensure the datagram's IPID uniqueness within the MDL, thereby preventing fragment misassociation or overlaps. No work has verified whether the hosts comply with RFC 6864, i.e., whether they rate limit themselves when sending non-atomic datagrams.

IPv6 and TCP

While Gilad and Herzberg [6] provided some IPv6 fragmentation statistics, and thus we know that IPv6 fragmentation is actually observed on the Internet, no work has conducted a comprehensive analysis of IPv6 fragments' characteristics (fragment series, fragmented services, etc.). Moreover, we do not know if IPv6 fragment overlaps occur

on the Internet. The only work that studied TCP segment overlaps dates back to 2008; therefore, TCP overlap analysis should be revisited. Note that we address none of these limitations in the thesis.

Section takeaways

Several works recommended avoiding IPv4 and IPv6 fragmentation; however, recent evidence shows that some UDP-based applications or VPN tunnels have relied on fragmentation. The last work that we know of, which provided some statistics regarding fragmentation, dates back to 2013 and measured a total of 0.15% IPv4 and 0.11% IPv6 fragmented packets. In 2008, full or partial data overlaps were observed in some IPv4 and TCP traffic. In the corresponding IPv4 work, the authors classified the (very few) observed partial overlaps as coming from a buggy or suspicious host, and some of the full overlaps are not just sheer retransmissions. Differently, the TCP overlaps come from data retransmissions; however, the authors measured that 20 to 30% of these retransmissions covered a smaller data portion than the original data segment (i.e., partial overlaps).

1.3 Evasion and insertion attacks

Numerous attacks exploiting IPv4 fragmentation have been described in the literature. One very well-known is the *Ping of the Death* [54], in which an attacker can split an abnormally big (greater than 65,535 bytes) ICMP ping packet into many fragments. This caused some OSes to crash and, thus, a Denial Of Service (DoS)[8] Other works abused fragmentation to perform DoS attacks [6]. Others described how an attacker could use tiny fragments to bypass a firewall rule [55], [56] or how to attack UDP applications (e.g., DNS cache poisoning [57], [58]). All these possible fragment-based attacks steered the research community to recommend avoiding as much as possible IP fragmentation [5].

Other attacks aim to find IP or TCP packet sequences differently treated by a monitored implementation and a monitoring one. Although we also found implementation error (see Section 4.4.2), we focus on this last attack type in this document. They are named insertion and evasion attacks.

[8]The goal of DoS is to make the targeted service unavailable.

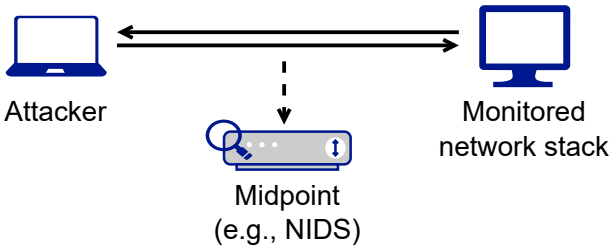


Figure 1.8: The threat model to perform insertion and evasion attacks.

1.3.1 Ptacek and Newsham model

Ptacek and Newsham [1] introduced nine IP and TCP ambiguities that may lead to NIDS misassemblies with the monitored hosts and, NIDS circumvention. As illustrated in Figure 1.8, the ambiguities come from the fact that 1) NIDSes receive a copy of the network traffic coming from and to the hosts and 2) NIDSes and monitored hosts are distinct machines. An ambiguity arises when a packet does not have the same effect on the NIDS and the monitored host. The authors described, for example, that a NIDS accepted a TCP packet with an invalid checksum while the monitored host rejected it. NIDS editors can easily overcome these kinds of ambiguities, which come from RFC specification non-conformance. However, other ambiguities are much more challenging to overcome. This is especially the case for the IP and TCP data overlaps, since IPv4 and TCP RFC specifications permit overlaps but do not state which overlapping data should be preferred, as mentioned in Sections 1.1.1.5 and 1.1.3.5. The authors discussed two attack types derived from these IP and TCP ambiguities: *insertion* and *evasion*.

Definition 1 — Ptaceck's insertion and evasion attacks. In the insertion attacks, the NIDS accepts some packets that the monitored host rejects. Conversely, the NIDS rejects some packets the monitored host accepts in evasion attacks.

1.3.2 Wang et al. model

In 2020, Wang et al. [59] refined Ptacek and Newsham's insertion and evasion definitions for TCP-based attacks. First, the authors defined *accepted* and *dropped* packets in an opaque box[9] testing context, i.e., the model does not require direct access to the targeted machines.

[9]Also known as black box.

Definition 2 — Wang's accepted and dropped packet characterization. A packet is *dropped* by a protocol implementation if it causes neither a TCP state[10] change nor any new data to be pushed to the application; otherwise, the packet is said to be *accepted*.

The authors' goal is to desynchronize a Deep Packet Inspection (DPI) engine, e.g., Censorship System (CS) or NIDS, and host TCP (Mealy) state machines to make the end host reassemble a forbidden, or malicious, payload without the DPI knowing it.

Definition 3 — Wang's state machine synchronization. The DPI and host state machines are said to be *synchronized* if the generated application data streams from the initial LISTEN state are the same.

De-synchronization is performed with a well-chosen *insertion* or *evasion packet* that the DPI and the host process differently, resulting in discrepant application reassembled data. The forbidden payload is either located in the insertion/evasion packet or a follow-up packet.

Definition 4 — Wang's insertion and evasion packets. The NIDS *accepts* an insertion packet and the monitored host *drops* it. Differently, the NIDS *drops* the evasion packet and the monitored *accepts* it, or both implementations *accept* the evasive packet but in a different way.

To summarize, in comparison to Ptaceck and Newsham's model, Wang et al. added some opaque box model testing considerations, which is a significant step forward because one does not always have access to the targeted implementations (e.g., CSes). Wang et al.'s model includes the case in which two implementations accept the packets, but differently. The authors actually observed such discrepant reassembly behavior for overlapping TCP data between Linux 4.9.3 and Zeek 2.6.

1.3.3 Limitations

We identify the following shortcomings:

- Ptacek and Newsham consider a data chunk as being either *totally* accepted or dropped but not *partially* accepted. With the overlap from Figure 2, the "ATTACK"/"AT00CK" reassembly divergence is impossible.

[10] In the paper, the state refers to the "high-level TCP states (e.g., LISTEN, ESTABLISHED), or low-level/implementation-level states (e.g., the number of challenge ACKs that have been sent)".

- Wang et al. do not consider on purpose malicious (or "filtered") payload insertion inside the midpoint or DPI flow data, nor do they consider IP-based attacks.

Section takeaways

This thesis focuses on insertion and evasion attacks that exploit protocol ambiguities. It makes a given packet sequence of such a protocol being treated differently by one monitoring host and one monitored host, ultimately resulting in discrepant flow data. These protocol ambiguities either come from imprecise or poorly implemented specifications. One such ambiguity is the data overlap between IP fragments and TCP segments. Indeed, implementations can prefer different data (e.g., oldest, newest), since IPv4 and TCP specifications do not state which chunk portion an implementation must reassemble with.

1.4 Stack reassembly policies in the presence of overlaps

This section presents the related works that analyzed and described IPv4, IPv6, and TCP implementation reassembly policies. We categorize the works that tested implementation reassembly policies into two families according to the test case generation approach.

1.4.1 Manually generated overlap cases

In 1998, Ptacek and Newsham [1] first showed OSes' distinct behaviors when reconstructing overlapping chunks. They tested two overlapping types for IPv4 and TCP protocols, within two separate chunk sequences. Windows and Solaris (resp. Windows) OS families favored the oldest IPv4 (resp. TCP) overlapping data, while BSD, Linux, HP-UX, and Irix (resp. Solaris, BSD, Linux, HP-UX, and Irix) favored the newest or oldest data depending on the case. All the commercial NIDSes the authors tested did not perform IPv4 defragmentation, and thus, all could be bypassed by simply fragmenting a malicious datagram. Since the NIDSes applied a unique TCP reassembly policy, the authors showed that they could be subject to overlap-based attacks depending on the target OS.

In light of the uncovered ambiguities, Shankar and Paxson proposed actively probing the monitored network and building host profiles in [61]. These profiles included



Figure 1.9: Shankar and Paxson model. There are as many fragments as rectangle colors, and numbers encompass the fragment sending order. Source: [60]

topology information (number of hops to the NIDS, path MTU) and IPv4 and TCP reassembly policies. The NIDS could subsequently perform a profile lookup to resolve any ambiguity the profile covers. The probe sends carefully crafted packets and interprets the responses from the target (i.e., opaque box approach). Regarding overlaps, they obtained IPv4 stack policies through ICMP Echo messages and TCP policies via SSH session initiation attempts. The authors tested three IPv4 overlap types altogether (i.e., within a unique ICMP Echo Request fragment sequence) that are illustrated in Figure 1.9, contrary to the two TCP overlaps they probed within two separate segmented SSH initiation attempts. Shankar and Paxson observed five IPv4 and three TCP reassembly policies from almost 50 tested stacks (general-purpose OSes, routers, printers).

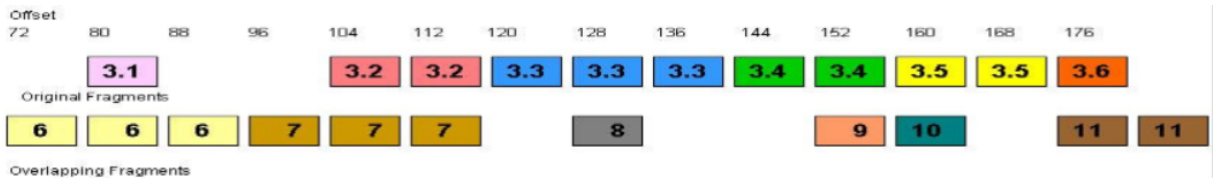


Figure 1.10: Novak model addition. There are as many fragments as rectangle colors, and numbers encompass the fragment sending order. Source: [60]

In 2005, Novak [60] extended the Shankar and Paxson test model, as illustrated Figure 1.10. With the Novak extension, the Shankar and Paxson model covers the nine overlapping types that can link two data fragments. The author used a unique *ICMP Echo Request* fragment sequence to obtain IPv4 policies. However, differently from [61], she introduced a way to make ICMP checksum valid no matter the targeted stack reassembled data due to well-chosen payloads. More precisely, she chose the fragment payloads so that 1) they align with the fragment `offset` IP header field unit (8 bytes), 2) no matter which overlapping data is preferred, the higher layer `checksum` is valid, and, 3) the preferred fragment can be distinguished from the other. The pat-

terns are combinations of "11", "22", "33", "44" strings, and, for example, "11223344", "11334422", or "44221133" are valid patterns. Finally, the extended Novak model enriched the five previously described IPv4 policies and uncovered two new policies.

Two years later, Novak and Sturges used Novak's model to test TCP stacks in [62]. They tested the overlaps in two ways: altogether or separately. They used the netcat tool [63] in the first case and attempted to initiate HTTP sessions in the second one. Since not all the stacks supported netcat, they could not test all of them with the two modes. Surprisingly, Windows before Vista and Solaris 9 and 10 did not reassemble some overlap types similarly depending on the tested mode. The Novak's model enriched Shankar and Paxson's TCP policies, setting the number of different policies to eight.

Atlasis [56] described for the first time IPv6 overlap reassembly policies in 2012, three years after the RFC 5722 [46] stated that IPv6 implementations must silently discard any datagram whose fragments overlap. The author verified whether the stacks adopted the standard. Atlasis used combinations of three fragments for the testing. He introduced a test case generation strategy that yielded one or two distinct overlap types inside every fragment sequence. He showed that none of the tested OSes complied fully with RFC 5722. More than 10 years later, Lin et al. [64] showed that some OSes and security equipment still do not comply regarding IPv6 data overlaps, even though the new IPv6 standard [47] adopted RFC 5722's recommendation in 2017. The authors only tested a unique overlap type; therefore, they may have overestimated implementation compliance.

Censorship systems (CSes) operate similarly to NIDSes [65]–[67]. We thus also consider works that evaluated the effectiveness of evasive techniques to circumvent the censorship. In 2013, Khattak et al. [68] highlighted the Great Firewall of China (GFW), i.e., the Chinese CS, TCP/IP analysis model. The authors could infer such a model by placing a censored keyword in IP, TCP, and HTTP packets. For example, they showed that the GFW accepted packets with incorrect acknowledgement numbers or checksums, which could lead to a packet insertion in the CS flow data. They also described the IPv4 and TCP reassembly policies of the Great Firewall of China (GFW) based on nine overlap types. Khattak et al. associated a fixing cost score with every CS evasive opportunity, and they explained that solving the overlap-related opportunity would be very expensive because it would require either traffic normalization [69] or knowledge of every censored host reassembly policy. The first solution could impact the

performance, while the second one would require maintaining up-to-date host profiles by actively probing the entire nation's hosts, which would not scale well.

Four years later, Wang et al. [70] showed that the GFW model had evolved and that some evasive techniques were no longer usable. In particular, they reported that IPv4 (resp. TCP) overlap-based attacks succeeded in only 1.6% (resp. 30.8%) of the cases. However, Wang et al. only tested one overlap type. The authors and Li et al. [71] also reported that middleboxes could significantly impact the success of an evasive technique. Aliyun (Chinese cloud service provider) discarded IPv4 fragments, while QCloud (Chinese cloud service provider), China Unicom (Chinese ISP), and T-Mobile US reassembled the fragments. Both cases eliminated the overlap ambiguity and explained the low success of the IP overlap-based strategy.

1.4.2 Semi-automatically generated overlap cases

Bock et al. [72] used a genetic algorithm named Geneva (short for genetic evasion) to automatically find IP/TCP packet sequences differently processed between Chinese, Indian, and Kazakh CSes and national-level hosts. Geneva accomplished four types of packet manipulations: `drop`, `tamper headers`, `duplicate`, and `fragment`. It performed very well since it succeeded in automatically re-deriving previously found evasion strategies and uncovering new ones. Theoretically, the algorithm can mutate original packets into overlapping IPv4 and TCP chunk sequences, as it can modify the corresponding header fields and payload. However, Geneva found no such chunk sequences for this or the latter [73], [74] testing campaigns. Yet, since Geneva found successful strategies that leveraged TCP segmentation (without overlap), it is likely that overlap-based technique would also be effective, at least at the TCP level.

In 2020, Wang et al. [59] also tried to automatically find evasive TCP packet sequences against Zeek and Snort NIDSes and the GFW. Unlike Geneva, their tool, SymTCP, used symbolic execution. The method first required manually identifying accept and drop points in Linux's code, corresponding to the lines where the Linux implementation chooses to accept or drop/ignore a given TCP packet. The authors aimed to gather all the symbolized packet sequences that reached the previously identified accept and drop points. These sequences are considered to be good evasion or insertion candidates for circumvention. The authors used one to three symbolized packets to perform the state machine desynchronization, and eventually some follow-up packets if there was

no evidence of state machines' (de)synchronization with only the symbolized packets. SymTCP, just as Geneva, re-derived already discovered evasion and insertion strategies and found new ones. It successfully found one overlap-based strategy to evade Zeek version 2.6. However, while the used method could theoretically cover all overlap types, SymTCP cannot find exhaustive overlapping test cases because of its inability to model retroactive behaviors on data buffers (as rightly explained in Section IX of the paper).

One year later, Wang et al. [75] extended the SymTCP framework with Themis so as to ensure symbolic execution exhaustivity. To address the path explosion problem that arises, Themis performed state merging. The authors conducted discrepancy analysis with three symbolic packets between Linux kernel versions 3.0, 3.10, 4.4, 5.4, and 5.10. Themis could not find any TCP-based attack strategy that exploited the data overlap ambiguity. Since all the tested implementations were Linux-based, Wang et al. explained that this result was expected[11]. In theory, the tool can show discrepancies in reassembly policies if there are any. However, if none are found, one cannot easily retrieve the reassembly policy, as we can see with the tested Linux versions. The authors also highlighted that adapting the tool to any OS may require significant efforts.

In 2023, Di Paolo et al. [76] used a fuzzing-like approach to check OS conformity with the IPv6 specification [47] when processing overlapping data fragments. They derived the Shankar and Paxson model [61] by permuting and duplicating the chunks and found that none of the tested OSes (Linux, Windows, or BSD-based) complied. In their second and most exhaustive testing scenario, they performed the permutation and four duplications of the original fragment sequence (composed of six fragments). In total, 30 fragments composed the sequence, and three overlap types are covered. Oprea et al. [77] re-used Di Paolo's model to uncover IPv6 evasion and insertion opportunities on Suricata. If configured with Extreme Performance Tuning guidelines [78] and running on a Linux Ubuntu 22.04 host, they showed that exploitable attacks can target Suricata when monitoring Windows 10, OpenBSD 7.4, or FreeBSD 14.0 hosts. But, in such a configuration case, Suricata's host OS performs the reassembly. Therefore, if the monitored OS is not Linux, the traffic reaching Suricata does not correspond to the one reaching the monitored host OS.

[11]We doubt, however, of such assumption validity since we found different TCP reassembly policies between Linux kernel versions 3.16.0 and 4.9.0. See Section 4.4.1.

1.4.3 Limitations

The related works that focused on stack reassembly policies have several shortcomings. First of all, the tested overlap types differ across works, from only one type, e.g., [64], [70], to nine types, e.g., [60], [62], [68], for pairs of chunks. None of the works ensures overlap test case exhaustivity for pairs of chunks (or more). This non-exhaustiveness is at least partly due to the test case generation method. Manually, one cannot ensure exhaustiveness, while the analyzed semi-automatic methods (i.e., fuzzing or symbolic execution) are good at finding new elusive chunk sequence examples but not at exhaustive testing. Second, the used reassembly policy description formats often make their analysis and use difficult, as they consist of figures or the number of reassembled cases. Natural language sentences are easily understandable but only suitable for simple implementation policies, which may not always be true when addressing any n-tuples of overlapping chunks. Table 4.4 summarizes the limits of the analyzed works.

Section takeaways

In the last 25 years, several works have tested network stacks, such as general-purpose OSes, routers, or printers, with different overlapping chunk sequences. Throughout the years, they showed that the stacks can reassemble the sequences differently and described five IPv4 [60], five IPv6 [56] , or six TCP [62] reassembly policies regarding data overlaps. Some tried to leverage these discrepant policies to perform evasion or insertion attacks on NIDSes or CSes. Since they can be installed locally, NIDSes have been tested on laboratory conditions (i.e., the test cases did not go through the Internet), and this method has been found successful against Snort, Suricata, and Zeek with some overlap chunk sequences [59], [77], [79]. Researchers have not had easy access to CS instances; thus, they have had to infer CS behavior under real conditions (i.e., the test cases did go through the Internet). Until 2020, some works [59], [68], [70], [71] reported the (more and more) relative success of overlapping chunk-based attacks against CSes.

1.5 Implemented countermeasures inside the NIDSes

One main goal of this thesis is to improve defense capabilities against overlap-based insertion and evasion attacks. A NIDS is a security equipment that passively monitors

the network packets reaching the hosts it is tasked to defend. A NIDS can be signature or anomaly-based. A signature-based NIDS performs pattern-matching. It raises an alert if one reconstructed flow's characteristic corresponds to a signature entry from a list of known malicious signatures. Anomaly-based NIDSes generally use machine learning (ML) methods to learn normal network operating behaviors. This way, if any behavior deviates from the expectation, it is classified as suspicious, and the NIDS raises an alert.

This thesis analyzes the signature-based NIDSes Suricata [30], Snort [31], [80], and Zeek [32]. To address the overlapping data ambiguity, these NIDSes proposed two countermeasures: raise alerts whenever there is an overlap or associate host IP addresses with the reassembly policies they implement.

1.5.1 Snort

Novak and Sturges's works [60], [62] directly aimed at improving Snort regarding overlapping-based attacks. Indeed, Sourcefire owned and developed Snort at that time, and Novak and Sturges were Sourcefire's employees. They implemented the obtained IPv4 and TCP policies inside Snort. This allows NIDS users to associate monitored IP addresses with implemented reassembly policies. Users can thus configure their Snort instances to reassemble with Windows IPv4 and TCP policies for a subnet entirely composed of Windows machines, theoretically disambiguating the overlapping packet flows. Frag3 and stream5 preprocessors are Snort's related features. In the (latest) Snort version 3, users can perform the association with `stream_ip` and `stream_tcp` configuration variables in `snort.lua`. Table 1.2 details Snort's implemented reassembly policies. Snort also proposes raising alerts in case of IP or TCP data overlaps.

1.5.2 Suricata

Suricata also offers reassembly policy configurability as a countermeasure to overlap-based attacks. Like Snort, the policies are based on the work of Novak and Sturges[12]. Suricata allows one to associate an IP address or a range of IP addresses to reassembly policies in the `host-os-policy` field from the configuration file `suricata.yaml`. Table 1.3 details Suricata's implemented reassembly policies. Suricata also proposes raising alerts in case IP or TCP data overlaps.

[12]We can see in the file `defrag.c` that tests regarding the Novak and Sturges model are performed. We can also see in the file `stream-tcp-reassemble.c` that Novak and Sturges work [62] is referenced.

Protocol	Feature name	Reassembly policies	
		[60]/[62] names	implemented
IPv4/IPv6	frag3	bsd	bsd
		bsd-right	bsd-right
		linux	linux*
		first	first
		windows	windows
		solaris	solaris
		last	last
TCP	stream5	windows	-
		linux	linux, hpux10, irix
		bsd	bsd, macos, windows, win_2003*
		solaris	solaris, hpux11
		linux-old	old_linux
		macos	-
		first	first, vista, proxy
		last	last

Table 1.2: Implemented policies inside Snort. * means this is the NIDS default policy.

Protocol	Feature name	Reassembly policies	
		[60]/[62] names	implemented
IPv4/IPv6	host-os-policy	bsd	bsd, irix, hpux10*
		bsd-right	bsd-right
		linux	linux, old-linux
		first	hpux11, macos, old-solaris
		windows	windows, vista, windows2k3
		solaris	solaris
		last	-
TCP	host-os-policy	windows	-
		linux	linux
		bsd	bsd, bsd-right, hpux10, irix,macos, windows, windows2k3*
		solaris	hpux11, solaris, old-solaris
		linux-old	old-linux
		macos	-
		first	vista
		last	-

Table 1.3: Implemented policies inside Suricata. * means this is the NIDS default policy.

1.5.3 Zeek

Zeek is the Bro NIDS [81] successor. There is some evidence showing that several TCP and IP reassembly policies were implemented in Zeek/Bro, as part of the "Active Mapping" function introduced in [61] (see, for example, [82], [83]). But in 2010, Vern Paxson explained that it was not possible to use this function anymore [84]. We did not find any inconsistent information on that matter after that: for example, in files `Frag.cc` and `Reassem.cc`[13], there is only one way for fragments and segments to be reassembled. Instead, Zeek proposes raising an alert whenever it processes overlapping chunks and even inconsistent overlapping chunks (i.e., different data for the same byte offset) to address overlap-related issues.

1.5.4 Limitations

As mentioned, Snort and Suricata's implemented policies are based on [60], [62] works that date respectively from 2005 and 2007. No study has performed a complete reassembly discrepancy analysis with OSes regarding test cases and implemented policies. Indeed, Oprea et al. [77] actually tested Linux's (not Suricata's) reassembly consistency with three OSes, and none of the works proved test case exhaustiveness. The policies of the addressed implementations, i.e., OSes, routers, and printers, may also have evolved since Novak et al.'s studies. Finally, the configurable Suricata and Snort NIDSes currently apply $n = 2$ pair policies to any n overlapping chunk sequence. However, no work has ever verified that such pair policies could be generalized to any n.

Section takeaways

To address the overlap ambiguity issue, NIDSes proposed two countermeasures: associate the monitored hosts' IP addresses with a given reassembly policy from a set of implemented policies or raise an alert whenever a data overlap occurs. In this thesis, we test Snort and Suricata, which propose the reassembly policy configurability, and Zeek, which always applies the same policy. Snort and Suricata policies are based on Novak and Sturges [60], [62] pair-based reassembly policies published in 2005 and 2007.

[13] See https://github.com/zeek/zeek/blob/release/6.1/src/Reassem.cc and https://github.com/zeek/zeek/blob/release/6.1/src/Frag.cc.

Chapter 2

Overlap-based attack characterization

Contents

2.1 Problem formulation 49
2.1.1 Definitions 50
2.1.2 Illustration 50
2.2 Threat Model 51
2.2.1 Gaining monitored host and NIDS reassembly policy knowledge 52
2.2.2 Crafting the data chunks 53
2.3 Desynchronization persistence across protocol data buffers after the overlap attack 54
2.3.1 *p* protocol data buffer desynchronization persistence 54
2.3.2 Connection desynchronization of an upper protocol 56

This section first defines insertions and evasions in the specific context of overlap-based attacks. It then details the attacker capabilities required to exploit OS and NIDS reassembly discrepancies. Finally, it discusses the attack's impact duration. Part of this section's contributions were introduced in our paper [33].

2.1 Problem formulation

An attacker can use the overlap data ambiguity to exploit NIDS and host reassembly divergence. They would perform a so-called *insertion* or *evasion* attack. We extend Ptacek and Newsham [1] and Wang et al. [59] insertion and evasion attack definitions to fit the context of IP and TCP chunk overlaps. To recall, the main shortcomings were that:

- Ptacek and Newsham consider a data chunk as being either *totally* accepted or dropped but not *partially* accepted. With the overlap from Figure 2, the "ATTACK"/"AT00CK" reassembly divergence is impossible.
- Wang et al. do not consider on purpose malicious (or "filtered") payload insertion inside the NIDS flow data, nor do they consider IP-based attacks.

We treat the NIDSes and the host OSes as opaque boxes in the following. We consider the IP and TCP data pushed to the upper layer; thus, our definitions apply to both IP and TCP levels.

2.1.1 Definitions

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

Definition 5 — p protocol data buffer synchronization. Given a chunk sequence $c_f ... c_l \in C^p$, we say that the NIDS and the monitored host have their p data buffers synchronized between t_f, the arrival time of the first considered chunk, and t_l, the arrival time of the last considered chunk, if the next upper-layer data streams are the same.

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

Definition 6 — Evasion in a data overlap-based context. An evasion attack consists of *some malicious payload* that is not visible in the data analyzed by the NIDS while it is visible on the *monitored host*'s reassembled payload.

Definition 7 — Insertion in a data overlap-based context. Symmetrically, an insertion attack consists of *some malicious payload* that is visible in the flow data analyzed by the *NIDS* while it is not on the host's reassembled payload.

2.1.2 Illustration

We use the overlap chunk sequence introduced in Figure 2 to illustrate insertion and evasion attack types in Table 2.1. As we can see, the non-targeted host either reassembles differently (with a benign payload, for example) or completely ignores the chunk sequence. Data buffer desynchronization is one of the most dreaded risks for a NIDS since it eventually allows an attacker to bypass all its security mechanisms. See some related CVEs [85]–[89], whose scores are high or critical. On the one hand, attackers

Attack type	Implem.	Target	Reassembled data	Attack scenario	Works [1]	Works [59]	Works Us
Evasion	NIDS		-	E1	✓	✓	✓
	Sup. host	✗	"ATTACK"				
	NIDS		"AT00CK"	E2		✓	✓
	Sup. host	✗	"ATTACK"				
Insertion	NIDS	✗	"ATTACK"	I1	✓		✓
	Sup. host		-				
	NIDS	✗	"ATTACK"	I2			✓
	Sup. host		"AT00CK"				

Table 2.1: Attack types illustrated with Figure 2 reassembly cases and based on Definitions 6 and 7. - means the implementation *ignores* the flow chunk data.

can use evasions to circumvent the NIDS inspection function. An alert pattern can thus reach the monitored host without the NIDS noticing it, as [1], [59] did. In this case, "ATTACK" consists of a well-chosen NIDS filtered pattern. An evasion can also impact other NIDS functionalities, such as file or TLS certificate extractions, since a unique (overlapped) bit is sufficient to corrupt them. "ATTACK" is the file or TLS certificate in this case. On the other hand, the insertion attack can alter the NIDS's normal behavior, for instance, by exploiting a known NIDS vulnerability (e.g., [90]–[92]). "ATTACK", thus, corresponds to the NIDS vulnerability exploit. Or it can raise false positive alerts, wasting analysts' time, in which "ATTACK" is a NIDS filtered pattern.

Section takeaways

An attacker can use insertion or evasion attacks to desynchronize NIDS and monitored host data buffers. The standard (i.e., from related works) model adapted to IP and TCP overlap-based evasion and insertion attacks comprises three reassembly possibilities. Either the host reassembles with the malicious payload, a benign payload, or ignores the overlapping chunks. The malicious payload is contained in the monitored host (resp. NIDS) flow data in evasion (resp. insertion) attacks.

2.2 Threat Model

The threat model we consider consists of an attacker, a NIDS, and a monitored host, as illustrated in Figure 2.1. The NIDS gets a copy of all the network packets the host

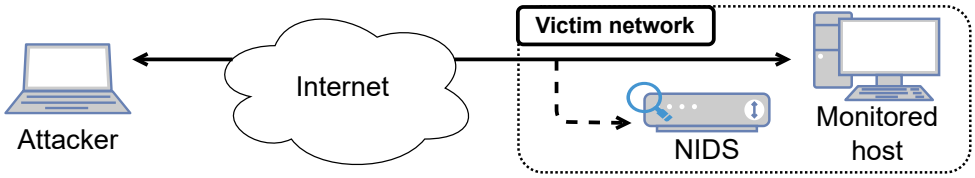


Figure 2.1: The threat model considered in the thesis. The dotted line corresponds to a copy of the network traffic.

receives and sends. Depending on the context, we will consider that the NIDSes offering the reassembly policy configurability are instrumented by associating the monitored host IP with the corresponding policy[1] or using the default policy. The attacker should be able to:

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

2.2.1 Gaining monitored host and NIDS reassembly policy knowledge

The attacker may want to learn about the monitored host and NIDS reassembly policies to increase the chances of a successful attack. A good approximation to learn about the host reassembly policy is to determine the host OS. Several tools exist to perform OS fingerprinting. Active ones, such as Nmap [93], Hershel(+) [94], gait [95] or JA4TCPScan [96], use specifically crafted packet sequences or retransmission times to identify a host OS, while passive tools, such as p0f [97], nPrintML [98], or JA4TCP/JA4TCPServer from the JA4+ tool suite [99], analyze packet header fields in existing communications. The attacker can either assume that the NIDS is correctly configured, and thus use the previously obtained OS knowledge to determine the related policy, or that the NIDS uses its default policies. Finally, they can craft an overlap chunk sequence based on the results reported in Sections 4.3 and 4.4.1 and their objectives.

[1]Based on the current knowledge, we consider that hosts with more recent OS versions than the ones tested by Novak and Sturges in [60], [62] (e.g., Debian 12) should have the latest OS family representative reassembly policy (e.g., *linux* because Linux 2.4 was the latest tested version) associated in the NIDS configuration file.

2.2.2 Crafting the data chunks

According to the selected overlap case, the attacker chunks the original malicious packet into several pieces. They must appropriately manipulate the header fields `Fragment Offset` and `More Fragments` (resp. `Sequence Number`) to perform the IP (resp. TCP) attack. But, the other header fields of the crafted chunks must be consistent with the carried data (e.g., correct IP or TCP `checksum`, correct `length`). The original payload must be in plaintext[2], and choosing the malicious, (i.e., "ATTACK" in Table 2.1) payload depends on what the attacker wishes to do. However, there are some constraints on the overlapping data portion which is *not* visible in the non-target flow data, especially concerning the syntactic and semantic correctness of the upper-layer protocols. This correctness depends on the performed attack scenario, i.e., E1, E2, I1, or I2, as described in Table 2.1.

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

E2 and I2-related constraints The NIDS and the monitored host both push data to the upper-layer protocol. For the "AT00CK" reconstruction to be harmless, it must syntactically and semantically conform with all the upper protocols. In particular, as for any upper-layer checksum: 1) if the checksum is contained within the overlapping portion, then it should be correctly adjusted to fit the "AT00CK" reassembled payload; otherwise, 2) the overlapping data should be adapted to fit the checksum[3]. Any upper-layer syntax or semantic disrespect may cause unwanted side effects (e.g., NIDS alerts, NIDS or monitored host failure) that could affect the attacker's stealthiness.

Note that we considered that "ATTACK" and "AT00CK" payloads have the same length. However, this may not be the case if several rightmost IP fragments overlap or a TCP implementation does not reassemble overlapping data. In that case, using an upper-layer

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

[3]Only two dedicated octets are required to make a payload fit any internet checksum because it is computed with 2-octet words [8]. See Appendix C "Deceiving TCP checksum" of Feng et al. work [100] for a payload crafting example. We will use this same principle to generate the data for our test cases in Section 4.1.1.3.

protocol with a `length`-related header field can be more difficult. See Section 4.1.1.3 for more details.

Section takeaways

The considered threat model consists of an attacker sending crafted (overlapping) fragments or segments, an NIDS, and a monitored host. The attacker can determine the NIDS and monitored host reassembly policies so as to find suitable evasive or insertion chunks. If the NIDS or host ignores the chunks, the overlapping data portion can carry any data. Otherwise, the overlap data portion should be semantically and syntactically correct.

2.3 Desynchronization persistence across protocol data buffers after the overlap attack

An attacker may need to hide data from multiple p protocol packets. However, it is possible that the NIDS and monitored host p's data buffers re-synchronize after the overlapping chunks. This section gives some insights regarding the desynchronization persistence of the p buffers and higher stateful protocol buffers, if any is present in the p data.

2.3.1 p protocol data buffer desynchronization persistence

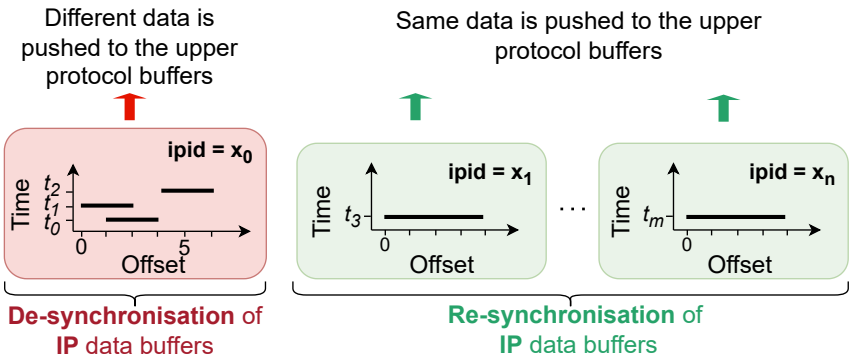


Figure 2.2: IP data buffer re-synchronization illustration.

IPv4 and IPv6 Due to its statelessness, when performing IP overlap-based attacks, the IP data buffer desynchronization only lasts for the duration of the fragment reassembly.

In Figure 2.2, it corresponds to fragments c_0, c_1, and c_2 arriving from t_0 to t_2. Data is pushed to the upper layer only after all the fragments have arrived. Theoretically, one IPv4 datagram can carry up to 65,535 data octets, and thus, the data length sum of the datagram's fragments can also reach 65,535 octets. However, since the reassembly timeout is usually very short (e.g., 30 seconds maximum in Linux [101]) and the Maximum Transmission Unit (MTU) on the Internet is generally at most 1,500 octets [102], [103], the original datagram length and related fragment number should be limited. The NIDS and monitored host buffers are re-synchronized once the `IPID` moves to the next value if subsequent packets are complete datagrams or non-overlapping fragments. When this happens, both IP implementations generate the same data again. c_3 to c_m fragments illustrate the IP data buffer re-synchronization in Figure 2.2.

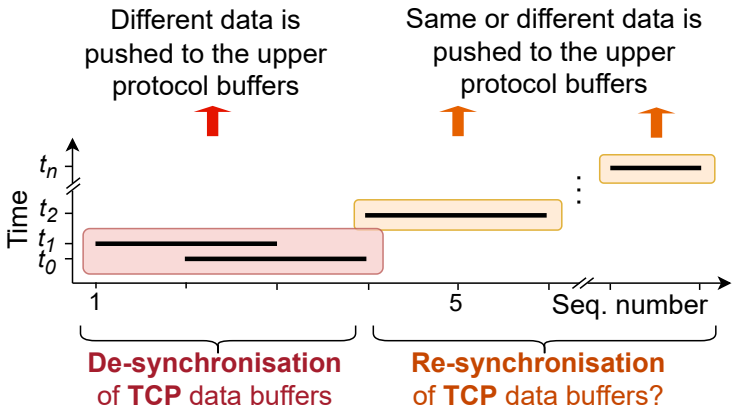


Figure 2.3: TCP data buffer re-synchronization illustration.

TCP Since TCP is a streaming protocol, data is regularly pushed to the upper layer for a given TCP session. The overlap attack that occurs from t_0 to t_1 in Figure 2.3 may have a lasting impact on the subsequent segment reassembly. Segments arriving after the overlap segment sequence (i.e., from t_2 in Figure 2.3) are *reassembled differently* when the overlapping segment reassembly ends in E1 and I1 scenarii. The NIDS (resp. the monitored host) indeed drops at least one of the two overlapping segments in E1 (resp. I1). A hole exists in its TCP buffer; thus, subsequent contiguous segment data is not pushed to the upper-layer data stream. In this case, TCP data buffer *desynchronization persists*. On the contrary, when E2 or I2 attack scenarii are performed, the NIDS and the monitored host *push the same data* to the upper-layer protocol after t_2. The TCP desynchronization is here only *temporary*.

Note that Figure 2.3 is a simplified view of the implementation behavior since the chunks that the end host sends are not represented.

2.3.2 Connection desynchronization of an upper protocol

Since the considered IP fragments and TCP segments are data chunks, one or several higher protocols are encapsulated within their payload. Well-chosen overlapping data may provoke the NIDS and monitored host connection desynchronization of a stateful upper-layer protocol sp^i, with $i \in [1, 2, ...]$. Suppose the attacker aims to hide data at this stateful protocol level or any higher layer. In that case, the attacker can use the connection sp^i protocol desynchronization in conjunction with the p desynchronization to continue hiding some malicious data until the remaining sp^i connection lifetime. The malicious payload can thus be located within the overlap data portion or in any follow-up p packet as long as this connection holds.

For example, if we consider a $p = IP$-based evasion attack and TCP as the next protocol. We wish to desynchronize sp^1. The fragment data portion overlaps the sequence number field of the TCP header, with one of the values being carefully manipulated. An artificial hole in the NIDS's TCP stream is introduced, and no data will subsequently be pushed to the NIDS application data stream.

The sp^i protocol specification or some works exploiting either specification ambiguities or bad specification implementations [1], [59], [72], [75], [104]–[106] give other examples of how to alter the connection state of stateful protocols.

Section takeaways

Evasion and insertion aim to desynchronize the NIDS and monitored host IP or TCP data buffers. At some point, these hosts' buffers can resynchronise, and the generated data can start to be the same again. Yet, by carefully manipulating the header fields of any stateful upper protocol, an attacker could make the desynchronisation last as long as this upper protocol connection holds.

Chapter 3

IPv4 fragmentation characteristics from Internet traffic

Contents

3.1 Method 57
3.1.1 MAWI dataset 58
3.1.2 Analyzing tools 58
3.1.3 Series construction methods 59
3.2 IPv4 fragment characteristics 60
3.2.1 Overview 60
3.2.2 Series types 61
3.2.3 Size analysis 62
3.2.4 Upper protocol analysis 65
3.2.5 Source IP host diversity 67
3.3 Analysis of series with misassociation evidence 69
3.3.1 RFC compliance 69
3.3.2 Header field analysis 71
3.4 Conclusion 73

In this chapter, we (temporarily) take a step aside from the overlap-based attack analysis by studying some characteristics of IPv4 fragmentation. First of all, we outline the method used to conduct the analysis. Second, we describe the general attributes of the IPv4 fragments in the analyzed MAWI dataset. Finally, we quantify the extent to which hosts are subject to generating data overlaps through the analysis of fragment misassociation.

3.1 Method

This section describes the analyzed dataset and the tools used to study them. Then, it slightly modifies the *fragment series* definition from related works and details two

construction methods for the series analysis.

3.1.1 MAWI dataset

MAWI traffic repository [7] is a collection of daily PCAP traces, captured at a transit link in WIDE project (AS2500) with tcpdump. From July 1, 2006, Samplepoint-F has captured 15 minutes of traffic every day at 2:00 p.m. Japan Standard Time. This network traffic is particularly valuable as it can catch Internet traffic characteristics' evolution, and particularly IPv4 fragment ones. Only some anonymized outermost protocol headers (e.g., IPv4, IPv6, TCP, UDP, HTTP) are exposed publicly. We analyze 100 random traces a year from the first complete year, i.e., 2007, to 2024. Our dataset thus comprises 1,800 traces of 15-minute traffic captures, totaling 503 GB of fragmented IPv4 traffic (over 103 TO IPv4 traffic) and 492,091,401 fragments (over 145,441,073,422 IPv4 packets).

The main limits related to the MAWI dataset in our case are that i) the analyzed fragments may not be ordered and structured the same as the ones received by the end hosts because of fragment delays or alteration by on-path equipment, ii) the time of the capture is the same for every trace, and iii) the WIDE network's users may not be representative of users, since WIDE only provides Internet to universities and research institutes. Despite these limitations, the MAWI dataset remains a valuable resource for measuring IPv4 fragmentation characteristics inside real Internet traffic.

3.1.2 Analyzing tools

We use the Capinfos tool [107] to compute generic statistics on the complete MAWI traces (with and without the fragments). To build and analyze the fragment series, we developed a plugin within the Pcap Analysis Library (PAL) [108], a set of Rust tools for analyzing PCAP files. PAL behaves as an end-host reassembly engine. In other words, it has a fragment queue, reconstructs the original datagrams when possible, and has a reassembly timeout. We use a 30-second reassembly timeout, which corresponds to Linux's [109]. As in Linux, this timeout is initialized when processing the first series fragment but is not reset at every series fragment's arrival because, otherwise, a fragment series could outlive the 2-minute Maximum Segment Lifetime (MSL).

3.1.3 Series construction methods

Shannon et al. [2] defined a fragment series as "the ordered list (as monitored on the network) of fragments whose source is a single original datagram". In practice, they grouped in series all fragments with the same four-tuple ft, consisting of ⟨`srcIP`, `dstIP`, `IPID`, `next proto`⟩, every 600 seconds. John and Olovsson [3] used 1 second. In other words, starting from time t_0 when the first ft fragment appears in the trace, all the following ft fragments captured within $[t_0, t_{0+x}]$ are grouped altogether to form one unique series. The next ft fragment that is located after t_{0+x} will start forming another series that will last x seconds, and so on. The authors of these works could, therefore, not be sure that the fragments of the constructed series were from a "single original datagram" as the definition states, because a four-tuple's `IPID` could be reused in multiple original datagrams within any chosen time window.

To clarify this ambiguity, we rather consider a fragment series as *the ordered list (as monitored on the network) of fragments that have the same four-tuple*. We use two methods to build the series: PAL's stateful method and the related works' time window method. It is important to note that these two methods construct *different* series, but we use them for different purpose.

In Section 3.2, where we analyze some general fragmentation characteristics, we use the stateful method because it is closest to the end-host behaviors and it works fine with hosts sending many data and reusing `IPID`, as it produces as many series as filled fragment queues. In this section, we also measure the overlaps and we consider that a fragment overlap is very likely to occur at the end-host if the stateful method constructs *overlap* and *duplicate* series (as defined in Section 1.2.1.1).

In Section 3.3, we analyse the fragment overlaps from a different perspective. As described in Section 1.1.1.4, `IPID` reuse for a new original datagram within the receiver's reassembly timeout can (benignly) cause the fragments from different original datagrams to be misassociated, and in some cases, to overlap. RFC 6864 [9] described that the increasing host throughput could result in `IPID` that are reused frequently. This RFC thus reaffirmed that the hosts sending non-atomic datagrams, i.e., fragments that can be fragmented on-path, must rate-limit themselves to prevent `IPID` reuse within the receiver's reassembly timeout. The rate limitation should prevent fragment misassociation, and the possible related data corruption at the receiver's side. Therefore, we measure to what extend fragments have chances to be misassociated, or to overlap, in Section 3.3. When constructing series using the time window method, we argue that

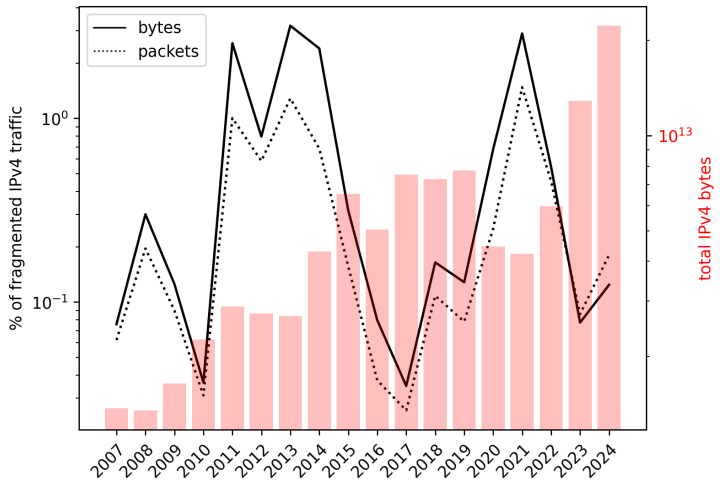


Figure 3.1: Evolution of IPv4 fragmentation in MAWI.

overlapping fragments inside the series can be a sign of (benign) misassociation as long as a malicious sender did not intentionally create such overlap.

NB We do not filter the first and last seconds of each PCAP trace (as [3] did) because it would just move the cold start and end problem. We are aware that some series could be wrongly classified as *incomplete* because the traces lack some fragments before and after the capture boundaries. We will quantify such event occurrence in the related result parts.

3.2 IPv4 fragment characteristics

After a brief fragment series overview, we will analyze some size characteristics (PMTU, original datagram size), next fragment protocol attributes, and the diversity of source IP addresses which traffic is fragmented.

3.2.1 Overview

Between 2007 and 2024, at least 0.03% of the overall IPv4 traffic is fragmented yearly, as shown in Figure 3.1. Even if the fragmentation proportion is not stable over time, the overall fragment volume increases since the IPv4 traffic volume increases. The fragmentation peaks in 2011, 2013, 2014, and 2021 are not correlated with particular

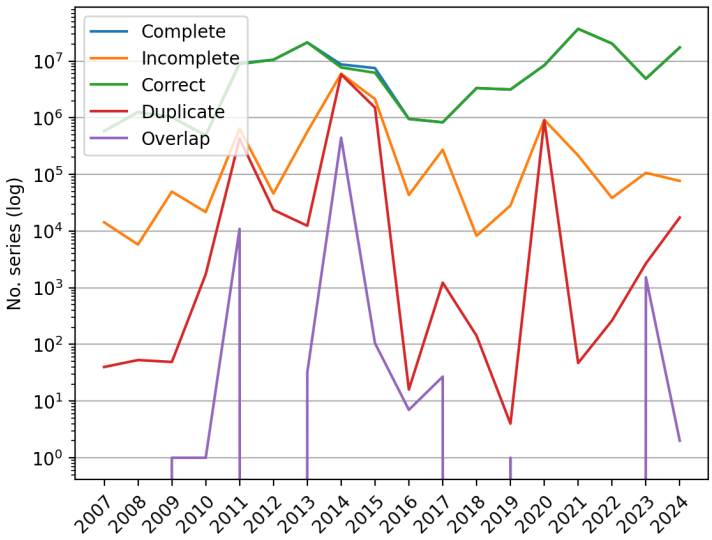


Figure 3.2: Series type distribution. There are 168,498,212 series in total.

IPv4 traffic peaks. As expected, on average, fragmented packets carry more bytes than non-fragmented packets, since the packet number's curve is almost always under the bytes' curve. We also observe a high increase in the overall IPv4 traffic volume in 2023 and 2024.

3.2.2 Series types

The constructed fragment series can be described with types, as defined in Section 1.2.1.1 and illustrated in Figure 1.7. To recall, a *complete* series has no data hole, a *correct* series has no data hole and no data overlap, while there are at least two fragments that fully (resp. partially) overlap in a *duplicate* (resp. *overlap*) series.

3.2.2.1 Overview

Figure 3.2 illustrates the series type evolution in the analyzed traces, based on the stateful construction method. We remark that *complete* and *correct* curves are very close, which is due to the quite low number of *duplicate* and *overlap* series. Moreover, 8.67% of the *incomplete* series are located in the first and last 30 seconds, meaning that these series could actually arrive complete at the end-host.

3.2.2.2 Analysis of series with overlaps

At least a few duplicate series are observed every year, which is not the case of overlap series that are only observed some years. We remark that duplicate series peaks always go along with overlap peaks, though. During the 18 years, we count a total number of 8,729,600 duplicate and 457,978 overlap series, representing 5.18% and 0.27% of the series, respectively. Duplicate series can be composed of network duplicates; but, in MAWI traces, we do not have access to the packet data to directly check that. As [3] did, we can, however, check the consistency of the next protocol IP checksum inside fragments with a null `offset`, when such fragments overlap. We count 5,017,472 duplicate series that have at least two different leftmost fragments, representing 3.00% of the overall series. These overlap and duplicate series will be analyzed from a different perspective in Section 3.3.

3.2.3 Size analysis

The subsection focuses on PMTUs and original datagram sizes.

3.2.3.1 PMTU

The PMTU, i.e., the minimal MTU value across the followed links, between the sending and receiving hosts has a major impact on the IPv4 fragmentation. The data sizes of the fragments with the `MF` bit set (or the series' byte-wise leftmost fragments) should correspond to the PMTU at the moment of the capture, unless a suspicious sending host actively fragmented the application data uncorrelated with the next hop's MTU.

Table 3.1 reports the overall observed top 10 PMTUs and compares them with those described by Shannon et al. Interestingly, 1480 only represents two third of the left fragment sizes in the analyzed traces. This is a 20% drop in comparison with Shannon et al. [2], and thus, the nine other values observed represent a much bigger proportion in our case. We also observe only one value under 1376 bytes: 792 or 1280 with weighted data. MTUs thus tend to stabilize between 1376 and 1480 bytes since Shannon et al.'s study. The high difference for 1440 between our raw and weighted columns (i.e., 7%) is due to the value over-representation since 2022, which accounts for between 23.59% and 84.38% of the series during these three years, while it never exceeds 2.65% in the other years. In addition to 1480, the values 1472, 1376, and 1432 are observed in both

Shannon et al. [2] (2001)		Ours (2007-2024)		Ours (2007-2024)*	
Size	**Series**	**Sizes**	**Series**	**Size**	**Series**
1480	85.314%	**1480**	67.613%	**1480**	66.917%
1464	11.112%	1440	15.286%	**1432**	9.318%
552	1.186%	1408	5.325%	1440	8.523%
1472	1.086%	**1432**	5.234%	1408	2.889%
1476	0.455%	1424	1.881%	1424	2.417%
1336	0.183%	1416	1.530%	1416	1.873%
1376	0.120%	**1376**	0.598%	**1376**	1.413%
104	0.115%	1400	0.531%	**1472**	1.023%
744	0.097%	792	0.288%	1280	0.961%
1432	0.068%	**1472**	0.283%	1456	0.925%

Table 3.1: Top 10 left fragment sizes. Bold values are shared with [2]. * means the data is normalized so that every year has the same weight.

of our columns. While 1472 is likely due to PPPoE, which adds an 8-byte overhead in DSL connections, the other values are likely caused by overhead related to VPN tunnels or by mobile traffic [103]. These PMTUs over 1376 are consistent with the TCP stacks' advertised MSS that Custura et al. [103] reported in 2018.

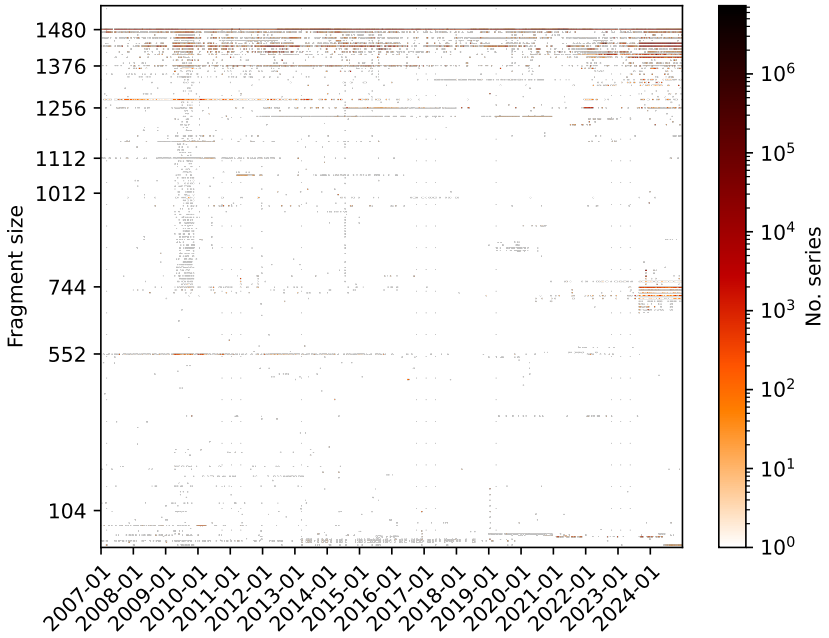


Figure 3.3: Left, i.e., with bit `MF` unset, fragment sizes.

Figure 3.3 provides a fine-grain evolution representation of the left fragment sizes. Except the 1480-related line, we do not observe other full horizontal lines, which represent a presence in all the 1,800 traces. The other lines above 1400 are discontinued, showing that these PMTUs are present in all years but not in all the traces. Others clearly decreased a lot at some point; for example, 552, 1280, 1376 stopped in 2017. To conclude, PMTUs have evolved in line with the Internet, i.e., the increase of VPN tunnel usage and mobile traffic.

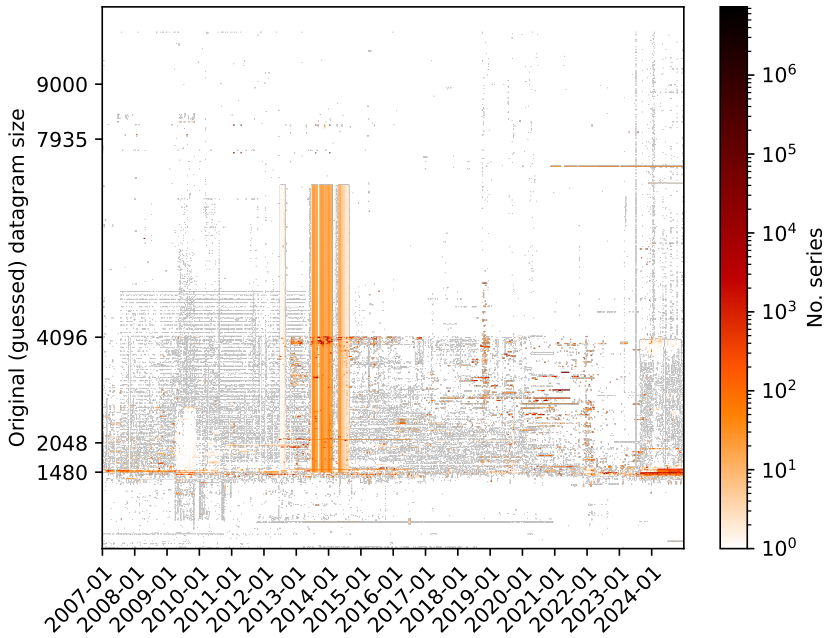


Figure 3.4: Original (guessed) datagram size across fragment series.

3.2.3.2 (Guessed) original datagram sizes

For each fragment series, we deduce the original datagram sizes by looking at the rightmost, i.e., with bit `MF` set, fragment of the series. Figure 3.4 is a representation of the (guessed) original datagram sizes in the analyzed MAWI traces. First of all, we observe few original sizes under 1480 across the years. It means that the size of the original IPv4 packets (which get fragmented) is at least 1480. We remark horizontal dots, or a line, at 1480 bytes, which is due to the PMTUs under 1480, as well as a high concentration just above 1480. These sizes are typical encapsulation evidence (VPN, encryption, routing), since encapsulation adds some extra header bytes to the sender's original frag-

ment size. Interestingly, there are no clear lines at 1480 bytes' multiple, as we could have expected. Instead, the original datagram sizes seem quite randomly distributed between 1480 and 4000 bytes.

3.2.4 Upper protocol analysis

This subsection first studies the next IP protocol; then, it analyzes the UDP services to understand why UDP is by far the most fragmented following protocol.

3.2.4.1 Next IP protocol

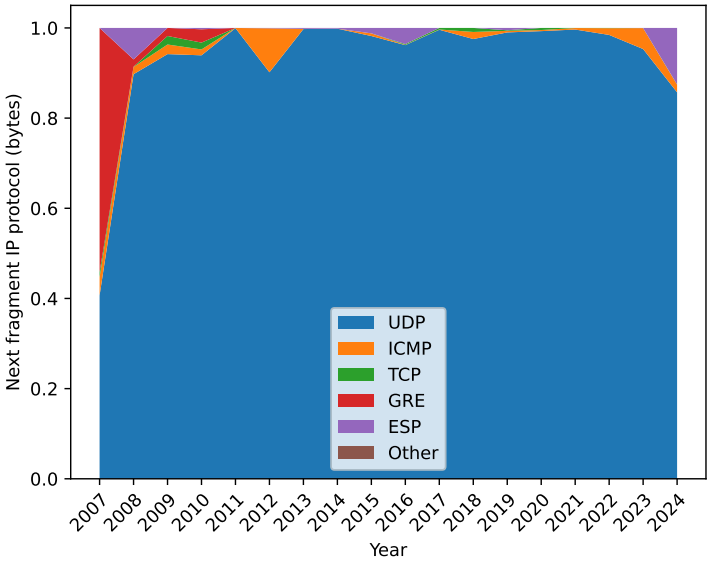


Figure 3.5: Next IP protocol distribution (bytes).

Five protocols are quite well distributed across the years: UDP, ICMP, TCP, GRE, and ESP. Figure 3.5 plots the next IP protocol distribution over the years. As expected, UDP is the most fragmented next protocol, representing 93% of every year's fragmented traffic (packets and bytes) on average. The UDP fragmented traffic accounted for less than a percent only in five out of the 18 analyzed years, and it reached more than 10% of the overall UDP traffic in 2011, 2013, and 2021. ICMP represents 1% of the fragments on average, with two peaks in 2012 and 2021/2022. TCP fragmentation is not very common since it represents only 7.60×10^{-4}% of the IPv4/TCP traffic. Notably,

fragmented GRE constantly decreased over the years until 2015, from 18% (resp. 58%) of the overall IPv4/GRE (resp. fragmented) traffic in 2007.

3.2.4.2 UDP services

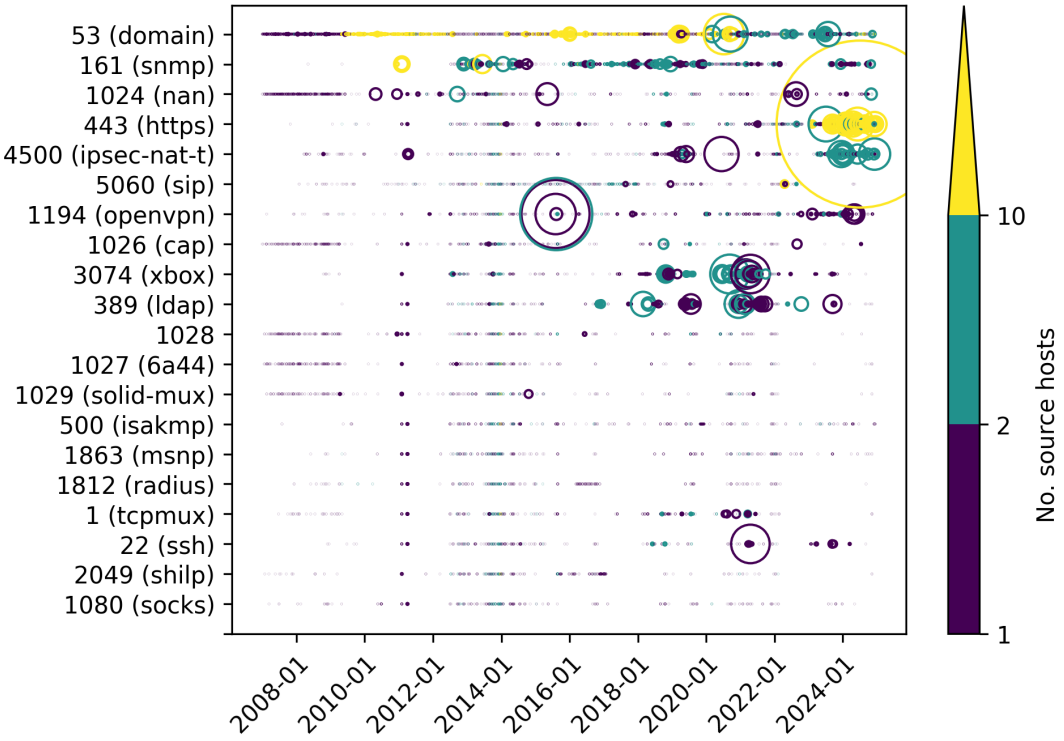


Figure 3.6: Top 20 observed UDP services in the MAWI traces. Labels are IANA's [110].

The service is identified with the destination and source ports. If both ports are assigned in IANA [110], the series is classified as the lowest value-related service. Throughout the years, Figure 3.6 reports the top 20 observed UDP services that this method successfully classified. The more a service is distributed among the traces, the higher it is ranked. One circle is related to one trace, and the circle size is proportional to the service traffic volume in the trace. Furthermore, the color indicates the diversity of source IP addresses within a PCAP file. If a circle is yellow, it means that the number of different series source hosts is greater than 10 in the corresponding trace.

As expected, DNS is the most fragmented UDP services across the years. The use of DNS Security Extensions (DNSSEC), which protect DNS Resource Record through digital signatures, is the cause of such fragmentation as already reported in the literature

(e.g., [111], [112]). There is a high host diversity causing fragmented DNS from 2009 to 2020. Figure 3.6 also shows an increasing host diversity and fragment series volume for HTTPS/QUIC since 2007. There are peaks for some services (CHARGLEN, XBOX, LDAP, OPENVPN, IPSEC-NAT-T, HTTPS/QUIC); otherwise, all the services are observed fragmented at least a bit all the years. Two traces show some abnormally high volume of HTTPS/QUIC and XBOX traffic. Weirdly, SNMP (from 2013 to 2024) and LDAP (from 2017 to 2022) services are present in the traces, while they should not be present on the Internet. When comparing with Shannon et al.[2], we do not observe common UDP services. They observed that most of the traffic they could classify was tunnel-related, which is not the case here. However, we also observe some OPENVPN and IPSEC-NAT-T tunneled traffic. Since these protocols may use random ports, the number of fragmented OPENVPN and IPSEC-NAT-T is likely underestimated. Finally, 8 out of the 20 services are quite unconventional (port numbers 1, 19, 1024, 1026 to 1029, and 1080). Some of the related horizontal dots, or lines, seem to overlap, with a unique host sending such traffic. We thus presume a suspicious host sent this unrepresentative traffic.

3.2.5 Source IP host diversity

The next characteristics we focus on here is the diversity of fragment series `srcIP` hosts and three-tuples $\langle$`srcIP`, `dstIP`, `next proto`$\rangle$. Indeed, the more diverse the hosts are, the more devices or IPv4 implementation stacks are concerned by fragmentation, and thus, the more the fragmentation is expanded over the Internet.

Figure 3.7 plots the number of source hosts and that of three-tuples over the years. We remark that the number of source hosts which traffic is fragmented is not stable over time, without a general increase nor decrease. The minimum is at 493 in 2021, the maximum at 25,567 in 2009, and the mean is 5,687. To understand whether only one or few problematic routers cause the fragmentation on the datagrams's path, we group /32 addresses that belong to the same /24 prefixes. On average, we find that the number of /32 hosts is two times the number of /24 prefixes. Because the theoretical IPv4 address space of /24 prefix space is $2^{32-24} = 2^8 = 256$ times lower /32's one, a lot of /32 hosts are unique in their corresponding /24 prefix. Just as the source IP address numbers, the number of three-tuples is not stable. There is not a perfect correlation with the number of source hosts, since there can be i) a relative low number of source hosts but high number of three-tuples as in 2008, 2023, or 2024, and ii) a close number

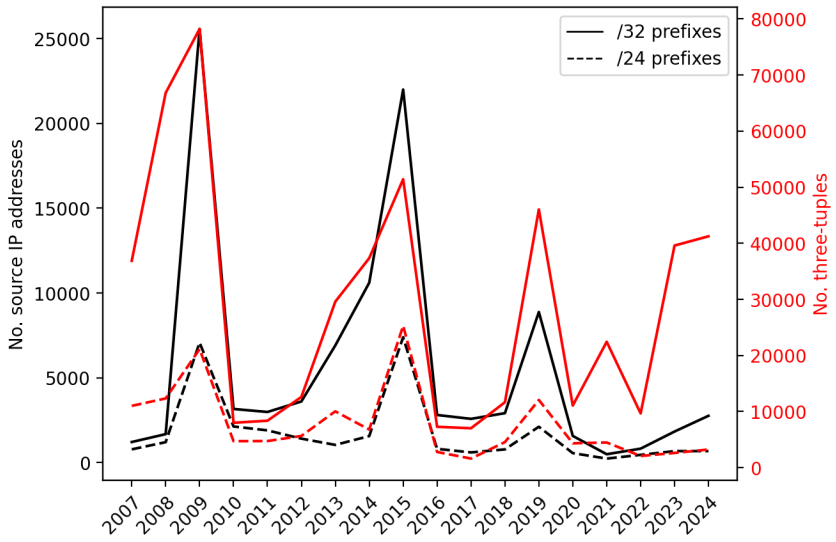


Figure 3.7: Number of source hosts and three-tuples.

of source hosts and three-tuples as between 2009 and 2014 and between 2016 and 2018. However, even if there is a large number of hosts which traffic is fragmented at some points, only one host is responsible for more than 40% of the series in seven years, and it even reaches 70% in 2011 and 80% in 2022. It is only in 2009, 2010, and 2023 that it took more than five hosts to reach 50% of the fragmented series.

Section takeaways

IPv4 fragmentation is still observed and accounts for at least 0.03% every year. UDP, TCP, ICMP, GRE, and ESP are the next IP protocols the most observed. They are well distributed across the years, except for GRE that stopped in 2015. UDP/DNS is the most fragmented service, and represents 15.84% of all the UDP/DNS IPv4 traffic. Other UDP services are constantly observed over the years, such as QUIC, IPsec, or OpenVPN. On average, 5,687 hosts send some IPv4 traffic that is at some point fragmented, but a few talkative hosts usually concentrate most of the traffic. Finally, fragments overlap partially in 0.27% of the series and fully in 3.00 to 5.18% of the series.

3.3 Analysis of series with misassociation evidence

There have been concerns that the increasing host throughput would lead to more fragment misassociations, and thus, data corruption, as explained in Section 3.1.3. In this section, we first assess whether the hosts implemented the RFC 6864 countermeasure, i.e., rate-limited themselves, to prevent this problem from happening when sending non-atomic datagrams. Second, we perform some header field analysis to evaluate whether the series were not generated by suspicious (malicious, buggy) hosts.

3.3.1 RFC compliance

This subsection details the method used to evaluate if the hosts rate-limit themselves. Then, it describes the results.

3.3.1.1 Method

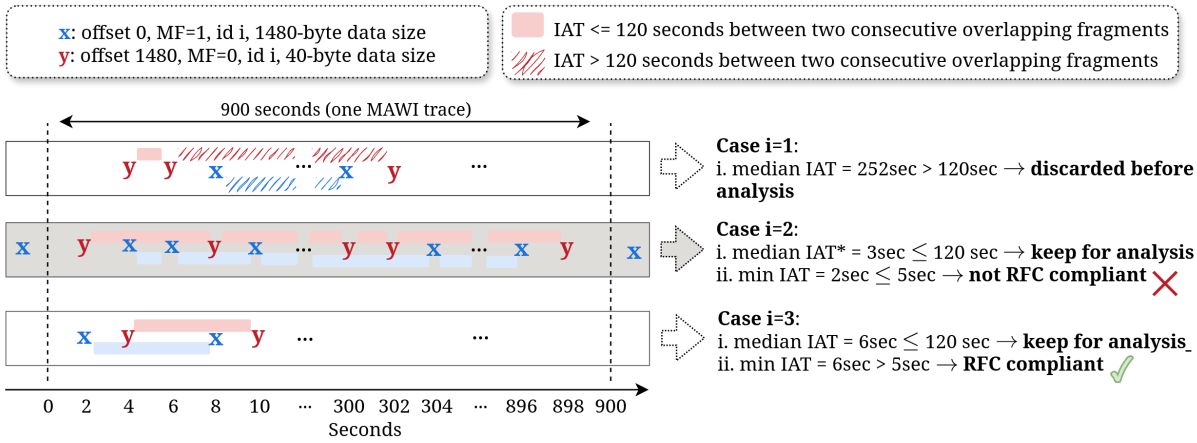


Figure 3.8: Illustration of series filtering and assessment of a `srcIP` host compliance with RFC 6864. * means that the IAT median is computed with only the represented IATs (other x and y fragments are located in the . . . , but the x and y shown are representative of the other x and y).

To evaluate the hosts' compliance with RFC 6864, we use a multi-step method, which is represented in Figure 3.8. First, we gather all the fragments that have the same four-tuples ⟨`srcIP`, `dstIP`, `IPID`, `next proto`⟩ in the traces. It corresponds to using a 900-second time window to group the fragments in series [2], [3]. Inside these series, overlapping fragments, i.e., fragments that carry data for the same fragment offset, are

likely to be from different original datagrams, except if they are network duplicates. By analyzing the inter-arrival times between the series' overlapping fragments, we deduce if an host rate limit itself. Such series correspond to *overlap* and *duplicate* series in Shannon et al. [2] terminology. However, we only study *duplicate* series with fragments of `offset` zero that have different transport header checksums, thereby preventing the inclusion of network duplicates. In the following, *we consider only series with fragments that overlap within 120 seconds on average*[1], which is the example of MDL given by RFC 6864, and we consider this as an upper bound. In Figure 3.8, the first two y in case 1 illustrate a network duplicate. The IAT median of case 1 is greater than 120 seconds; the series is thus not analyzed.

Then, for every source `srcIP` host present in the traces, we select the series with the largest number of fragments, which we consider the worst case scenario. In Figure 3.8, it would correspond to case 2 if the series have the same `srcIP`.

Finally, we compute the hosts' MDL estimate as the minimum of the inter-arrival times (IATs) between the overlapping fragments of the hosts' selected series. *We consider that a host does not rate limit itself if it reuses the `IPID` within 5 seconds*. This time, considering that the series illustrated in Figure 3.8 are sent by different hosts, we thus consider that the host represented by the series in case 2 is not RFC compliant, while the host of case 3 is.

3.3.1.2 Analysis

We counted 175 and 569 hosts that reused the `IPID` within 120 seconds, for a total of 43,157 and 53,709 hosts, where the first number corresponds to the year range 2007-2013 and the second to 2014-2024. Overall, there is thus a relatively small chance (between 0.4% and 1.0%) that a host sends fragmented traffic that could be misassociated in the end. Furthermore, the proportion of hosts that generated fragments with constant `IPID` doubled, from 14% (24 hosts) to 29% (162 hosts). Since RFC 6864 lifted the `IPID` uniqueness constraint for atomic datagrams, some hosts may have incorrectly implemented it[2]. Some malicious hosts could also be the source of these constant `IPID` series.

Figure 3.9 plots an estimated MDL of the hosts that reused the `IPID` within 120

[1]Here, the terms "average" refers to the median.

[2]Salutari et al. [113] measured 39% constant `IPID` in 2017 and Feng et al. [100] measured 17% in 2022. We suppose they did not distinguish between atomic and non-atomic datagrams, though.

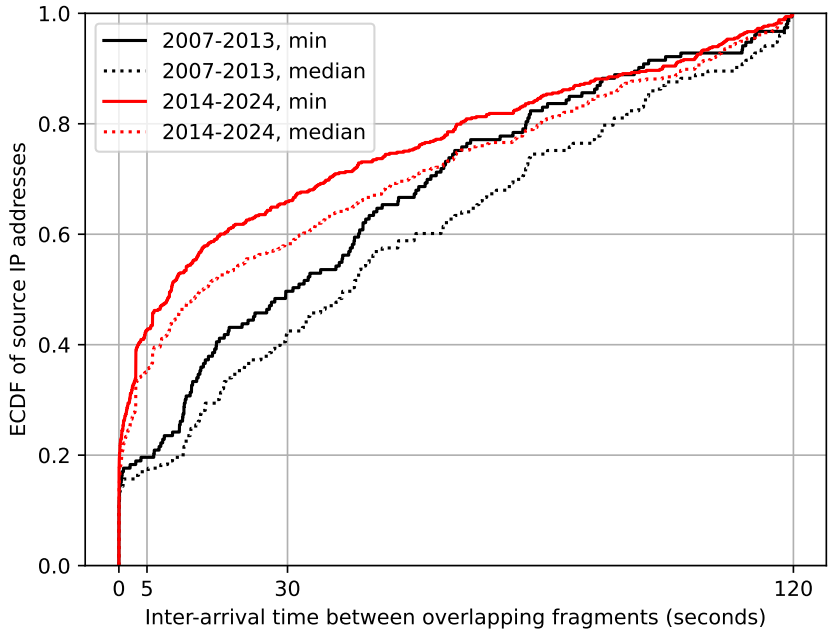


Figure 3.9: Source host MDL estimation.

seconds before and after RFC 6864's publication. IAT medians are also plotted to provide a simple representation of typical IAT values. Between 2007 and 2013, these hosts accounted for approximately 20% of all the hosts that reused the `IPID` within 120 seconds. This host proportion has doubled after the RFC release. This observation is consistent with the general trend of increasing host throughput, but does not comply with RFC 6864. We cannot see clear vertical steps (after the one around zero) that would indicate a shared behavior. Thus, if a maximum egress traffic rate is set, it likely means the hosts have actively set it. In other words, there is likely no hardcoded or default value in OSes' IPv4 stacks.

3.3.2 Header field analysis

This subsection evaluates if the *overlap* and *duplicate* series analyzed in the previous subsection, i.e., series with at least two overlapping fragments, are not generated by suspicious senders. To do so, we analyze some fragments' header fields in Table 3.2. We particularly measured whether any fragment in series has the `DF` bit set (because `DF` and `MF` bits must not be set at the same time) and a low `TTL` (because such fragments likely correspond to IDSes and middleboxes evasion attempts). Within the duplicate se-

Series type	Years	Total nb of...		Number with...							
				DF bit set		TTL < 10		MF=1 frag. with...			
								mult. sizes		size < 576	
		hosts	series	hosts	series	hosts	series	hosts	series	hosts	series
Duplicate (w/ incons. checksum)	07-24	702	4,607,914	14	**16,314**	6	1,397	23	142	8	711
	07-13	162	1,498,150	4	2,126	0	0	3	115	3	668
	14-24	547	3,109,764	13	**14,188**	6	1,397	20	27	5	43
Overlap	07-24	249	730,585	5	99	0	0	**44**	130	5	568
	07-13	44	456,759	2	4	0	0	5	44	4	532
	14-24	205	273,826	3	95	0	0	**39**	86	1	36

Table 3.2: Summary of overlap and duplicate series with suspicious header field values. There are 93,393,577 series.

ries from 2014 to 2024, a relatively high number of series have the `DF` bit set (14,188) and a `TTL` value under 10 (1,397); however, they are generated by only a few hosts. We also analyzed the size of fragments with $MF = 1$ bit set, i.e., the leftmost fragments. We observe that series' leftmost fragments can have up to seven different sizes, which is a clear sign of suspiciousness, as it is unlikely that a datagram is fragmented more than twice (to accommodate more than three distinct MTUs). They represent a few 142 duplicate and 130 overlap series, but are caused by 3% and 18% of the related series hosts(!). Finally, fragments with sizes lower than 576 should not be observed because all IPv4 implementations must be able to accept them [8]. These fragments have almost disappeared since 2014. To conclude, the majority of the series with overlapping fragments analyzed in Section 3.3.1 do not exhibit suspicious behavior and are, therefore, unlikely to be due to buggy or malicious hosts. However, a significant proportion of the hosts that generated the *overlap* series (where the fragment `offsets` do not perfectly align) are suspicious.

Section takeaways

The RFC 6864 prediction has come true: throughput increase has gone along with more possible fragment misassociations. In the analyzed traces prior to this RFC release in 2013, approximately 20% of the hosts that reused the `IPID` did so within 5 seconds. *This proportion has reached 40% after 2013*. Therefore, many hosts do not comply with the RFC countermeasure, since they have not rate-limited themselves when sending non-atomic datagrams or have used constant

IPID. Our fragment header field analysis did not reveal evidence of suspicious activity for the majority of hosts that reused the IPID.

3.4 Conclusion

In this chapter, we conducted an 18-year longitudinal study of IPv4 fragmentation in some MAWI dataset traces. For some years now, fragmentation has been considered "harmful" because of possible attacks (e.g., DoS or NIDS evasion, as studied in this thesis) or resource overhead [5], [6], [114]. Our analysis showed that the fragmented IPv4 traffic has not disappeared and accounted between 0.03% and 1.48% (bytes and packets) of the total traffic every year. As we have seen, there can be some episodes, such as the evolution of services (e.g., UDP/DNSSEC) or the introduction of new services (e.g., UDP/QUIC), that (more or less) temporarily require the senders' original IPv4 datagrams to be fragmented. Moreover, a non-negligible number of hosts (from 493 to 25,567 each year) sent IPv4 traffic that was fragmented at some point in our dataset. The NIDSes should, therefore, not ignore the IPv4 fragments when reconstructing the monitored host data streams.

We chose to study the fragment data overlaps through the misassociation issue prism described in RFC 6864. Indeed, a host that does not rate limit itself when sending non-atomic datagrams, i.e., datagrams that can be fragmented, meets the conditions for possible fragment misassociation and data overlap at the end host. We assessed the hosts' conformance with RFC 6864 and found that approximately 40% of the hosts do not comply with an MDL of less than 5 seconds after this RFC release. The problem, however, seems limited so far, as a relatively small number of hosts actually had to reuse IPID within 120 seconds.

To summarize, IPv4 fragmentation still benignly occurs on the Internet. Hence, NIDSes should not ignore or raise an alert whenever a monitored host receives IPv4 fragment, because it would generate too many false positive alerts. With the increasing host sending throughput, we observed that there are higher chances for fragment misassociation and overlaps to occur benignly at the end-host. Yet, there is a difference between the hosts sending full and partial overlaps. Approximately 18% of the hosts that sent (partial) overlap series appear suspicious after the analysis of some header fields. Hence, raising a NIDS alert when a partial fragment overlap occurs is possible,

but it may generate a non-negligible number of false positive alerts. The NIDSes should also carefully tune their fragment reassembly timeouts to faithfully represent those of the monitored hosts.

Chapter 4

IPv4, IPv6, and TCP stack reassembly policy analysis

Contents

4.1 Method 76
4.1.1 Overlap test case model 76
4.1.2 PYROLYSE tool 86
4.1.3 Comparison with related methods 89
4.2 Tested stacks 89
4.2.1 Network stacks 91
4.2.2 Midpoint stacks 93
4.3 Detailed NIDS insertion and evasion opportunities for $n = 2$ 93
4.3.1 OS reassembly policies 93
4.3.2 NIDS/OS reassembly consistency 96
4.4 Stack reassembly characterization for n $\leq$ 3 99
4.4.1 Reassembly diversity 99
4.4.2 Reassembly errors 111
4.4.3 Inconsistencies between $n = 2$ and $n = 3$ reassemblies 115
4.5 Discussion and limits 121
4.5.1 Reassembly compliance with RFCs 121
4.5.2 From theory to practice: test cases' usability in real-world attack scenarii 122
4.5.3 Reassembly policy configurability cost 125
4.5.4 Configurable NIDS challenges to address $n > 2$ overlapping chunks 127
4.6 Conclusion 128

This chapter characterizes the IPv4, IPv6, and TCP overlap-based reassembly policies of a wide range of network and midpoint stacks. First, it introduces the method we use to obtain such stack policies and the related tool we developed, PYROLYSE. Second, it details the network and midpoint stacks we tested. Third, it reports some NIDS

insertions and evasion opportunities based on $n = 2$ overlap fragment and segment sequences. Finally, it characterizes the reassemblies of more than 60 network and midpoint stacks for $n \leq 3$. Most of this chapter contributions were published in [33], [34]. The NIDS insertion and evasion attack opportunities as well as any stack reassembly error were disclosed to the concerned editors (see Appendix B).

NB this chapter describes the main thesis contributions. They include numerous interdependent concepts that could not be separated and imply a substantial chapter length.

4.1 Method

This section first depicts the theoretical contributions of this thesis regarding modeling overlap test cases. Second, it introduces the tool PYROLYSE that we developed and publicly released to test the overlap-related behavior of different IP and TCP implementation types. Third, it compares our work and related works regarding the overlap-addressed scope.

4.1.1 Overlap test case model

We provide four main contributions regarding the overlap test case model.

1. We ensure that the generated overlap test cases are exhaustive for any n chunk sequence, with the Allen's interval algebra (see Section 4.1.1.1). To the best of our knowledge, this is the first time this level of exhaustivity has been provided.
2. We introduce new testing scenarii that aims to reproduce diverse and real world contexts, e.g., contiguous extra chunks around the overlapping chunks (see Section 4.1.1.2).
3. IP and TCP testing require upper services that may carry a `checksum` header field. We present new sets of payload patterns that have no impact on the upper layer checksum, ensuring, possibly, any reassembled test case payload (see Section 4.1.1.3).
4. We propose a new reassembly policy formalism that can cope with the test case exhaustivity (see Section 4.1.1.4)

For the very specific $n = 2$ case, we also introduce two testing modes, which indicates if the overlaps are tested individually or altogether (see Section 4.1.1.5).

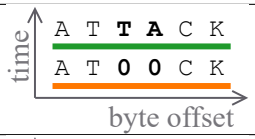
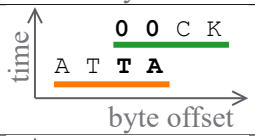
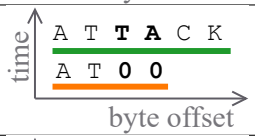
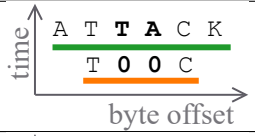
Descr.	Relation $\mathcal{R}$	Relation $\mathcal{R}$ inverse	Example for $\mathcal{R}$
Meet	X M Y	X Mi Y	
Before	X B Y	X Bi Y	
Equal	X Eq Y	-	
Overlap	X O Y	X Oi Y	
Start	X S Y	X Si Y	
During	X D Y	X Di Y	
Finish	X F Y	X Fi Y	

Table 4.1: Allen's interval algebra relations.

4.1.1.1 Test case chunk sequence

Spatio-temporal reasoning is particularly well suited to model packets of protocols that allow chunking and, thus, overlapping. Indeed, we can associate byte offset with one spatial dimension and arrival time with one temporal dimension. Allen's interval algebra [10] is such a spatio-temporal algebra. Thirteen Allen relations describe all the possible spatio-temporal relationships between two chunks. The relations are described in Table 4.1 and they are used to generate the test cases.

Definition 8 — Test case. Since an Allen 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.

Four relations are non-overlapping (i.e., M, Mi, B, and Bi); thus, some test cases show no overlap at all. Furthermore, not all the possible $\binom{n}{2}$ relation combinations make sense. For example, the (B, M, B) test case is impossible. We say that these test cases

are incoherent, and some tools (such as SparQ [115]) can help us filtering them.

- $\boldsymbol{n = 2}$: $\binom{2}{2} = 1$ Allen's relation describes the link between two chunks, and the 13 relations are coherent test cases for $n = 2$ chunks.

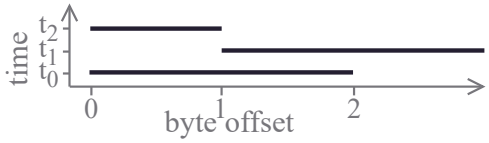


Figure 4.1: The (O, Si, Mi) triplet of chunks.

- $\boldsymbol{n = 3}$: Since $\binom{3}{2} = 3$, three Allen relations describe all the $n = 3$ chunk test cases. We represent the triplet of chunks from Figure 4.1 with the notation $(p_{01}, p_{02}, p_{12}) = (O, Si, Mi)$, where indexes refer to the chunk pairs' time position. We find 409 coherent and unique triplet test cases.

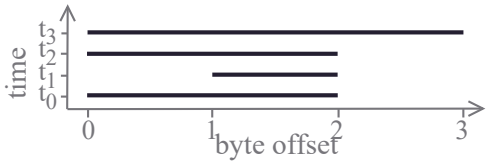


Figure 4.2: The (Fi, Eq, S, F, D, S) quadruplet of chunks.

- $\boldsymbol{n > 3}$: As already mentioned, we will focus on $n \leq 3$ in the following 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 4.4.3) for most implementations, we don't need to test $n > 3$. However, the same principle can be used to generate any n-related coherent test cases. For instance, the quadruplet of chunks from Figure 4.2 is encoded with $\binom{4}{2} = 6$ relations as $(p_{01}, p_{02}, p_{03}, p_{12}, p_{13}, p_{23}) = (Fi, Eq, S, F, D, S)$.

4.1.1.2 Testing scenarii

As mentioned in Section 1.4, some related works [61], [62] added an extra segment before (bytewise) the test case chunks and sent it after (timewise) them, to delay the targeted implementation reassembly. Therefore, the testing scenario, i.e., the context surrounding the original test case chunks described with the $\binom{n}{2}$ Allen relations, seems

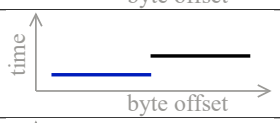
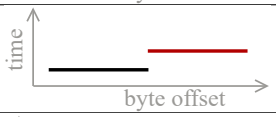
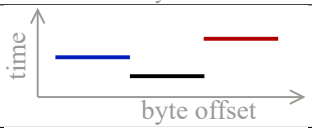
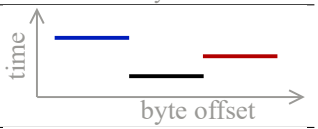
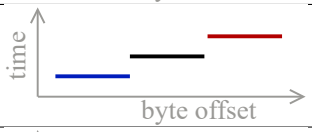
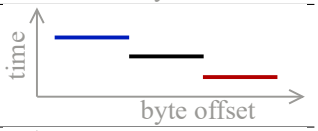
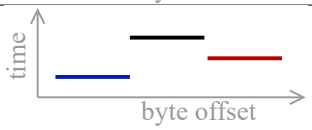
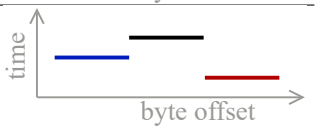
Name/description	Illustration	Name/description	Illustration
s^{c}: continuous		-	-
s^{sp}: Start precedes		s^{ep}: End precedes	
s^{sf}: Start follows		s^{ef}: End follows	
$s^{sf,ef}$: Start follows and End follows		$s^{ef,sf}$: End follows and Start follows	
$s^{sp,ef}$: Start precedes and End follows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

Table 4.2: 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.

to impact the test case reassembly and, ultimately, the obtained implementation reassembly policy. We thus introduce scenarii to capture as much of the implementation policies' complexity regarding context as possible. We differentiate two context types: one that is *protocol-agnostic* and another that is *protocol-dependent*.

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 4.2 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 is 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.

Protocol-dependent context Protocol semantics includes 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 [8], [47]. 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, s_{of} means that the oldest rightmost finishing fragment has the `MF` bit unset.

Association between protocol-agnostic and protocol-dependent parts Test scenarii comprise a protocol-agnostic part and an optional protocol-dependent part. We thus index test scenario names with the protocol-dependent part when necessary and possible: e.g., s^c_{of} 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[1], the TCP scenarii correspond to the one described in Table 4.2. In total, there are 42 IP and 11 TCP scenarii, representing 10,362 and 4,642 test cases, respectively. Note that some IP test cases are the same in some IP-dependent scenarii

[1]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.

(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).

4.1.1.3 Chunk payload population with checksum-impactless patterns

IP and TCP test cases are composed of overlapping and non-overlapping chunks, which must carry some data so as to complete the testing. An implementation can reassemble a given test case in several ways, and the reassembled payload lengths may vary. The reconstructed lengths may vary for IP because of the scenarii with a protocol-dependent part we have just introduced. For TCP, this can occur if an implementation ignores overlapping segments. The patterns that Novak introduced in [60] (and that others used [76], [77]) do not permit varying reassembled payload lengths when testing IPv4 or IPv6 with ICMP or ICMPv6. This section aims to fill this gap.

Upper service requirements IP and TCP testing requires upper services in an opaque-box testing context so as to retrieve the overlap chunk reassemblies of the targeted implementation. These upper services may require a header, which may contain some `checksum` and `length`-related fields. Since the reassembled payload possibilities for a given test case may vary in length, *an upper service containing a header field related to the reassembled payload length cannot be used*. This is, for example, the case of UDP; thus, it cannot be used inside the fragments for IPv4 and IPv6 testing. The `checksum` values are computed differently depending on the protocol. Checksums are basically computed with the 2-byte one's complement of the one's complement of some header(s) and data. We categorize two `checksum` types.

- The checksums that do make use of an IP pseudo-header (e.g., ICMPv6). Such pseudo-header are composed of `source` and `destination` IP addresses, the `protocol` (e.g, ICMPv6), and the protocol `length` (comprising, the protocol header and the data). The protocol `length` is important in our case, since the reassemble payload lengths may vary for a given test case. A target implementation *dynamically* computes the `length` field of the pseudo-header; therefore, there is room for checksum manipulation so that the varying reassembled payloads fit the final payload length. Note that this dynamic length computation is a key difference from the upper services mentioned above, which carry a *fixed* length value in their header.
- The checksums that do not make use of an IP pseudo-header (e.g., ICMP).

Consequently, the patterns to use are different depending on the IP or TCP upper layer `checksum` type, if there is any. If there is no `checksum` field, there is no constraint on the patterns to use. Note that IPv4 and TCP carry `checksum` header fields which are chunk-dependent; thus, there is no ambiguity regarding their values.

Pattern requirements First, the overlapping data portions must be different to distinguish which data chunk is preferred. Second, the patterns must be 8-byte-long since this corresponds to the IP `fragment offset` unit. Third, the IP and TCP upper-layer `checksum` must be correct no matter the final test case reassembly; that is to say, no matter the chosen overlapping portion pattern and no matter the reassembled payload length, as mentioned in the previous part. As a result, we need *unique 8-byte-long patterns that do no impact on the IP and TCP upper-layer* `checksum`.

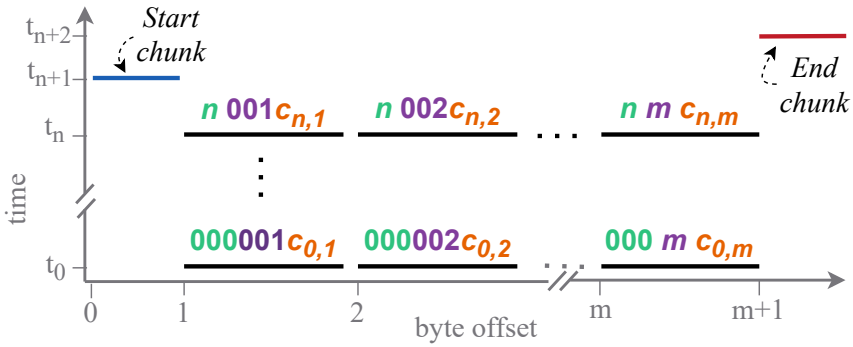


Figure 4.3: The checksum-impactless patterns, in a $s^{sf,ef}$ testing scenario. The first three bytes (green parts) correspond to the chunk ID. The following three bytes (purple parts) correspond to the pattern offset. The last two bytes (orange parts) are the checksum correction bytes.

Design Figure 4.3 illustrates the set of chunk patterns. The first six bytes of a pattern correspond to its unique ID while the last two bytes are the 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*[2].

- For the upper layers using checksums which do not employ an IP pseudo-header, it practically means that every pattern's one complement is equal to 0.

[2]Some works outlined this checksum trick and used it in fragment-based attacks with UDP and TCP as upper-layers [57], [100], [116].

- For the upper layers using checksums that do make use of an IP pseudo-header, it means that every pattern's one complement is equal to −8 in order to counterbalance the impact of the pattern (i.e., +8) in the pseudo-header `length` and, therefore, in the `checksum`.
- For the upper layers without a checksum field, it means that the last two bytes can be random.

For a given pattern ID, the difference between the sets of patterns thus lies in the last two correction bytes.

4.1.1.4 Time policies as a reassembly policy formalism

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

Table 4.3: Description of the time policies used to describe implementation reassembly. n is the overlapping chunk number.

Description We choose to describe test case reassemblies regarding the temporal aspect of the preferred chunk data. Table 4.3 details every possible time policy for $n \leq 3$. For $n = 2$, there can only be one overlapping data portion between the two chunks; however, for $n = 3$, we differentiate two types of overlapping portions. A triple overlap means that the three chunks overlap on a same data portion. A residual pair overlap is an overlap that is exclusive to a chunk pair. The set TP_{pairs} encodes the $n = 2$ pair time policies, and the set $TP_{triplets}$, that is equal to $TP_{t,t} \times TP^3_{rp,t}$, encodes the overall $n = 3$ triplet time policies.

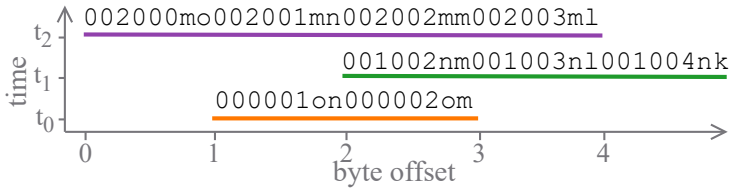


Figure 4.4: The triplet (O, D, Oi) representation populated with the IPv4 checksum-impactless patterns.

Figure 4.4 illustrates the (O, D, Oi) triplet test case. If an OS reassembles with

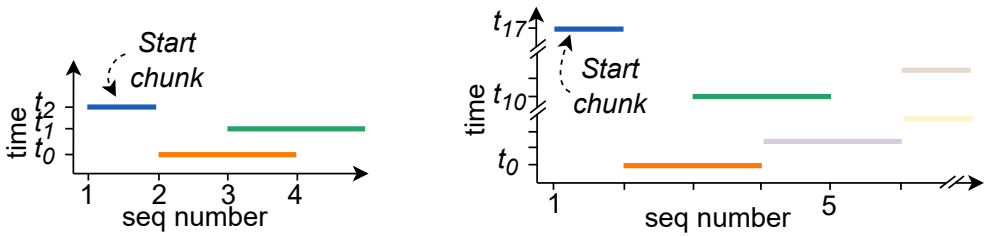
002000mo 000001on 000002om 001003nl 001004nk payload, the time policy is thus: tp_t = `old` and ($tp_{rp_{01},t}$, $tp_{rp_{02},t}$, $tp_{rp_{12},t}$) = (`none`, `old`, `old`). $tp_{rp_{01},t}$ is `none` since the entire overlap of the orange and green chunks does not overflow the boundaries of the triple overlap.

Remarks In this thesis, we only address up $n = 3$ test cases. As a result, we do not model $n > 3$ reassemblies. But, the same logic can be applied. For $n = 4$, it would require at most $\binom{4}{2} = 6$ residual pairs, $\binom{4}{3} = 4$ residual triplet, and $\binom{4}{4} = 1$ quadruple time policy fields.

Furthermore, the introduced time policy format can capture and describe fully the implementation reassembly behaviors for $n \leq 3$. However, one cannot easily abstract or deduce stack general behaviors with this time policy format. Natural language, for example, has the advantage of encompassing and explaining the policies with only a few words, e.g., "Linux favors an original fragment with an offset that is less than a subsequent fragment" is how Novak described Linux IPv4 policy in [60]. Besides this shortcoming, our time policy formalism can be used in several ways:

- by performing a hashmap lookup when processing a tested overlap case.
- as an intermediate reassembly description format to further abstract the stack policy with, for example, some natural language sentences.

4.1.1.5 Testing modes



(a) Single (b) Multiple

Figure 4.5: Testing mode illustration for the (O) test case and the s^{sf} TCP scenario.

For the very specific case of $n = 2$ testing, overlaps can be tested *single* within nine separate chunk sequences as [1], [62] did ("individual overlap tests" in [62]). Or, differently, *multiple* overlaps can be tested altogether within a unique chunk sequence, resulting in one reassembly (Novak and Sturges named it "model overlap tests" in [62]).

Figure 4.5 illustrates the two modes for the (O) test case. Novak and Sturges [62] observed different reassemblies for $n = 2$ when testing *single* and *multiple* modes for TCP, and this last mode has been the most tested in the related works that targeted the OSes [60]–[62], [76]. In the following, we will specifically use the Novak *multiple* chunk sequence, which is illustrated in Figures 1.9 and 1.10, when testing this mode.

Above $n = 2$, only the *single* mode makes sense; therefore, only this mode is tested for $n = 3$.

4.1.2 PYROLYSE tool

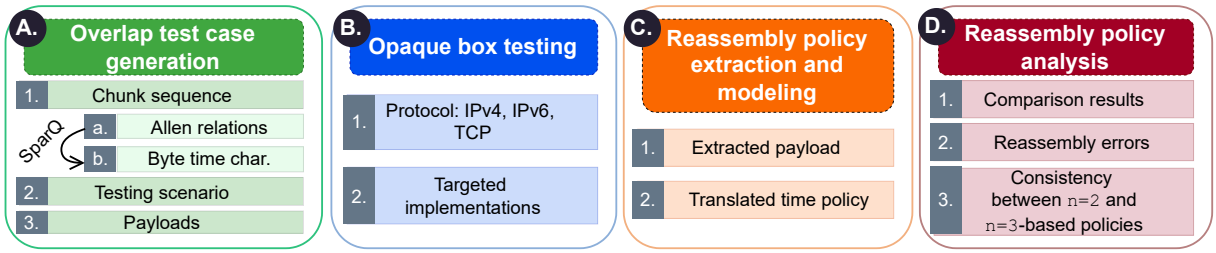


Figure 4.6: PYROLYSE testing pipeline.

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 4.6 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 the time policy 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.

Note regarding testing modes We designed PYROLYSE to test implementations with the *single* mode, which permits exhaustive n testing as well as reassembly policies' expression and analysis with the introduced time policy format. However, one can design their own custom chunk sequences and test the stack reassembly. Here, PYROLYSE steps from A.1.b to C.1 can be used. We specifically did that when testing the *multiple* mode with the Novak model for $n = 2$.

4.1.2.1 Overlap test case generation

The test case generation step starts with the chunk sequence generation. As documented in Section 4.1.1, the chunk sequence exhaustiveness is ensured with a spatio temporal reasoning, i.e., Allen's interval algebra [10]. This algebra is composed of 13 relations, as documented in Table 4.1. PYROLYSE generates all the combinations of $\binom{n}{2}$ relations. Using SparQ [115], 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 ensure the upper-layer checksum correctness.

4.1.2.2 Opaque box testing

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

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 [117] 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 a RST packet. The tool acknowledges any echoed data that the targeted implementation sends. Finally, tcpdump [118] captures IP and TCP host communications.

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

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 all be tested similarly; we thus use three different setups.

i. The OSes are Vagrant/Virtualbox boxes publicly available from the Vagrant Cloud [119]. 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.

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

4.1.2.3 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, as introduced in Section 4.1.1.4.

4.1.2.4 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 4.1.1.3 testing payload patterns specifically 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 4.4.3) to determine whether implementation reassembly policies are generalizable to any n.

4.1.3 Comparison with related methods

Table 4.4 compares the related works analyzed in Section 1.4 with this chapter overlap modeling contributions. The table uses the concept and terminology introduced in the two previous subsections to ease the comparison.

We consider that a work is exhaustive at testing the overlaps if the nine (resp. 409) overlap chunk sequences were tested for $n = 2$ (resp. $n = 3$), as reported in Section 4.1.2.1. Overall, only three of the 15 analyzed works were exhaustive. In respect to testing scenarii, they tested at most two of them, while we explore 42 IP and 11 TCP scenarii. Hence, our method can capture more real-world contexts around the overlap test cases. Almost all the works shared their code; however, only few of them can be easily extended to test more n overlapping chunks. Furthermore, the related works described the addressed overlap test cases and obtained reassembly policies through, for instance, figures, natural language, or number of answered overlap test cases. These formalisms were sufficient up to $n = 2$ chunk sequences, but they are not for $n \geq 3$ chunk sequences. The Allen relation sequences and the pair/triplet time policies we previously described fill this gap.

4.2 Tested stacks

This section lists the protocol implementations that we will analyze in the following. We test the latest stable version for all the targeted protocol implementations during testing. In addition, we examine a range of versions from 10 years to the latest for

Author	Work	Year	Protocol	n	Test case exhaus.	Tested Allen relations	Scenario	Mode	Tool characteristics: available	Tool characteristics: target ext.	Tool characteristics: n ext.	Reassembly policy format	Target stack: midpoint	Target stack: network
Ptaceck and Newsham	[1]	1998	IPv4/ TCP	2	✗	Fi, D	s^c	*single*	✓ [120]	✗	✗	natural language	NIDS	OS
Shankar and Paxson	[61]	2003	IPv4	2	✗	O, Oi, Eq	s^c	*multiple*	✗	✗	✗	natural language		OS/router/ printer
			TCP	2	✗	O, D	s^{sf}	*single*						
Novak and Sturges	[60]	2005	IPv4	2	✓	all	s^c	*single*	*	✗	✗	natural language		OS
	[62]	2007	TCP	2	✓	all	s^{sf}	*single/ multiple*						
Atlasis	[56]	2012	IPv6	3	✗	all	s^c_{nf} s^c_{mnf}	*single*	*	✗	✗	figures		OS
Khattak et al.	[68]	2013	IPv4/ TCP	2	✓	all	?	*single*	✗	✗	✗	natural language	CS	
Wang et al.	[70]	2017	IPv4	2	✗	Eq	s^{sf}_{af}	*single*	✓ [121]	✓	✗	natural language	CS	
			TCP	2	✗	Eq	s^{sf} s^c	*single*						
Bock et al.	[72]	2019	IPv4/ TCP	-	✗(1)	Unknown	-	-	✓ [122]	✓	-	Geneva's genetic building blocks + natural language	CS	
	[73]	2020												
	[74]	2021												
Wang et al.	[59]	2020	TCP	2/3	✗(1)	F, D, Oi	-	-	✓ [123]	✓(2)	✓	natural language	CS/ NIDS	OS(2)
	[75]	2021	TCP	3	✗(1)	-	-	-	✓ [124]	✓(2)	✓	natural language		
Di Paolo et al.	[76]	2023	IPv6	2	✗	O, Oi, Eq	s^c_{af}	*multiple*	✓ [125]	✓	✗	nb. of target responses		OS
Lin et al.	[64]	2024	IPv6	2	✗	Eq	s^c	*single*	✓ [126]	✓	✗	compliance with RFC 5722 [46]	NIDS/ firewall	OS
Oprea	[77]	2025	IPv6	2	✗	O, Oi, Eq	s^c_{af}	*multiple*	✓ [127]	✓	✗	nb. of exploitable overlap. sequences	NIDS	OS
Us	-	-	**IPv4/ IPv6**	**2/3**	✓	**all**	S^{IP}	***single/ multiple***	✓ [11]	✓	✓	**Allen relation sequence + pair/triplet time policies**	**NIDS**	**OS/IoT/ unikernel/NIC/ DPDK-comp.**
			TCP	**2/3**	✓	**all**	S^{TCP}							

Table 4.4: Summary regarding overlap-based works. S^{IP} (resp. S^{TCP}) corresponds to the 42 IP (resp. 11 TCP) scenarii introduced in Section 4.1.1.2. * means that the source code is present inside the paper. (1) 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. (2) 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.

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 Sections 4.3 and 4.4. Overall, we tested 61 protocol implementations that comprise network (general purpose OSes, embedded/IoT OSes, unikernel or DPDK-based applications, and NICs) and midpoint stacks (NIDSes). The test of various network stack types aims to verify whether the implementations re-implement their own TCP/IP stacks. Since the NIDSes can be deployed in very diverse networks, they can monitor various stack types. Table 16 provides more details for every implementation.

4.2.1 Network stacks

This subsection briefly describes every network stack type and lists all the tested stacks.

General purpose OSes General purpose OSes are generalist OSes that are usually installed on laptops, such as Windows, Linux-based, FreeBSD.

OS			Addressed protocols		
Name	**Version**	**Additional version info**	**IPv4**	**IPv6**	**TCP**
Windows	Server 2019/10/11	<2022/10/25 build	✓	✓	✓
		≥2022/10/25 build			
Debian	8	Linux 3.16.0-9	✓	✓	✓
	9	Linux 4.9.0-12			
	12	Linux 6.1.0-37			
FreeBSD	10.2	RELEASE-p8	✓	✓	✓
	11.2	RELEASE-p10			
	11.3	RELEASE-p9			
	12.1	RELEASE-p10			
	12.2	RELEASE-p6			
	14.1	RELEASE			
OpenBSD	6.0	GENERIC.MP#2319	✓	✓	✓
	6.1	GENERIC.MP#8			
	7.7	GENERIC.MP#625			
NetBSD	7.0	GENERIC.201509250726Z		✓	✓
	8.0	GENERIC#0			
	9.3	GENERIC#0			
Solaris	11.2 → 11.4	SunOS 5.11	✓	✓	✓

Table 4.5: Description of the tested general purpose OSes.

For NetBSD, the IPv4 test cases with fragment sizes lower than 40 bytes do not

receive any answer. We hypothesize that the patterns used are not well-suited to test the NetBSD IPv4 policy. We thus do not provide it.

IoT/embedded stacks Devices with computation and communication capabilities have massively spread in recent years. Internet of Things (IoT) devices are particularly concerned. Equipped with sensors or actuators, they help in varying areas, such as agriculture, healthcare, transportation, industry, or household. Not every device firmware reimplements all the OS components; instead, proprietary firmwares often reuse lightweight and open-source TCP/IP stacks. We test four of these light OSes:

- lwIP version 2.2.1 for IPv4, IPv6, and TCP.
- picoTCP version 1.7.0 for IPv4, IPv6, and TCP.
- uIP version 1.0 for IPv6 and TCP.
- smoltcp version 0.12.0 for IPv4, IPv6, and TCP.

Unikernels Unikernels run directly on the hardware or the hypervisor. One instance is highly specialized and dedicated to one application. Only the kernel components that the application requires are compiled; thus, the generated instances are very light. In this chapter, we test mirage-tcpip version 9.0.0 and provide its IPv4, IPv6, and TCP policies.

DPDK-compatible frameworks DPDK, short for Data Plane Development Kit, provides a set of libraries to process network packets in the user space instead of the kernel space. The goal is to enhance packet processing performance. Some frameworks use DPDK libraries to provide high-speed packet processing applications. This is, for instance, the case of Seastar. We test its IPv4, IPv6, and TCP stacks for the version 22.11.0.

NICs A Network Interface Controller (NIC) is a hardware component that permits the reception and transmission of network packets at the OSI layers 1 or 2. It connects a machine to a network. NICs often have offloading functionalities, which enables the host OS to offload part of the traffic reassembly to increase packet processing performances. Therefore, if enabled, some NIC offloads can alter and disambiguate the overlapping packets reaching the host OS or NIDS. We test the TCP offloading of Chelsio T520-CR (driver version 9.0.1 and firmware version 6.5.1.1023) and Xilinx SFN8522PLUS-Onload (driver version 3.19.0.3 and firmware version 1.27.5.0 TP 0.1.4.9) cards.

4.2.2 Midpoint stacks

We test the IPv4, IPv6, and TCP reassemblies of three widely used open-source NIDSes:

- Suricata versions 7.0.2 to 7.0.9.
- Snort versions 3.1.58.0 and 3.7.1.0.
- Zeek versions 6.1.0 and 7.1.1.

4.3 Detailed NIDS insertion and evasion opportunities for $n = 2$

As reported in Table 4.4, related works obtained the reassembly policies with some s^c-related IP and s^{sf} TCP scenarii. The tested modes for $n = 2$ were *single* and/or *multiple* depending on the work. The configurable NIDSes Suricata and Snort based their IP and TCP implemented policies on [60], [62] works, and applied the $n = 2$ for any n overlap chunk sequence. Before diving into $n = 3$, all the testing scenarii and more diverse network stack types we introduced in the previous sections, we take a look at the latest version general purpose OS[3] and NIDS reassembly policies with the testing parameters used in related works. First, this sections details OS reassembly policies. Then, it qualifies and quantifies attack opportunities on Snort, Suricata, and Zeek NIDSes, configured with the appropriate policies if applicable[4].

4.3.1 OS reassembly policies

This subsection details IPv4, IPv6, and TCP policies for recent OSes with *multiple* and *single* testing modes. When relevant, we also compare our findings with the latest related work: IPv4 *multiple* mode testing from [60] findings, IPv6 *multiple* mode from [76], and TCP *multiple* and *single* modes from [62].

4.3.1.1 IP protocols

Table 4.6 reports OS IPv4 and IPv6 test case reassembly policies.

[3]Except NetBSD because we could not test its IPv4 policy, as mentioned in Section 4.2. Furthermore, the latest tested FreeBSD and OpenBSD versions reassemble the same way the overlap test cases; therefore, we only report FreeBSD policies in this section.

[4]To recall, we consider that hosts with more recent OS versions than the ones tested by Novak and Sturges in [60], [62] (e.g., Debian 12) should have the latest OS family representative reassembly policy (e.g., *linux* because Linux 2.4 was the latest tested version) associated in the NIDS configuration file.

OS family	Kernel/build info	Protocol version	Testing mode	Test case: Overlapping relation								
				F	*Fi*	*S*	*Si*	*O*	*Oi*	*D*	*Di*	*Eq*
Windows	≥2022/10/25 build	v4	*multiple*	**Ø**	∅	**Ø**	**Ø**	∅	∅	**Ø**	**Ø**	**Ø**
			single	**n**	∅	**n**	**o**	∅	∅	**n**	**o**	**n**
		v6	*multiple*	**Ø**	∅	**Ø**	**Ø**	∅	∅	**Ø**	**Ø**	**Ø**
			single	**n**	∅	**n**	**o**	∅	∅	**n**	**o**	**n**
Debian	Linux 6.1.0-37	v4	*multiple*	**Ø**	∅	**Ø**	**Ø**	∅	∅	**Ø**	**Ø**	**Ø**
			single	**n**	∅	**n**	**o**	∅	∅	**n**	**o**	**n**
		v6	*multiple*	**Ø**	∅	**Ø**	**Ø**	∅	∅	**Ø**	**Ø**	**Ø**
			single	**n**	∅	**n**	**o**	∅	∅	**n**	**o**	**n**
Solaris	SunOS 5.11	v4	*multiple*	n	**o**	**o**	o	o	o	n	o	**o**
			single	n	**Ø**	**n**	o	o	o	n	o	**n**
		v6	*multiple*	n	**o**	**o**	o	o	o	n	o	**o**
			single	n	**Ø**	**n**	o	o	o	n	o	**n**
FreeBSD	14.1 (RELEASE)	v4	*multiple*	n	**o**	**o**	o	o	n	n	o	**o**
			single	n	**Ø**	**n**	o	o	n	n	o	**n**
		v6	*multiple*	**Ø**	∅	**Ø**	**Ø**	∅	∅	**Ø**	**Ø**	**Ø**
			single	**n**	∅	**n**	**o**	∅	∅	**n**	**o**	**n**

Table 4.6: OS IP reassembly policies. o (resp. n) means that oldest (resp. newest) chunk data is prefered and ∅ means that the OS ignores the overlap. Bold blue means that testing modes are reassembled differently. IPv4 (resp. IPv6) cell backgrounds encodes in consistency with [60] (resp. [76]).

IPv4 Windows and Linux policies have evolved for the *multiple* testing mode since Novak [60]. Both OSes now ignore overlap chunks. The other OSes have not changed their policies. However, the newly tested mode, namely *single*, shows different reassemblies for all the OSes when compared to the *multiple* mode. In total, six out of nine overlapping relations are reassembled differently for Windows and Linux-based OSes, while it accounts for three out of nine for the remaining OSes. We hypothesize that the context introduced by the adjacent chunks used inside *multiple* mode causes the discrepancies between the two testing modes. We thus argue that the *single* mode reassemblies should be used to obtain context-agnostic reassemblies inside the NIDSes. In this testing mode, all the latest OSes reassemble F, S, Si, D, Di, and Eq relations the same way, never ignoring the test cases. Fi relation is never reassembled, possibly due to the fragment with the *MF* bit unset being dropped. Finally, O and Oi are the only overlap test cases that show different reassemblies depending on the OS.

IPv6 FreeBSD does not reassemble O and Oi Allen relations, which differs from the observed IPv4 behavior. Except for FreeBSD, all OSes reassemble IPv4 and IPv6 overlapping fragments the same way, and thus, the previous paragraph descriptions also apply to IPv6 fragments. The Windows and Linux-based OSes ignore all the overlapping relations in a *multiple* test mode, which is inconsistent with Di Paolo et al.'s [76] findings for O, Oi, and Eq relations. Since the OS versions that Di Paolo et al. tested are very close to ours, we hypothesize that their lack of tested relation set exhaustiveness for that mode impacts their extracted reassembly.

4.3.1.2 TCP

Table 4.7 describes OS TCP reassembly policies and compares findings with state-of-the-art ones [62]. Windows and FreeBSD reassemble similarly the overlaps across testing modes. The former OS always reassembles with the oldest segment data. These policies are consistent with [62] description. The latest Linux-based OSes reassemble S relation differently depending on the testing mode, favoring old (resp. new) data for *multiple* (resp. *single*) mode. The SunOS also reassembles the Eq overlap differently, which is consistent with [62] findings.

OS family	Kernel/build info	Test case									
		Testing mode	Overlapping relation								
			F	*Fi*	*S*	*Si*	*O*	*Oi*	*D*	*Di*	*Eq*
Windows	≥2022/10/25 build	*any*	o	o	o	o	o	o	o	o	o
Debian	Linux 6.1.0-37	*multiple*	n	o	**o**	o	o	n	n	o	o
		single	n	o	**n**	o	o	n	n	o	o
Solaris	SunOS 5.11	*multiple*	n	o	n	o	n	o	n	o	**n**
		single	n	o	n	o	n	o	n	o	**o**
FreeBSD	14.1 (RELEASE)	*any*	n	o	o	o	o	n	n	o	o

Table 4.7: OS TCP reassembly policies. o (resp. n) means that oldest (resp. newest) chunk data is prefered. Bold blue means that testing modes are reassembled differently. Cell backgrounds encodes in consistency with [62].

Subsection takeaways

IPv4, IPv6, and TCP $n = 2$ reassembly policies are more complex than described in the state-of-the-art as overlap reassemblies change depending on the test mode. IP policies have evolved; for example, the Windows and Linux families now show the same reassemblies for the s^{c}_{nf} scenario. TCP reassembly policies for s^{sf} have been unchanged since 2007, except Linux's. Furthermore, OSes continue to reassemble some overlap test cases differently by favoring old or new data or ignoring the chunks.

4.3.2 NIDS/OS reassembly consistency

This section compares the NIDS reassembly policies we observed with the ones of OSes (see Section 4.3.1) for IPv4, IPv6, and TCP protocols and the *single* mode. To illustrate, we first check reassembly discrepancies between NIDSes and one OS family, namely FreeBSD. We choose this OS because it offers the most insertion and evasion attack[5] opportunities when configured inside Snort and Suricata using the *bsd* policy. We finish by summarizing results for all the tested OSes and providing metrics on NIDS attack opportunities. Full results can be found in [12].

4.3.2.1 NIDS consistency with FreeBSD version 14.1

Table 4.8 gathers IP and TCP NIDS reassembly policies in the *single* mode, which is the easiest an attacker can exploit.

IP NIDSes are all vulnerable to overlap-based attacks with at least two overlap types, except for Snort with IPv4 chunks. Zeek and Suricata prior version 7.0.5 can be subject to IPv4 insertion attack with Fi relation and IPv4 evasion or insertion with just Oi. Moreover, with only two consistent IPv6 test case reassemblies with the FreeBSD, many other relations can be used to perform insertion or evasion attacks on Zeek. Despite being based on the same *bsd* policy description as [60], Suricata reassembled IPv4 overlaps differently until version 7.0.4. Snort is, interestingly, perfectly consistent with the considered OS for IPv4 protocol even though no previous work had described

[5]Based on Section 2.1.1 definitions, if the NIDS and the host reassemble differently a test case by not ignoring it (i.e., E2 and I2 from Table 2.1), the attacker can either perform an insertion or an evasion. The other attack cases E1 and I1 are straightforward.

Protocol	Implementation		Overlapping relation							
	Name	Tested versions	*F*	*Fi*	*S*	*Si*	*O*	*Oi*	*D*	*Di*	*Eq*
IPv4	**FreeBSD**	**14.1**	**n**	∅	**n**	**o**	**o**	**n**	**n**	**o**	**n**
	Suricata-*bsd*	7.0.2 → 7.0.4	n	o	n	o	o	o	n	o	n
	Suricata-*bsd**	7.0.5 → 7.0.9	n	∅	n	o	o	n	n	o	n
	Snort-*bsd*	3.1.58.0, 3.7.1.0	n	∅	n	o	o	n	n	o	n
	Zeek	6.1.0, 7.1.1	n	o	n	o	o	o	n	o	n
IPv6	**FreeBSD**	**14.1**	**n**	∅	**n**	**o**	∅	∅	**n**	**o**	**n**
	Suricata-*bsd*	7.0.2 → 7.0.4	n	o	n	o	o	o	n	o	n
	Suricata-*bsd**	7.0.5 → 7.0.9	n	∅	n	o	o	n	n	o	n
	Snort-*bsd*	3.1.58.0, 3.7.1.0	n	∅	n	o	o	n	n	o	n
	Zeek	6.1.0, 7.1.1	o	o	o	o	o	o	o	o	o
TCP	**FreeBSD**	**14.1**	**n**	**o**	**o**	**o**	**o**	**n**	**n**	**o**	**o**
	Suricata-*bsd*	7.0.2 → 7.0.9	n	o	o	o	o	n	n	o	o
	Snort-*bsd*	3.1.58.0, 3.7.1.0	n	o	o	o	o	n	n	o	o
	Zeek	6.1.0, 7.1.1	o	o	o	o	o	o	o	o	o

Table 4.8: NIDS reassembly consistency with FreeBSD 14.1 (RELEASE) in a *single* testing mode. o (resp. n) means that oldest (resp. newest) chunk data is prefered and ∅ means the OS ignores the test case. Green (resp. red) means that NIDS reassembly is the same as (resp. different from) FreeBSD. * It corresponds to Suricata reassemblies after the fix related to [13].

reassemblies for the *single* test mode. Additionally, IPv4 and IPv6 fragments are reassembled similarly by Snort and Suricata. Zeek notably reassembles all the overlaps by favoring the oldest IPv6 fragment data. Suricata has fixed the IPv4 misassemblies with FreeBSD from version 7.0.5.

TCP Snort and Suricata policies are consistent with the tested FreeBSD for all the overlapping test cases. The NIDSes, thus, consistently implemented the reassembly policies that Novak and Sturges described [62]. Zeek, which does not have such a reassembly policy configuration capability, reassembles F, Oi, and D inconsistently, as it always reassembles with the oldest segment's data. An attacker can use these overlaps to perform insertion and evasion.

4.3.2.2 Consistency with all OSes

Snort reassembles perfectly consistently IPv4 test cases with the BSD and Sun-based OSes and IPv6 test cases with SunOS. We can find at least one IP test case that is

reassembled differently for the remaining OSes, i.e., the Linux and Windows ones. Neither Suricata until version 7.0.4 nor Zeek consistently reassembles all IP overlapping test cases with any of the characterized OSes. Therefore, at least an insertion or an evasion attack can target these NIDS-OS couples. Suppose that Snort or Suricata TCP reassembly policy is correctly associated with the host; there is no possible overlap-based attack except in one case: TCP *solaris* policies and SunOS-based OSes with the Eq test case. On the contrary, Zeek reassembles segments consistently with Windows OSes but does not with the remaining OSes. See [12] for more details.

Protocol	NIDS		Test case inconsistencies		Tested OS kernels with possible attack	
	Name	Tested version			Evasion	Insertion
IPv4	Suricata	7.0.2 → 7.0.4	8	22%	F	W, L, S, F
		7.0.5 → 7.0.9	6	17%		W, L, S
	Snort	3.1.58.0, 3.7.1.0	4	11%		W, L
	Zeek	6.1.0, 7.1.1	9	25%	F	W, L, S, F
IPv6	Suricata	7.0.2 → 7.0.9	9	25%		W, L, S, F
		7.0.5 → 7.0.9	8	22%		W, L, S, F
	Snort	3.1.58.0, 3.7.1.0	6	17%		W, L, F
	Zeek	6.1.0, 7.1.1	28	78%	W, L, S, F	W, L, S, F
TCP	Suricata	7.0.2 → 7.0.9	1	3%	S	S
	Snort	3.1.58.0, 3.7.1.0	1	3%	S	S
	Zeek	6.1.0, 7.1.1	11	31%	L, S, F	L, S, F

Table 4.9: NIDS inconsistencies with OS reassemblies and corresponding attack opportunities for a *single* testing mode. W, L, S, and F respectively correspond to the tested Windows, Linux, SunOS and, FreeBSD/OpenBSD kernels.

Table 4.9 gives more general consistency metrics and related attack opportunities. In particular, Zeek and Suricata reassemble IPv4 overlaps inconsistently for about 20% of them. Snort performs better with 11% inconsistent test cases. Snort and Suricata globally perform better than Zeek for IPv6, with the same or fewer OSes that can be targeted with an insertion or evasion attack. Zeek, which has different IPv4 and IPv6 reassembly policies, performs worse for version 6 (28 inconsistencies) than for version 4 (9 inconsistencies). TCP enables fewer attack opportunities, with only one overlap that Suricata and Snort reassemble incorrectly. Zeek exhibits TCP-based evasion or insertion attacks for 3 OSes out of 4. Since one inconsistency is enough to perform an insertion and/or an evasion, the OSes (i.e., columns 4 and 5 in Table 4.9) that can be targeted are of importance.

Subsection takeaways

As expected, Snort and Suricata (which allow policy configuration) perform overall better than Zeek (which uses a unique policy), when analyzing *single* test mode reassemblies. Using their policy configurability feature, Snort and Suricata from version 7.0.5 can protect themselves completely against IPv4 and IPv6 evasion attacks (for the tested OSes) and almost entirely against TCP segment-based attacks. Every reassembly inconsistency is a possible security issue and, Suricata modified its default IPv4, i.e., bsd, reassembly policy [13] thank to these results.

4.4 Stack reassembly characterization for $n \leq 3$

This section characterizes the reassembly policies of more than 60 network and midpoint stacks for $n \leq 3$. First, it details the obtained policy diversity across stacks and testing scenarii. Third, it describes eight reassembly errors we discovered and reported during the tests; one of them was assigned a CVE [13]. Fourth, it discusses reassembly policy generalization from $n \leq 3$ overlap chunk sequences to any n sequences.

4.4.1 Reassembly diversity

This subsection gives a high-level overview regarding the policies of the implementations detailed in Section 4.2. It first describes the implementations' reassembly policy diversity. Then, it characterizes the impact of the multiple testing scenarii on the test case reassembly. Full reassembly policies can be found at [11].

Use of default NIDS policies In this subsection, we use the corresponding protocol default policy when analyzing the NIDS $n \leq 3$ policies. Indeed, even configured, the last Section 4.3 showed that Snort and Suricata do not reassemble fully in accordance with the corresponding OSes. Furthermore, 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, the situation closest to reality certainly uses the default NIDS reassembly policies.

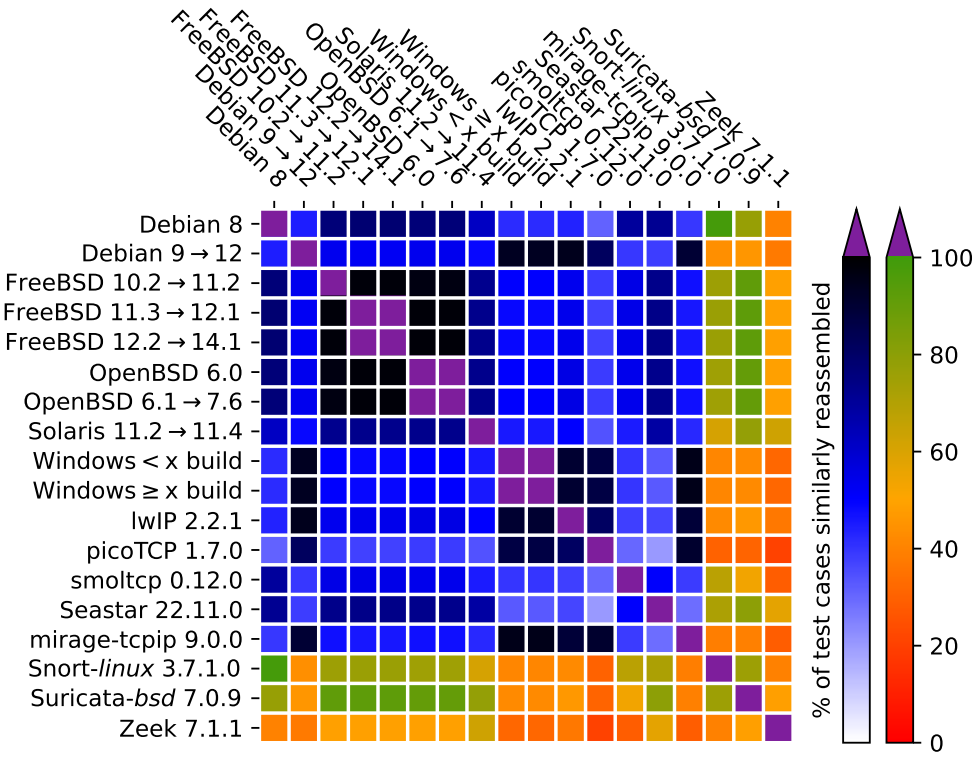


Figure 4.7: IPv4 reassembly similarities across tested implementations. Purple cells encodes perfect reassembly similarities. Green, orange, and red cells encodes similarities between the NIDSes with their default reassembly policies and the other implementations. **x build** corresponds to the 2022/10/25 build of the Windows OSes.

4.4.1.1 Reassembly diversity across implementations

Overall, we observe 15 IPv4, 14 IPv6, and 14 TCP distinct behaviors for 18 to 23 protocol stacks. Figures 4.7 to 4.9 illustrate 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.

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

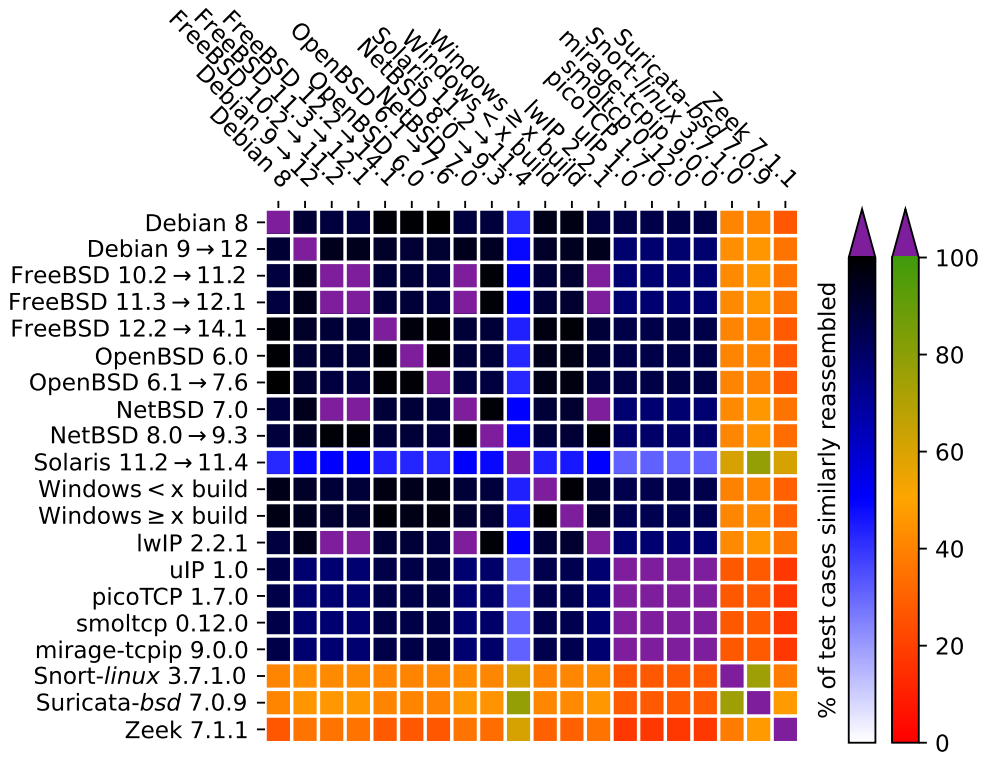


Figure 4.8: IPv6 reassembly similarities across tested implementations. Purple cells encodes perfect reassembly similarities. Green, orange, and red cells encodes similarities between the NIDSes with their default reassembly policies and the other implementations. **x build** corresponds to the 2022/10/25 build of the Windows OSes.

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 changed their IPv6 policies slightly, with less than a 2% difference. The versions introducing the change are the 2022/10/25 Windows builds[6], OpenBSD 6.1, and an unidentified NetBSD version between 7.0 and 8.0[7]. The latest Windows OSes now exhibit the same reassembly behavior when processing IPv4 and IPv6 overlapping or non-overlapping fragments,

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

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

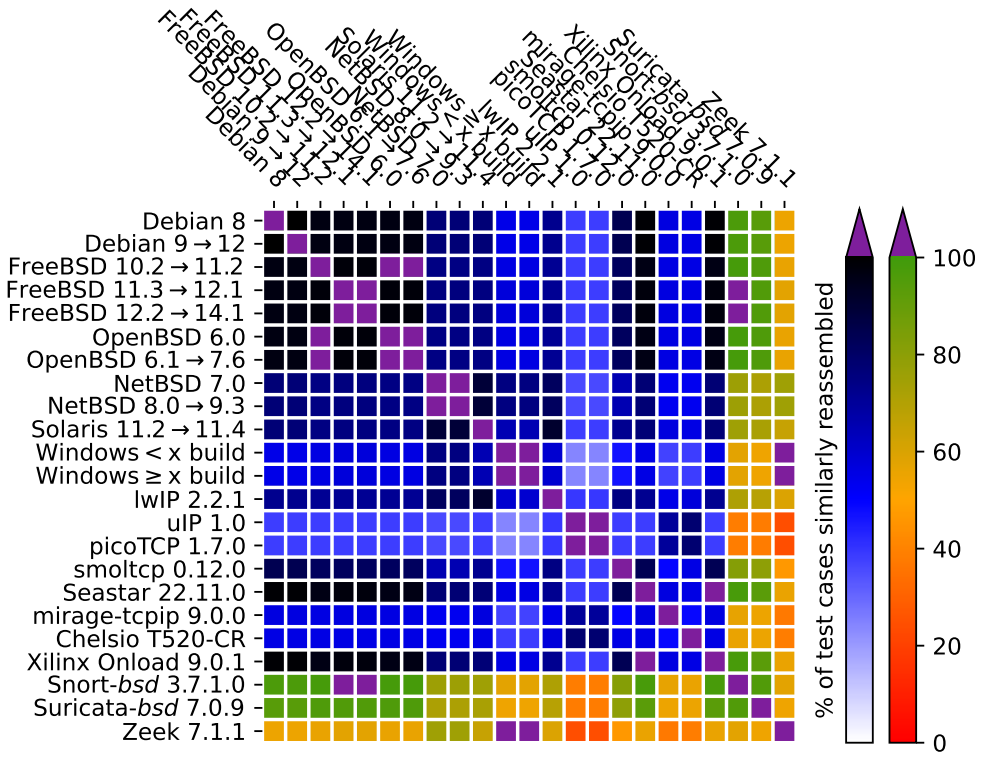


Figure 4.9: TCP reassembly similarities across tested implementations. Purple cells encodes perfect reassembly similarities. Green, orange, and red cells encodes similarities between the NIDSes with their default reassembly policies and the other implementations. **x build** corresponds to the 2022/10/25 build of the Windows OSes.

which is also true for Solaris and the latest Debian OSes.

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 .

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 re-

assemble 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*.

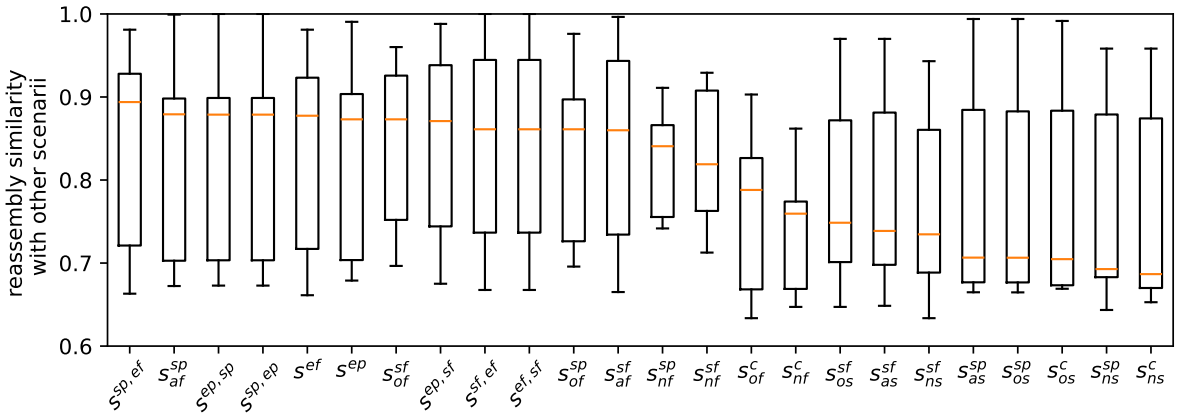
Interesting similarities and dissimilarities IPv6 implementations reassemble the test cases quite similarly. This can be explained for two reasons. First, most of the embedded stacks do not reassemble IPv6 fragments at all. Second, all the tested OSes but Solaris *ignore* about 80% of the test cases. 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.

4.4.1.2 Reassembly diversity across testing scenarii

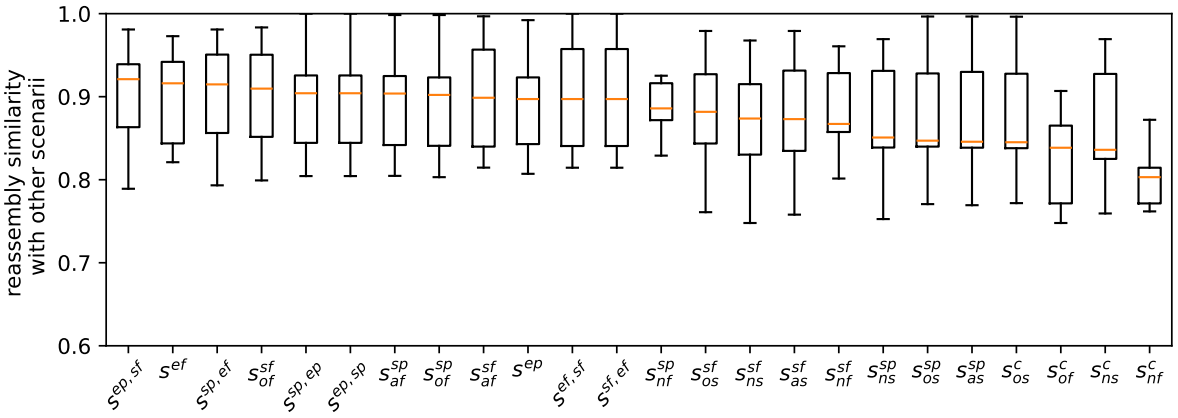
We here analyze the scenario reassembly differences and similarities for every analyzed protocol. Figures 4.10a to 4.10c report the reassembly similarity between the scenarii. For a given scenario on the x axis, every test case reassembly is compared with the 23 other IP (resp. 10 TCP) scenarii for all the considered implementations. The test case reassembly comparison between the x scenario and every other individual scenario provides 23 IP and 10 TCP similarity scores, or y points. Similarity scores are ratios computed as the number of similarly reassembled test cases over the total number of test cases (which is always 422 here). Since orange lines represent the medians, an horizontal line inside the boxplot located at 0.90 means that half of the 23 IP (resp. 10) scenarii have at least 90% of their test cases that are reassembled in the same way as the considered scenario. We remark that the caps of $s^{ep,sp}/s^{sp,ep}$ on the one hand and

[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.

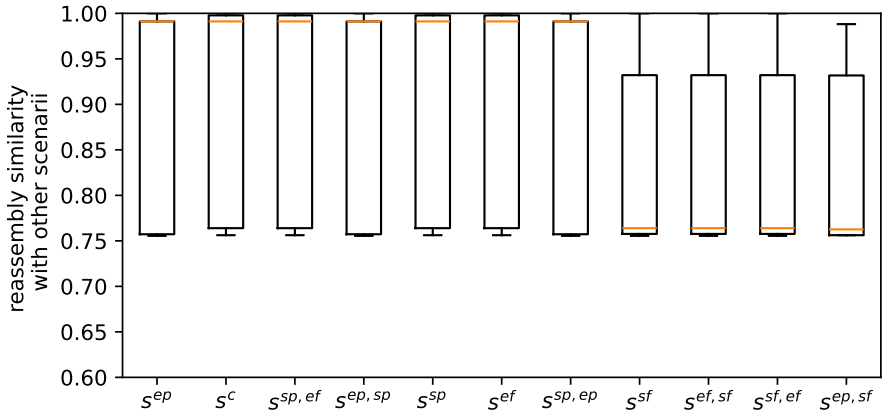
$s^{sf,ef}/s^{ef,sf}$ on the other hand reach 1.0 for every protocol. These pairs of boxplots are also always the same. Therefore, all the protocol implementations reassemble these scenario pairs similarly. We cannot find other such identical scenario tuples across all protocols. All the other introduced scenarii are therefore needed to characterize the reassembly policies.



(a) IPv4



(b) IPv6



(c) TCP

Figure 4.10: Reassembly similarities between scenarii across implementations. Orange lines are the medians. The lowest and upper whiskers correspond to the minimum and maximum y values.

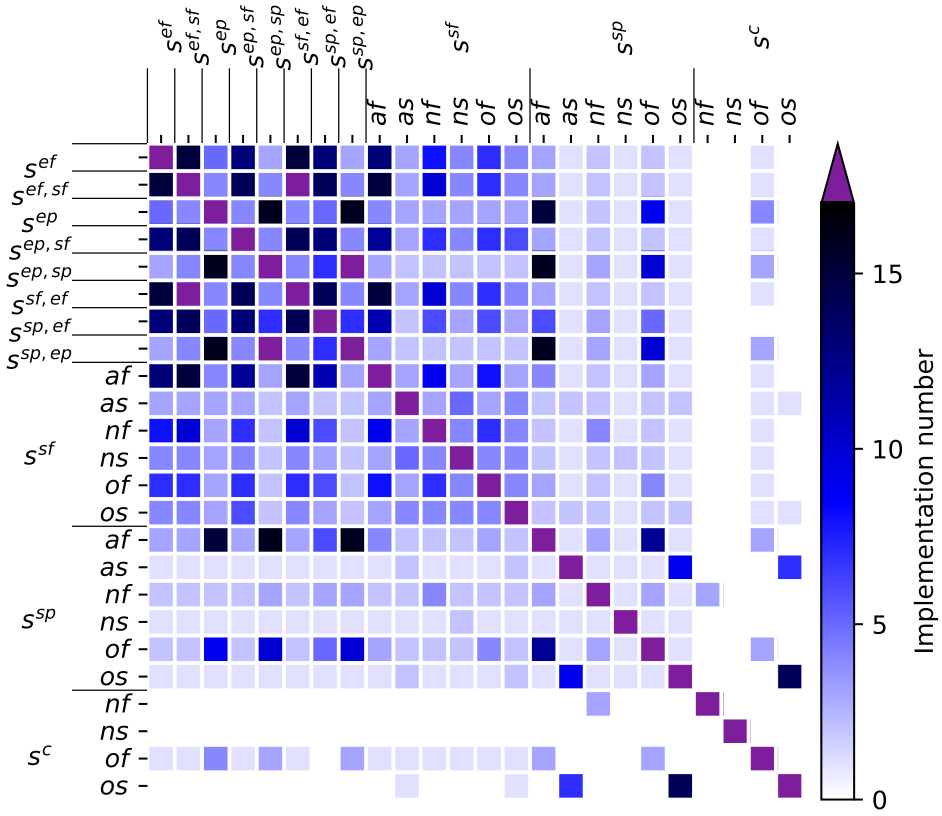


Figure 4.11: IPv4 reassembly similarity between pairs of scenarii that comprise 422 test cases. Purple cells encodes perfect reassembly similarities.

IPv4 This paragraph first analyzes the scenarii that comprise only a protocol-agnostic part. It then focuses on protocol-dependent ones.

Protocol-agnostic scenarii Figure 4.11 encodes reassembly similarities between pairs of IPv4 scenarii across the implementations. Similarity scores are computed as the number of implementations that exhibit the same time policies for every test case. We observe two main groups of IPv4 scenarii. Around 15 implementations reassemble s^{ef}, $s^{ef,sf}$, $s^{ep,sf}$, $s^{sf,ef}$, $s^{sp,ef}$ (resp. s^{ep}, $s^{ep,sp}$, $s^{sp,ep}$) in the same manner. Thus, the implementations tend to reassemble the scenarii with a Start and/or an End fragment that *follows* the test case fragments more similarly. The second group comprises scenarii with extra chunk(s) only *preceding* the test case fragments.

Protocol-dependent scenarii Unsetting the MF bit of the rightmost *finishing* or *starting* fragment(s) impacts significantly test case reassemblies, since there is a small number of implementations that reassemble similarly the s_{af}/s_{as}, s_{of}/s_{os}, and s_{nf}/s_{ns} couples of scenarii. The average reassembly similarity of these couples is between 0.67 and 0.79. The other couples s_{mf}/s_{ms}, s_{mnf}/s_{mns}, s_{onf}/s_{ons}, and s_{omf}/s_{oms}, which are not represented in Figure 4.11 because they are only composed of a subset of the 422 test cases, exhibit the same behavior. We also note that half the implementations reassemble s^{sf}_{nf} or s^{sf}_{of} with s^{sf}_{af} similarly. Thus, there is no shared behavior across IPv4 implementations that would consist in preferring the oldest or newest rightmost finishing fragment. There are very few implementations that behave the same across starting sub-strategies for s^{sf}. We observe more similarities inside the sub-strategies for s^{sp}, with s^{sp}_{af} and s^{sp}_{of} on the one hand and s^{sp}_{as} and s^{sp}_{os} on the other hand which are quite close. This behavior is expected because reassembly conditions are met before all the fragments arrived in some test cases; thus, it is normal that s^{sp}_{af} and s^{sp}_{as} are closest to s^{sp}_{of} and s^{sp}_{os} than s^{sp}_{nf} and s^{sp}_{ns}, respectively. However, the fact that s^{sp}_{af} (and to a lesser extent s^{sp}_{of}) is close to some protocol-agnostic scenarii is more surprising, because of the exact same reasons. Except s^{c}_{os} with s^{sp}_{os}, few implementations reassemble perfectly similarly s^{c}-like scenarii with the others, as the first s^{c} representative in Figure 4.10a is at the 15th position or s^{c}_{ns}-related blank line in Figure 4.11 show it.

Protocol-dependent vs protocol-agnostic only scenarii In Figure 4.10a, we note that s^{sp}_{af} box is really close to $s^{sp,ep}$ and $s^{ep,sp}$. Three implementations (Suricata, Solaris, and NetBSD) cause the difference, with only few test cases that are differently reassembled. But, despite this protocol-dependent and protocol-agnostic reassembly closeness, the scenarii with only a protocol-agnostic part are more similarly reassembled with the others than the scenarii with a protocol-dependent part, as the rightmost and leftmost scenarii show. As regard s^{sf} and s^{sp}-like scenarii, we remark that the MF unsetting strategy that exhibits the closest reassembly with agnostic scenarii is af (in which all the rightmost finishing fragments have the MF bit unset). Therefore, one should unset any rightmost finishing fragment so that the MF ambiguity exposed in Section 4.1.1.2 has the minimal impact on the implementation reassemblies.

IPv6 Nine IPv6 implementations reassemble test cases similarly across the testing scenarii with only a protocol-agnostic part, which explains that the IPv6-related heatmap

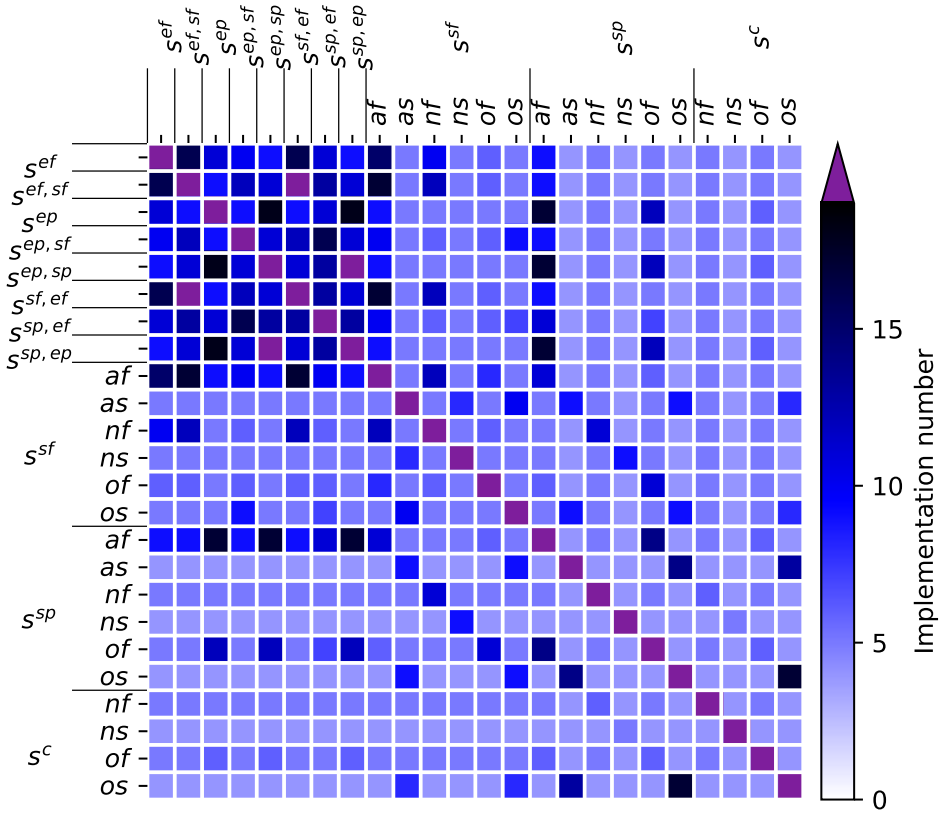
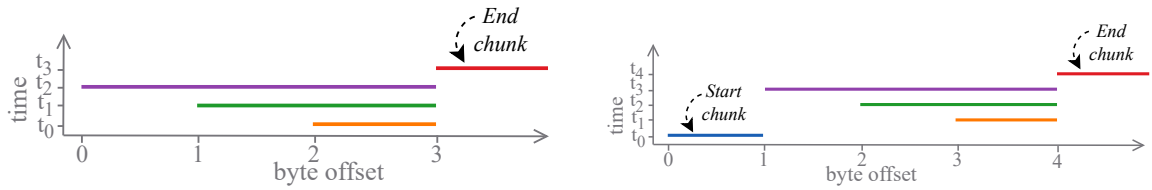


Figure 4.12: IPv6 reassembly similarity between pairs of scenarii that comprise 422 test cases. Purple cells encodes perfect reassembly similarities.



(a) s^{ef} (b) $s^{sp,ef}$

Figure 4.13: The (F, F, F) test case representation.

represented in Figure 4.12 is darker than IPv4. Zeek favors old data; 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. (F, F, F) represented in Figure 4.13 is one such overlapping test case example.

We can group the scenarii in three main categories for the remaining IPv6 stacks: 1) s^{ef}, $s^{ef,sf}$, $s^{sf,ef}$, s^{sf}_{af}, 2) s^{ep}, $s^{ep,sp}$, $s^{sp,ep}$, s^{sp}_{af}, and 3) $s^{ep,sf}$, $s^{sp,ef}$. Here, if one extra fragment (no matter which one) is sent before and another is sent after timewisely the test case chunks, the reassemblies differ. We believe this behavior is due to implementations trying to conform to RFC [47] concerning overlapping data fragments. The concerned implementations (OpenBSD OSes, Debian 8, FreeBSD 14.1) reassemble the overlapping test cases of the first and third groups differently. In the same way as FreeBSD 10.2 to 12.1, NetBSD 7.0, and lwIP mentioned above, these implementations reconstruct test cases of group 1 scenarii that exhibit no overlap at all or overlaps, but with any newest overlapping fragment that is entirely included in the oldest ones. After the $p_{0,1}$ overlap resolution, the fragment flow is somehow reset because chunks 0 and 1 overlap. There is enough data to reassemble with the extra fragment(s) and the test case's chunk 2 in group 1, which is not the case in group 3, since one extra chunk is cleared during the reset. Figure 4.13a (resp. Figure 4.13b) represents one such test case for group 1 (resp. 3), where the chunks 0, 1, and 2 refer to the chunk orange, green, and purple, respectively. Other than that, IPv4 and IPv6 scenarii follow the same heatmap colouring trend; thus, we draw the same conclusions as IPv4 for the rest of the scenarii.

TCP Except $s^{ep,sf}$, all the scenarii have their caps reaching 1.0 in Figure 4.10c, meaning that the tested TCP implementations always reassemble similarly these scenarii with at least one other scenario. We also remark that the similarity medians of the leftmost TCP scenarii (i.e., from s^{ep} to $s^{sp,ep}$) are higher than any IPv4 and IPv6 scenario, which is due to the few TCP scenario numbers and the close reassembly of these rightmost scenarii. Figure 4.14, which plots the reassembly similarities between all pairs of scenarii, confirms this observation. More precisely, we can see four scenario groups: 1) $s^{sp,ef}$, s^{sp}, s^{ef}, s^{c}, 2) s^{ep}, $s^{ep,sp}$, $s^{sp,ep}$, 3) $s^{ef,sf}$, s^{sf}, $s^{sf,ef}$, and 4) $s^{ep,sf}$. The plots of groups 1 and 2 and the plots of groups 3 and 4 are very close. Here, the main difference lies in the presence of $Start$ segment *following* the test case sequence (thus introducing a data hole on the left). Four (resp. five) stacks cause the difference between group 1 and 2 (resp. 3 and 4), namely NetBSD OSes, Suricata, and lwIP (resp. Debian and NetBSD OSes, and lwIP). Seven implementations (i.e., Windows OSes, FreeBSD 11.3 to 14.1, Zeek, Snort, and mirage-tcpip) reassemble all the scenarii similarly, with Windows OSes and Zeek notably always preferring the oldest chunk data.

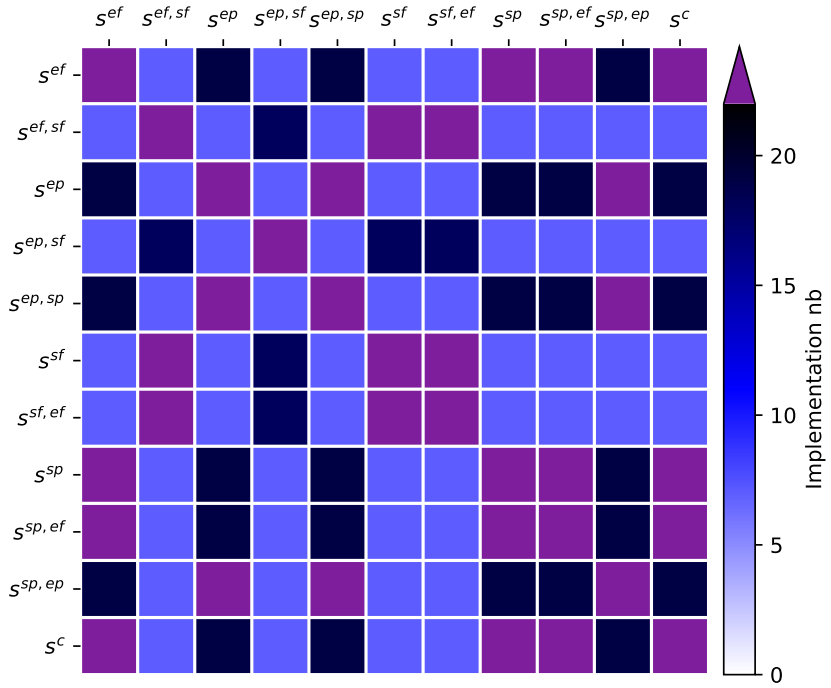


Figure 4.14: TCP reassembly similarity between pairs of scenarii across implementations. Purple cells encodes perfect reassembly similarities.

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.

Subsection takeaways

The 42 IP and 11 TCP scenarii and $n \leq 3$ test cases uncovered diverse implementation reassembly policies (fully available at [11]). They have increased from 5 IP and 6 TCP policies [56], [60], [62] to 12 IPv4, 14 IPv6, and 14 TCP stack behaviors. PYROLYSE provides an automated way to extract such reassembly time

policies. It also eases testing the reassembly consistency of security equipment (such as NIDSes) with the network stack implementations.

4.4.2 Reassembly errors

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

Table 4.10: Reassembly error summary uncovered by PYROLYSE. ! means that the error is partially fixed.

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 4.10 summarizes the encountered errors.

4.4.2.1 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.

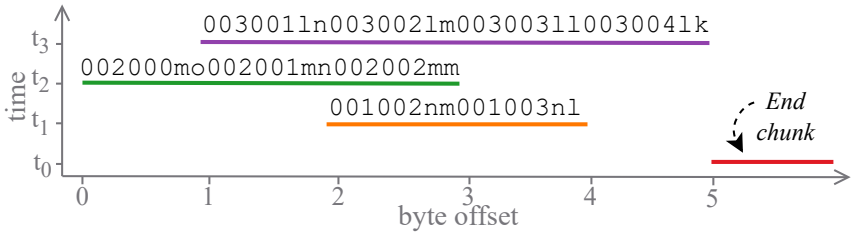


Figure 4.15: The (Oi, O, D) test case representation in a s^{ep} testing context.

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.

Suricata The error is present in 93 to 801 test cases (over the 10,362 IP 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 4.15. With the *default* policy, the reconstructed datagram is 002000mo 002001mn 001002nm 001003nl ..... Since Suricata raises no alert related to the 003004lk 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 [13] and is partially fixed from Suricata version 7.0.5. At the time of writing this thesis, there are still 20 test cases that the *bsd* (default) policy still reassembles with a data hole.

Snort The error concerns all reassembly policies (including the default one *linux*) for 26 to 211 test cases (over the 10,362 IP 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 4.15. 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.

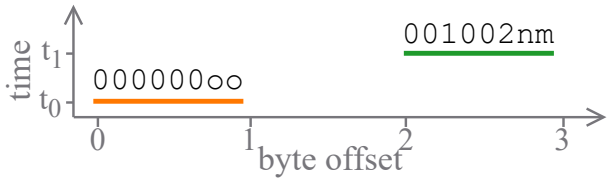


Figure 4.16: The (B) test case representation in a s^c testing context.

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 4.16 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. At the time of writing this thesis, there is no data located after the data hole in Suricata's TCP stream anymore; however, Suricata still raises alerts on this data.

Early response OpenBSD responded to the fragment sequence for 14 triplet cases (over the 422 $n \leq 3$ test 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[9] 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 [128].

4.4.2.2 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

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

test case-related TCP connections. The impacted test cases were, however, different across the implementations.

OpenBSD The OS 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. The error is observed only if the firewall pf is enabled, which is the case by default. This reassembly error was observed in all the testing scenarii and is now patched [129].

Mirage-tcpip and lwIP 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.

4.4.2.3 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 4,642 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.

Subsection takeaways

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.

4.4.3 Inconsistencies between $n = 2$ and $n = 3$ reassemblies

This subsection 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. We first show 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, we identify some root causes for the observed inconsistencies. Finally, considering these root causes, we explore the extension of $n = 2$ pair time policies (as described in Section 4.1.1.4) with manually implemented reassembly algorithms, as a way for the NIDSes to reassemble any n sequence. These algorithms can be described with a few natural language sentences, and thus, if consistent, can easily explain the stack policies.

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

Table 4.11: TCP test case inconsistency number across couple latest OS/NIDS reassembly policy, for the s^{sf} testing scenario.

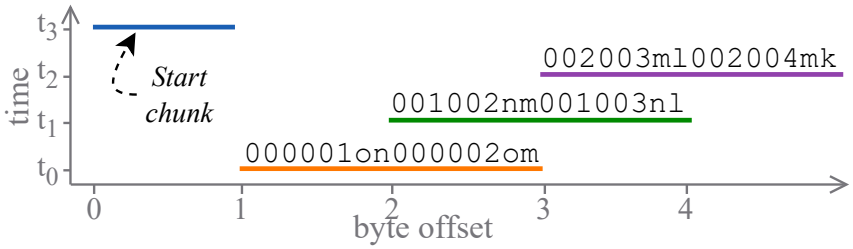


Figure 4.17: The (O, M, O) test case representation in a s^{sf} testing context.

4.4.3.1 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 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 [62], we use our related TCP scenario s^{sf}. Table 4.11 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). Note that these reassembly inconsistencies are not due to any of the errors described in Section 4.4.2. We also observe inconsistencies for other protocols and scenarii, and most of them are not caused by the NIDS reassembly errors."

4.4.3.2 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.

Observations from the $n \leq 3$ observed policies As discussed in Section 4.4.1.1, 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 4.17, 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.

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()` [130]. This function seems to merge data segments, as the source code function description mentions "before queuing `skb`[10], 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 4.18 illustrates such a test case.

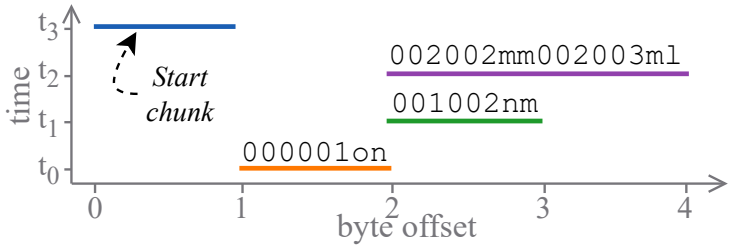


Figure 4.18: The (M, M, S) test case representation in a s^{sf} testing context.

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.

[10]Short for socket buffer.

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

4.4.3.3 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 4.4.1). 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 last paragraph of Section 4.4.3.2, 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 4.4.3.2. 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 reassembly consistency between the algorithms and the implementations.

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 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

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	1/15	picoTCP	6/15	Windows OSes, lwIP, picoTCP, Seastar, mirage-tcpip
IPv6	4/17	picoTCP, uIP, mirage-tcpip, smoltcp	10/17	Windows OSes, FreeBSD 10.2 to 11.2, FreeBSD 11.3 to 12.1, NetBSD 7.0, lwIP, picoTCP, mirage-tcpip, uIP, smoltcp
TCP	2/20	Windows OSes	10/20	Windows, OpenBSD, and FreeBSD OSes, smoltcp, Seastar, Xilinx Onload

Table 4.12: Reassembly consistency of the 3,376 IP and 4,642 TCP ($n \leq 3$) protocol-agnostic test cases across network stacks.

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 [130]) 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 without a 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 4.4.3.2. These mechanisms explain, at least to some extent, the Suricata and Snort's inconsistent reassemblies for $n = 3$ reported in Section 4.4.3.1. We 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 therefore implemented custom reassembly algorithms to verify this point. The algorithms are further detailed in Appendix C.3.

Triplet test case consistency analysis The consistency performance of custom reassembly algorithms is exposed in Table 4.12. The second column lists implementations whose time policy of residual pairs inside triplets (i.e., we ignore triple overlap as defined in Section 4.1.1.4) 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 4.4.1.2. 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 explainable. See a detailed description of the related mechanisms in Appendix C.2.

Among the easiest behaviors the algorithms succeeded to capture, there are implementations that behave uniformly across scenarii.

- FreeBSD 10.2 to 11.2 (for IPv6), NetBSD 7.0 (for IPv6), and lwIP (for IPv4 and IPv6) resolve the overlaps immediately and always drop the newest fragment when they ignore the $n = 2$-related pair.
- If picoTCP (for IPv4 and IPv6) and mirage-tcpip (for IPv4) ignore one of the triplet test case's pair in $n = 2$ tests, it actually means that the entire triplet is ignored.
- Seastar (for IPv4) merges any overlapping or non-overlapping IPv4 fragments after the overlap resolution.
- Windows (for TCP) always favors oldest data; thus, any implemented algorithm is consistent.
- For TCP, several algorithms are consistent with FreeBSD 11.3 to 14.1, smoltcp, Seastar, and Xilinx Onload reassemblies. It means that the related mechanisms do not change the overlap time policy or, in other words, one relation r_1 is changed to another r_2 and, r_1 and r_2 time policies are the same in $n = 2$ testing. The algorithm that merges overlapping (after the overlap resolution) and contiguous chunks particularly suits all these implementations.

Other stacks behave differently depending on the testing scenario.

- For TCP, FreeBSD 10.2 to 11.2 and OpenBSD OSes resolve the overlaps delayedly

and does not apply any segment merging for s^{sf}-like scenarii. For the other scenarii, several algorithms but the one described are consistent with these OSes' reassemblies.

- For IPv4 and IPv6, Windows resolves the overlaps immediately for s^{sf}-like scenarii. For the other scenarii, several other algorithms are consistent with Windows reassemblies.

However, more than 10 stack policies cannot wholly be inferred for each protocol.

Subsection takeaways

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 52 protocol stacks.

4.5 Discussion and limits

The section discusses the chapter results. First, it analyzes protocol stacks' conformance with IPv4, IPv6, and TCP RFCs. Second, it gives insights regarding the usability of the $n \leq 3$ overlap test case inconsistencies in real-world attacks. Third, it debates the cost of the policy configurability inside the NIDSes. Finally, it lists recommendations for the configurable NIDSes to use the described $n \leq 3$ stack policies.

4.5.1 Reassembly compliance with RFCs

IPv4 and TCP RFC [8], [49] do not specify the behavior implementations must adopt in the presence of fragment or segment data overlaps (see Section 1.1). 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 [47] states that implementations must drop all the fragments if any of the original datagram's bytes overlap. As reported in Section 4.4.1, 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, we still observe overlapping test cases that are reassembled, which is non-compliant with the RFC but consistent with the related works [56], [64], [76].

4.5.2 From theory to practice: test cases' usability in real-world attack scenarii

This subsection discusses the practical usability of this chapter's results from the attacker's perspective. First, it differentiates the reassembly inconsistency usability as regards the constituent test case Allen relations. Second, it analyzes the testing setup's bias. Third, it lists some causes (e.g., equipment, network events) that can actively alter the original attacker chunk sequence. Finally, it discusses the impact of our findings on threat models that are partly different from the one described in Section 2.2.

4.5.2.1 Relation differences

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

4.5.2.2 Testing setup impact

This part lists some practical limitations from our testing setup that can impact the results' use.

Context importance In addition to the test cases modeled with the Allen relations and the byte and time characteristics, the test cases require some contextualization, which we implemented in two ways: with the testing *mode* for $n = 2$ and with the testing *scenarii* for $n \leq 3$.

First, the mode tested the pair overlaps either individually (*single*) or altogether (*multiple*). IP showed the most reassembly discrepancies between the two modes across OSes, but we also observed two TCP differences. Second, the scenarii introduced 1) some extra contiguous chunks before and/or after the test case chunks (protocol-agnostic part) and 2) different `MF` unsetting strategies in case of overlapping rightmost fragments (protocol-dependent part). We found two pairs of scenarii that are always reassembled the same across tested stacks; however, we could not find any other similarly reassembled scenario tuples. TCP stacks reassemble more similarly across testing scenarii since we found three triplets of scenarii that are always reassembled the same across implementations.

Since the context surrounding the overlapping chunks significantly impacts implementations' reassemblies, one who wishes to reproduce our findings should reuse the test case sequences as is. Extracting part of or all the chunks of a given test case and integrating them into one bigger chunk sequence may not produce the intended behavior. To link with Section 2.3, note that, if the used test case sequence has desynchronized any stateful upper protocol, the communication should remain desynchronized, as long as this protocol connection holds, and thus, contiguous non-overlapping chunks can be used after the chosen overlap test case chunk sequence.

Test case chunk length The chunk payload patterns are 8-byte long, and the tested chunks' sizes are from 8 to 40 bytes. Using bigger chunk sizes could modify the implementations' reassemblies at the receiver side. As reported in Section 4.2, we could not test the IPv4 stack of NetBSD because it ignored IPv4 fragments lower than 40 bytes. Furthermore, we observed some IPv6 stacks ignoring all our test cases; however, we think this is not due to ignoring small-sized chunks but rather due to non-implemented IPv6 defragmentation.

4.5.2.3 Chunk sequence alteration before reaching the NIDS or monitored host

Offloaded stacks on NIC might impact the network stack and NIDS policies. For instance, the NICs we tested have their own TCP policies (see Section 4.4.1). NIDS

developers advise configuring NIDS instances so nothing alters the monitored traffic, such as NIC offloading. By configuring Suricata with the Extreme Performance Tuning guidelines [78], one can consider that Oprea et al. [77] made the overlapping IPv6 traffic reaching Suricata alteration possible, as the NIDS' host (i.e., Linux here) stack performed the packet reassembly. These recommendations, however, do not guarantee that such alteration does not occur on the monitored hosts themselves.

The works [70], [71] show that some middleboxes drop or reassemble IPv4 fragments, but which middlebox types actually perform this is unclear (e.g., routers, end hosts' firewall). The reassembly of such middleboxes should be tested to clarify this point.

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

4.5.2.4 Practical impacts in other threat models

Three other threat models could leverage this chapter's results.

CS and firewall evasion and insertion Stack reassembly policy diversity can be used to bypass other network equipment than the NIDSes, such as CSes or firewalls. CSes and NIDSes indeed have a lot in common in the way they monitor network traffic. As mentioned in Section 1.4, some works reported that CSes could be bypassed using differentiated overlap-based reassemblies until 2017. Then, other works reported the technique unusability; but, we think this result is as least partly due to incomplete testing. As Khattak et al. rightly explained in [68], disambiguating traffic with overlapping data is very costly for a CS because it requires either traffic normalization or some reassembly policy knowledge of the monitored hosts. Evidence shows that some middleboxes normalize IP fragments by dropping or reassembling them. However, there is no such normalization evidence for TCP. Some other works [64], [131], [132] showed that overlapping fragments could be used to evade firewalls, and more precisely, to evade the upper-layer-related filters. Most firewalls can indeed inspect header fields of the IP `next proto`, such as the UDP or TCP ports. For instance, one could bypass a rule that would filter any (insecure) HTTP traffic with overlapping TCP fragments. In both CS and firewall evasion cases, their reassembly policies should be tested and documented to enable one to choose suitable evasion or insertion test case candidates.

TCP and IP stack fingerprinting Stack reassembly policies can be used to perform host stack/OS fingerprinting during the discovery phase before an attack. In this phase, the attacker gathers information on the victim host they plan to attack. The victim host's OS name and version are highly valuable data because they restrict the exploitable vulnerability list. Since 1) we observe almost one reassembly policy for every stack and 2) we observe reassembly differences across general purpose OS versions, one could use reassembly policies to fingerprint and identify the victim host stack.

Exploiting reassembly errors We reported eight reassembly errors in Section 4.4.2 that impacted diverse stack types, i.e., NIDSes, OSes, embedded/IoT. Half of them have not yet been fixed, and can thus be used to attack the related stacks. As reported in Table 4.10, an attacker could leverage the disclosed lwIP and mirage-tcpip TCP error to perform a DoS attack or the Snort IP error to bypass its pattern-matching functionality. Generating and testing $n > 3$ chunk sequences with PYROLYSE could provoke other stack reassembly errors that an attacker could further use to abuse a stack.

4.5.3 Reassembly policy configurability cost

Suricata and Snort allow one to configure reassembly policies according to the monitored host OS, while Zeek does not. We analyze and compare configurable and non-configurable reassembly policy costs in the following. Table 4.13 summarizes the main pro and cons points.

Reassembly policy	Reassembly engine complexity	Reassembly policy usability	Benign overlap-related false alerts	Developper effort	User effort during conf.
Configurable	−	+	+	−	−
Non-configurable	+	−	−	+	+

Table 4.13: Pros and cons summary for the reassembly policy configurability.

4.5.3.1 Configurability cost

Making a NIDS configurable regarding various implementation reassembly policies requires several steps. As network protocol implementations may evolve over time, check-

ing whether their policies have changed regularly is necessary. NIDSes should be able to easily modify and add reassembly policies as well as extend the mapping between stack versions and reassembly policies. NIDS reassembly policies must be carefully tested to ensure the NIDS reassembles consistently with the stacks and that no bug was introduced. Finally, NIDS users must correctly configure their NIDS instance to associate IP addresses with reassembly policies. This configuration task is challenging as an organization's IT infrastructure may rapidly evolve and comprise hundreds (or many more) of monitored hosts. Moreover, IP addresses may be non-static, increasing the human cost of such a configuration even more.

4.5.3.2 Non-configurability cost

A NIDS that does not make the reassembly policy configurable must propose another countermeasure to face overlap-based attacks. For example, an alert-based solution is possible and would consist in raising an alert whenever an inconsistent data overlap is detected (Zeek, Suricata and Snort implement such a countermeasure). Several approaches may be adopted depending on whether a chunk sequence with overlapping data is inherently considered malicious [11]. If so, there may not be the need for extra information logging, but if not, such a NIDS must log the beginning and finishing byte offsets as well as data on overlapping portions for further analysis. In any case, reassembled data from these chunk sequences must not be used (e.g., for TLS certificate extraction) because the NIDS would not know the monitored host reassembled payload.

4.5.3.3 Lessons learnt from Snort, Suricata, and Zeek observed reassemblies

Snort and Suricata's implemented policies are based on Novak et al.'s works [60], [62], which are around 20 years old. Network stacks have been updated since then and their reassembly policies too. Because these NIDSes have not included the network stack (e.g., OS) policy test in their testing pipeline, there was a small chance that the NIDSes could reassemble consistently with these network stacks' recent versions. Snort and Suricata should include a stack policy evolution test in their pipeline if they still wish to offer the reassembly policy configurability functionality. In such a case, they should also perform intensive reassembly consistency testing, for example with PYROLYSE. Other-

[11] There are benign reasons for complete or partial overlaps to occur, as reported and quantified in Chapter 3 and [4], [9].

wise, besides being inconsistent with stack reassemblies, the proposed (implemented) policies could also introduce reassembly errors, as described for Snort and Suricata in Section 4.4.2. Such reassembly errors can be more harmful than not proposing the policy configurability but implementing a unique, light, and well-maintained reassembly policy. NIDS proposing the policy configurability option should take into account the significant development efforts required.

Zeek, which does not offer reassembly policy configurability, is also vulnerable to insertion and evasion attacks with overlapping IP or TCP data. Since it is the only proposed countermeasure, Zeek editor should verify that the NIDS effectively raises alerts when any IP or TCP overlap occurs, for example, when processing the test cases generated by PYROLYSE. We did not observe reassembly errors when testing Zeek; thus, it supports the idea formulated in Table 4.13 that Zeek's reassembly engine is probably less complex than Snort and Suricata's one, requiring fewer development efforts and leading to fewer development errors.

4.5.4 Configurable NIDS challenges to address $n > 2$ 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 52 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 52 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.

4.6 Conclusion

This chapter provided an overview of stack reassembly policies when processing overlapping IP fragments and TCP segments. We enhanced related works' testing models to ensure test case exhaustiveness and to cover more real-world contexts around the overlap chunks. We implemented this new model inside PYROLYSE, which we used to test the behavior of some IPv4, IPv6, and TCP stacks and NIDSes. After the investigation of PYROLYSE artifacts, we observed much more diverse reassembly behaviors than previously, as almost every stack has its own policy when addressing $n \leq 3$. We showed that, most of the time, $n = 3$ test case reassemblies of network stacks cannot easily be deduced from $n = 2$-based policies. Hence, the configurable NIDSes' behavior that directly applies $n = 2$-based reassemblies to any n chunk sequence is incorrect. We found that the tested NIDSes, Snort, Suricata, and Zeek, reassemble inconsistently with general purpose OSes, even if they are tested under the same conditions as the works they are based on and if the NIDSes are configured with the related implemented OS policy (if applicable). Every uncovered inconsistency is a possible security issue. Finally, we disclosed eight problematic reassembly behaviors, including reassembly logic, connection termination, and NIDS log display errors. One was assigned a CVE [13], and five are completely or partially fixed.

Conclusion

Contents

Contribution summary **129**
IPv4 fragment overlaps in real Internet traffic 129
IPv4, IPv6, and TCP policies and related NIDS evasions and insertions 130
Perspectives and Future works **132**
Measure IPv6 and TCP overlaps in Internet traffic 132
Extend the targeted stacks and protocols inside PYROLYSE 133
Extend the configurable NIDSes 133
Improve the security of network stacks with respect to data overlaps . 134

This chapter summarizes the contributions and discusses the perspectives of the thesis.

Contribution summary

In this thesis, we focused on IPv4, IPv6, and TCP data overlaps, which are protocol ambiguities that Ptaceck and Newsham originally described in 1998 [1]. An ambiguity arises whenever one or several network packets are differently treated by two hosts (e.g., one host drops or ignores a packet while the other accepts it). In our case, the two hosts are a NIDS and a monitored host. Throughout this thesis, we provided several contributions related to the abuse of IPv4 and IPv6 fragment and TCP segment overlaps, that are valuable for both the research community, and NIDS and network stack editors. We notably enhanced the theoretical model, the testing method, tested stacks and tooling in comparison to previous works.

IPv4 fragment overlaps in real Internet traffic

The countermeasures to apply inside the NIDSes when fragment or segment overlaps may vary depending on whether an overlap is inherently considered malicious. Until 2008, some works showed that IPv4 fragment and TCP segment overlaps are actually observed on the Internet [2]–[4]. Some other works [5], [6] discouraged the use of IPv4

fragmentation due to its potential for abuse (e.g., NIDS evasion or insertion, DoS) or resource overhead. In Chapter 3, we verified whether IPv4 fragmentation has disappeared from the Internet, and we provided updated IPv4 fragment overlap measurements using backbone traffic.

RQ.1 – What are the characteristics of IPv4 fragments in the wild, and to what extent does data overlap occur?

Our 18-year longitudinal analysis of Internet traces from the MAWI dataset showed that IPv4 fragmentation has accounted from some tenths of a percent to three percents of packets, yearly. There can be some episodes, such as the evolution of services (e.g., UDP/DNSSEC) or the introduction of new services (e.g., UDP/QUIC), that (more or less) temporarily required the senders' original IPv4 datagrams to be fragmented. Varying PMTUs between 1396 and 1500 bytes (a sign of mobile networks and VPN tunnels-related PMTUs) also explained that fragmentation was observed throughout the years. Even if a few hosts concentrated most of the fragmented traffic, between 493 to 25,567 hosts sent yearly some IPv4 traffic that was fragmented at some point. Fragments overlapped partially in 0.3% of the fragment series and completely in 3.0 to 5.2% of the series, in the Internet traces we analyzed. Depending on the period, we also measured that between 0.1 and 0.4% of the hosts sent fragments that have chances to overlap at the end-host, because they reused the same `IPID` within 5 seconds for different original datagrams. Such senders did not comply with RFCs [8], [9], as they did not rate-limit themselves or used constant `IPID` for non-atomic datagrams (i.e., datagrams that can be fragmented on-path).

The NIDSes should, therefore, not ignore the IPv4 fragments when reconstructing the monitored host data streams. And, raising a NIDS alert when a partial or a full fragment overlap occurs is possible, but it may generate a non-negligible number of false positive alerts.

IPv4, IPv6, and TCP policies and related NIDS evasions and insertions

In Chapter 2, we extended the insertion and evasion attack model from previous works so that it better fits the IP and TCP overlap ambiguity's exploitation. Evasion (where the target is an host monitored by the NIDS) and insertion (where the target is the NIDS) attacks aim to desynchronize the NIDS and monitored host data buffers. We identified

four attack scenarii to perform an evasion or an insertion, which constrain the overlapping chunk (fragment or segment) payloads. If either the NIDS or the monitored host ignores the overlapping packets, the overlapping data portion can carry any data. Otherwise, the overlap data portion should be both semantically and syntactically correct. Finally, we explained that when carefully crafting the overlapping data portion(s) and using a stateful protocol on top of IP fragments or TCP segments, the desynchronization of the NIDS and monitored host data buffers can persist, even after the overlap attack.

In Chapter 4, we described the reassemblies of network stacks and evaluated to what extent the well-known open source NIDSes Suricata, Snort, Zeek could be attacked with overlap-based insertions and evasions. To that end, we introduced a new model that permits the *complete* testing of stacks: i) exhaustive set of overlap test cases for any n overlapping chunks with the Allen's interval algebra, ii) 42 IP and 11 TCP testing scenarii that reproduce real world context around the overlapping chunks, iii) two sets of patterns to populate the chunk payloads, and iv) a formalism that describes the obtain reassembly policies. We tested and described the policies of a wide range of network stacks (OSes, embedded/IoT, unikernel, DPDK-compatible, and NICs) and of the Suricata, Snort, Zeek NIDSes, for up to three ($n \leq 3$) overlapping chunks. We make the related tool PYROLYSE we developed and artifacts publicly available [11], [12].

RQ.2 – How diverse and correct are IPv4, IPv6, and TCP implementation reassemblies when exploring test case context and up to $n = 3$ overlaps?

Reassembly policies are highly diverse, as our tests revealed 12 IPv4 behaviors out of the 15 tested stacks, 14 out of the 17 IPv6 stacks, and 14 out of the 20 TCP stacks, in comparison to the 5 IP and 6 TCP policies described in previous works. Testing the exhaustive set of possible overlaps for $n \leq 3$ with various contexts (representing 10,362 IP and 4,642 TCP test cases in total) has therefore allowed us to demonstrate that almost every tested stack has its own reassembly policy. Some of these stacks not only have their own policy, but they also reassemble with errors. We indeed discovered eight reassembly errors, which we disclosed to the concerned editors (i.e., Suricata, Snort, OpenBSD, lwIP, and mirage-tcpip). One CVE [13] was assigned to one of the Suricata errors. At the time of the thesis writing, half of the errors were partially or fully corrected.

RQ.3 – Can the NIDSes Suricata, Snort, and Zeek be subject to overlapping-based attacks when monitoring hosts with general purpose OSes?

When Suricata, Snort, and Zeek's default configurations are used, only Snort is perfectly consistent with the latest FreeBSD OSes for TCP, and Zeek is consistent with Windows OSes for TCP, for all ($n \leq 3$) the generated IP and TCP test cases. However, when the policies of Snort and Suricata are configured, they perform overall better than Zeek, which always uses a unique policy. For example, Snort and Suricata from version 7.0.5 can protect themselves completely against IPv4 and IPv6 evasion attacks (for the tested OSes) and almost entirely against TCP segment-based attacks, when we tested one scenario for the pairs ($n = 2$) of chunks. We reported the $n \leq 3$ reassembly policies of the tested stacks to the NIDS editors, because every inconsistency is a possible security issue.

Perspectives and Future works

In this section, we discuss four research tracks that could be investigated in the future.

Measure IPv6 and TCP overlaps in Internet traffic

In Chapter 3, we measured the occurrence and structure of IPv4 overlaps in MAWI fragment series. Up to our knowledge, IPv6 overlaps have never been quantified in Internet traces, and the only work that have analyzed TCP overlaps dates back to 2008 [4]. To strengthen our IPv4 analysis and help NIDS editors and users take the correct actions when facing IP fragment or TCP segment overlaps, the characteristics of IPv6 and TCP chunks should be studied. Moreover, a more in-depth study could be conducted regarding the structure of IPv4 fragment series overlaps. That way, a NIDS editor or user that wishes to raise a security alert when a data overlap occurs could differentiate the alert criticity level depending on the overlap types, or relations.

RFC 6864 [9] now allows the IPv4 stacks to use any `IPID` value when sending atomic datagrams, because they should not be fragmented on-path. The works that measured the IPv4 stacks behavior in respect to `IPID`s did not distinguish between atomic and non-atomic datagrams [100], [113]. In Chapter 3's analysis, we only looked at constant values for a small subset of the non-atomic datagrams (i.e., series where the fragments overlap within 120 seconds). Since the `IPID` predictability for non-atomic datagrams can be exploited to perform attacks (e.g., [6], [100]), the IPv4 stack behavior when setting non-atomic datagrams' `IPID` should particularly be studied.

Extend the targeted stacks and protocols inside PYROLYSE

In this thesis, we tested new types of network stacks in comparison to the literature. However, more stacks for each stack type should be tested to have a broader picture of the stack reassembly policies on the Internet. First of all, more instances of the stack types we addressed in this thesis should be tested (e.g., F-Stack, uTCP, mTCP, or mOS). Second, there are some stack types that we did not address, but which would require small efforts to test with PYROLYSE, since they do not require changing the pipeline (e.g., ICSes). Finally, some middleboxes can alter the fragment or segment sequence before it reaches the end-host, such as routers, load balancers, censorship systems, or firewalls. This equipment testing can be challenging because it can be more difficult or impossible to access the reassembly logs. The stack reassembly for a given test case (as defined in PYROLYSE) might require sending multiple chunk sequences. In these cases, the PYROLYSE pipeline needs to be adapted.

We tested the reassembly of IPv4, IPv6, and TCP stacks when processing overlapping data. Other protocols propose this chunking mechanism, with some specificities: Wifi 802.11, 6LowPan, or IPsec (IKEv2). One could assess whether these protocol implementations have different reassembly policies. Because if so, an attacker may also exploit these protocol ambiguities to perform NIDS evasions or insertions.

Extend the configurable NIDSes

In Chapter 4, we attempted to explain and formalize the stack $n \leq 3$ reassembly policies with some (real) mechanisms that have a side effect on the stack policies. The implemented mechanisms succeeded in easily and fully explaining some policies, and they improved reassembly predictability compared to the baseline for the others. Security-wise, however, everything except perfect reassembly consistency (or explainability, in that case) is unsatisfactory from a configurable NIDS perspective. More mechanisms could be implemented and tested inside PYROLYSE to try to explain the remaining unexplained policies for $n \leq 3$. However, we think that some of the stack policies are too specific and cannot be abstracted by a small set of generic reassembly mechanisms. In this light, we recommended that configurable NIDSes modify their policy implementation by using the pair ($n = 2$) policies in all cases and raise an alert as soon as they process a chunk sequence with more than one overlap. As almost every stack has its own policy, this could be done painlessly by storing policies inside HashMaps and per-

forming look-ups on the fly.

The NIDS manual policy configuration task is challenging as an organization's IT infrastructure may rapidly evolve and comprise hundreds (or many more) of monitored hosts. Moreover, IP addresses may be non-static, increasing the human cost of such a configuration even more. This configuration could be painlessly automated through passive stack fingerprinting [97], [98] or active fingerprinting [61], [94]. As several host OSes may be behind an IP address, NIDSes should consider changing the IP address-based reassembly to a flow-based reassembly. These challenges should be investigated in future works.

Improve the security of network stacks with respect to data overlaps

PYROLYSE uncovered eight reassembly errors during the thesis testing campaign. For the NIDSes, one drawback of proposing the policy configurability feature is that the reassembly engine is more complex and, thus, errors can be introduced, as in Snort and Suricata. These NIDSes would benefit to incorporate PYROLYSE inside their testing pipeline. We argue that this is also the case for IPv4, IPv6, and TCP stacks which often update their reassembly code, to ensure that the newly introduced code has not introduced any errors. Generating test cases with more than three overlapping chunks could also uncover other reassembly errors in the stacks.

Glossary

CS Censorship System. 38, 41, 42, 44, 124

CVE Common Vulnerabilities and Exposures. 50, 99, 112, 128, 131

DF The field `Don't Fragments` bit of the IPv4 header. 23–25, 27, 71, 72

DoS Denial of Service. 33, 36, 113, 125, 130

DPDK Data Plane Development Kit. 88, 91, 92, 131

ICS Industrial Controle System. 133

IDS Intrusion Detection System. 16

IoT Internet of Things. 15, 17, 88, 91, 92, 125, 131

IT Information Technology. 126, 134

IXP Internet Exchange Point. 33

MAWI Measurement and Analysis on the WIDE Internet (working group). 12, 17, 57, 58, 60, 62, 64, 66, 73, 130, 132, 158

MF The field `More Fragments` bit of the IPv4 or IPv6 header. 23–25, 27, 62–64, 71, 80, 107, 113, 119, 123, 158

MTU Maximum Transmission Unit. 24–30, 40, 62, 72

NIC Network Interface Controller (also known as Network Interface Card). 88, 91, 92, 123, 124, 131

NIDS Network Intrusion Detection System. 12, 13, 16–19, 21, 22, 37–42, 44, 45, 47, 49–56, 73–76, 78, 87, 88, 91–103, 110–116, 118, 119, 121–134

OS Operating System. 11, 13, 15, 16, 18, 36, 39–41, 43, 44, 47, 49, 50, 52, 71, 75, 84, 86, 88, 91–103, 109–112, 114–116, 120, 121, 123, 125, 126, 128, 131, 132, 134

PMTU Path Maximum Transmission Unit. 28, 29, 60, 62

PMTUD Path Maximum Transmission Unit Discovery. 28, 29

RFC Request For Comments. 12, 16, 22–25, 27–30, 35, 37, 41, 57, 59, 69–73, 80, 109, 121, 122, 132, 158

TTL The field `Time To Live` of the IPv4 header. 71, 72

VPN Virtual Private Network. 23, 36, 63, 64, 130

WIDE Widely Integrated Distributed Environment (project). 58

Bibliography

[1] T. H. Ptacek and T. N. Newsham, “Insertion, evasion, and denial of service: eluding network intrusion detection”, Technical report, Secure Networks, Inc, Tech. Rep., 1998 (pages 5, 6, 16, 37, 39, 49, 51, 56, 85, 90, 129).

[2] C. Shannon, D. Moore, and K. Claffy, *Characteristics of fragmented IP traffic on Internet links*, 2001 (pages 6, 25, 32–35, 59, 62, 63, 67, 69, 70, 129).

[3] W. John and T. Olovsson, “Detection of malicious traffic on back-bone links via packet header analysis”, *Campus-Wide Information Systems*, vol. 25, no. 5, pp. 342–358, 2008 (pages 6, 32, 34, 35, 59, 60, 62, 69, 129).

[4] M. Vutukuru, H. Balakrishnan, and V. Paxson, “Efficient and robust TCP stream normalization”, in *SP 2008*, IEEE, 2008 (pages 6, 34, 126, 129, 132).

[5] C. A. Kent and J. C. Mogul, *Fragmentation considered harmful*. Citeseer, 1987, vol. 17 (pages 6, 36, 73, 129).

[6] Y. Gilad and A. Herzberg, “Fragmentation considered vulnerable”, *ACM Transactions on Information and System Security (TISSEC)*, vol. 15, no. 4, pp. 1–31, 2013 (pages 6, 33–36, 73, 129, 132).

[7] K. Cho, K. Mitsuya, and A. Kato, “Traffic data repository at the {WIDE} project”, in *2000 USENIX Annual Technical Conference (USENIX ATC 00)*, 2000 (pages 7, 58).

[8] J. Postel, *RFC 791: Internet Protocol*, 1981 (pages 7, 22–24, 27, 28, 31, 53, 72, 80, 121, 130).

[9] J. Touch, *RFC 6864: updated specification of the IPv4 id field*, 2013 (pages 7, 23, 25, 35, 59, 126, 130, 132).

[10] J. F. Allen, “Maintaining knowledge about temporal intervals”, *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, 1983 (pages 8, 77, 87).

[11] *PYROLYSE tool and paper artifacts*, 2025. [Online]. Available: https://github.com/ANSSI-FR/pyrolyse (pages 8, 19, 90, 99, 110, 131, 153).

[12] *DIMVA paper artifacts*, 2025. [Online]. Available: https://gitlab.inria.fr/laubard/dimva_2025_artifacts (pages 8, 96, 98, 131).

[13] NIST. “CVE-2024-32867”. (2024), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-32867 (visited on 11/16/2025) (pages 9, 97, 99, 112, 128, 131, 151).

[14] M. Castells, *The information age*. Oxford Blackwell Publishers, 1996, vol. 1 (page 15).

[15] H. Ritchie, L. Rodés-Guirao, E. Mathieu, *et al.* “Number of people using the Internet (1960-2021)”. (), [Online]. Available: https://archive.ourworldindata.org/20250925-233948/grapher/number-of-internet-users.html (visited on 11/16/2025) (page 15).

[16] H. Ritchie, E. Mathieu, M. Roser, and E. Ortiz-Ospina. “Share of the population using the Internet (1960-2023)”. (), [Online]. Available: https://archive.ourworldindata.org/20250909-093708/grapher/share-of-individuals-using-the-internet.html (visited on 11/16/2025) (page 15).

[17] T. Mulla, “Assessing the factors influencing the adoption of over-the-top streaming platforms: A literature review from 2007 to 2021”, *Telematics and Informatics*, vol. 69, p. 101 797, 2022 (page 15).

[18] S. Shirvani Moghaddam, “The past, present, and future of the internet: A statistical, technical, and functional comparison of wired/wireless fixed/mobile internet”, *Electronics*, vol. 13, no. 10, p. 1986, 2024 (page 15).

[19] Z. Schillaci, “LLM adoption trends and associated risks”, in *Large Language Models in Cybersecurity: Threats, Exposure and Mitigation*, Springer Nature Switzerland Cham, 2024, pp. 121–128 (page 15).

[20] P. K. Sadhu, V. P. Yanambaka, and A. Abdelgawad, “Internet of things: Security and solutions survey”, *Sensors*, vol. 22, no. 19, p. 7433, 2022 (page 15).

[21] T. N. Theis and H.-S. P. Wong, “The end of moore’s law: A new beginning for information technology”, *Computing in science & engineering*, vol. 19, no. 2, pp. 41–50, 2017 (page 15).

[22] T. M. Chen and J.-M. Robert, “The evolution of viruses and worms”, in *Statistical methods in computer security*, CRC press, 2004, pp. 289–310 (page 15).

[23] “ANIMAL Source Code”. (), [Online]. Available: https://www.fourmilab.ch/documents/univac/animalsrc.html (page 15).

[24] E. H. Spafford, “The Internet worm program: An analysis”, *ACM SIGCOMM Computer Communication Review*, vol. 19, no. 1, pp. 17–57, 1989 (page 15).

[25] The European Parliament and the Council of the European Union. “Directive (EU) 2022/2555 on measures for a high common level of cybersecurity across the Union”. (2022), [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32022L2555 (visited on 11/16/2025) (page 15).

[26] The White House. “FACT SHEET: New Executive Order on Strengthening and Promoting Innovation in the Nation’s Cybersecurity”. (2025), [Online]. Available: https://bidenwhitehouse.archives.gov/briefing-room/statements-releases/2025/01/15/fact-sheet-new-executive-order-on-strengthening-and-promoting-innovation-in-the-nations-cybersecurity/ (visited on 11/16/2025) (page 15).

[27] La Moncloa. “Óscar López is committed to cooperation with the private sector to consolidate Spain’s leadership in cybersecurity”. (2025), [Online]. Available: https://www.lamoncloa.gob.es/lang/en/gobierno/news/Paginas/2025/20250507-cybersecurity-leadership.aspx (visited on 11/16/2025) (page 15).

[28] “IEEE SP conference”. (2025), [Online]. Available: https://www.ieee-security.org/TC/SP-Index.html (visited on 11/16/2025) (page 15).

[29] B. Mukherjee, L. T. Heberlein, and K. N. Levitt, “Network intrusion detection”, *IEEE network*, vol. 8, no. 3, pp. 26–41, 2002 (page 15).

[30] “Suricata”. (2025), [Online]. Available: https://suricata.io/ (visited on 11/16/2025) (pages 18, 45).

[31] “Snort”. (2025), [Online]. Available: https://www.snort.org/ (visited on 11/16/2025) (pages 18, 45).

[32] “Zeek”. (2025), [Online]. Available: https://zeek.org/ (visited on 11/16/2025) (pages 18, 45).

[33] L. Aubard, J. Mazel, G. Guette, and P. Chifflier, “Overlapping data in network protocols: bridging OS and NIDS reassembly gap”, in *DIMVA*, 2025 (pages 19, 49, 76).

[34] L. Aubard, J. Mazel, G. Guette, and P. Chifflier, “Overlapping IPv4, IPv6, and TCP data: exploring errors, test case context, and multiple overlaps inside network stacks and NIDSes with PYROLYSE”, in *RAID*, 2025 (pages 19, 76).

[35] J. Postel, *RFC 760: Internet Protocol*, 1980 (page 22).

[36] Google, *Google IPv6 statistics*, 2025. [Online]. Available: https://www.google.com/intl/en/ipv6/statistics.html (pages 22, 27).

[37] R. Braden, *RFC 1122: Requirements for Internet hosts-communication layers*, 1989 (page 23).

[38] J. Heffner, M. Mathis, and B. Chandler, *RFC 4963: IPv4 Reassembly Errors at High Data Rates*, 2007 (page 25).

[39] F. Gont, *RFC 6274: security assessment of the Internet Protocol version 4*, 2011 (page 25).

[40] B. Hinden and D. S. E. Deering, *RFC 2460: Internet Protocol, Version 6 (IPv6) Specification*, 1998 (pages 27, 28).

[41] J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, and M. Bailey, “Measuring IPv6 adoption”, in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 87–98 (page 27).

[42] L. Prehn, F. Lichtblau, and A. Feldmann, “When wells run dry: the 2020 IPv4 address market”, in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, 2020, pp. 46–54 (page 27).

[43] J. C. Mogul and S. E. Deering, *RFC1191: Path MTU discovery*, 1990 (page 28).

[44] J. McCann, S. Deering, and J. Mogul, *RFC 8201: Path MTU Discovery for IP version 6*, 2017 (page 28).

[45] S. Deering and R. Hinden, *RFC 1883: Internet Protocol, version 6 (IPv6) specification*, 1995 (page 28).

[46] S. Krishnan, *RFC 5722: Handling of Overlapping IPv6 Fragments*, 2009 (pages 28, 41, 90).

[47] S. Deering and R. Hinden, *RFC 8200: Internet Protocol, Version 6 (IPv6) Specification*, 2017 (pages 28, 31, 41, 43, 80, 109, 122, 155).

[48] J. Postel, *DoD Standard–Transmission Control Protocol–RFC 761*, 1980 (page 28).

[49] W. Eddy, *RFC 9293: Transmission Control Protocol (TCP)*, 2022 (pages 28–31, 121).

[50] G. Fairhurst, T. Jones, M. Tüxen, I. Rüngeler, and T. Völker, *RFC 8899: Packetization Layer Path MTU Discovery for Datagram Transports*, 2020 (page 29).

[51] J. Postel, *RFC 793: Transmission Control Protocol*, 1981 (page 31).

[52] W. John and S. Tafvelin, "Analysis of Internet backbone traffic and header anomalies observed", in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007, pp. 111–116 (pages 33, 34).

[53] "The CAIDA Anonymized Internet Traces 2012 Dataset". (), [Online]. Available: https://www-old.caida.org/data/passive/passive_2012_dataset.xml (visited on 11/16/2025) (pages 33, 34).

[54] M. Kenney. "Ping of death". (), [Online]. Available: https://insecure.org/sploits/ping-o-death.html (visited on 11/16/2025) (page 36).

[55] G. Ziemba, D. Reed, and P. Traina, *RFC1858: Security considerations for IP fragment filtering*, 1995 (page 36).

[56] A. Atlasis, "Attacking IPv6 implementation using fragmentation", 2012 (pages 36, 41, 44, 90, 110, 122).

[57] X. Zheng, C. Lu, J. Peng, *et al.*, "Poison over troubled forwarders: A cache poisoning attack targeting {DNS} forwarding devices", in *USENIX Security*, 2020 (pages 36, 82).

[58] A. Herzberg and H. Shulman, "Fragmentation considered poisonous, or: one-domain-to-rule-them-all. org", in *2013 IEEE Conference on Communications and Network Security (CNS)*, IEEE, 2013, pp. 224–232 (page 36).

[59] Z. Wang and S. Zhu, "SymTCP: eluding stateful deep packet inspection with automated discrepancy discovery", in *Network and Distributed System Security Symposium (NDSS)*, 2020 (pages 37, 42, 44, 49, 51, 56, 90).

[60] J. Novak, *Target-based fragmentation reassembly*, 2005 (pages 40, 44–47, 52, 81, 85, 86, 90, 93, 94, 96, 110, 126).

[61] U. Shankar and V. Paxson, "Active mapping: Resisting NIDS evasion without altering traffic", in *2003 Symposium on Security and Privacy, 2003.*, IEEE, 2003, pp. 44–61 (pages 39, 40, 43, 47, 78, 86, 90, 134).

[62] J. Novak and S. Sturges, *Target-based TCP stream reassembly*, 2007 (pages 41, 44–47, 52, 78, 85, 86, 90, 93, 95, 97, 110, 116, 126).

[63] G. Giacobbi. "Netcat". (2004), [Online]. Available: https://netcat.sourceforge.net/ (visited on 11/16/2025) (page 41).

[64] 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 (pages 41, 44, 90, 122, 124).

[65] R. Clayton, S. J. Murdoch, and R. N. Watson, "Ignoring the great firewall of China", *ISJLP*, vol. 3, p. 273, 2007 (page 41).

[66] J. C. Park and J. R. Crandall, "Empirical study of a national-scale distributed intrusion detection system: Backbone-level filtering of HTML responses in China", in *2010 IEEE 30th International Conference on Distributed Computing Systems*, IEEE, 2010, pp. 315–326 (page 41).

[67] D. Anderson, "Splinternet Behind the Great Firewall of China: Once China opened its door to the world, it could not close it again.", *Queue*, vol. 10, no. 11, pp. 40–49, 2012 (page 41).

[68] S. Khattak, M. Javed, P. D. Anderson, and V. Paxson, "Towards illuminating a censorship monitor's model to facilitate evasion", in *3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI 13)*, 2013 (pages 41, 44, 90, 124).

[69] M. Handley, V. Paxson, and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and {End-to-End} Protocol Semantics", in *10th USENIX Security Symposium (USENIX Security 01)*, 2001 (page 41).

[70] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. V. Krishnamurthy, "Your state is not mine: A closer look at evading stateful internet censorship", in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 114–127 (pages 42, 44, 90, 124).

[71] F. Li, A. Razaghpanah, A. M. Kakhki, *et al.*, "Lib• erate,(n) a library for exposing (traffic-classification) rules and avoiding them efficiently", in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 128–141 (pages 42, 44, 124).

[72] K. Bock, G. Hughey, X. Qiang, and D. Levin, "Geneva: Evolving censorship evasion strategies", in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2199–2214 (pages 42, 56, 90).

[73] K. Bock, G. Hughey, L.-H. Merino, *et al.*, "Come as you are: Helping unmodified clients bypass censorship with server-side evasion", in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 586–598 (pages 42, 90).

[74] K. Bock, G. Naval, K. Reese, and D. Levin, "Even Censors Have a Backup: Examining China's Double HTTPS Censorship Middleboxes", in *Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet*, 2021, pp. 1–7 (pages 42, 90).

[75] Z. Wang, S. Zhu, K. Man, *et al.*, "Themis: Ambiguity-Aware Network Intrusion Detection based on Symbolic Model Comparison", in *Proceedings of the 8th ACM Workshop on Moving Target Defense*, 2021, pp. 31–32 (pages 43, 56, 90).

[76] E. Di Paolo, E. Bassetti, and A. Spognardi, "A New Model for Testing IPv6 Fragment Handling", in *European Symposium on Research in Computer Security*, Springer, 2023, pp. 277–294 (pages 43, 81, 86, 90, 93–95, 122).

[77] I.-C. Oprea, *Lost in Reassembly: Exploiting IP Fragmentation in Computer Networks* (pages 43, 44, 47, 81, 90, 124).

[78] 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 (pages 43, 124).

[79] Z. Zhang, B. Yuan, K. Yang, D. Zou, and H. Jin, "StateDiver: Testing Deep Packet Inspection Systems with State-Discrepancy Guidance", in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 756–768 (page 44).

[80] M. Roesch *et al.*, "Snort: Lightweight intrusion detection for networks.", 1999 (page 45).

[81] V. Paxson, "Bro: a system for detecting network intruders in real-time", *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999 (page 47).

[82] "File Reassem.cc in a old Zeek/Bro version". (), [Online]. Available: https://github.com/zeek/zeek/blob/2b6ad76bd53ff69eec5ebcc47f7f5b42558163ae/src/Reassem.cc#L246 (visited on 11/16/2025) (page 47).

[83] "File Active.cc in a old Zeek/Bro version". (), [Online]. Available: https://github.com/zeek/zeek/blob/2b6ad76bd53ff69eec5ebcc47f7f5b42558163ae/src/Active.cc#L188 (visited on 11/16/2025) (page 47).

[84] "Vern Paxson response regarding reassembly policy configurability and Active Mapping functionality inside Bro/Zeek". (), [Online]. Available: https://community.zeek.org/t/fragmentation-and-tcp-overlapping-issues/1720/3 (visited on 11/16/2025) (page 47).

[85] NIST. "CVE-2021-37592". (2021), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-37592 (visited on 11/16/2025) (page 50).

[86] NIST. "CVE-2019-18792". (2019), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-18792 (visited on 11/16/2025) (page 50).

[87] NIST. "CVE-2019-18625". (2019), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-18625 (visited on 11/16/2025) (page 50).

[88] NIST. "CVE-2024-37151". (2024), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-37151 (visited on 11/16/2025) (page 50).

[89] NIST. "CVE-2024-55629". (2024), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-55629 (visited on 11/16/2025) (page 50).

[90] NIST. "CVE-2019-12175". (2021), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2019-12175 (visited on 11/16/2025) (page 51).

[91] NIST. "CVE-2023-7242". (2021), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2023-7242 (visited on 11/16/2025) (page 51).

[92] NIST. "CVE-2024-47522". (2021), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2024-47522 (visited on 11/16/2025) (page 51).

[93] "Nmap". (), [Online]. Available: https://nmap.org/ (visited on 11/16/2025) (page 52).

[94] Z. Shamsi and D. Loguinov, "Unsupervised clustering under temporal feature volatility in network stack fingerprinting", in *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, 2016, pp. 127–138 (pages 52, 134).

[95] "Gait". (), [Online]. Available: https://github.com/sandialabs/gait (visited on 11/16/2025) (page 52).

[96] "JA4TScan". (), [Online]. Available: https://github.com/FoxIO-LLC/ja4tscan (visited on 11/16/2025) (page 52).

[97] M. Zalewski. "P0f". (2014), [Online]. Available: https://lcamtuf.coredump.cx/p0f3/ (visited on 11/16/2025) (pages 52, 134).

[98] J. Holland, P. Schmitt, N. Feamster, and P. Mittal, "New directions in automated traffic analysis", in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3366–3383 (pages 52, 134).

[99] "JA4+ tool suite". (), [Online]. Available: https://github.com/FoxIO-LLC/ja4 (visited on 11/16/2025) (page 52).

[100] X. Feng, Q. Li, K. Sun, *et al.*, "PMTUD is not panacea: Revisiting IP fragmentation attacks against TCP", in *Proceedings of the Network & Distributed System Security Symposium (NDSS), San Diego, CA, USA*, 2022, pp. 24–28 (pages 53, 70, 82, 132).

[101] "Linux IP fragment lifetime." (), [Online]. Available: https://www.kernel.org/doc/html/v6.2/networking/ip-sysctl.html (visited on 11/16/2025) (page 55).

[102] C. Bormann, G. Fairhurst, D. Grossman, *et al.*, *Rfc 3819: advice for internet subnetwork designers*, 2004 (page 55).

[103] A. Custura, G. Fairhurst, and I. Learmonth, "Exploring usable Path MTU in the Internet", in *2018 Network Traffic Measurement and Analysis Conference (TMA)*, IEEE, 2018, pp. 1–8 (pages 55, 63).

[104] G. S. Reen and C. Rossow, "Dpifuzz: A differential fuzzing framework to detect dpi elusion strategies for QUIC", in *Proceedings of the 36th Annual Computer Security Applications Conference*, 2020, pp. 332–344 (page 56).

[105] Y.-H. Zou, J.-J. Bai, J. Zhou, J. Tan, C. Qin, and S.-M. Hu, "{TCP-Fuzz}: Detecting Memory and Semantic Bugs in {TCP} Stacks with Fuzzing", in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 489–502 (page 56).

[106] H. Gascon, C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck, "Pulsar: Stateful black-box fuzzing of proprietary network protocols", in *Proceedings of the 11th EAI International Conference on Security and Privacy in Communication Networks, SecureComm 2015, Dallas, TX, USA, October 26-29, 2015*, Springer, 2015, pp. 330–347 (page 56).

[107] I. Schorr, G. Combs, and J. Young. "Capinfos tool". (), [Online]. Available: https://www.wireshark.org/docs/man-pages/capinfos.html (visited on 11/16/2025) (page 58).

[108] "Pcap Analysis Library (PAL)". (), [Online]. Available: https://github.com/rusticata/pcap-analyzer (visited on 11/16/2025) (page 58).

[109] "Linux IP reassembly timeout (30 seconds)". (), [Online]. Available: https://elixir.bootlin.com/linux/v6.17.7/source/include/net/ip.h#L147 (visited on 11/16/2025) (page 58).

[110] "Service Name and Transport Protocol Port Number Registry". (), [Online]. Available: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml (visited on 11/16/2025) (page 66).

[111] G. Van Den Broek, R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "DNSSEC meets real world: dealing with unreachability caused by fragmentation", *IEEE communications magazine*, vol. 52, no. 4, pp. 154–160, 2014 (page 67).

[112] G. C. Moura, M. Müller, M. Davids, M. Wullink, and C. Hesselman, "Fragmentation, truncation, and timeouts: are large DNS messages falling to bits?", in *International Conference on Passive and Active Network Measurement*, Springer, 2021, pp. 460–477 (page 67).

[113] F. Salutari, D. Cicalese, and D. J. Rossi, "A closer look at ip-id behavior in the wild", in *Passive and Active Measurement: 19th International Conference, PAM 2018, Berlin, Germany, March 26–27, 2018, Proceedings 19*, Springer, 2018, pp. 243–254 (pages 70, 132).

[114] R. Bonica, F. Baker, G. Huston, R. Hinden, O. Troan, and F. Gont, *Rfc 8900: ip fragmentation considered fragile*, 2020 (page 73).

[115] "SparQ tool". (), [Online]. Available: https://github.com/dwolter/SparQ (visited on 11/16/2025) (pages 78, 87).

[116] M. Brandt, T. Dai, A. Klein, H. Shulman, and M. Waidner, "Domain validation++ for mitm-resilient pki", in *CCS*, 2018 (page 82).

[117] "Tcpreplay tool". (), [Online]. Available: https://tcpreplay.appneta.com/ (visited on 11/16/2025) (page 87).

[118] "Tcpdump tool". (), [Online]. Available: https://www.tcpdump.org (visited on 11/16/2025) (page 87).

[119] "Vagrant Cloud". (), [Online]. Available: https://app.vagrantup.com/boxes/search (visited on 11/16/2025) (page 88).

[120] D. Song, *Fragroute tool*, 2002. [Online]. Available: https://www.monkey.org/~dugsong/fragroute/ (page 90).

[121] *INTANG tool*, 2017. [Online]. Available: https://github.com/seclab-ucr/INTANG (page 90).

[122] *Geneva tool*, 2019. [Online]. Available: https://github.com/Kkevsterrr/geneva (page 90).

[123] *SymTCP tool*, 2020. [Online]. Available: https://github.com/seclab-ucr/SymTCP (page 90).

[124] *Themis tool*, 2021. [Online]. Available: https://github.com/seclab-ucr/Themis (page 90).

[125] *IPv6-fragmentation tool*, 2023. [Online]. Available: https://github.com/netsecuritylab/ipv6-fragmentation (page 90).

[126] *Ipv6-fragmentation tool*, 2023. [Online]. Available: https://github.com/linbin89/FragEva6-Guard (page 90).

[127] *Oprea's tool*, 2025. [Online]. Available: https://github.com/OpreaCristian2002/reassembly-policies-5g-using-ipv6 (page 90).

[128] A. Bluhm, *OpenBSD patch related to pf's IP reassembly error.* [Online]. Available: https://github.com/openbsd/src/commit/9915416fe885bd67bf19209c5fe2f9bbbb7191ab (pages 113, 151).

[129] A. Bluhm, *OpenBSD patch related to pf's TCP reassembly error.* [Online]. Available: https://github.com/openbsd/src/commit/12e4c257ea8089c7f56eb0c2c4493a0458273f4f (pages 114, 151).

[130] *The try_to_coalesce() function source code.* [Online]. Available: https://elixir.bootlin.com/linux/v6.1/source/net/ipv4/tcp_input.c#L4653 (pages 117, 119).

[131] D. Reed, P. S. Traina, and P. Ziemba, *RFC 1858: Security Considerations for IP Fragment Filtering*, 1995 (page 124).

[132] I. Miller, *RFC 3128: Protection Against a Variant of the Tiny Fragment Attack (RFC 1858)*, 2001 (page 124).

[133] J. Postel, *RFC 768: User datagram protocol*, 1980.

[134] J. Postel, *RFC 792: Internet Control Message Protocol*, 1981.

[135] M. Gupta and A. Conta, *RFC 4443: Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, 2006.

[136] S. McCann and M. A. West, *RFC 4413: TCP/IP Field Behavior*, 2006.

[137] J. Iyengar and M. Thomson, *RFC 9000: QUIC: A UDP-Based Multiplexed and Secure Transport*, 2021.

[138] G. Fairhurst, T. Jones, M. Tüxen, I. Ruengeler, and T. Völker, *RFC 8899: Packetization Layer Path MTU Discovery for Datagram Transports*, 2020.

[139] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, *RFC2883: An extension to the selective acknowledgement (SACK) option for TCP*, 2000.

[140] G. Taleck, "Ambiguity resolution via passive OS fingerprinting", in *International Workshop on Recent Advances in Intrusion Detection*, Springer, 2003, pp. 192–206.

[141] Z. Shamsi, A. Nandwani, D. Leonard, and D. Loguinov, "Hershel: Single-Packet OS Fingerprinting", *SIGMETRICS Perform. Eval. Rev.*, vol. 42, no. 1, pp. 195–206, Jun. 2014, ISSN: 0163-5999. DOI: 10.1145/2637364.2591972. [Online]. Available: https://doi.org/10.1145/2637364.2591972.

[142] S. Bai, H. Kim, and J. Rexford, "Passive OS fingerprinting on commodity switches", in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, IEEE, 2022, pp. 264–268.

[143] W. Li, D. Zhang, G. Xie, and J. Yang, "TCP and ICMP in network measurement: An experimental evaluation", in *International Symposium on Parallel and Distributed Processing and Applications*, Springer, 2005, pp. 870–881.

[144] T. Beardsley, “Snacktime: A perl solution for remote OS fingerprinting”, *Plan B Security, Austin, TX, Jun*, 2003.

[145] S. Zhu, S. Li, Z. Wang, *et al.*, “You do (not) belong here: detecting DPI evasion attacks with context learning”, in *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*, 2020, pp. 183–197.

[146] D. Nobori and Y. Shinjo, “{VPN} Gate: A {Volunteer-Organized} Public {VPN} Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls”, in *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 229–241.

[147] E. Kenneally and D. Dittrich, “The Menlo Report: Ethical principles guiding information and communication technology research”, *Available at SSRN 2445102*, 2012.

[148] NIST. “CVE-2018-5391”. (2018), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2018-5391 (visited on 11/16/2025).

[149] P. Dikshit, M. Kosek, N. Faulhaber, J. Sengupta, and V. Bajpai, “Evaluating DNS Resiliency and Responsiveness with Truncation, Fragmentation & DoTCP Fallback”, *IEEE Transactions on Network and Service Management*, 2024.

[150] “Meek pluggable transport.” (), [Online]. Available: https://support.torproject.org/glossary/meek/ (visited on 11/16/2025).

[151] “Obfs4 pluggable transport.” (), [Online]. Available: https://support.torproject.org/glossary/obfs4/ (visited on 11/16/2025).

[152] “Suricata reassembly policies.” (), [Online]. Available: https://docs.suricata.io/en/suricata-7.0.4/configuration/suricata-yaml.html#host-os-policy (visited on 11/16/2025).

[153] “Snort IP reassembly policies.” (), [Online]. Available: https://www.snort.org/faq/readme-frag3 (visited on 11/16/2025).

[154] “Snort TCP reassembly policies.” (), [Online]. Available: https://www.snort.org/faq/readme-stream5 (visited on 11/16/2025).

[155] “Snort flow reassembly option.” (), [Online]. Available: https://docs.snort.org/rules/options/non_payload/flow (visited on 11/16/2025).

[156] “Suricata flow reassembly option.” (), [Online]. Available: https://docs.suricata.io/en/suricata-7.0.4/rules/flow-keywords.html (visited on 11/16/2025).

[157] “Suricata advice regarding NIC offloading.” (), [Online]. Available: https://docs.suricata.io/en/suricata-7.0.4/performance/packet-capture.html#offloading (visited on 11/16/2025).

[158] “Zeek advice regarding NIC offloading.” (), [Online]. Available: https://docs.zeek.org/en/v6.2.0/cluster-setup.html#offloading-and-ethtool-tuning (visited on 11/16/2025).

[159] Y. Gilad and A. Herzberg, “Fragmentation considered vulnerable: blindly intercepting and discarding fragments”, in *5th USENIX Workshop on Offensive Technologies (WOOT 11)*, 2011.

[160] A. Dunkels. “Lwip”. (2014), [Online]. Available: https://www.nongnu.org/lwip/2_1_x/index.html.

[161] NIST. “CVE-2021-37592”. (2021), [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2021-37592 (visited on 11/16/2025).

[162] J. C. Bennett, C. Partridge, and N. Shectman, “Packet reordering is not pathological network behavior”, *IEEE/ACM Transactions on networking*, vol. 7, no. 6, pp. 789–798, 1999.

[163] F. Hilal and O. Gasser, “Yarrpbox: detecting middleboxes at internet-scale”, *Proceedings of the ACM on Networking*, vol. 1, no. CoNEXT1, pp. 1–23, 2023.

[164] J. P. Anderson, “Computer security threat monitoring and surveillance”, *Technical Report, James P. Anderson Company*, 1980.

[165] C. Stoll, *The cuckoo’s egg: tracking a spy through the maze of computer espionage*. Doubleday, 1989.

[166] F. Zhu, P. Xu, and J. Zong, “Moore’s Law: The potential, limits, and breakthroughs”, *Applied and Computational Engineering*, vol. 10, no. 1, pp. 307–315, 2023.

Appendices

A Tested implementation details

The Table 16 extends the information given in Section 4.2 regarding the tested protocol stacks.

B Detailed responsible disclosure process

Table 14 gives details regarding the responsible disclosure process.

Disclosure type	Date	Recipient	Description	Comments
Reassembly errors/ bugs	06/11/2023	Suricata	IP reassembly logic error	Partially fixed, credited with CVE [13]
	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 [128]
	28/01/2025	OpenBSD	TCP connection termination error	Fixed [129]
	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	Snort/Zeek/ Suricata	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 [13]
	26/08/2024	Snort/Zeek/ Suricata	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	Snort/Zeek/ Suricata	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	Snort/Zeek/ Suricata	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

Table 14: Responsible disclosure summary. Provided artifacts are composed of a technical report and the related PCAP or log materials.

Implementation				Addressed protocol	Comments
Type	**Name**	**Version**	**Additional version info**		
OS	Windows	Server 2019/ 10/11	<2022/10/25 build	IPv4/IPv6/TCP	Windows' firewall is enabled by default, and we did not observe any reassembly difference if disabled.
			≥2022/10/25 build		
	Debian	8	Linux 3.16.0-9	IPv4/IPv6/TCP	-
		9	Linux 4.9.0-12		
		12	Linux 6.1.0-37		
	FreeBSD	10.2	RELEASE-p8	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.
		11.2	RELEASE-p10		
		11.3	RELEASE-p9		
		12.1	RELEASE-p10		
		12.2	RELEASE-p6		
		14.1	RELEASE		
	OpenBSD	6.0	GENERIC.MP#2319	IPv4/IPv6/TCP	OpenBSD's firewall is enabled by default, and we oberve IPv4 and IPv6 reassembly difference if disabled. Note that we found the related reassembly errors (Section 4.4.2) with OpenBSD version 7.6 GENERIC#332 and that the tested OpenBSD 7.7 has fixed the IPv4 and TCP errors.
		6.1	GENERIC.MP#8		
		7.7	GENERIC.MP#625		
	NetBSD	7.0	GENERIC. 201509250726Z	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	GENERIC#0		
		9.3	GENERIC#0		
	Solaris	11.2 → 11.4	SunOS 5.11	IPv4/IPv6/TCP	Solaris' firewall is disabled by default, and we oberve IPv4 and IPv6 reassembly difference if enabled.
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 4.4.1 and 4.4.3 report NIDS reassemblies with the *default* policies. Sections 4.3 and 4.4.2 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.

Table 16: Protocol implementation target descriptions. The OS reassemblies provided in the paper are those with the firewall *default* state. If any reassembly is different when the firewall is enabled/disabled, they are documented in [11].

C Custom reassembly algorithm test

This appendix section gives details regarding the custom reassembly algorithm test. First, it describes the pipeline. Second, the section reports the mechanisms with a reassembly side effect we observed in the wild. Finally, it lists the implemented algorithms across protocols.

C.1 The pipeline

Figure 19 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.

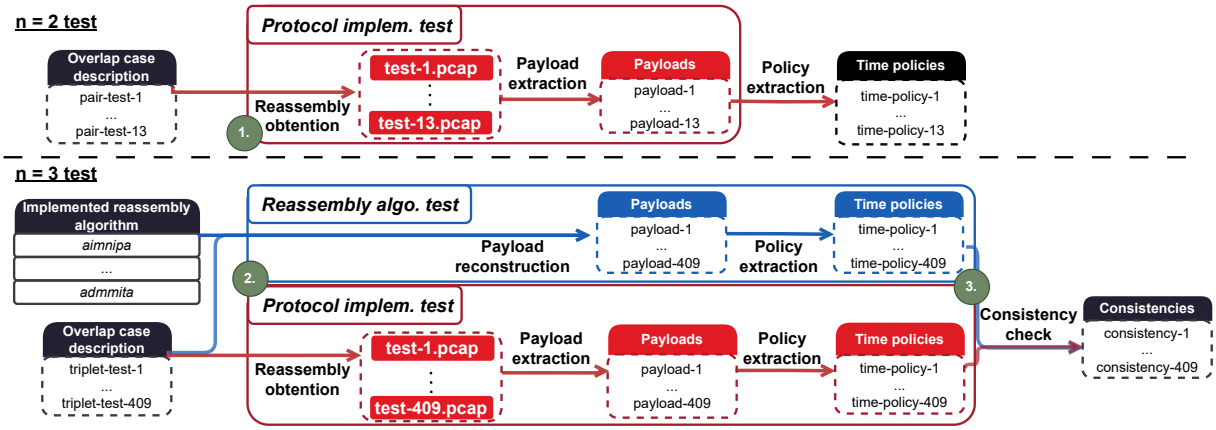


Figure 19: The pipeline for the reassembly algorithm test. This process is repeated for every (protocol, scenario, implementation) tuple. Black items are input or output elements.

Let's take the $s^{sf,ef}$ and Debian 12 OS as an example for the IPv4 protocol. First (cf ① on Figure 19), we run Debian 12 test in order to retrieve the $n = 2$ time policies. We get 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 19), 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 19), this finally enable us to get the number of test

case reassembly inconsistencies for every implemented algorithm by comparing their reassemblies with Debian 12.

C.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 4.1.

C.2.1 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 4.4.3.2. 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.

C.2.2 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.2.3 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 [47] 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 therefore, 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.

C.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 C.2.

C.3.1 IP protocols

We implemented the 12 reassembly algorithms reported in Table 17.

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.

C.3.2 TCP protocol

We implemented the six reassembly algorithms reported in Table 18.

Table 17: 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 18: 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

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.

Table of Figures

1 Data overlap ambiguity illustration (french summary) 6
2 Data overlap ambiguity illustration. 16

1.1 The IPv4 header 23
1.2 IPv4 fragmentation illustration 24
1.3 IPv4 fragmentation with loss 26
1.4 IPv4 fragmentation with overlap 26
1.5 The TCP header 29
1.6 TCP segmentation illustration 30
1.7 Fragment series types 33
1.8 The threat model for insertion and evasion attacks 37
1.9 Shankar and Paxson model 40
1.10 Novak model addition 40

2.1 The threat model considered in the thesis 52
2.2 IP data buffer re-synchronization illustration. 54
2.3 TCP data buffer re-synchronization illustration. 55

3.1 Evolution of IPv4 fragmentation in MAWI. 60
3.2 Series type distribution 61
3.3 Left, i.e., with bit `MF` unset, fragment sizes. 63
3.4 Original (guessed) datagram size across fragment series. 64
3.5 Next IP protocol distribution (bytes). 65
3.6 Top 20 observed UDP services in the MAWI traces 66
3.7 Number of source hosts and three-tuples. 68
3.8 Series filtering and hosts' RFC 6864 compliance assessment 69
3.9 Source host MDL estimation. 71

4.1 The (O, Si, Mi) triplet of chunks. 78
4.2 The (Fi, Eq, S, F, D, S) quadruplet of chunks. 78
4.3 The checksum-impactless patterns 82
4.4 A triplet populated with the introduced patterns 84

4.5 Testing mode illustration for the (O) test case and the s^{sf} TCP scenario. 85
4.6 PYROLYSE testing pipeline. 86
4.7 IPv4 reassembly similarities across tested implementations 100
4.8 IPv6 reassembly similarities across tested implementations 101
4.9 TCP reassembly similarities across tested implementations 102
4.10 Reassembly similarities between scenarii across implementations 105
4.11 IPv4 reassembly similarity between pairs of full test case scenarii 106
4.12 IPv6 reassembly similarity between pairs of full test case scenarii 108
4.13 The (F, F, F) test case representation. 108
4.14 TCP reassembly similarity between pairs of full test case scenarii 110
4.15 The (Oi, O, D) test case representation in a s^{ep} testing context. 112
4.16 The (B) test case representation in a s^{c} testing context. 113
4.17 The (O, M, O) test case representation in a s^{sf} testing context. 116
4.18 The (M, M, S) test case representation in a s^{sf} testing context. 117
19 The pipeline for the reassembly algorithm test 154

COLLEGE DOCTORAL BRETAGNE
MATHS, TELECOMS
INFORMATIQUE, SIGNAL
SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Recouvrements de données dans les protocoles IPv4, IPv6 et TCP : exploration des réassemblages de divers piles réseaux et supervision réseau

Mots clés : Détection d'intrusion, évasion, IP, TCP, recouvrement de données

Résumé : La fragmentation IP et la segmentation TCP permettent de diviser des paquets réseaux trop volumineux en morceaux plus petits. Ce découpage peut donner lieu à du recouvrement, c'est-à-dire que plusieurs morceaux ainsi créés peuvent se chevaucher, de manière complète ou partielle, avec des données non nécessairement identiques. Les politiques de réassemblage, c'est-à-dire le morceau de données préféré en fonction du type de recouvrement, diffèrent selon les implémentations IPv4, IPv6 et TCP. Dès lors, un système de détection d'intrusion réseau (NIDS) qui ne ré-assemble pas les recouvrements de la même manière que l'hôte surveillé est aveugle au flux réellement traité par cet hôte, laissant la place à son contournement.

L'objectif principal de cette thèse est d'évaluer dans quelle mesure les NIDS sont vulnérables à des attaques basées sur les recouvrements IPv4, IPv6 et TCP. Nous proposons tout d'abord une nouvelle méthode pour modéliser les recouvrements de fragments et de segments afin de garantir la complétude des tests. Nous instancions ce modèle dans notre outil PYROLYSE et testons les politiques de réassemblage de différents types de piles IP et TCP. Nous avons découvert que les politiques sont beaucoup plus diverses et complexes que décrites dans l'état de l'art et que les NIDS Suricata, Snort et Zeek présentent des incohérences de réassemblage avec ces piles, ce qui les rend vulnérables aux attaques par recouvrement. Nous avons également trouvé des erreurs de réassemblage dans cinq piles, dont une CVE.

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

Keywords: Intrusion detection, evasion, IP, TCP, data overlap

Abstract: IP fragmentation and TCP segmentation enable the splitting of large data packets into smaller ones. These mechanisms permit full or partial overlaps with different data on the overlapping portions. Reassembly policies, i.e., the data chunk preferences that depend on the overlap types, differ across IPv4, IPv6, and TCP implementations. This can lead to vulnerabilities, as Network Intrusion Detection Systems (NIDSes) may interpret the packet differently from the monitored hosts.

The main goal of this thesis is to evaluate to what extent NIDSes are vulnerable to overlap-based attacks. We first propose a new way to model the fragment and segment overlaps to ensure complete testing. We instantiate this model in our tool PYROLYSE and test the reassembly policy of various IP and TCP stack types. We discovered that the policies are much more diverse and complex that described by related works. Suricata, Snort and Zeek NIDSes exhibit reassembly inconsistencies, which make them vulnerable to overlap-based attacks. We also found reassembly errors in five stacks, including one CVE.