Attacks on Dynamic Protocol Detection of Open Source Network Security Monitoring Tools

Jan Grashöfer
Karlsruhe Institute of Technology
Institute of Telematics
jan.grashoefer@kit.edu

Christian Titze
Karlsruhe Institute of Technology
Institute of Telematics
christian.titze@alumni.kit.edu

Hannes Hartenstein
Karlsruhe Institute of Technology
Institute of Telematics
hannes.hartenstein@kit.edu

Abstract

Protocol detection is the process of determining the application layer protocol in the context of network security monitoring, which requires a timely and precise decision to enable protocol-specific deep packet inspection. This task has proven to be complex, as isolated characteristics like port numbers are not sufficient to reliably determine the application layer protocol. Hence, more dynamic detection approaches have been developed. In this paper, we analyze the Dynamic Protocol Detection mechanisms employed by popular and widespread open-source network monitoring tools. We show on the example of HTTP that all analyzed detection mechanisms are vulnerable to evasion attacks, which pose a serious threat to real-world monitoring operations. We find that the underlying fundamental problem of protocol disambiguation is not adequately addressed in two of three monitoring systems that we analyzed. To enable adequate operational decisions, this paper highlights the inherent trade-offs within Dynamic Protocol Detection.

1 Introduction

Common Network Security Monitoring and Intrusion Detection Systems make use of Deep Packet Inspection (DPI) techniques to allow application layer specific analysis of the monitored traffic. Once the application layer protocol is identified, these systems attach appropriate decoders to extract detailed meta data about or contents of the communication for further analysis. While the primary focus of a Network Security Monitoring (NSM) system is to provide as detailed information about the observed traffic as possible, access to the high-level semantics of the traffic is extremely valuable for Intrusion Detection Systems as well. For example, in case of signature-based intrusion detection, the additional information can be used to improve the accuracy of signatures to reduce false positives. In the following, we will refer to NSM systems in the sense of a superclass, which includes network Intrusion Detection and Prevention Systems (IDS/IPS).

Determining the correct application layer protocol decoder for a connection based on port numbers has proven insufficient. On the one hand, there are arbitrary deviations from using standardized ports for multiple reasons ranging from web interfaces operated on peculiar ports to users or applications actively trying to bypass port-based restrictions. On the other hand, there are protocols that use unpredictable ports by design, because ports are automatically negotiated. Popular examples are FTP and SIP. Hence, the concept of Dynamic Protocol Detection (DPD) has evolved, which denotes a flexible approach that takes the actual content of a connection into account to determine the protocol in use, i.e. to perform protocol disambiguation. DPD has been introduced as a key element to virtually almost all modern NSM systems, because failing to detect the protocol in use prevents the appropriate decoding of traffic and hence spoils the intended visibility.

In this paper, to attack the monitoring, we revisit DPD and transfer general evasion strategies to DPD. By analyzing state of the art DPD implementations, we deduce two attack techniques, *Deferred Start* and *Misleading Start*, which exploit the underlying problem of protocol disambiguation. Applying the DPD attack techniques for text-based protocols on the example of HTTP, we implement three practical attack realizations. Although most of the traffic is encrypted nowadays, HTTP still represents a highly relevant use case: Apart from scenarios in which the lack of encryption constitutes the reason for careful monitoring, the presented attacks would also affect monitoring operations that deliberately decrypt HTTPS traffic for monitoring purposes [1], [2]. We show that the attacks pose a threat to real-world deployments by an evaluation of the behavior of popular web servers, such as nginx, when facing our attack traffic. We find that the web server behavior exploited for the proposed evasion is quite common in the top 500 websites. Based on the outlined attacks, we point out the inherent trade-offs within DPD and discuss approaches to address them. Given the fundamental nature of the underlying problem, we intend to raise the awareness for the need to carefully balance these trade-offs.

This paper is structured as follows: First, we present related work in 2. In 3 we define the attack scenario and introduce our attack strategy. Then we analyze the DPD mechanisms employed by two popular and widespread open-source NSM tools, namely Bro[1] and Snort in 4. Building upon our insights, in Section 5, we introduce two general DPD attack techniques that are tailored to our attack scenario and construct different types of attack traffic to conduct evasion attacks. In 6 we show that both analyzed NSM systems as well as a third, popular IDS, Suricata, are vulnerable to these evasion attacks. In this context, we also discovered a Denial of Service (DoS) attack against Bro that has since been reported and fixed. Furthermore, we evaluate the real-world applicability and impact of the presented attacks. Considering our findings, we discuss the challenges of DPD in 7. Finally, we conclude our paper including an outlook on future work in 8.

2 Related Work

Attacks on network monitoring in general are a well-known problem [3]–[5]. Already in 1998, Ptacek and Newsham [3] introduced a classification of monitoring attacks, described corresponding attacks on IP as well as TCP level and evaluated monitoring software against these attacks. Ptacek *et al.* showed that all tested systems were vulnerable and came to the conclusion that substantial efforts have to be undertaken to address the shortcomings they discovered. Nevertheless, more than a decade later Cheng *et al.* [5] still found popular NSM software vulnerable to well-known attacks. Roelker [6] described techniques to evade detection in context of HTTP focusing on exotic encoding to prevent a comprehensive analysis. Despite previous work in the NSM domain, our work demonstrates that already gathered insights have not been transferred thoroughly to DPD, which leaves state of the art monitoring software vulnerable to attacks.

Traffic classification is a field related to DPD. Numerous approaches have been suggested to infer the type of traffic [7] or the application in use [8]. Machine learning techniques have even been used to classify encrypted traffic [9]. However, there is a fundamental difference between traffic classification and protocol detection for NSM: Protocol detection requires a timely and precise decision to enable protocol-specific DPI, whereas traffic classification allows for a much higher degree of fuzziness, as it usually aims at providing a rather coarse-grained overview of traffic composition.

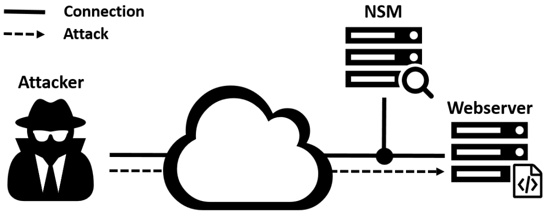

Figure 1: Attack scenario: The attacker aims at attacking a monitored web server, while hiding the attack traffic from the NSM.

To the best of our knowledge, the first scientific work on DPD in context of NSM is the work of Dreger *et al.* [10] that was published in 2006. Dreger *et al.* present a tree-based approach for dealing with ambiguities in the detection process. The authors carve out the fundamental trade-offs in protocol detection and foresee attacks on the monitor in general. In contrast, we will present attacks on the protocol detection mechanism itself. We believe that Dyer *et al.* [11] were the first to systematically investigate attacks on DPD in context of DPI. However, their threat model is substantially different from the established thread model for NSM: They assume that both endpoints are controlled by the attacker. To circumvent censorship, they tunnel traffic by mimicking benign protocols. Our work employs the established threat model for NSM [10], [12]: We assume that only one of the monitored connection endpoints is controlled by the attacker. This in turn requires attacks on the monitor to be significantly more advanced. Nevertheless, we show that DPD mechanisms can be exploited to spoil the visibility defenders seek to gain by deploying NSM systems.

3 Threat Model & Attack Strategy

In this section, we define the attack scenario that constitutes our threat model (3.1) and present our attack strategy (3.2).

3.1 Attack Scenario

In this work, we focus on text-based protocols using the example of HTTP. The attack scenario is depicted in Figure 1: The objective of the attacker is to execute an attack on a web server without being accurately observed by the NSM that monitors the traffic of the web server. The fundamental assumption in the field of NSM is that only a single connection endpoint is controlled by the attacker. If both endpoints were under the control of an attacker, the attacker would be able to establish arbitrary covert channels, which in turn are impossible to analyze by an observer [12]. In our scenario the attacker is able to send arbitrary traffic to the web server, but has no additional means

[1] In October 2018 Bro was renamed Zeek. As our analysis focused on Bro version 2.5.1, we use the old name for the remainder of this paper.

to influence the server's behavior, i.e. the web server does not cooperate in hiding the attack traffic. As the NSM receives a full copy of the traffic, the overall attack is twofold: In addition to the primary objective of attacking the web server, the attacker also needs to attack the monitor. Therefor the attacker is required to craft the attack traffic so that it is processed by the web server but evades the NSM.

3.2 Attack Strategy

Our strategy to evade monitoring focuses on the detection of text-based application layer protocols like HTTP, FTP or SMTP. If we are able to prevent the NSM to correctly detect the protocol in use, we take away the ability to adequately analyze the observed traffic and thus spoil the visibility that operators of the NSM seek to gain.

While text-based protocols allow for easy analysis by a human observer, parsing these kinds of protocols often represents a difficult task. Whereas binary protocols encode the length of Application Data Units (ADUs), either in the protocol's definition or in form of a length field, text-based protocol parsers need to accumulate the stream of characters until the end of an instruction, i.e. a line ending, is identified. The accumulated character string can then be parsed and interpreted according to the protocol. Unfortunately, it is common for text-based protocols not to limit the total length of ADUs, which theoretically requires unlimited buffers for accumulating a single ADU. However, a NSM has to make a timely decision on the protocol. The less information is available to decide on the protocol in use, the likelier a decision cannot be definitive due to ambiguities.

Application Data Units of text-based protocols are typically represented as distinct lines. Lines are usually separated using either a single linefeed symbol (LF) or a combination of carriage-return (CR) and linefeed [13]. To evade the monitor, we will exploit this general structure of text-based protocols in the context of DPD. Based on our analysis of protocol detection mechanisms employed in popular NSM systems in the following section, we will deduce two attack techniques, *Deferred Start* and *Misleading Start*. These techniques will allow us to artificially delay the point at which a proper decision can be made or deliberately mislead the monitor to make a wrong decision.

Our example focuses on HTTP, due to its prevalence in the modern Internet. While all of the analyzed DPD mechanisms can be extended to work on HTTP/2 in its unencrypted form[2], only one of the investigated monitoring applications comes with an HTTP/2 analyzer. Thus, we narrowed our analysis down to HTTP/1.1 and below. Nevertheless, the techniques we apply can also be used for other protocols.

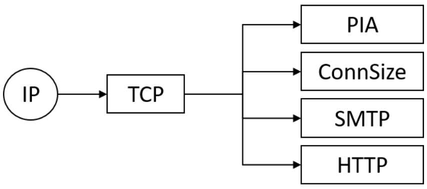

Figure 2: Exemplary analyzer tree for an IP-based TCP connection in Bro. Branches indicate parallel execution of analyzers.

4 Protocol Detection Mechanisms

In this section we will describe the Dynamic Protocol Detection mechanisms of two different open-source NSM systems, namely Bro [15] and Snort 3 [16]. To detect the protocol in use independently of the port, both systems apply protocol-specific signatures. Yet, we will see that the underlying architectures significantly differ. Note that we omit in this section a detailed analysis of Suricata due to its similarity to Snort. Furthermore, libraries like nDPI [17] and libprotoident [18] do not implement mechanisms to enable DPI based on their protocol classification. As our work focuses on the inherent trade-offs of such mechanisms, these libraries are out of scope.

4.1 Bro

The Bro NSM serves as a flexible platform for the analysis of network traffic. Incoming packets are processed by an *event engine* that utilizes protocol analyzers to parse the traffic and generate a high-level event stream.

To detect which protocols are used in the observed network traffic, Bro uses the DPD framework introduced in 2006 by Dreger *et al.* [10]. It orchestrates inspection of a connection by composing a pipeline of protocol analyzers to which the given stream of traffic is fed. Analyzers can be chained to account for protocols nested into each other. As it is not always clear which analyzer has to be attached, a protocol analyzer may have multiple child analyzers. This results in each connection managing its analyzers in the form of a tree as shown in Figure 2. If an analyzer signals a serious inconsistency with its protocol, the analyzer is removed from the tree. Following this trial-and-error approach, possible ambiguities in the protocol detection can be solved eventually.

Figure 2 shows an exemplary analyzer tree for a TCP connection. In this case the *TCP Analyzer* is set as root analyzer and takes care of reassembling the stream. By default, the *Connection Size Analyzer (ConnSize)*, which keeps track of the connection's statistics like bytes sent and received, and the so called *Protocol Identification Analyzer (PIA)* are added as child analyzers. Furthermore, the initial analyzer tree is built based on well-known ports (e.g. 80 for HTTP). Besides well-known ports, each protocol analyzer might specify

[2] Note that the preface of a HTTP/2 connection is text-based. [14]

```
signature dpd_http_client {
  ip-proto == tcp
  payload /^[[:space:]]*(OPTIONS|GET|HEAD|
                         POST|...)[[:space:]]*/
  tcp-state originator
}

signature dpd_http_server {
  ip-proto == tcp
  payload /^HTTP\/[0-9]/
  tcp-state responder
  requires-reverse-signature dpd_http_client
  enable "http"
}
```

Figure 3: Excerpt from Bro’s signature file that defines a pair of patterns to detect the HTTP protocol.

```
http_methods = { ’GET’, ’HEAD’, ’POST’, ... }

default_wizard = {
  spells = {
    { service = ’http’, proto = ’tcp’,
      client_first = true,
      to_server = http_methods,
      to_client = { ’HTTP/’ } },

    { service = ’smtp’, proto = ’tcp’,
      client_first = true,
      to_server = { ’HELO’, ’EHLO’ },
      to_client = { ’220*SMTP’, ’220*MAIL’ } }, ...
  }, ...
}
```

Figure 4: Excerpt from `snort_defaults.lua` (default configuration) that defines patterns called spells to detect text-based protocols.

signatures to get triggered. For example, the SMTP analyzer in Figure 2 might have been attached due to a signature match. Figure 3 shows a pair of signatures for HTTP: The client signature specifies a set of common HTTP methods and the server signature matches the generic beginning of an HTTP response. The HTTP analyzer is triggered by the server signature, which requires the client signature to have matched by specifying the `requires-reverse-signature` condition. The PIA is responsible for matching the DPD signatures and attaching new analyzers to the tree. To allow the activation of additional analyzers in the course of a connection, the PIA buffers the beginning of the connection. The buffer size is configurable and by default set to 1024 bytes. If an additional analyzer is added to the tree, the buffer is replayed to that analyzer. Once the buffer is filled, the dynamic detection process is stopped.

4.2 Snort 3

Snort is likely today’s most popular open source IDS and IPS solution and mainly focuses on matching patterns in network traffic. The patterns are defined in the form of rather low-level rules, matching on the binary stream of network traffic. Snort 3 allows the restriction of rules to certain application layer protocols like HTTP, whereas previous versions only supported TCP, UDP, ICMP and IP [19]. In August 2018, a beta version of Snort 3 was released [20] but it has not been declared production ready, yet. Nevertheless, in context of this paper, we will consider Snort in version 3.

In Snort 3 application layer analysis is realized by so called service inspectors. A service inspector consists of a stream splitter that preprocesses the data stream and splits it into protocol-specific ADUs, e.g. lines in case of HTTP. The ADUs are subsequently processed by the actual service inspector, which also manages the protocol state.

DPD is implemented by a special service inspector. To identify text-based protocols, patterns called spells are matched. Spells are realized as acceptors, i.e. finite state machines that accept a regular language. Figure 4 shows the definition of the patterns for HTTP and SMTP. Alongside the service identifier (`service`), which is used for assigning the corresponding inspector, each spell defines the patterns expected in a first message to the server (`to_server`) and in its reply (`to_client`). While the development notes and code architecture suggest that the final decision on the application layer protocol should be on the service inspector that is triggered, as of writing this paper, Snort 3 relies on the DPD done by the wizard inspector. Once the wizard attached a service inspector, the classification of the connection is fixed.

Summing up, we find three important facets of DPD: First, traffic has to be buffered until a decision on the protocol is made. Second, signatures are used to infer the protocol in use and third, the applied heuristics might not be definite so that the resulting ambiguities need to be handled. Considering a third NSM system for our evaluation in Section 6, we will see that Bro is the only system that actively deals with ambiguities by implementing an analyzer tree.

5 DPD Attack Techniques

In this section we present two techniques to conduct novel insertion attacks on DPD in context of text-based protocols (5.1 & 5.2) and develop three practical realizations of these techniques (5.3). Based on the practical realizations, we will evaluate the DPD mechanisms implemented in popular open source NSM systems in Section 6.

5.1 Deferred Start

A *Deferred Start* attack is based on the observation that NSM systems are forced to focus on the beginning of a connection to identify the protocol. The basic idea is to defer the actual start of the connection by flooding the monitor with useless data, causing the protocol detection to fail.

Considering that every byte of the connection is relevant for analysis, the monitored traffic has to be buffered for analysis until the protocol is finally determined. This requirement is further aggravated by the fact that the beginning of a connection usually contains information of particular interest: For example, request and response headers in the beginning of an HTTP connection provide meaningful meta data. As extensive buffering would pose a Denial-of-Service vector, the buffer size has to be limited, which requires the DPD mechanism to come to a timely decision.

In general there are two possible approaches to deal with the need to buffer traffic in the light of delayed protocol determination: First, one could use a reasonable sized **fixed buffer**. The size has to be chosen large enough to make sure that the protocol can be determined but as small as possible to save valuable resources. If the protocol cannot be detected based on the buffered traffic, the DPD mechanism would give up and report a detection failure. Considering an attacker who might be able to flood that buffer this approach seems suboptimal, but in fact, this technique implements a trade-off: For a protocol that is unknown to the monitoring software, the DPD mechanism will never be able to detect the correct protocol. In this case, continuously trying to detect the protocol would again waste valuable resources. Thus a fixed buffer implicitly realizes a protocol detection timeout.

Second, a **ring buffer** could be used. In contrast to a fixed buffer, a ring buffer of capacity n always provides access to the last n bytes of the connection under consideration. This sliding window approach would be able to mitigate a deferred start as described above. In theory, a ring buffer also allows ongoing protocol detection as there is no implicit limit that would stop the process. Although continuous detection might be undesirable, due to binding resources, the ring buffer approach is more flexible as it allows to decouple the buffering of connection data and the protocol detection timeout.

5.2 Misleading Start

A *Misleading Start* attack exploits the focus on the connection start as well. To detect a protocol on a non-standard port, signatures are employed for identification of known protocols. For example, a signature for HTTP might match version strings like `HTTP/1.1` to trigger an HTTP-related analysis. Signatures can be further divided into unidirectional and bidirectional signatures. Whereas unidirectional signatures match on flows independently, i.e. communication from client to server as well as from server to client, bidirectional signatures match on a combination of patterns across both directions.

When using signatures to detect protocols, there is an inherent trade-off with respect to the strictness of the signatures. If a signature is too strict, it will fail to correctly flag all connections that use a given protocol. If a signature is too loose, ambiguities will arise that have to be resolved to come to a final decision. Both situations might be exploited by an attacker: Strict signatures can be evaded by leveraging edge-cases that are accepted by liberal endpoint implementations but not covered by the signature. Loose signatures can be misused to cause additional load on the monitor either by tricking the software into a resource-intensive inspection of actually non-conforming traffic, or by complicating the process of solving intentionally induced ambiguities.

For example, an HTTP request starts by specifying the method. The HTTP method has to be assumed as arbitrary string given the protocol's extensibility [21]. Hence, after analyzing the first token of a line there can be multiple options for the underlying protocol.

5.3 Practical Realization

Based on the previously presented DPD attack techniques, we have deduced three approaches to realize practical attacks:

CRLF Stuffing To realize a *Deferred Start*, we prepend a valid HTTP request with whitespace characters to deliberately delay the protocol detection. We have chosen the combination of CR and LF characters as they serve as line delimiter for HTTP and thus also occur in valid requests [22].

Unknown Method The simplest possible approach to realize a *Misleading Start* attack is to use an HTTP method unknown to the monitoring software. We implement this approach by sending a request that uses the string `UNKNOWNMETHOD` as method.

HELO Method A more advanced option for a *Misleading Start* is to exploit potential ambiguities that are for example caused by overlapping signatures. We implement this approach by using the string `HELO` as method, which is also the first command in an SMTP session.

The last two attacks make use of methods that are not supported by the web server to avoid detection by the NSM system. We added a `Connection: keep-alive` to our first, manipulated request and send a second, valid request after the first one. By reusing the same connection we intend to hide the valid, potentially malicious request from the monitor. The attack is successful if the second request is not detected by the NSM system but correctly processed by the web server.

6 Evaluation

In this section, we assess the real-world consequences of the developed attacks with respect to the effect on the monitoring software (6.1). For our tests, we use Bro, Snort and Suricata, a third popular open-source NSM system. Furthermore, we evaluate the susceptibility of web servers to determine the effectiveness of our attacks under real-world conditions (6.2). Finally we will consider the combined effectiveness (6.3), as our threat-model requires a twofold attack.

6.1 Attacking the Monitor

Our primary interest is the detection behavior for traffic on non-standard ports, as we focus on the dynamic detection of protocols. Nevertheless, the generated traffic might also impact protocol detection on standard ports. Hence we conducted our experiments utilizing both a standard port for HTTP (80) as well as a non-standard port (4242). Table 1 provides an overview of our findings. Note that evasions for traffic on port 80 are of particular severity, because traditional port-based heuristics would have covered them. To infer the impact of the discovered attacks, we also assess the consequences of successful attacks and discuss possible mitigations.

Bro As described in Section 4.1, the Bro NSM uses a fixed buffer to realize a cutoff threshold for protocol detection. Given a default PIA buffer size of 1024 bytes, correct protocol detection on non-standard ports is susceptible to the **CRLF Stuffing** approach, if a request is prefixed by a sufficiently large number of bytes. The standard detection signature for HTTP is bidirectional, i.e. it encompasses patterns for request and response headers. This means that the PIA needs to buffer a complete request and at least the beginning of a response to match the signature. Hence, the number of stuffing characters required to exhaust the buffer also depends on the length of the request. In theory detection could be avoided using a sufficiently large, valid HTTP request that suppresses the recognition of the response header. The possibility of buffer exhaustion has been foreseen by the creators of the DPD framework and was legitimated as deliberate design decision [10]. To allow the user to balance the resulting trade-off, the buffer size is configurable.

When confronted with the same traffic on a standard port, Bro successfully detects and analyzes the HTTP session. This is due to the fact that the HTTP analyzer is already added based on the port when building the initial tree of analyzers. Furthermore, the HTTP analyzer does not consider the consecutive CRLFs as protocol violation but as empty request lines and thus keeps attached to the session. However, each empty request triggers a so called weird event causing hundreds of events generated per packet. Because triggering events on a per packet basis is already considered overly expensive [23], this allows to perform a Denial of Service attack. We reported the Denial of Service attack to the project and it was addressed in the Bro 2.5.5 security release by introducing a sampling mechanism for these events [24].

Aside from the Deferred Start, Bro's protocol detection can be evaded in its standard configuration by both Misleading Start attacks, **Unknown Method** and **HELO Method**. To cause the HTTP analyzer to be attached, the request part as well as the reply part of the signature need to match. Because the request part is based on known methods only, the signature can easily be evaded using unknown methods. Like the buffer size, the protocol signatures can be configured freely by the user. There are two possible solutions to relax the signature: First, the combined, bidirectional signature can be split into two separate, unidirectional signatures. In Table 1 this variant is listed as "Bro (unidirectional)". Second, the request part can be improved by matching HTTP version information as well.

In case of Bro, evading protocol detection for HTTP traffic prevents the protocol specific analysis. In particular, there will be no meta data about the session in the `http.log` file and no HTTP specific event will be generated. However, general meta data about the connection, e.g. size and duration, is still being gathered and written to the `conn.log` file. Although Bro is vulnerable to all of the attacks in its standard configuration, the system can be configured to withstand them. With respect to the fixed size of the PIA buffer, which causes the vulnerability to Deferred Start attacks, a ring buffer approach is preferable, as it provides more flexibility to counter this type of attack.

Snort 3 The Dynamic Protocol Detection mechanism of Snort 3 is not vulnerable to the **CRLF Stuffing** attack approach. As described in Section 4.2, the detection of text-based protocols is realized by protocol signatures called spells. The corresponding acceptors explicitly ignore leading Spaces (SP), Tabs (TAB), Carriage Returns (CR) and Line Feeds (LF). The set of ignored characters is hard-coded, leaving the opportunity to defer the connection start by using other prefix characters. While this is theoretically possible, Section 6.2 will show that the approach is not exploitable in practice. Although the DPD mechanism of Snort 3 is not vulnerable to the stuffing attack, we noticed that the attached HTTP inspector is unable to reliably parse the session. For example, in case of 512 CRLFs, two responses and no request is found and in case of 32 CRLFs only one response could be parsed. This observation likely indicates a reassembling issue, which is, however, out of scope for this paper. Because spells implement unidirectional protocol signatures, Snort 3 is also not vulnerable to the **Unknown Method** attack approach. Although the request in the attack traffic does not trigger the HTTP detection, the following reply causes a match and thus causes the whole session to be flagged and analyzed as HTTP.

As Snort 3 does not explicitly consider the case of multiple matching protocol signatures, it is vulnerable to the **HELO Method** attack approach: Once an in-

Table 1: Dynamic Protocol Detection vulnerabilities of open source NSM software.

NSM System	Attack:	CRLF Stuffing		Unknown Method		HELO Method	
	Port:	4242	80	4242	80	4242	80
Bro 2.5.1		Evasion[1]	DoS	Evasion	–	Evasion	–
Bro 2.5.1 (unidirectional)		Evasion[1]	DoS	–	–	–	–
Snort 3		–[2]	–[2]	–	–	Evasion	Evasion
Suricata 4.1.2		Evasion[1]	Evasion[1]	–	–	Evasion	Evasion

– stands for *not* vulnerable

[1] To evade the (configurable) buffer has to be exhausted.

[2] HTTP Inspector is attached but cannot cope with the traffic.

spector for a connection is selected, the decision cannot be reverted. This can be used to trick Snort 3 into a wrong classification, causing an evasion. Because the `HELO` sequence triggers the SMTP inspector while the sequence is not part of the pattern set to match HTTP (see Figure 4), Snort 3 attaches the SMTP inspector and from here on fails to decode the connection properly. Given that the attached inspector has to cope with non-conforming traffic, this behavior bears the potential for Denial of Service attacks.

Due to its strong focus on rule matching, the consequences of evading DPD primarily concern this domain: Snort 3 offers the possibility to refine rules by specifying HTTP as the protocol (see Section 4.2) and match on selected protocol elements like the headers. Note that the latter also requires the attached HTTP inspector to correctly parse the session, which turned out to be problematic during CRLF Stuffing attacks. If a connection is not classified as HTTP and cannot be inspected accordingly, Snort does not match any HTTP related rule. To estimate the real-world impact, we have analyzed popular open rule sets with respect to their use of HTTP-specific rules. Table 2 shows that these sets contain a significant number of rules that can be evaded by preventing the correct detection of HTTP traffic. To work around this issue, one can weaken the specification of the affected rules: The protocol constraints have to be relaxed, e.g. using TCP instead of HTTP, and references to protocol elements that would have been made available by the inspector have to be converted into more general expressions. While this allows the rule to match again, it introduces overhead by increasing the number of rules that have to be considered for matching on lower protocol levels. Thus, the workaround increases the chance of false positive matches. All in all, a significant percentage of common rules can be evaded, while the available workaround comes with a substantial performance and quality degradation.

Table 2: HTTP related IDS rules, i.e. rules that rely on correct protocol detection.

Rule Set	Rules Total	HTTP-related	
ET Snort Edge open	19.673	8.530	43%
Snort 3 Community	829	487	58%
ET Suricata 4.0 open	19.328	10.654	55%
Positive Research	317	52	16%

All rule sets have been obtained on 12th of March 2019.

Suricata A third, popular open-source Network Security Monitoring system is Suricata [25], which focuses on intrusion detection and is heavily influenced by Snort. For our experiments, we used Suricata in version 4.1.2 operated under a standard configuration. Our black-box test revealed that Suricata is vulnerable to the **CRLF Stuffing** attack. But, as the attack requires about one hundred thousand CRLFs to evade protocol detection, it does not pose a serious threat in real-world scenarios as described in Section 6.2. In the course of our experiments, we tried to mitigate the attack by increasing the reassembly buffers for TCP without success. Suricata is not vulnerable to the the **Unknown Method** attack. However, we have been able to successfully execute the **HELO Method** attack, leaving all tested systems vulnerable.

In the case of Suricata, the consequences of evading the Dynamic Protocol Detection are twofold: First, Suricata is capable of generating an HTTP log that records meta data about the observed HTTP sessions. Evading the protocol detection mechanism prevents the software from extracting the meta data and thus suppresses logging of the corresponding information. Second, given that Suricata was developed as an alternative to Snort, it supports similar rule matching functionality. Like for Snort, a significant number of rules for Suricata are HTTP-specific as can be seen in Table 2. Again, attacking the DPD mechanism allows evading these rules. While Snort 3 is officially in beta state, Suricata is deployed in productive environments. Hence, the severity in case of Suricata should be considered even higher.

6.2 Susceptibility of Web Servers

The strategies presented to evade the different monitoring solutions will only pose a threat if the used protocol deviations do not affect the intended recipient, which in our case is an HTTP server. In this section, we review how popular web servers handle the protocol variations we seek to use for monitoring evasion.

Table 3: Leading characters ignored by web servers.

Web Server (Version)		**Ignored Characters**	**Maximal Repetitions**
Apache	(2.4.49)	CRLF	20
nginx	(1.14.0)	CR, LF	>10m
IIS	(8.5)	TAB, SP, CR, LF	16.271
lighttpd	(1.4.45)	-	-
nodejs	(8.10.0)	CR, LF	81.797

Deferred Start Our analysis of Bro's DPD mechanism revealed the possibility of a Denial of Service attack. As the DoS vector is solely based on the request, it can be exploited independently of the communication endpoint. Furthermore, it is possible to evade correct protocol classification for connections to non-common ports, if we are able to fill the PIA buffer. For this attack vector to pose a threat, we need to determine characters that can be prepended to valid HTTP requests without affecting the interpretation by the web server. To find suitable prefixes, we generated requests preceded by every possible 16 bit permutation, covering two ASCII characters. We consider a prefix as ignored, if the server replies with status code 200 (OK) to our request. Table 3 lists ignored characters for each tested web server together with the maximal number of repetitions tolerated. While most of the tested web servers accept any permutation of CR and LF, Apache just ignores the CRLF sequence. Only lighttpd does not accept any leading character. With respect to the default buffering capabilities of Bro, the web servers nginx, nodejs and IIS will offer the opportunity for an attack. The ignored characters can be prepended to a request without affecting its interpretation by the web server, but will exhaust the DPD buffers of the NSM system.

Misleading Start To evade Snort 3 and Suricata, we need to mislead the wizard inspector to recognize a different protocol. As there should be no overlapping between HTTP and another text-based protocol, this requires to send invalid HTTP requests that will not be processed by the web server. However, if the server does not close the connection, we could send further requests in the same connection that would evade proper analysis by the NSM. Although we cannot send arbitrary data, the first portion of an HTTP request constitutes the method to use and is thus variable. According to RFC 7230 [22], web servers should respond with the status codes 501 (Not Implemented) or 405 (Method Not Allowed) if the method is unknown to the server or not allowed for the requested resource, respectively. Table 4 shows that nginx, IIS and the hardened Apache [26] keep the connection open, leading to an exploitable situation. According to a recent survey, these three web server implementations are used to serve about 75% of all websites [27]. Furthermore, we surveyed the web servers hosting the top 500 websites based on the TRANCO ranking [28]. We found that 28% of the reachable web servers keep connections open, when confronted with an unimplemented method, which would offer a potential attack surface. Note that a large request method string might also be used to fill detection buffers as described in Section 5.1.

Table 4: Web servers confronted with an unimplemented request method*.

Web Server (Version)		**Reaction**	
Apache	(2.4.29)	501 Not Implemented	→ closed
Apache	(2.4.29†)	403 Forbidden	→ **open**
nginx	(1.14.0)	405 Method Not Allowed	→ **open**
IIS	(8.5)	405 Method Not Allowed	→ **open**
lighttpd	(1.4.45)	501 Not Implemented	→ closed
lighttpd	(1.4.45†)	501 Not Implemented	→ closed
nodejs	(8.10.0)	closed immediately	

† The web server was set up to use a "hardened" configuration, which limits the available request methods.
* For testing the unimplemented method behavior the string "UNKNOWNMETHOD" was used as HTTP method.

6.3 Combined Effectiveness

As discussed in Section 3.1, the overall attack is twofold. While all analyzed NSM systems can be evaded, the attacker needs to combine an attack on the target web server with a suitable evasion approach for the NSM system in place. For example, the **HELO Method** approach prevents the port-independent protocol detection for all NSMs in their default configuration. In case of Snort 3 and Suricata even traffic on well-known ports remains undetected. To allow a follow-up request after the misleading one, the web server is required to keep the connection open in case of encountering an unimplemented method. In this scenario, nginx, IIS and the hardened Apache server would offer an attack surface.

7 Discussion

The underlying issue in detecting an application layer protocol is the fact that a connection's ports serve as a session identifier and at the same time only implicitly codify the type of the session, i.e. the protocol in use. The endpoints themselves know about the services they operate by definition and only have to verify that the other endpoint adheres to the corresponding protocol. While the ports can be used by an observer to track a session, an observer lacks the background information about which services are operated on which ports. Therefore, a more general solution to the problem would be to make that information available to the monitor. If it is not available to the monitor, e.g. because the ports are dynamically negotiated, the protocol in use has to be inferred based on the observed communication. There are two fundamental challenges that have to be addressed in this context.

First, **ambiguities** have to be explicitly considered: The ideal acceptor for a protocol would be its analyzer. As many protocols are based on high-level grammars, i.e. they form context sensitive (Chomsky 1) or recursively enumerable (Chomsky 0) languages, the corresponding analyzers exhibit a significant complexity (or even face decidability issues) [29]. Thus, to decide on the protocol in use, all analyzers would have to be executed in parallel, which is inapplicable with respect to the resulting performance needs. Hence, simpler signatures, e.g. based on context-free grammars (Chomsky 2), are used (see 4). While this approach allows for efficient filtering of the possible options, it is obvious that the protocol identification cannot be decided based on these signatures. This means that, after matching signatures, there are ambiguities, which leave multiple possible options for further analysis. These options have to be considered by design to come to a well-grounded decision.

Second, the **cost-benefit ratio** of ongoing protocol detection has to be taken into account: The trade-off between performance and accuracy by either giving up on a protocol as soon as any protocol violation is encountered or continuous protocol detection throughout the stream was already described by Dreger *et al.* [10] in the context of Bro. While they also described attacks on the analyzers, attacks on the DPD mechanisms itself have not been considered explicitly. But, an attacker might try to mislead a protocol detection mechanism, either by exploiting an ambiguity or by deliberately switching the protocol in the course of the connection. In the first case, dealing with ambiguities in general also mitigates this attack. In the second case, the attached analyzer starts to fail without a possibility to resynchronize again. To address this situation, protocol detection could be restarted when an analysis begins to fail. This example underlines that buffering and continuous detection are separate aspects: While the process of detection is potentially ongoing, buffering is only needed back to the point where the protocol changes. Thus buffer size and the threshold for protocol detection should be decoupled. In addition, buffer contents could optionally be stored in persistent storage for later analysis if the protocol detection failed.

Recapitulating the presented attacks, we would also like to emphasize the exemplary nature of the selected scenario. Given the trend towards **encryption** of communication, in particular regarding the Web and the Internet, we expect the majority of HTTP traffic to be encrypted sooner or later, which prevents content analysis at arbitrary points of the communication path. However, NSM is usually conducted within the scope of a single management domain. Thus, the use of devices that terminate encrypted connections and allow access to the unencrypted traffic in a trusted environment is common. Examples range from load balancing scenarios in which encrypted connections are terminated upfront to the deployment of dedicated TLS-proxies that decrypt traffic on the fly [1], [2]. Consequently, the challenges and attacks presented in this paper also apply to the analysis of decrypted traffic. Even without access to the plain text of encrypted traffic, DPD retains its relevance: On the one hand, DPD is relevant in encrypted environments as collecting protocol-specific meta data is still valuable (e.g., for TLS) and requires the identification of the protocol in use. On the other hand, apart from the Internet, there are numerous kinds of networks in which monitoring is indispensable because of a lack of encryption. This can be either due to external constraints, e.g. the requirement to operate legacy systems, or a deliberate decision in order to provide transparency: The alternative to monitoring a networked system in a scenario in which the communication cannot be observed anymore is to gather data on the endpoints. However, this approach is inapplicable if the endpoints cannot be trusted.

Furthermore, the investigation of DPD also brings a new perspective to the discussion of **Postel's Law**. Postel's Law, also known as the Robustness Principle, refers to the implementation of protocols and states that one should strictly adhere to the standards when sending, but be liberal in what to accept when receiving [30]. While the benefits of this approach are obvious, numerous discussions have revealed significant issues for the Internet ecosystem caused by following this rule [31], [32]. So far security considerations primarily criticized the increased complexity of liberal implementations, which in turn increases the potential for introducing bugs. Our work demonstrates that following the Robustness Principle can also severely impede the observation of a networked system: The lax handling of protocol divergences and potential ambiguities result in a vast amount of possible interpretations that have to be considered by a monitor. Facing this problem, Kreibich *et al.* proposed to introduce a component that normalizes traffic [4]. This way a consistent, unambiguous standard is enforced, which can be relied on by the monitor. However, the suggested approach just shifts the attack surface away from the monitor, towards the normalizer. Overall, following Postel's law introduces room for interpretation that, in the end, allows for misinterpretation.

All in all, our work shows that the complexity of Dynamic Protocol Detection is prevalently underestimated. Although well-suited approaches exist that allow to deal with this complexity and balance the resulting trade-offs, we showed that all tested NSM systems are vulnerable to evasion by exploiting their DPD mechanisms. With respect to the fundamental nature of the underlying problem and the possible consequences for operating NSM systems, awareness has to be raised: Developers need to consider the lessons learned in their designs, to allow practitioners to balance the inevitable trade-offs.

8 Conclusion

In this paper, we analyzed the Dynamic Protocol Detection mechanisms employed by two popular and widespread open-source network monitoring tools. Building upon our insights, we deduced different DPD attack approaches that focus on the example of HTTP. Confronting three network monitoring tools with the generated traffic, we were able to evade all of them. In addition, we discovered a DoS attack in one of the systems, which has been reported and is now fixed. Given the shortcomings of the state of the art DPD mechanisms, we evaluated the real-world applicability and impact of the presented attacks. Based on our results, we come to the conclusion that deficient protocol detection can have a serious impact on the monitoring and security operations. Considering our findings, we discussed the main challenges of DPD for Network Security Monitoring: Detection decisions are neither clear nor definite given potential ambiguities in the detection process and various trade-offs have to be carefully taken into account when trading resources like computation time and memory for accuracy. The detection process can be tuned in terms of strictness of the detection signatures (e.g., uni- or bidirectional), endurance of the analysis (resource-saving cutoff vs. continuous detection) as well as buffer type (ring vs. linear) and size. For future work, we want to investigate mechanisms that allow resynchronization of partial streams, e.g. the mechanism employed by [33], with respect to their resilience against attacks.

Ethical Considerations

Following the concept of responsible disclosure, we have reported the DoS vulnerability we found in Bro during our experiments to the developers. Accordingly, the issue was fixed and a security release was published. In contrast to the DoS attack, the possibilities to evade monitoring are based on a fundamental problem. By publishing our results we hope to raise the awareness of the resulting trade-offs for practitioners, who have to balance them, and developers, who have to provide the means to do so.

References

[1] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. Halderman, and V. Paxson, "The security impact of HTTPS interception," presented at the Network and Distributed System Security Symposium (NDSS), Jan. 2017. DOI: `10.14722/ndss.2017.23456`.

[2] L. Waked, M. Mannan, and A. Youssef, "To intercept or not to intercept: Analyzing TLS interception in network appliances," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security - ASIACCS '18*, Incheon, Republic of Korea: ACM Press, 2018, pp. 399–412. DOI: `10.1145/3196494.3196528`.

[3] T. H. Ptacek and T. N. Newsham, "Insertion, evasion, and denial of service: Eluding network intrusion detection," Secure Networks Inc. Calgary Alberta, 1998.

[4] C. Kreibich, M. Handley, and V. Paxson, "Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics," in *Proc. USENIX Security Symposium*, vol. 2001, 2001.

[5] T.-H. Cheng, Y.-D. Lin, Y.-C. Lai, and P.-C. Lin, "Evasion techniques: Sneaking through your intrusion detection/prevention systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 1011–1020, 2012, ISSN: 1553-877X. DOI: `10.1109/SURV.2011.092311.00082`.

[6] D. Roelker, "HTTP IDS evasions revisited," *Sourcefire Inc*, 2003, DEF CON 11. [Online]. Available: `https://www.defcon.org/images/defcon-11/dc-11-presentations/dc-11-Roelker/dc-11-roelker-paper.pdf` (visited on 06/04/2019).

[7] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008, ISSN: 1553-877X. DOI: `10.1109/SURV.2008.080406`.

[8] T. Bujlow, V. Carela-Español, and P. Barlet-Ros, "Independent comparison of popular DPI tools for traffic classification," *Computer Networks*, vol. 76, pp. 75–89, Jan. 2015, ISSN: 13891286. DOI: `10.1016/j.comnet.2014.11.001`.

[9] S. Rezaei and X. Liu, "Deep learning for encrypted traffic classification: An overview," *IEEE Communications Magazine*, vol. 57, no. 5, pp. 76–81, May 2019, ISSN: 0163-6804, 1558-1896. DOI: `10.1109/MCOM.2019.1800819`.

[10] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer, "Dynamic application-layer protocol analysis for network intrusion detection," in *15th USENIX security symposium*, USENIX Association, 2006, pp. 257–272.

[11] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton, "Protocol misidentification made easy with format-transforming encryption," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13*, Berlin, Germany: ACM Press, 2013, pp. 61–72, ISBN: 978-1-4503-2477-9. DOI: `10.1145/2508859.2516657`.

[12] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer Networks*, vol. 31, no. 23, pp. 2435–2463, 1999. [Online]. Available: `http://www.icir.org/vern/papers/bro-CN99.pdf`.

[13] J. Forshaw, *Attacking network protocols: a hacker's guide to capture, analysis, and exploitation.* San Francisco: No Starch Press, 2017, 310 pp., ISBN: 978-1-59327-750-5.

[14] M. Belshe, R. Peon, and M. Thomson, "Hypertext transfer protocol version 2 (HTTP/2)," RFC Editor, RFC 7540, May 2015, Published: Internet Requests for Comments. [Online]. Available: `http://www.rfc-editor.org/rfc/rfc7540.txt`.

[15] The Zeek Project. (2019). The bro/zeek network security monitor, [Online]. Available: `https://www.zeek.org/` (visited on 08/29/2019).

[16] Cisco. (2019). Snort 3, [Online]. Available: `https://www.snort.org/snort3` (visited on 08/29/2019).

[17] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano, "nDPI: Open-source high-speed deep packet inspection," in *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, Nicosia, Cyprus: IEEE, Aug. 2014, pp. 617–622. DOI: `10.1109/IWCMC.2014.6906427`.

[18] S. Alcock and R. Nelson, "Libprotoident: Traffic classification using lightweight packet inspection," 2012. [Online]. Available: `https://wand.net.nz/sites/default/files/lpi.pdf` (visited on 11/25/2019).

[19] M. Roesch and S. Team, *Snort 3 user manual*, 2019. [Online]. Available: `https://www.snort.org/downloads` (visited on 06/12/2019).

[20] Cisco. (Aug. 29, 2018). Snort blog, [Online]. Available: `https://blog.snort.org/2018/08/snort-3-beta-available-now.html` (visited on 08/29/2019).

[21] R. Fielding and J. Reschke, "Hypertext transfer protocol (HTTP/1.1): Semantics and content," RFC Editor, RFC 7231, Jun. 2014, Published: Internet Requests for Comments. DOI: `10.17487/RFC7231`. [Online]. Available: `http://www.rfc-editor.org/rfc/rfc7231.txt`.

[22] ——, "Hypertext transfer protocol (HTTP/1.1): Message syntax and routing," RFC Editor, RFC 7230, Jun. 2014, Published: Internet Requests for Comments. [Online]. Available: `http://www.rfc-editor.org/rfc/rfc7230.txt`.

[23] The Zeek Project. (2019). Bro/zeek documentation, [Online]. Available: `https://docs.zeek.org/en/stable/scripts/base/bif/event.bif.bro.html#id-new_packet` (visited on 08/29/2019).

[24] ——, (2019). Bro release notes, [Online]. Available: `https://www.zeek.org/manual/2.5.5/install/release-notes.html#bro-2-5-5` (visited on 08/29/2019).

[25] Open Information Security Foundation (OISF). (2019). Suricata, [Online]. Available: `https://suricata-ids.org/` (visited on 08/29/2019).

[26] Open Web Application Security Project (OWASP). (Apr. 4, 2016). SCG WS apache, [Online]. Available: `https://www.owasp.org/index.php/SCG_WS_Apache` (visited on 09/20/2019).

[27] Netcraft. (Aug. 15, 2019). August 2019 web server survey, [Online]. Available: `https://news.netcraft.com/archives/2019/08/15/august-2019-web-server-survey.html` (visited on 09/19/2019).

[28] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2019, ISBN: 978-1-891562-55-6. DOI: `10.14722/ndss.2019.23386`.

[29] L. Sassaman, M. L. Patterson, S. Bratus, and A. Shubina, "The halting problems of network stack insecurity," *;login:*, vol. 36, 2011.

[30] B. E. Carpenter, "Architectural principles of the internet," RFC Editor, RFC 1958, Jun. 1996, Published: Internet Requests for Comments. [Online]. Available: `http://www.rfc-editor.org/rfc/rfc1958.txt`.

[31] M. Thomson, "The harmful consequences of the robustness principle," IETF Secretariat, Internet-Draft, May 2019, Published: Working Draft. [Online]. Available: `http://www.ietf.org/internet-drafts/draft-iab-protocol-maintenance-03.txt`.

[32] L. Sassaman, M. L. Patterson, and S. Bratus, "A patch for postel's robustness principle," *IEEE Security & Privacy Magazine*, vol. 10, no. 2, pp. 87–91, Mar. 2012, ISSN: 1540-7993. DOI: `10.1109/MSP.2012.31`.

[33] R. Sommer, J. Amann, and S. Hall, "Spicy: A unified deep packet inspection framework for safely dissecting all your data," in *Proceedings of the 32nd Annual Conference on Computer Security Applications - ACSAC '16*, Los Angeles, California: ACM Press, 2016, pp. 558–569, ISBN: 978-1-4503-4771-6. DOI: `10.1145/2991079.2991100`.