Research Report: Synthesizing Intrusion Detection System Test Data from Open-Source Attack Signatures

Jared Chandler
Tufts University
Medford, Massachusetts
jared.chandler@tufts.edu

Adam Wick
Fastly
Portland, Oregon
awick@fastly.com

***Abstract*—Intrusion Detection Systems (IDS) act as a first line of defense for network infrastructure by identifying malicious traffic and reporting it to administrators. Signature-based IDS identify this traffic by attempting to parse packets according to user-supplied rules based on well-known examples of bad traffic. However, test data can be difficult to come by (due to its sensitive nature) which makes evaluating new rules difficult. In this work we discuss the limitations of an existing SMT-based synthesis approach to automatically generating malicious network traffic. We then present a survey of how IDS rules are written in practice using an open-source corpus of over 30,000 rules and discuss a road-map towards extending the existing approach with the goal of generating security test data characterizing a broad range of threats, as well as ancillary uses assisting users in writing IDS rules and identifying IDS implementation bugs. Finally, we share early results from an evaluation of one such extension which successfully generated IDS test data for over 90% of the rules evaluated.**

***Index Terms*—network security, intrusion detection system, synthesis**

I. Introduction

Intrusion detection systems (IDS) perform a critical security function by identifying potentially malicious traffic moving across the network. An IDS performs this recognition by attempting to parse an incoming packet according to a collection of user-supplied rules or signatures. These rules are written in domain specific languages specifying the circumstances and conditions under which a packet is successfully parsed, with a successful parse by a rule meaning that the packet is labeled as potentially malicious. The system can then alert an administrator, log the traffic for further inspection, and/or take an action (such as preventing the packet from moving across the network).

An IDS can only identify malicious traffic when it has a corresponding rule. Network administrators, researchers, and cybersecurity professionals create rules in response to malicious activity, and often share them with the security community at large. As a result, open-source rule sets can include a wide variety of rules that, when installed, allow an IDS to identify a correspondingly wide range of potential malicious activity.

However, without access to the original test cases usually in the form of malicious traffic captures, the behavior of an IDS rule cannot be easily verified and often must be taken on faith by the user. These captures can be sensitive, however, as they may contain information about the original network, or betray particular configuration states that may be considered critical secrets.

Since IDS rules function by parsing input packets and determining if they match a rule, it may be tempting to think of IDS parser security in the same way that one thinks of classic parser security. In a classic PDF parser, for example, security issues often stem from the parser being too permissive, and thus triggering unforeseen behavior on a malicious input. IDS parsers, however, offer an interesting contrast, in that risk occurs when a malicious packet passes through unrecognized. Instead of attempting to recognize only legitimate network traffic, an IDS tries to recognize only potentially illegitimate traffic. In addition, while parsers designed to handle legitimate input, such as XML, SQL, and PDFs for example, have ready access to examples of valid input for testing, examples of malicious traffic to test the recognition capability of an IDS are actually quite hard to come by. Example malicious traffic can identify an organization whose systems were compromised, or a system being targeted [1], [2]. Captured malicious traffic can also divulge collection methods and sensor placement that security organizations do not want to disclose [3], [4]. As a result, the recognition capability of an IDS rule from a community dataset often must be taken on faith, barring some accompanying input to test it on.

This lack of appropriate test data goes further than simply validating that an IDS is functioning correctly. Without access to representative test data of security threats, researchers are limited to public captures, and thus must expend significant effort building infrastructure to either collect / capture data on their own, or accept publication restrictions on their research imposed by the owner of the test data.

In this paper we propose using our SoundTheAlarm SMT-based synthesis technique to automatically generate testing data from open-source IDS rules [5]. We discuss the limitations of our technique, how the IDS rules are used in

2023 IEEE Security and Privacy Workshops (SPW) | 979-8-3503-1236-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/SPW59333.2023.00023

© 2023, Jared Chandler. Under license to IEEE.

DOI 10.1109/SPW59333.2023.00023
Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.

practice, and potential solutions addressing the limitations. Our approach takes IDS rules as input, and generates packets satisfying the constraints of the rule, as validated by the IDS itself. Originally developed for cyber-deception of an adversary, we believe this technique is well suited to automatically generating representative attack data for testing network security in a variety of contexts. This approach has the benefit of being able to generate testing data for a wide variety of malicious network behaviors without requiring disclosure of sensitive or identifying data. Further, instances where our approach fails to generate testing data can be useful for identifying malformed IDS rules, and instances in which the generated data fails to trigger the IDS are useful for finding bugs in the IDS implementation.

A. Contributions

We summarize our contributions as the following:

1) A examination and discussion of the limitations of our earlier work synthesizing malicious network traffic from IDS rules in Section III.
2) An analysis of how IDS rules and features of the Suricata IDS rule language are used (or not used) in an open-source corpus of more than 30000 individual rules spanning 12 years in Section IV.
3) A discussion of proposed extensions to our approach to allowing greater coverage when generating synthesized attack messages in Section V, and early results from evaluating one such extension in Section VI.

B. Roadmap

Our paper is structured as follows: First, we provide a brief background on intrusion detection systems, their use, and functions. Second, we introduce the salient aspects of the domain-specific pattern the rules are written in and provide a summary of our approach to automatically generating messages from rules which we call SoundTheAlarm. Third, we discuss the limitations uncovered in earlier evaluations of our proof-of-concept tool. Next, we examine and discuss the open-source Proofpoint Emerging Threats ruleset from different perspectives and observed patterns of use. We then discuss a set of proposed extensions and enhancements geared toward allowing SoundTheAlarm to be used to generate effective test data for IDS as well as aiding a user in the development of new IDS rules. We then present early results from evaluating one such extension to SoundTheAlarm. Finally, we present relevant related work and a brief conclusion.

II. Background

Intrusion Detection Systems (IDS) function as network sensors allowing network administrators to detect malicious activity, and then take action based on that detection. They perform this task by comparing network traffic (at the packet level) against a set of rules or signatures, with each rule/signature tailored to detect a specific type of malware or security vulnerability. These rules are created by security researchers, network administrators, and analysts working with observed instances of malicious activity. To benefit the broader community, these rules are often shared publicly or commercially so that network administrators can proactively identify malicious activity and implement appropriate defenses. Similar to how a vaccine primes an immune system to recognize a pathogen, shared rules help prime IDS systems to recognize a threat before it is observed on the local network.

IDS Systems are characterized by the mechanism they use to observe potentially malicious behavior. Signature-based systems, such as Snort [1] and Suricata [2], match observed traffic against fixed rules to identify threats. Behavioral systems such as Bro [3] and later Zeek [4], measure behaviors on the network according to thresholds learned from normal activity. When thresholds are exceeded, the system alerts the administrator and optionally takes action.

Behavior-based systems (also called anomaly-based systems) have the advantage of being able to incorporate a baseline of what is considered normal activity for a specific network. This baseline allows these systems to avoid false-positives through continual refinement of the thresholds for normal activity. However, these baselines often vary between networks. For example, what is considered normal on the network at an educational institution likely differs from that of a financial institution. Because of these differences, the metrics and thresholds used by a behavioral IDS for one network are of limited use to an administrator at another institution.

In contrast, signature-based IDS focus on identifying only the threat, assuming that everything else is benign. Signature-based intrusion detection systems use rules to identify traffic, and then perform an action: logging an alert, blocking packets from moving further on the network, etc.[5][6] However, because the rules used by signature-based IDS are fixed, they may incorrectly identify benign network traffic as malicious, creating a false positive for an administrator to investigate. Alternatively, they may be defined too narrowly, and only detect particular variants of a given attack. However, in contrast to behavioral IDS, signature-based IDS rules are readily transferable between different networks; even if the underlying patterns of network use differ, many of the applications and their vulnerabilities will overlap.

For this work, we focus on signature-based network IDS systems [6] such as Snort and Suricata. The development of rules to identify traffic involves non-trivial effort, especially considering the need to have concrete instances of malicious traffic to test the rule while it is being developed and as it is refined. The Snort IDS was first released in 1998, while Suricata released in 2009. Suricata improved on Snort, in terms of performance, while still being able to use the Snort rule format. This commonality allowed Suricata users to leverage the existing base of Snort rules, but with additional extensions

[1] https://www.snort.org/
[2] https://suricata.io/
[3] https://old.zeek.org/manual/2.5.5/intro/index.html
[4] https://zeek.org
[5] https://docs.snort.org/rules/
[6] https://suricata.readthedocs.io/en/suricata-6.0.0/rules/index.html

Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any
(msg:"ET TROJAN Possible Metasploit Payload Common
Construct Bind_API (from server)";
flow:from_server,established; content:"|60 89 e5
31|"; content:"|64 8b|"; distance:1; within:2;
content:"|30 8b|"; distance:1; within:2;
content:"|0c 8b 52 14 8b 72 28 0f b7 4a 26 31 ff|";
distance:1; within:13; content:"|ac 3c 61 7c 02 2c
20 c1 cf 0d 01 c7 e2|"; within:15; content:"|52 57
8b 52 10|"; distance:1; within:5; metadata:
former_category TROJAN; classtype:trojan-activity;
sid:2025644; rev:1; metadata:affected_product Any,
attack_target Client_and_Server, deployment
Perimeter, deployment Internet, deployment Internal,
deployment Datacenter, tag Metasploit,
signature_severity Critical, created_at 2016_05_16,
updated_at 2018_07_09;)
```

Fig. 1. Example Suricata rule (Reprinted from Chandler et al. [5]).

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 21 (msg:"ET
SCAN PRO Search Crawler Probe";
flow:to_server,established; content:"PASS "; nocase;
depth:5; content:"crawler"; nocase; within:30;
pcre:"/^PASS\s+PRO(-|\s)*search\s+Crawler/smi";
reference:url,sourceforge.net/project/showfiles.php?
group_id=149797;
reference:url,doc.emergingthreats.net/2008179;
classtype:not-suspicious; sid:2008179; rev:3;
metadata:created_at 2010_07_30, updated_at
2010_07_30;)
```

Fig. 2. Example Suricata rule using regular expression (shown in red).

to the Snort rule format that enable easier targeting of specific packet attributes, such as HTTP headers and DNS query counts. While we focus on rules written for the Suricata IDS, we note that Snort rules use almost identical syntax. A limitation of Suricata is the lack of an official grammar describing the rule syntax despite the maturity of the platform.[7] [8] Users writing rules must rely on a combination of documentation, example rules, and repeated trial and error to develop new rules to match malicious network traffic.

Our earlier work on automatically generating attack data through synthesis focused on demonstrating a proof-of-concept system: SoundTheAlarm [5]. This system was limited to generating attack data for a subset of the Suricata rule language. Our interest here is in generating IDS testing data for any rule provided.

A. Suricata Rule Domain Specific Language

Suricata rules specify the properties that a candidate packet must satisfy to be considered a match. These properties either specify packet features that can be examined directly (such as IP address, length, and network port), or describe search

[7]https://redmine.openinfosecfoundation.org/issues/4662
[8]https://forum.suricata.io/t/rule-grammar-specification/1664

```
alert http any any -> [$HOME_NET,$HTTP_SERVERS] any
(msg:"ET EXPLOIT eMerge E3 Command Injection Inbound
(CVE-2019-7256)"; flow:established,to_server;
http.method; content:"GET"; http.uri;
content:"/card_scan"; startswith; fast_pattern;
content:".php"; distance:0; within:15;
content:"=|60|"; reference:cve,2019-7256;
classtype:attempted-admin; sid:2033757; rev:1;
metadata:created_at 2021_08_22, cve CVE_2019_7256,
former_category EXPLOIT, updated_at 2021_08_22;)
```

Fig. 3. Example Suricata rule using named buffers for HTTP method (http.method) and URL (http.uri).

conditions to be applied to the packet byte-string. Example rules are shown in Figures 1 and 2. The search conditions, called payload keywords [9], allow a user to specify content values (`content`) that a packet byte-string must contain in order to match a rule. For more flexible matching, users can also provide regular expressions (`pcre`) to match sections of the packet. A user can optionally specify the location where a match must be located, either in relation to the start of the packet (`offset`, `depth`) or relative to an earlier match (`distance`, `within`).

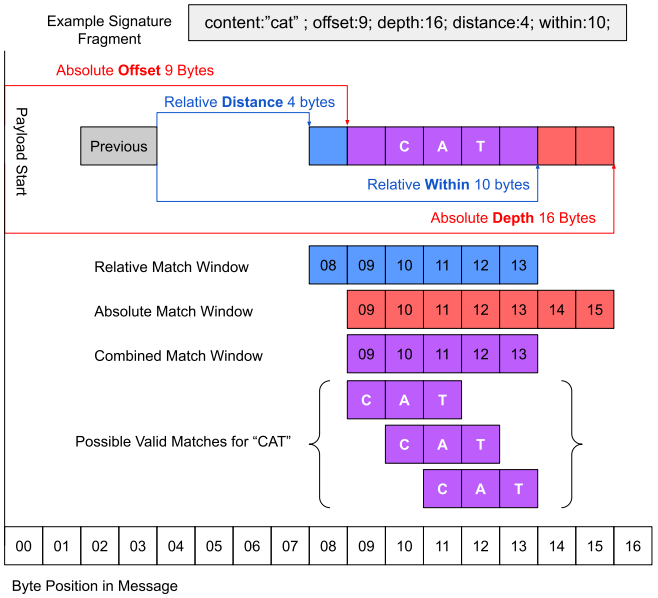

Fig. 4. Example of Suricata content position constraint meanings (Reprinted from Chandler et al. [5]).

We illustrate the use of these content position payload keywords and their meanings in Figure 4.While these keywords search anywhere in a packet for a match by default, Suricata also allows the scope of the keywords to be limited to a specific field or semantic region of a packet such as a HTTP Uniform Resource Identifier (URI), or a DNS Query. Suricata

[9]https://suricata.readthedocs.io/en/suricata-6.0.0/rules/payload-keywords.html

Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.

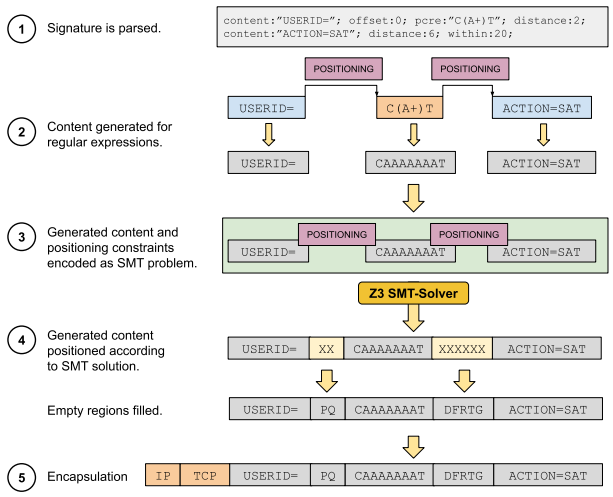

Fig. 5. Steps for generating payloads from signatures (Reprinted from Chandler et al. [5]).

implements this scoping through the use of named buffers for portions of well known protocols. The use of these is shown in Figure 3 for buffers `http.method` and `http.uri`. Suricata handles the parsing of a packet into protocol specific named buffers and then applies the rules to search those buffers which are referenced. These named buffers can be thought of as syntactic sugar for selecting protocol specific fields and regions of messages, allowing the user to concentrate on the content to be matched in the buffer rather than the conditions necessary to restrict the search to the bytes of interest.

B. Message Synthesis Approach

SoundTheAlarm's present approach to synthesizing messages which match a rule relies on five steps shown in Figure 5. SoundTheAlarm first generates concrete strings by directly copying `content` keywords, and generating string instances from `pcre` keywords. Formally, SoundTheAlarm introduces a set of integer valued constraints describing each concrete string as shown in Equations 1 through 6. Values which are fixed such as the size of the content being placed in the message ($\mathbf{Content_{width}}$ and $\mathbf{Payload_{start}}$) are shown in **black**. Fixed values which are extracted directly from the rule are shown in `red` and `blue`. Variables for the SMT-solver to determine the integer values of are shown in *purple*.

$$\mathbf{Content_{width}} = Content_{end} - Content_{start} \quad (1)$$

$$Content_{start} < Content_{end} \quad (2)$$

$$\mathbf{Payload_{start}} + \texttt{offset} <= Content_{start} \quad (3)$$

$$\mathbf{Payload_{start}} + \texttt{depth} >= Content_{end} \quad (4)$$

$$Previous_{end} + \texttt{distance} <= Content_{start} \quad (5)$$

$$Previous_{end} + \texttt{within} >= Content_{end} \quad (6)$$

In the case of a rule with multiple instances of `content` or `pcre`, we add an appropriate set of constraints for each instance. Finally, we ask the solver to produce a solution satisfying all introduced constraints. If the solver is able to find a satisfying assignment we extract the corresponding interger valued $Content_{start}$ entries and use them to position the generated byte strings as shown in steps (3) and (4) of Figure 5.

III. Limitations Uncovered in Earlier Evaluation

While our research goal here is to produce network attack test data automatically from IDS signatures, SoundTheAlarm originated as a method to automatically create deceptive network attack data for the purposes of confusing and deceiving a network attacker [5]. As part of a proof-of-concept evaluation for that purpose SoundTheAlarm was given a selection of both TCP and UDP rules, and then asked to first synthesize an attack message from the rule, and then if a message was produced, to verify that it could trigger instances of the Suricata and Snort IDS to validate that it was consistent with the appearance of malicious network traffic, with evaluation by subject matter experts planned as follow-on work. The details of this evaluation can be found in our paper: Deceptive Self-Attack for Cyber-Defense [7]. However, in the course of this and subsequent evaluations, instances where SoundTheAlarm failed to either generate a message from a rule, or failed to trigger an IDS with the generated message proved interesting. In this section we present a summary of those cases, followed by a discussion of common problems we observed in writing rules. We then discuss how SoundTheAlarm might help a user, followed by the limitations of the present proof-of-concept tool.

A. Mistaken Assumptions

In our earlier evaluations SoundTheAlarm was unable to synthesize messages for some rules. These message generation failures initially appeared to be caused by incorrectly written rules due to what we thought was an erroneous ordering of the `content` keywords, with content later in the message appearing first in the rule. When re-written in the same order as their `offset` values, SoundTheAlarm generated messages which triggered an alert. We discovered that while the overwhelming majority of rules are written with `content` and `pcre` portions ordered as they would appear in the byte-string, there is no strict requirement that they do so. Only instances of relative positioning using `distance` and `within` are required to come directly after an already matched element.

B. False Negatives

For cases where SoundTheAlarm synthesized a message from the rule constraints, but the IDS failed to alert when given the message (False Negatives), we examined the failures to identify the underlying root causes. Among these were several rules which escaped specific characters that Suricata's rule parser ignores, but which were a part of our generated messages. We believe these rules were written before a bug

Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.

relating to escape characters in Suricata's signature parser was reported and subsequently fixed.[10][11] The result is that the packets matched by these rules prior to the bug fix are no longer matched afterwards. However, as the rules are syntactically valid, they load correctly even though they will never detect their intended threats. *These rules underscore the importance of having testing data to verify that a rule functions as intended.* In this case, the rules were likely tested only at the time they were written, and were deemed satisfactory. In the future, a tool like SoundTheAlarm could be used to ensure that rules are properly validated across IDS upgrades, as a form of continuous regression testing. An interesting characteristic of Suricata rules is that while there is no formal grammar, different types of keywords have been introduced and subsequently deprecated. Having to method to verify that rules with deprecated syntax either work as intended or not would be useful to system administrators and rule authors.

C. Constraints Embedded in Regular Expressions

We also found multiple rules with regular expressions that referenced the start of a line (^) and end of a line ($) characters. Suricata interprets these as positioning matches at the start and end of a buffer byte-string, a constraint we had not originally anticipated. We believe these constraints could be automatically extracted from the regular expression syntax and added to SoundTheAlarm 's synthesis approach. Both of these constraints can be expressed in other ways in the rule DSL; using `offset` to constrain a match at the start of a payload or buffer, and using the `endswith` keyword to constrain a match at the end of a payload or buffer.

D. Overlapping Content for Performance

Finally, we observed a limited number of rules which used both `content` and `pcre` keywords to match overlapping sections of a message byte-string. We had initially assumed that each `content` or `pcre` keyword matched an independent and disjoint region of the payload. We believe these overlapping patterns were used to enhance the performance of the rules, as the `content` elements are used to match or reject a byte-string before the computationally more expensive `pcre` search is run.

```
content:"foo";offset:0;
pcre:"fooba+r";offset:0;
```
(7)

To address such overlaps, our synthesis approach could be extended to consider constraints on which values can occur at each byte-position in our synthesized string. Presently, the approach simply looks for a non-overlapping layout which respects all appropriate distance constraints. Adding such byte-value constraints would substantially increase the complexity of the synthesis problem, but is a worthwhile trade-off for reasons we discuss in Section V.

E. What can go wrong with rules?

While we propose using SoundTheAlarm mainly to automatically generate test data appropriate for an IDS, we have identified several use-cases related to helping users writing rules.

The opaque interface of the the IDS itself is the first obstacle facing users when writing rules identify malicious traffic. Given a candidate rule, and a capture of the network traffic, the IDS only reports back whether or not the rule was triggered by some traffic in the capture. The user must proceed strictly through trial and error, as the system provides no additional feedback. This limitation may be trivial when a rule is simply matching a single piece of content, but as rules grow in complexity it adds significant difficulty to the development process. Compounding this difficulty is the ability to write syntactically valid rules that are logically impossible. For example, while rule fragment 8 is valid, rule fragments 9 and 10 can never be satisfied.

```
content:"a";content:"b";
```
(8)

```
content:"a";depth:1,content:"b";depth:1;
```
(9)

```
content:"a";depth:1,content:"cb";depth:2;
```
(10)

Even when a rule is written and can identify a piece of malicious traffic, issues remain. First, the rule could be too specific, correctly identifying some malicious packets, but missing others. Second, the rule could be too general, in which case it will erroneously identify benign traffic as being a threat. These situations can be be difficult to foresee as the user is constrained by the same trial and error interface to determine if a rule is working correctly or needs to be adjusted.

F. How else can SoundTheAlarm help?

We believe SoundTheAlarm can help with these problems in two ways. First, by alerting a user when their rule contains a logical impossibility, as evidenced by SoundTheAlarm's inability to synthesize an example message from the rule. This warning would allow the user to correct the problem and move on. Second, once the user has a rule which is valid for some message, SoundTheAlarm can generate multiple example messages giving a user context as to what data their rule is actually describing, such as is illustrated in Figure 10 and discussed in Section V. In the case that the rule is too general, the user can add constraints. In the case that the rule is too specific, constraints can be removed. We believe that SoundTheAlarm can interactively provide useful feedback similar to interactive tools for working with regular expressions[12] while improving on wizard based tools that only help with rule input.[13] We also plan to explore situations where our synthesized test data would be useful to security practitioners, analysts and researchers outside of testing IDS configuration, such as with evaluating network forensics tools [8], as input

[10]https://github.com/theY4Kman/parsuricata/issues/3
[11]https://redmine.openinfosecfoundation.org/issues/2626
[12]https://regex101.com/
[13]https://github.com/chrisjd20/Snorpy

Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.

to machine learning systems [9], and for adding realism to honeypots [10].

G. Limitations of Current Approach

While our approach has demonstrated the ability to generate testing data using only IDS rules, it remains subject to several limitations.

First, SoundTheAlarm's SMT encoding only supports the two most commonly used payload keywords: `content`, `pcre`, and the four corresponding position keywords: `offset`, `depth`, `within`, and `distance`. However, Suricata rules allow a user to specify long-distance dependencies using `isdataat`, `byte_jump`, and `byte_extract` keywords. Synthesis from rules using these keywords is not supported at present, but could be added to our SMT encoding. Similarly, packet features such as IP time-to-live and various HTTP, DNS, and TLS named buffer keywords are not presently supported. In particular, synthesis of HTTP content is more complex due to the size of the underlying byte-strings being generated; the use of encryption and compression; and the number of unique fields a request and response can include. We discuss this complexity in Section IV, and share early results from a proof-of-concept extension in Section VI.

At present we generate byte-strings from `pcre` elements before using Z3 to solve positioning constraints. This design choice allows us to treat regular expressions and static content identically from the solvers perspective. In future work we plan to explore allowing Z3's regular expression functionality to perform this work. Doing so will allow the strings generated from the regular expression to take into account additional constraints introduced by other keywords, such as when pieces of content overlap. We discuss proposed solutions to the above limitations in Section V.

Finally, there is no guarantee that SoundTheAlarm will be able to synthesize test data from a rule in a reasonable amount of time. While we were able to return useful results very quickly in our evaluations of SoundTheAlarm–generally in less that a second–these timings may not continue to hold as we add new keywords to our SMT-encoding or perform synthesis of byte-values rather than integer positions for fixed strings.

To better understand these limitations as they relate to rules as they are used in practice, we next turn to an examination of a real-world Suricata data-set of over 30,000 rules.

IV. Rules in Practice

Our goal in this section is to understand rules as they are used in practice as a means to extending the SoundTheAlarm tool to generate appropriate traffic for any rule. Specifically, we focus on understanding changes in rules over time, the distributions of protocols, how different rule features are used, and the relative complexity of the rules themselves.

A. The Proofpoint Emerging Threats Community Ruleset

In this analysis we focus on one open-source ruleset: started in 2003, the Proofpoint Emerging Threats Community Ruleset

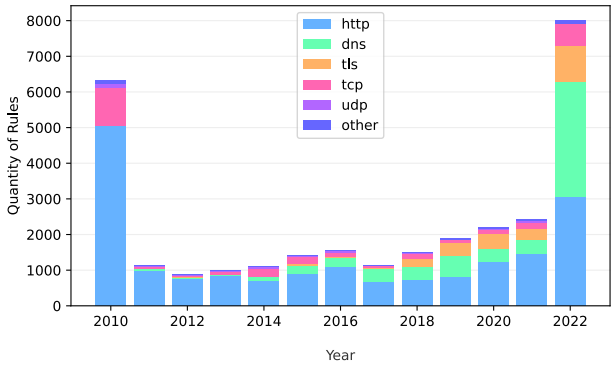

Fig. 6. Quantity of rules added to Emerging Threats dataset by year and protocol. Rules for 2010 were a basis set of Snort rules.

is the largest public collection of signatures available for both Snort and Suricata.[14] Our interest in these rules is two-fold. First, these rules characterize actual threats of concern to network administrators and security professionals. Second, when thought of as individual parser programs written by users in a domain specific language, these rules illustrate which language constructs are most commonly used to identify malicious network traffic, and how these language constructs are used in practice. For our analysis we used the Suricata version of the rule-set containing more than 30,000 signatures spanning more than 12 years.[15]

B. Rules over Time

In Figure 6 we show the the quantity of new rules added each year. A key feature of Suricata was the ability to use Snort rules, which is reflected by the 6333 rules initially ported from Snort. The significant increase in the number of rules for 2022 is due to organic growth in the number of rules combined with the addition of rules identifying domain names and specific TLS certificates. These DNS and certificate rules individually name specific domains and certificates which have been observed engaging in malicious activity. These rules were automatically generated from threat intelligence feeds by DNS hosting providers and security researchers for inclusion in the community ruleset. Figure 6 further illustrates the distribution of rules by protocol for year. This data shows growth of DNS and TLS rules in 2022 relative to earlier years. Overall, the growth in new rules from 2011 through 2022 is indicative, unsurprisingly, of new threats being uncovered and network administrators increasingly interested in detecting them. A major advantage of automatically generating appropriate network security test data from rules is that it can keep current with threats as the rules describing them are released.

C. Rules by Protocol

Overall, the majority of IDS rules focus on web traffic, as shown in Figure 7, with 18,321 rules for HTTP, compared to

[14]https://doc.emergingthreats.net/bin/view/Main/AboutEmergingThreats
[15]https://rules.emergingthreats.net/open/suricata-5.0/

Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.

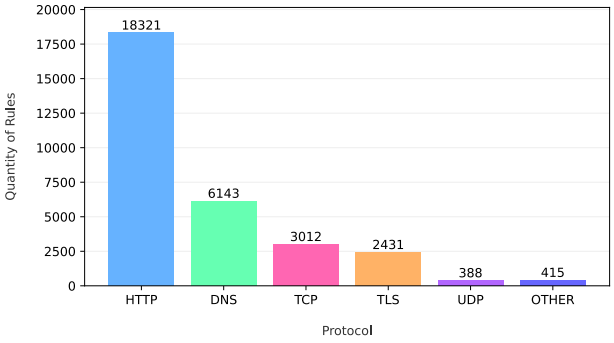

Fig. 7. Quantity of rules by protocol.

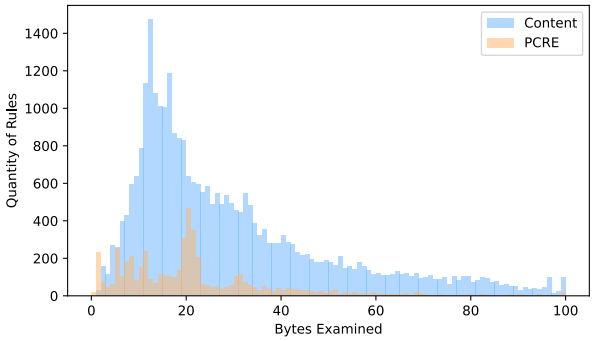

Fig. 9. Bytes examined using payload features (content, pcre)

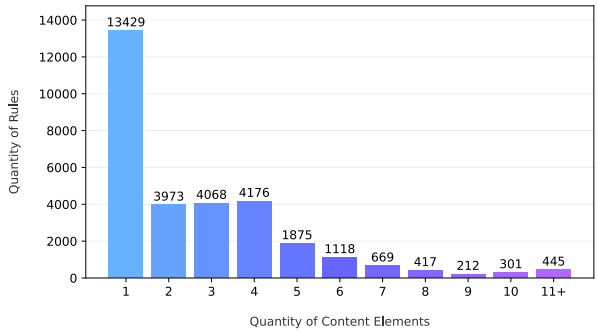

Fig. 8. Quantity of rules using payload features (content, pcre). 27 rules had no payload features.

6,143 rules for DNS and 6,246 rules for all other protocols combined. The quantities and distribution of rules reflect where network administrators are looking for malicious traffic. For example, it appears that generic UDP traffic is scrutinized less heavily than HTTP which comprised approximate 60% of the rules in the dataset. A significant limitation of SoundTheAlarm is the ability to synthesize payloads only for payload portions of TCP and UDP rules. Rules for protocols such as HTTP, DNS, and TLS make use of named buffers to match against specific fields or packet regions. Generating useful and representative IDS test data requires handling these protocols, both from a completeness perspective, and because they are clearly the focus of network administrator concerns.

D. How many pieces of Content?

In Figure 8 we show the frequency of content elements (`content` and `pcre`) in individual rules. While the highest frequency 13,429 (44%) is for a single element. 17,254 rules (56%) use two or more content elements, in which case relative positioning between elements becomes an important consideration as to whether a rule is overly general, or overly specific. We observed 84,601 individual instances of `content`, compared to only 6,448 instances `pcre`. Writing regular expressions is known to be difficult in spite of their expressive power [11], [12]. This difference between `content` and `pcre` may be due to user discomfort with writing and testing regular expressions in this context, This is a use case SoundTheAlarm can help address by automatically generating example packets from a signature to assist a user with the process of writing and refining rules.

E. How much data is being Examined?

We show an estimate as to the quantity of packet bytes described by these elements in Figure 9. For each rule we calculated the total size of all (content) elements. To estimate the quantity of packet bytes for each `pcre` element, we used an open-source library to generated 100 strings from the regular expression, and then averaged the lengths.[16] On average `content` elements described 33.3 packet bytes, while `pcre` elements described 65.6 packet bytes. Taken together, each rule in the dataset described 45.8 packet bytes on average. This data is useful as an estimate of that the scale of any byte-value synthesis we may wish to perform in excess of resolving positioning constraints. We discuss these cases in Section V.

F. Use of Payload Keywords

Next, we examined the use of the four content position payload keywords–`offset`, `depth`, `distance`, and `within`–to understand how they are used in practice. We show these results in Table I. 18,019 rules did not use any content position keywords, instead using only `content` and `pcre` to specify matches that could appear anywhere in the packet. This is notable as without ordering constraints the rules may be overly general, allowing for matches the author did not intend. 12,691 rules used one or more content position keywords. The most frequent keyword was `distance`, used 19,026 times, while the least frequent was `offset`, used only 995 times.

G. Use of Named Buffers

Finally, we examined buffer use for rules focusing on HTTP, DNS, and TLS protocols which comprise 86% of the total rules in the dataset. A significant limitation of SoundTheAlarm is the lack of handling for named buffers. To characterize buffers use we counted the number of times each buffer was used in a rule for these protocols as shown in Table II. For HTTP there were 24 distinct buffers with a total 31521 uses. Among them the `http.uri` buffer for the

[16]https://github.com/asciimoo/exrex

Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.

TABLE I
QUANTITIES OF RULES USING PAYLOAD POSITION KEYWORDS. •USED ○NOT USED.

Position Keyword				Qty.
offset	depth	distance	within	Rules
○	○	○	○	18019
○	○	○	•	520
○	○	•	○	5565
○	○	•	•	1189
○	•	○	○	2763
○	•	○	•	102
○	•	•	○	1184
○	•	•	•	469
•	○	○	○	114
•	○	○	•	1
•	○	•	○	14
•	○	•	•	23
•	•	○	○	281
•	•	○	•	19
•	•	•	○	77
•	•	•	•	370
Total Rules Using Position Keywords				
899	5265	8891	2693	12691
Individual Uses of Position Keywords				
995	5823	19026	7740	

TABLE II
INSTANCES OF NAMED BUFFER USE BY PROTOCOL.

Protocol	Named Buffer	Qty of Uses
HTTP	http.uri	10426
	http.method	6449
	http.request_body	2916
	http.host	2668
	http.header_names	2550
	http.user_agent	2354
	http.header	1251
	http.stat_code	813
	http.content_type	605
	http.request_line	326
	http.cookie	271
	http.accept	126
	http.start	121
	http.response_body	120
	http.content_len	110
	http.connection	109
	http.referer	102
	http.accept_enc	56
	http.protocol	36
	http.location	33
	http.accept_lang	28
	http.server	22
	http.response_line	21
	http.stat_msg	8
	HTTP Total	31521
DNS	dns.query	5959
	DNS Total	5959
TLS	tls.sni	929
	tls.cert_subject	816
	tls.cert_issuer	279
	tls.cert_serial	69
	tls.certs	9
	tls.version	6
	tls.cert_fingerprint	1
	TLS Total	2109

URL portion of a request was used 10426 times, and the `http.method` buffer referencing the type of HTTP request (commonly GET or POST) was used 6449 times. For DNS only a single buffer (`dns.query`) was used a total 5959 times, and for TLS, 7 buffers were used a total of 2109 times. This data indicates that handling named buffers, especially for HTTP, is essential to generate testing data covering these three protocols.

V. PROPOSED EXTENSIONS & ENHANCEMENTS

We now discuss proposed extensions and enhancements to SOUNDTHEALARM with the goal of allowing it to generate testing data for the majority of rules in the Proofpoint Emerging Threats Community Ruleset, as well as to provide meaningful feedback to a user while they write rules describing malicious traffic.

A. Generating Data for Named Buffers

At present SOUNDTHEALARM is unable to synthesize attack traffic from rules which leverage named buffers. A named buffer specifies a region of a parsed packet for Suricata to examine. In order for a packet to even be considered by a rule with a named buffer such as those shown in Table II, the message must at the very least be parsed as the an instance of the specified protocol format such as HTTP or DNS. This constraint requires that any synthesized content be embedded at the appropriate location in a parsable packet.

The most direct approach to this constraint is to generate an appropriate parsable instance of the required message and to insert to synthesized content in the appropriate locations as illustrated in Figure 10. Our analysis of the Proofpoint Emerging Threats Community Ruleset indicates that developing templates for three protocols (HTTP, DNS, and TLS) would enable coverage for the vast majority of the rules which were excluded from our earlier evaluations of SOUNDTHEALARM. We share early results from evaluating an approach to handle these buffers in Section VI.

B. Data Dependencies

The next limitation we seek to to address are those instances of rule keywords which look at data dependencies within messages. These include `isdataat` which looks to see if there is data at a specific byte location, `byte_jump` which interprets a location as a dynamic pointer to another location in a message, and `byte_test` which performs a computation such as xor or shift on bytes extracted at a location from the message and compares them to a known value. A synthesized message must ensure that the byte-values referenced by these

Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.

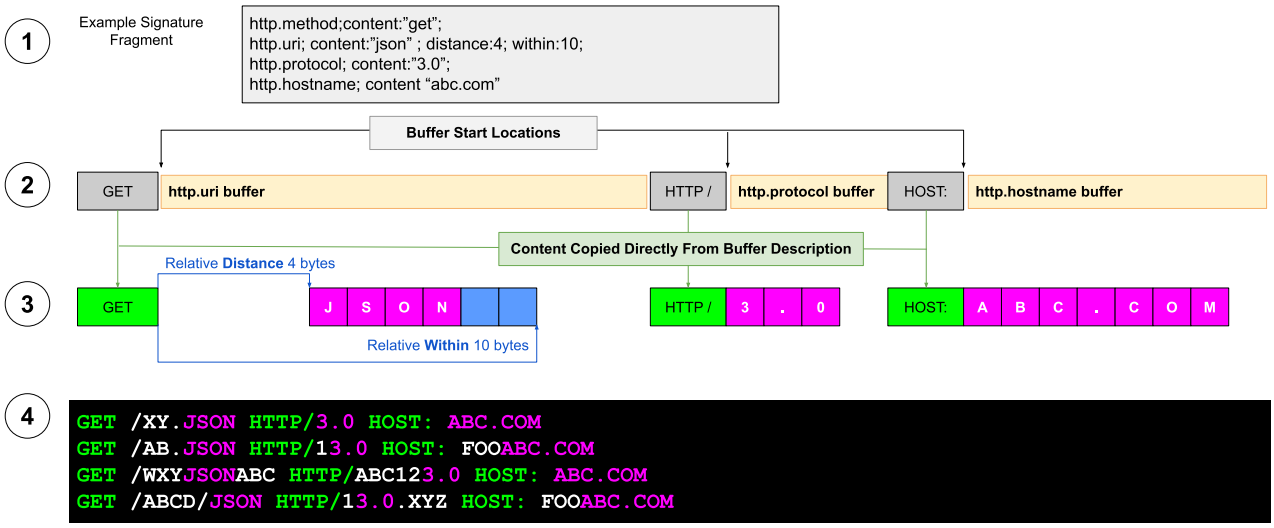

Fig. 10. Illustration of Suricata rule (1) using HTTP named buffers (2) to generate content and as reference points for constraint solving (3). Example synthesized byte-strings matched by the rule are shown in (4) with synthesized filler characters shown in white.

dependency keywords reflect the appropriate data and computational relationships.

C. Overlapping Content Constraints

As described previously in Section III-D, some rule instances explicitly reference the same portion of a message twice to reduce the resources required to scan large volumes of traffic. A synthesized message must respect that the overlapping regions take on byte-values which satisfy both references.

We believe encoding constraints on a byte-value level will allow SoundTheAlarm to address both data dependencies and overlapping content constraints. Additionally, we believe this extension will allow two useful enhancements related to the utility of the generated messages.

D. Minimization & Diversity

At present, SoundTheAlarm only solves constraints to position generated content as shown in Figure 5. Any gaps or unspecified bytes are either filled with random values, or copied from a user supplied example message.

We plan to explore performing synthesis such that a the generated message adheres closely to a template message. One way of expressing this is in terms of Levenshtein or edit-distance [13]. Levenshtein distance describes the number of transformations (insertions,deletions, and swaps) required to transform one string into another.[17] By attempting to synthesize a string with a minimum edit distance from a given example message, the synthesized message would both satisfy the constraints of the Suricata rule while incorporating as much of the existing content as possible: a desirable characteristic for producing human-readable and human-believable security test data. However, the major risk with this approach is that it may substantially increase the time required to generate a message.

Similarly, another desirable property of synthesized messages is representative diversity. Consider the following rule fragment.

```
content:"foo";content:"oobar";
```
(11)

The following strings all are matched by the rule.

```
foobar
fooobar
fooooobar
fooabcoobar
oobarfoo
oobarabcfooxyz
```
(12)

However, for assisting a user writing or debugging a rule synthesized examples which differ greatly from one another may be more useful than examples which simply increasingly repeat a single character through pumping. We propose using Levenshtein distance as a means to increase diversity in synthesized messages, such that all messages produced must exhibit a minimum edit distance from each other in the set. We plan to explore this in future work.

VI. Early Results

As a first step toward evaluating our proposed approach to handling named buffers, we implemented extensions to SoundTheAlarm to enable proof-of-concept support for the three most widely used named buffers: `http.uri`, `http.method`, and `http.host`. Our goal is to characterize the effort required to support named buffers across the three main protocols of interest from our analysis of rule use: HTTP, DNS, and TLS.

[17]https://en.wikipedia.org/wiki/Levenshtein_distance

Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.

We conducted our evaluation in the same manner as our earlier work [5]. We selected rules from the Proofpoint Emerging Threats community rule-set which used instances of the `http.uri`, `http.method`, or `http.host` named buffers, excluding those rules using keywords which SoundTheAlarm does not currently support. Our resulting evaluation set contained 6689 rules. All experiments were run on an instance of Google Colaboratory[18] with 2 Intel® Xeon® 2.20Ghz CPUs and 12GB of RAM. With our named buffer extensions, SoundTheAlarm was able to synthesize and trigger the Suricata IDS for 6145 rules (91.8 %). We consider this early result a strong positive indicator that SoundTheAlarm can be extended to handle named buffers in general. We plan a more detailed evaluation in future work, however we again noted interesting failure cases. In particular we observed rules which attempted to match white-space characters in the `http.uri` buffer generated messages successfully, but failed to trigger the IDS. An example fragment illustrating such white-space usage is shown below.

```
http.uri; content:"Flash Player.exe";     (13)
```

On deeper examination we confirmed that this is in fact a known bug in the Suricata IDS currently under review.[19][20] At present all rules using white-space in the `http.uri` named buffer will fail to match, allowing the malicious traffic to slip by undetected. This case in particular underscores the utility of our approach to help uncover bugs which might otherwise go unnoticed in critical security infrastructure.

VII. Related Work

Related work on evaluating IDS rules focuses on testing whether an IDS configuration [14] or measuring its ability to detect malicious network behavior [15], [16]. While some techniques replay packets, others generate entirely new messages. Approaches using these techniques include Erlacher et al.'s HTTP focused idsEventGenerator [17], Nidsbench[21], MUCUS [18], and IDSwakeup[22]. Cordero et al.'s ID2T [19]–[21] combines both approaches. ID2T [23] focuses on altering an existing network traffic capture to create testing data for an IDS. ID2T does this in two ways, first by adjusting the frequency of messages in order to trigger behavior based IDS systems. Second, ID2T can inject attack signatures into a PCAP, but is limited to approximately 13 different attacks coded by hand. Our approach, by contrast, generates attack data for thousands of attack rules, and with proposed extensions we believe can generate attack data for the majority of IDS rules. Other approaches to generated realistic IDS test data simply add and remove flows from a network stream [22] or use fixed profiles to generate network traffic, both benign and malicious [23]. SoundTheAlarm's synthesis approach uses a SMT-solver to automatically generate messages by leveraging positioning constraints extracted from Suricata rules. This approach differs from other works which simply copy content directly from the signature into a packet without considering order, or simply adjust overall packet frequency duplicating some packets while omitting others. SoundTheAlarm creates stateful sequences of packets by automatically wrapping generated messages in transport layer protocols. Other tools are less flexible. SoundTheAlarm was originally developed as a proof-of-concept cyber-deception tool [5] meant to deceive adversaries as to the presence of competitors on the network.

Other work related to generating network traffic focuses on creating realistic network messages, either for the purposes of deceiving of adversaries [24] or to skew trends in traffic flow data [25]. While these are instances of malicious traffic, they are more appropriate for testing either a human analyst's ability to uncover a covert channel, or testing a behavioral system's ability to detect flow data departing from a baseline. Another approach, focusing on embedding an encrypted channel in otherwise innocuous messages is Dyer et al.'s work on format-tranforming encryption [26]. Yu et al's work [27] takes an executable program and regular expression describing malicious input–similar in concept to IDS signatures– to generate new attack strings as malicious input. Finally, Chandler et al.'s work [7] on botnet cyber-deception introduces a network approach for generating deceptive command and control traffic for a single botnet protocol. SoundTheAlarm instead focuses on generating test traffic for a wider variety of threats, behaviors, and protocols.

VIII. Conclusion

In this paper, we discussed results from earlier evaluations of SoundTheAlarm, a proof-of-concept, SMT-based synthesis technique to generate network traffic from IDS signatures. We propose using SoundTheAlarm to help network administrators validate that an IDS rule will function as expected and that an IDS is properly configured. We examined a corpus of over 30,000 rules written for the Suricata IDS over the course of 12 years. Finally, we outlined proposed extensions to our synthesis technique to make it suitable to generate network security testing data for a larger set of rules and presented early results from one such extension.

Acknowledgments

This material is based upon work partly supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-19-C-0073. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

[18] https://colab.research.google.com
[19] https://redmine.openinfosecfoundation.org/issues/2881
[20] https://github.com/OISF/suricata/pull/8509
[21] https://packetstormsecurity.com/UNIX/IDS/nidsbench
[22] https://github.com/SavSanta/idswakeup
[23] https://github.com/tklab-tud/ID2T

References

[1] A. Albakri, E. Boiten, and R. De Lemos, "Risks of sharing cyber incident information," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, pp. 1–10, 2018.

Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.

[2] L. O. Nweke and S. Wolthusen, "Legal issues related to cyber threat information sharing among private entities for critical infrastructure protection," in *2020 12th International Conference on Cyber Conflict (CyCon)*, vol. 1300, pp. 63–78, IEEE, 2020.

[3] A. Albakri, E. Boiten, and R. De Lemos, "Sharing cyber threat intelligence under the general data protection regulation," in *Privacy Technologies and Policy: 7th Annual Privacy Forum, APF 2019, Rome, Italy, June 13–14, 2019, Proceedings 7*, pp. 28–41, Springer, 2019.

[4] I. Corona, G. Giacinto, and F. Roli, "Adversarial attacks against intrusion detection systems: Taxonomy, solutions and open issues," *Information Sciences*, vol. 239, pp. 201–225, 2013.

[5] J. Chandler and A. Wick, "Deceptive Self-Attack for Cyber Defense," in *Proceedings of the 56th Hawaii International Conference on System Sciences*, 2023.

[6] J. S. White, T. Fitzsimmons, and J. N. Matthews, "Quantitative Analysis of Intrusion Detection Systems: Snort and Suricata," in *Cyber sensing 2013*, vol. 8757, pp. 10–21, SPIE, 2013.

[7] J. Chandler, K. Fisher, E. Chapman, E. Davis, and A. Wick, "Invasion of the Botnet Snatchers: A Case Study in Applied Malware Cyberdeception," in *Proceedings of the 53rd Hawaii International Conference on System Sciences*, 2020.

[8] V. Corey, C. Peterman, S. Shearin, M. S. Greenberg, and J. Van Bokkelen, "Network forensics analysis," *IEEE Internet Computing*, vol. 6, no. 6, pp. 60–66, 2002.

[9] M. Sarhan, S. Layeghy, N. Moustafa, and M. Portmann, "Netflow datasets for machine learning-based network intrusion detection systems," in *Big Data Technologies and Applications: 10th EAI International Conference, BDTA 2020, and 13th EAI International Conference on Wireless Internet, WiCON 2020, Virtual Event, December 11, 2020, Proceedings 10*, pp. 117–135, Springer, 2021.

[10] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Wesley Professional, 2002.

[11] L. G. Michael, J. Donohue, J. C. Davis, D. Lee, and F. Servant, "Regexes are hard: Decision-making, difficulties, and risks in programming regular expressions," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 415–426, IEEE, 2019.

[12] C. Chapman, P. Wang, and K. T. Stolee, "Exploring regular expression comprehension," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 405–416, IEEE, 2017.

[13] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, pp. 707–710, Soviet Union, 1966.

[14] N. J. Puketza, K. Zhang, M. Chung, B. Mukherjee, and R. A. Olsson, "A Methodology for Testing Intrusion Detection Systems," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 719–729, 1996.

[15] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, *et al.*, "Evaluating intrusion detection systems: The 1998 DARPA offline intrusion detection evaluation," in *Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00*, vol. 2, pp. 12–26, IEEE, 2000.

[16] J. McHugh, "Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as performed by Lincoln Laboratory," *ACM Transactions on Information and System Security (TISSEC)*, vol. 3, no. 4, pp. 262–294, 2000.

[17] F. Erlacher and F. Dressler, "How to test an ids? genesids: An automated system for generating attack traffic," in *Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity*, pp. 46–51, 2018.

[18] D. Mutz, G. Vigna, and R. Kemmerer, "An experience developing an ids stimulator for the black-box testing of network intrusion detection systems," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pp. 374–383, IEEE, 2003.

[19] C. G. Cordero, E. Vasilomanolakis, N. Milanov, C. Koch, D. Hausheer, and M. Mühlhäuser, "Id2t: A diy dataset creation toolkit for intrusion detection systems," in *2015 IEEE Conference on Communications and Network Security (CNS)*, pp. 739–740, IEEE, 2015.

[20] E. Vasilomanolakis, C. G. Cordero, N. Milanov, and M. Mühlhäuser, "Towards the creation of synthetic, yet realistic, intrusion detection datasets," in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pp. 1209–1214, IEEE, 2016.

[21] C. G. Cordero, E. Vasilomanolakis, A. Wainakh, M. Mühlhäuser, and S. Nadjm-Tehrani, "On generating network traffic datasets with synthetic attacks for intrusion detection," *ACM Transactions on Privacy and Security (TOPS)*, vol. 24, no. 2, pp. 1–39, 2021.

[22] D. Brauckhoff, A. Wagner, and M. May, "Flame: A flow-level anomaly modeling engine.," in *CSET*, 2008.

[23] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[24] S. Maucione, "Loose lips may better Air Force security with 'Prattle'," *Federal News Network*, 2017.

[25] A. Wick, "I Want Your Flow To Be Lies," in *FloCon*, 2017.

[26] 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*, pp. 61–72, 2013.

[27] F. Yu, M. Alkhalaf, and T. Bultan, "Generating Vulnerability Signatures for String Manipulating Programs using Automata-Based Forward and Backward Symbolic Analyses," in *2009 IEEE/ACM International Conference on Automated Software Engineering*, pp. 605–609, IEEE, 2009.

Authorized licensed use limited to: IEEE Xplore. Downloaded on July 27,2024 at 19:21:23 UTC from IEEE Xplore. Restrictions apply.