How to Test an IDS? GENESIDS: An Automated System for Generating Attack Traffic

Felix Erlacher and Falko Dressler
[erlacher,dressler]@ccs-labs.org
Heinz Nixdorf Institute and Dept. of Computer Science
Paderborn University, Germany

ABSTRACT

Evaluating the attack coverage of signature-based Network Intrusion Detection System (NIDS) is a necessary but difficult task. Often, live or recorded real-world traffic is used. However, firstly, real-world network traffic is hard to come by at larger scale and the few available traces usually do not contain application layer payload. Secondly and more importantly, it contains only very few realistic attacks. So, the question remains how to test a NIDS? We propose GENESIDS, a system that automatically generates user definable HTTP attacks and, thus, allows for straightforward creation of network traces (or live traffic) where the number of different detectable events is only confined by the given attack definitions. By using an input format that follows the Snort syntax, the system can take advantage of thousands of realistic attack definitions. Our system can be used in combination with traffic generators to maintain typical load patterns as background traffic. Our evaluation shows that GENESIDS is able to reliably produce a very broad variation of HTTP attacks. GENESIDS is available as Open Source software.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**;

1 INTRODUCTION

NIDS are the tool of choice when it comes to defend against the increasing number of threats in the Internet [22]. Established taxonomies (e.g., [5]) categorize NIDS according to the applied detection method: Anomaly-based NIDS use behavior-based techniques by defining a model of normal network behavior and then detecting deviations to this model [2]. Knowledge-based systems use a precise definition of attacks and match incoming traffic against this definition. The most widespread variants of knowledge-based systems are signature or rule-based NIDS. While the methods proposed in this paper can, to a certain extent, also be used for anomaly-based systems, we focus on rule-based NIDS. NIDS are not limited to detect network attacks only, but are also able to detect every other activity that might be noteworthy to a network operator. We use the term *attack* in this paper, but want to make clear that this term denotes all network activity and events that can be detected by NIDS.

When developing a NIDS, it has to be validated that it works as expected in all possible scenarios. There have been various publications that propose and summarize methods for evaluating NIDS (e.g., [11, 14]). All of them agree that a vital evaluation step is to test the attack coverage and the ability to identify attacks precisely. This is the evaluation part that we are concentrating on.

One option is to use real traffic from a live network or one of the publicly available network traces [9, 15]. Except for carrier grade networks to which very few researchers have access to, network traces are the first choice for such tests. Among other drawbacks [13], they typically do not contain application layer payload. The biggest problem, however, is the following: Realistic network traffic contains only a fraction of all possible attacks that NIDS should be able to detect. Thus, most publications about novel NIDS [4, 8], ours included, choose a subset of manually crafted attacks and multiply and distribute them over a given network trace, resulting in a so called mixed workload containing benign and malicious traffic [5]. The problem, and this is frequently also reflected in review comments, is that manually generated traffic may contain more attacks than real-world traffic, but it mostly contains only few unique attacks. All in all, it covers only a small subset of all possible real-world attacks. The resulting evaluations show that these attacks are detected in various constellations and the system possibly does not produce any false-positives, but it gives no convincing evidence of the overall coverage of the system.

To overcome this issue, we propose *GENESIDS* (Generating Events for Signature-based Intrusion Detection Systems), a generator that takes as input a set of user definable attack descriptions and then statefully generates network packets for every attack in the set. A NIDS should then be able to detect all or subsets of these attacks depending on its usage scenario. Our system reduces the time consuming task of manually crafting packets to a minimum by relying on Snort rules as attack definitions. The set of Snort rules guarantees a broad coverage of real-world attacks. Snort rules are partitioned in different categories, which makes it easy to choose the appropriate rules for the test scenario. Rare or unknown site specific attacks can be manually added by writing a corresponding rule. We ease a detailed evaluation by uniquely labeling every created attack packet.

To generate a mixed traffic set with realistic traffic, we propose to combine GENESIDS with a L4-L7 traffic generator. Cisco's T-Rex,[1] for example, can statefully generate traffic at up to 200 Gbit/sec based on real-world traffic templates. As all traffic is artificially created, the configuration and the resulting traffic traces can now

WTMC'18, August 20, 2018, Budapest, Hungary
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-5910-8/18/08…$15.00
https://doi.org/10.1145/3229598.3229601

[1]trex-tgn.cisco.com

be published and shared within the community. This increases the repeatability of the evaluation results and allows for a better comparison of different systems.

Our GENESIDS system is written in C++ and made publicly available as Open Source under the GPL license[2].

2 RELATED WORK

We see now many tools that can generate traffic precisely following given time or burst patterns at high data rates and with very reliable timing [6]. They are build for load tests and other performance evaluations. Some are also able to produce meaningful application layer payload. However, we are not aware of tools that can automatically generate attacks in form of real-world malicious payload.

The vast majority of publications of novel NIDS uses publicly available traffic traces [3, 10] for all evaluation experiments. However, all such traces are heavily debated regarding their accuracy, timeliness, and completeness [21]. This is different for traffic data-sets of contests or cyber-warfare exercises, where the goal is to penetrate computer systems over a network [20]. They do contain more attacks than real-world traffic traces, but the attacks are not very diverse because of the relatively low number of attacked systems and, thus, the overall sum of attacks is still not satisfying.

Another approach is to purposely create traffic data-sets [4, 7, 8]. These consist of completely handcrafted network packets or real-world traffic traces mixed with custom data containing attacks. This way it is possible to generate traces that contain the desired category of attacks. But, again, as the attacks are mostly handcrafted they are usually rather small in coverage and quite debatable in their completeness. This can further be coupled with penetration testing frameworks like Metasploit[3] to generate malicious network traffic for NIDS evaluation [16]. However, there is still the lack of a sufficiently high number of different attacks.

To summarize the above: current traffic generators and available traces do not contain a high enough number of *unique* attacks to allow for a complete attack coverage evaluation for NIDS. GENESIDS does not only offer the possibility to manually craft malicious HTTP requests, but, by accepting Snort rules, currently includes more than 8000 unique HTTP attacks out-of-the-box. GENESIDS does not solve the issue of realistic timings on commodity hardware. This problem has been solved by various traffic generators like the already mentioned moongen or T-Rex and we propose a combination of both (see Figure 1)

We are aware that GENESIDS might also be used for so called squealing attacks [17], where an attacker generates synthetic attacks to overwhelm a system (and its operator) with alarms. But since the advent of such attacks, NIDS have evolved from simple stateless pattern checkers to sophisticated stateful systems that incorporate multiple means to avoid such evasion techniques. Furthermore, firewalls have evolved to be able to defend against attacks like these.

3 GENESIDS

Our envisioned NIDS evaluation setup is sketched in Figure 1. A stateful L4-L7 traffic generator takes as input two traffic templates.

[2]github.com/felixe/idsEventGenerator
[3]www.metasploit.com

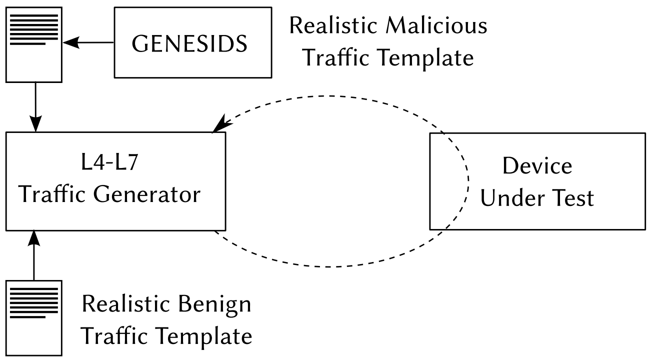
GENESIDS
Realistic Malicious Traffic Template
L4-L7 Traffic Generator
Device Under Test
Realistic Benign Traffic Template

Figure 1: Sketch of a NIDS evaluation experiment

One contains realistic benign traffic and the other template is build using the malicious TCP flows that we generate using GENESIDS. Depending on the application scenario, this traffic is mixed by the traffic generator and fed to the device under test. In this work, we focus on the generation of the malicious network traffic.

3.1 Overview

The goal of GENESIDS is to create malicious HTTP attack traffic to be used for testing rule-based NIDS. The largest set of machine-readable descriptions of HTTP attacks is contained in the different rule-sets for the NIDS Snort. Signature-based NIDS like Snort use Deep Packet Inspection (DPI) to detect attacks. Thus, a rule contains byte or ASCII patterns to be found in the traffic payload, to define an attack. For example, the well-known password cracker Brutus uses its name in the HTTP header. The rule to detect such an attack (rule sid: 26558), looks for the ASCII pattern “Mozilla/3.0 (Compatible) ;Brutus/AET” in the HTTP header. If GENESIDS is given this rule, it will create an HTTP packet containing this pattern in the HTTP header. If a NIDS receives this packet and this rule is part of its rule set, it should trigger an alert for this rule.

Because of the flexibility of the supported syntax (see also Section 3.3), the created traffic is not restricted to the available rules but can be adapted to the specific test case.

3.2 Input and Connection Management

GENESIDS accepts as input a text file with rules in Snort-like syntax. The first step is to parse the rule file. It will alert on rules that do not correspond to the accepted input syntax or warn if possible problems are detected. Afterwards it will go through the list of parsed rules and generate an HTTP request containing all patterns of the corresponding rule. This request is then sent to an HTTP Server address given at startup. GENESIDS will wait for a response (or the timeout) and then close the TCP connection and open a new TCP connection for the next HTTP request.

The rule-syntax allows for custom creation of the following HTTP fields: method, uniform resource identifier (uri), header names and values, request cookies, and the client body. The corresponding field content can be given as text or in form of a regular expression in the Perl Compatible Regular Expression (PCRE) syntax (detailed description in Section 3.3).

We use the well known `libcurl` library to create legitimate packets. To create an HTTP request that matches the given content pattern, we copy the pattern in the corresponding field of the generated request, the only change made is the exchange of hex patterns with their corresponding ASCII representation.

The generation of requests for rules containing PCRE patterns is not so straightforward. Here, we have to come up with a string that matches the given expressions. To generate a single string matching a given PCRE, we use the Python command exrex.[4] It allows for the creation of random matching strings for a given PCRE. It only has very few restriction on the given expression, e.g., it only supports 7-bit hex chars, it does not support some combination of quantifiers, and it does not support positive look-ahead. Whenever possible, we try to replace not supported quantifiers with equivalent ones. Because some NIDS might be sensitive to certain unusual characters, we exchange some patterns that may lead to such characters to equivalent patterns that can not lead to unusual characters. For example, the pattern ”SELECT.*FROM“ is changed to ”SELECT[a-z]FROM“ before handing it to the exrex command. GENESIDS can still be used for robustness tests against unusual characters by stating these characters explicitly in the PCRE or a content pattern. GENESIDS issues a warning if a pattern possibly generates irritating HTTP content, e.g., if a character in the HTTP uri is reserved according to the standard [1].

GENESIDS uses a stateful approach: For every single HTTP request a full fledged TCP connection is established. This is essential because most NIDS ignore and do not analyze packets if they are not part of a correct TCP connection. GENESIDS expects a response before ending the TCP connection, so the recommended standard procedure is to have an HTTP server running as a counterpart. For obvious reasons, it is not advisable to use a public web server.

3.3 Rules

One of the largest detailed description of real-world attacks is the set of available rules for the signature-based NIDS Snort [19]. Within this set, we focus on HTTP, which is by far the most used application layer protocol [18] and also covers most attacks to be detected by NIDS.

To take advantage of the many readily available Snort rules, we decided to closely follow the Snort rule syntax. GENESIDS accepts rules in Snort syntax (as of Snort 2.9.11). As GENESIDS produces HTTP requests, it accepts only HTTP related Snort rules with supported fields. The only supported rule action is `alert` as these are the only rules triggering an alert. The only supported protocol is `tcp`. The exact syntax is defined in the Snort manual[5]. The accepted fields and keywords are described below.

Snort rules consist of a rule header followed by the rule options. The rule header starts with a rule action, followed by the protocol and the source and destination addresses. HTTP responses are issued by an HTTP server and, thus, not controlled by GENESIDS. The rule options (enclosed in parenthesis) consist of a set of keywords possibly followed by a value. If GENESIDS detects a keyword which is unsupported and would alter the meaning of the rule it issues a warning (e.g., for the `depth` keyword, which is used to specify how far into a field Snort should search).

Mandatory keywords:

- `msg:` Following this keyword is the quoted event message description that accompanies the triggered event.

At least one of the following keywords must be present:

- `content:` This keyword is followed by a quoted pattern to look for in the HTTP part of the packet. As it is not limited to HTTP in Snort, it must be followed by one of the supported HTTP content-modifier keywords (see list below).
- `uricontent:` This keyword is followed by a quoted pattern to look for in the HTTP uri.
- `PCRE:` This keyword is followed by a PCRE enclosed in two slashes. As expressions are not limited to HTTP in Snort, it must be followed by one of the supported HTTP modifiers listed below.

Exactly one of the following HTTP related content-modifier keywords has to restrict the pattern following the `content:` or `pcre:` keyword to an HTTP part. If not the corresponding rule will not be accepted by GENESIDS. In parenthesis is the equivalent modifier for PCRE.

- `http_method (M)` - the preceding content or pattern will be inserted as the HTTP method. As this is a mandatory field in HTTP requests it defaults to “GET”.
- `http[_raw]_uri (U, I)` - the preceding content pattern will be inserted in the HTTP uri. Default is “/”.
- `http[_raw]_header (H, D)` - the preceding content pattern will be inserted as an own HTTP header field. GENESIDS always generates a name-value pair, adding a dummy name or value if missing in the pattern.
- `http[_raw]_cookie (C, K)` - the preceding content pattern will be set as a cookie in the HTTP request.
- `http_client_body (P)` - the preceding content pattern will be set in the HTTP request body.

The only Snort HTTP content-modifier keywords that are not supported are `http_stat_msg (Y)` and `http_stat_code (S)` as they are only used in HTTP responses. The keyword `flow:` is followed by other keywords denoting, e.g., the direction of the TCP flow of the packet. Again, this keyword is only used to check if the rule is looking for an HTTP response. So if keywords like `flow:from_server` or `to_client` are used, a warning is issued and the rule is ignored.

The last two meaningful keywords for GENESIDS rules are `sid:` and `rev:`. Both are mandatory. The `sid:` keyword is used to assign a unique identifier and the `rev:` denotes the revision number of the rule. They are both followed by a number. To ease the evaluation of attack coverage experiments, GENESIDS adds the sid number of the given rule as an additional HTTP header to the HTTP request in the form of `Rulesid:<sid number>`. This enables an automated test of which packet should have triggered which alert.

A minimal example rule could look like the following:

```
alert tcp any any -> any any (msg:"This is
an example rule"; content:"GET";
http_method; uricontent:"|2F|evil.jpg";
sid:1234567; rev:0;)
```

[4]pypi.python.org/pypi/exrex
[5]snort.org/documents

This rule contains two patterns. First the string ”GET“ is defined to be found in the HTTP method field and, second, the hex byte ’2F’ followed by the string ”evil.jpg“ is to be found in the HTTP uri field.

This rule generates a TCP flow with the following HTTP request (textual representation). Note the conversion of the hex byte ’2F’ to the corresponding ASCII sign “/”:

```
GET /evil.jpg HTTP/1.1
Host: 10.0.0.1
Rulesid: 1234567
```

The TCP flow will also contain the response from the server but is not depicted here. An equivalent rule could also be defined by using the `content:` keyword followed by the `http_uri` content modifier instead of the `uricontent:` keyword.

As already stated, patterns can also be expressed as a PCRE. Example:

```
alert tcp any any -> any any (msg:"This is
an example rule"; content:"POST";
http_method; pcre:"/AttackBody-V[0-9].*/P";
sid:2345678; rev:0;)
```

Here the string ”POST“ has to be found in the HTTP method field and the second pattern must match the PCRE ”AttackBody-V[0-9].*“. This pattern matches strings starting with ”AttackBody-V“ followed by a single digit, followed by a random string. The ’P’ after the regular expression denotes that this pattern has to be found in the HTTP body field. The resulting traffic generated by GENESIDS could look like the following HTTP request (textual representation):

```
POST / HTTP/1.1
Host: 10.0.0.1
Rulesid: 2345678
Content-Length: 14

AttackBody-V1x
```

Please note that we are generating one random string of a possibly infinite set of matching strings for the PCRE. This implies that the generated HTTP request might differ in another run. More examples can be found on the author’s homepage[6].

3.4 Limitations

As already mentioned, GENESIDS has been built to craft HTTP requests according to the given rules. Thus, no rules other than HTTP request related rules are accepted. Also, GENESIDS will only produce legal HTTP requests.

GENESIDS also accepts hex encoded characters the same way Snort does. But only the first 128 readable hex chars plus `\n` and `\r` are allowed. PCRE expressions are limited to what the exrex command accepts. We will show in the evaluation that these limitation only have a marginal impact on the acceptance of real-world rules.

The set of Snort-rules contains most of today’s relevant network attacks. However, it does not contain rare or unknown attacks and events.

[6]www.ccs-labs.org/~erlacher/resources/

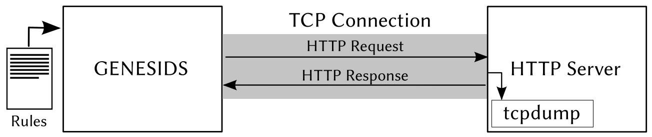

Figure 2: Experimental setup for traffic generation

4 EVALUATION

In the following, we show that GENESIDS can generate a broad variety of possible attacks and that the generated attacks reliably trigger the expected alerts in a NIDS. All the material used in the following experiments is available online[6] to allow for better comparison and repeatability.

To be able to test against real attack definitions, we used Snort rules (as of 22 January 2018) from the following sources:

- Snapshot 29111 provided to Snort.org subscribers;
- the community rule-set from Snort.org; and
- all rules from the Emerging Threats rule-set.[7]

This guarantees that a broad variety of up-to-date attacks are included in our tests. From this rule-set, we use every rule that is applicable to HTTP requests. We only exclude rules using some special Snort features. The final rule-set for the experiments consists of 8101 different rules.

The evaluation consists of two experiment steps. First, we generate traffic with GENESIDS using the above rule-set as input. We capture this traffic with the network capturing tool *tcpdump*.[8] Secondly, we use the captured traffic and analyze it with Snort[9] as the NIDS under test. The underlying assumption is that every rule written for Snort should trigger the corresponding alert in Snort.

The traffic generation experiment depicted in Figure 2 is conducted as follows. GENESIDS generates one HTTP-request per rule from the rule-set and sends the generated HTTP requests to an HTTP Server (Apache 2.4.10), which answers with the corresponding HTTP responses. As the majority of the HTTP requests contain an uri for a resource that is not available, the most common response in our experiment is a ‘404, not found’.

We then analyze the captured network traffic with Snort: We run Snort in IDS mode and configure it to take as input the captured pcap file and the same rule-set used by GENESIDS in the first experiment step. The only relevant changes made to the default configuration file are the deactivation of the reputation preprocessor and the logging of all alerts (Snort by default does not log more than 8 alerts per packet). To avoid Snort skipping packets with checksum errors, we used the `-k none` switch to turn it off.

After analysis, the Snort alert file contains an entry for every alert triggered. An entry, among other information, consists of the sid of the rule triggering the alert, and the TCP port of the connection belonging to the packet triggering the alert. As stated before, GENESIDS uses one TCP connection per request. Thus, a tuple composed of the TCP port number and the rule sid is unique as long as no TCP source ports are used twice in the same experiment

[7]rules.emergingthreats.net/open/snort-2.9.0/emerging-all.rules
[8]tcpdump.org
[9]Snort version 2.9.11, build from source

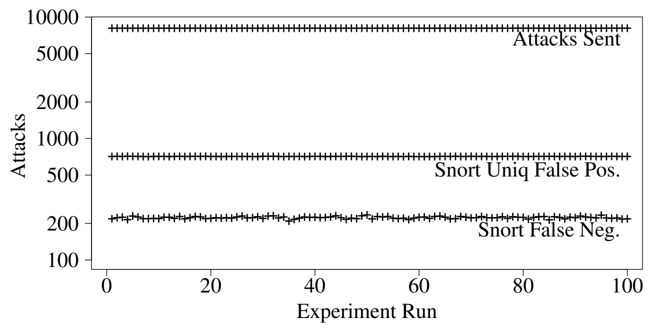

Figure 3: Results of 100 experiment repetitions

run. This allows to assign an HTTP request generated by GENESIDS to an alert triggered by Snort using the port-sid tuple.

For PCRE rules, we generate one random string that matches this expression. Thus, the generated requests will differ between different runs. To show that the numbers are stable over multiple runs, and to make sure we detect most flaws, we repeated this experiment 100 times.

The results are shown in Figure 3. In all runs, GENESIDS sends out exactly 8101 HTTP requests, one for every rule in the given rule file. It always used 8101 different TCP source ports. Comparing the generated TCP port-sid tuples with the port-sid tuples of the triggered alerts shows that on average more than 97 % of these tuples triggered the correct corresponding alert. This shows that GENESIDS generates HTTP requests triggering the correct alert in almost all cases.

On average Snort triggered 2847 false positive alerts. With false positives we denote alerts triggered by a TCP flow with a port-sid tuple that was not supposed to trigger that exact event. The majority of these false positives was triggered by the same few rules. On average the false positive alerts were triggered by 712 different unique rules. The fact that Snort triggers a high number of false positives is not new and caused by so called overlapping signatures [12]. In a normal environment, the used Snort rules are chosen very carefully because a high number of rules has a devastating negative impact on the packet throughput performance and, as is shown above, increases the number of false positives.

On average only 223 HTTP flows (2.8 %) did not trigger the corresponding event in at least one experiment (denoted as false negatives). To make sure we investigate every false negative, we picked all rules of all experiments that did not trigger an event. Over all 100 experiment runs, there was a total of 363 unique rules causing at least once a false negative. We manually inspected all and divided them into two categories: rules that did not trigger an event in any of the experiment runs (rules that caused only false negatives) and rules that triggered the correct event in at least one of the experiment runs (rules that also caused true positives). Rules from the latter category all have a PCRE pattern that, for reasons we will explain, generated at least once a pattern that did trigger the correct alert. Rules from the first category however, must contain an inherent issue that prevented the rule from triggering an alert.

We start with the first category of 179 false negatives that never triggered the correct alert:

- 61 rules use a content pattern to filter out possibly matching requests and only if this pattern matches a more detailed PCRE applied to refine the search. The PCRE is enclosed by ^ and $, meaning the string generated should be the only string in that field. However, GENESIDS adds, for example, all uri patterns to one long uri and, thus, the ^ and $ symbols do not match anymore and cause the rule to fail.
- 18 rules define multiple headers in one content or PCRE pattern. This does not work because GENESIDS adds headers according to the standard, which defines a \r\n in between headers.
- 28 rules are explicitly searching for one or multiple \r\n in a place. Again, GENESIDS/libcurl only support standard compliant HTTP requests. For example, a rule explicitly searching for the last field in the HTTP header with \r\n\r\n at the end of the pattern will always fail because currently the last header field is always the "Rulesid:" field. The same holds if a rule is searching for a line break at the beginning of the HTTP URI.
- 14 rules define explicitly \\ in a uri. In the standard configuration, Snort replaces them by a single \.
- 6 rules search for a pattern explicitly at the beginning of the client body. Because Snort only searches the client body if the length is greater than 6, we fill the client body with 5 characters before adding other client body data from rules.
- 15 rules ask for the not supported word character \w in a PCRE. Exrex simply ignores this pattern and, thus, produces a wrong string.
- 6 rules use the word boundary anchor \b in a PCRE at a position where it can not be applied. In all 6 cases, it positioned at the beginning of an uri pattern where we add another uri pattern first, without any non-word character in between.
- 5 of the rules use the reserved character # in a uri, which is ignored by libcurl.
- 5 of the rules use the disallowed character + in a uri. libcurl adds this character literally but Snort matches this character only to a whitespace character.
- The remaining 21 rules have similar singular problems, which we do not describe individually.

In the following, we explain the second category of false negatives. These are the 184 rules that at least in one experiment run triggered the correct alert, but failed in all other experiments. All rules in this category contain at least one PCRE. The reason for the failure is the random string generation to match this PCRE.

123 of these PCREs contain a character class starting with a negation, e.g., [^\x2F], which stands for every character but a /. In all the cases that did not trigger the correct event, these rules produced a non-supported character.

46 of the remaining rules contain a PCRE with a single . character, which, in the cases the rule did not trigger an alert, produced an unsupported character. We want to emphasize that GENESIDS already replaces the most common occurrences of [^ pattern and . character combinations. The ones described above are the very few instances, which are not covered by the replacements.

The remaining 15 rules contain PCRE patterns including alternations (separated with the | character), and one of the alternating patterns contained one of the problems described in first category of false negatives.

Overall, we can conclude that GENESIDS exactly fulfills its high expectations. It is able to automatically produce attack traffic that can be used (and possibly merged with background traffic) to test NIDS.

5 CONCLUSION

In this paper, we presented a novel system for generating attacks for NIDS evaluation named GENESIDS. By using an input format similar to Snort, one can not only generate self written attacks, but take advantage of thousands of readily available attack definitions maintained by a large community. More than 97 % of the attacks generated from 8101 different real-world rules triggered the desired alert. This makes us confident that GENESIDS works in a very precise and reliable way. We can also confirm that the limitations stated above only affect very few rules. We showed that GENESIDS can be used for attack coverage tests of signature-based NIDS. We are, however, confident that this tool can also be used for tests of anomaly-based NIDS or robustness tests. In future work we will show how to generate mixed traffic using GENESIDS and an L4-L7 traffic generator.

REFERENCES

[1] Tim Berners-Lee, Roy Fielding, and Larry Masinter. 2005. *Uniform Resource Identifier (URI): Generic Syntax.* RFC 3986. IETF.

[2] Monowar H Bhuyan, Dhruba Kumar Bhattacharyya, and Jugal K Kalita. 2014. Network Anomaly Detection: Methods, Systems and Tools. *IEEE Communications Surveys & Tutorials* 16, 1 (2014), 303–336. https://doi.org/10.1109/SURV.2014.072114.00000

[3] Alexander Branitskiy and Igor Kotenko. 2015. Network Attack Detection based on Combination of Neural, Immune and Neuro-Fuzzy Classifiers. In *18th International Conference on Computational Science and Engineering (CSE 2015)*. IEEE, Porto, Portugal, 152–159. https://doi.org/10.1109/CSE.2015.26

[4] Waleed Bul'ajoul, Anne James, and Mandeep Pannu. 2015. Improving Network Intrusion Detection System Performance through Quality of Service Configuration and Parallel Technology. *Elsevier Journal of Computer and System Sciences* 81, 6 (Sept. 2015), 981–999. https://doi.org/10.1016/j.jcss.2014.12.012

[5] Hervé Debar, Marc Dacier, and Andreas Wespi. 1999. Towards a Taxonomy of Intrusion-Detection Systems. *Elsevier Computer Networks* 31, 8 (April 1999), 805–822. https://doi.org/10.1016/S1389-1286(98)00017-6

[6] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. 2015. Moongen: A Scriptable High-Speed Packet Generator. In *15th Internet Measurement Conference (IMC 2015)*. ACM, Tokyo, Japan, 275–287. https://doi.org/10.1145/2815675.2815692

[7] Felix Erlacher and Falko Dressler. 2017. High Performance Intrusion Detection Using HTTP-Based Payload Aggregation. In *42nd IEEE Conference on Local Computer Networks (LCN 2017)*. IEEE, Singapore, 418–425. https://doi.org/10.1109/LCN.2017.18

[8] Felix Erlacher and Falko Dressler. 2018. FIXIDS: A High-Speed Signature-based Flow Intrusion Detection System. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*. IEEE, Taipei, Taiwan.

[9] Romain Fontugne, Pierre Borgnat, Patrice Abry, and Kensuke Fukuda. 2010. Mawilab: Combining Diverse Anomaly Detectors for Automated Anomaly Labeling and Performance Benchmarking. In *6th International Conference on emerging Networking Experiments and Technologies (CoNext 2010)*. ACM, Philadelphia, PA. https://doi.org/10.1145/1921168.1921179

[10] Shi-Jinn Horng, Ming-Yang Su, Yuan-Hsin Chen, Tzong-Wann Kao, Rong-Jian Chen, Jui-Lin Lai, and Citra Dwi Perkasa. 2011. A Novel Intrusion Detection System Based on Hierarchical Clustering and Support Vector Machines. *Elsevier Expert Systems with Applications* 38, 1 (Oct. 2011), 306–313. https://doi.org/10.1016/j.eswa.2010.06.066

[11] Elizabeth B. Lennon. 2003. *Testing Intrusion Detection Systems.* Information Technology Laboratory Bulletin Jul2003. National Institute of Standards and Technology. 1–4 pages.

[12] Frederic Massicotte and Yvan Labiche. 2011. An Analysis of Signature Overlaps in Intrusion Detection Systems. In *41st International Conference on Dependable Systems & Networks (DSN 2011)*. IEEE, Hong Kong, China, 109–120. https://doi.org/10.1109/DSN.2011.5958211

[13] John McHugh. 2000. Testing Intrusion DetectionSystems: 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)* 3, 4 (Nov. 2000), 262–294. https://doi.org/10.1145/382912.382923

[14] Aleksandar Milenkoski, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Bryan D Payne. 2015. Evaluating Computer Intrusion Detection Systems: A Survey of Common Practices. *Comput. Surveys* 48, 1 (Sept. 2015), 12. https://doi.org/10.1287/deca.1030.0001

[15] Nour Moustafa and Jill Slay. 2016. The Evaluation of Network Anomaly Detection Systems: Statistical Analysis of the UNSW-NB15 Data Set and the Comparison with the KDD99 Data Set. *ACM Information Security Journal: A Global Perspective* 25, 1-3 (Jan. 2016), 18–31. https://doi.org/10.1080/19393555.2015.1125974

[16] Khalid Nasr, Anas Abou-El Kalam, and Christian Fraboul. 2012. Performance Analysis of Wireless Intrusion Detection Systems. In *5th International Conference on Internet and Distributed Computing Systems (IDCS 2012)*. Springer, Fujian, China, 238–252. https://doi.org/10.1007/978-3-642-34883-9_19

[17] Samuel Patton, William Yurcik, and David Doss. 2001. An Achilles' Heel in Signature-Based IDS: Squealing False Positives in SNORT. In *4th International Symposium on Recent Advances in Intrusion Detection (RAID 2001)*. Springer, Davis, CA.

[18] Philipp Richter, Nikolaos Chatzis, Georgios Smaragdakis, Anja Feldmann, and Walter Willinger. 2015. Distilling the Internet's Application Mix from Packet-Sampled Traffic. In *Passive and Active Measurement Conference (PAM 2015)*. Springer, New York City, NY. https://doi.org/10.1007/978-3-319-15509-8_14

[19] Martin Roesch. 1999. Snort: Lightweight Intrusion Detection for Networks. In *13th USENIX Conference on System Administration (LISA 1999)*. Seattle, WA, 229–238.

[20] Benjamin Sangster, TJ O'Connor, Thomas Cook, Robert Fanelli, Erik Dean, Christopher Morrell, and Gregory J Conti. 2009. Toward Instrumenting Network Warfare Competitions to Generate Labeled Datasets. In *2nd workshop on Cybersecurity and Test (CSET 2009)*. Usenix, Montreal, Canada. https://doi.org/10.1.1.159.7948

[21] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*. INSTICC, Funchal, Portugal, 108–116. https://doi.org/10.5220/0006639801080116

[22] Benjamin Stritter, Felix Freiling, Hartmut König, Rene Rietz, Steffen Ullrich, Alexander von Gernler, Felix Erlacher, and Falko Dressler. 2016. Cleaning up Web 2.0's Security Mess - at Least Partly. *IEEE Security & Privacy* 14, 2 (March 2016), 48–57. https://doi.org/10.1109/MSP.2016.31