Automating Network Security Analysis at Packet-level by using Rule-based Engine

Martin Holkovič
Brno University of Technology
Brno, Czech Republic
iholkovic@fit.vutbr.cz

Ondřej Ryšavý
Brno University of Technology
Brno, Czech Republic
rysavy@fit.vutbr.cz

Jindřich Dudek
Brno University of Technology
Brno, Czech Republic
xdudek04@fit.vutbr.cz

ABSTRACT

When a network incident is detected, a network administrator has to manually verify the incident and provide a solution to stop the incident from continuing and prevent similar incidents in the future. The network analysis is a time-consuming and labor-intensive activity which requires good network knowledge. Creating a solution which automates the administrator's work can dramatically speed up the analysis process and can make the whole process easier for less experienced administrators. In this paper, we describe a method that uses a predefined set of rules to identify incident patterns. Though this principle is used by many security tools, the new aspect is that the presented approach uses the Wireshark tool which is well known among the administrators, and it is expressive enough to specify complex relations among source data thus being able to detect quite sophisticated attacks. The created rule's format uses the same language as the Wireshark filters.

CCS CONCEPTS

• **Networks → Error detection and error correction**; **Network monitoring**; *Network security*; • **Security and privacy** → *Intrusion detection systems.*

KEYWORDS

Network security, network monitoring, anomaly detection, threat detection, network forensics

ACM Reference Format:
Martin Holkovič, Ondřej Ryšavý, and Jindřich Dudek. 2019. Automating Network Security Analysis at Packet-level by using Rule-based Engine. In *6th Conference on the Engineering of Computer Based Systems (ECBS '19), September 2–3, 2019, Bucharest, Romania.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3352700.3352714

1 INTRODUCTION

Nowadays, one of the most important responsibilities of a network administrator is to secure a computer network against cyber threats. A threat can have a lot of different formats: brute-force login attempt, DDoS attack, phishing attack, data breach, and many others. Most of the protective actions are done proactively to prevent threats realization, but despite all the effort and implemented measures, some incident can occur in the network anyway. Then a network administrator has to identify, locate and stop incidents inside the network [23]. In this paper, we will use an example of a user who cannot access the Internet. The administrator has to investigate this problem and fix the user's Internet connection.

A very common way of network traffic analysis is using some network packet analyzer, e.g., *Wireshark* [18]. The analyzer process captured network traffic (PCAP files) and decodes individual packets. The administrator analyzes available information, checks the transferred data and compares it with expected values (e.g., from RFC standards). This manual process is time-consuming and requires a good knowledge of network protocol and various network technology. This paper aims to propose a method to automate this process.

1.1 Contribution

This paper describes a solution for automating the time-consuming and labor-intensive network analysis usually done manually by network administrators [24]. This automation will dramatically decrease the time required for analyzing PCAP files and allows less experienced administrators to check the files in the same way as a more experienced administrator would do.

To create a solution that can be easily used by network administrators, one needs to consider the following assumptions:

- Administrators often do not have enough programming skills;
- Adding support for new protocols should not require to create new protocol dissectors;
- Extending the tool with new detection capabilities must be straightforward for administrators;
- Administrators can work with packets as they are used to with manual analysis.

The presented solution can be compared to existing tools and other alternative approaches:

- Use an existing IDS or forensic tools: These tools are not made to work with network data which administrators usually work with. Adding new incident detection requires programming skills or understanding the nontrivial definition language. These tools usually have limited support of network protocols and protocol fields (even if the protocol is supported, not all fields are available). The usual workflow is to process each network flow individually to speed up the process which makes detection across multiple protocols and flows more complicated. More details about existing solutions are in Section 3;

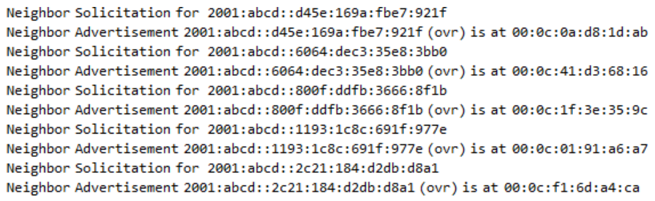
- Implement a new analysis tool from scratch: Creating a complete tool would require a lot of effort (time, people, money) and even after the tool would have been created, it would require additional human resources to add support for new protocols or new protocol versions;
- Use the machine learning approach [17]: This approach would require programming skills from administrators while extending the tool and the approach would be more like a magic black box instead of a straightforward approach.

It is really important to clarify that our goal is not to create another IDS tool or to create an alternative to existing IDS tools (these tools are more described in Section 3). IDS tools already contain a huge amount of predefined rules to detect well-known attacks and are highly optimized to maximize performance. Our goal is to create a complementary tool which would be used alongside existing IDS tools. This tool would allow administrators to automate their manual incident analysis by specifying incidents using an easy to understand and use threat specification language.

2 SECURITY ANALYSIS OF NETWORK COMMUNICATION

The goal of the network security analysis is to detect security incidents inside the network and provide as much information about these incidents as possible [11]. Example of such information can be the source, progress, and consequences of the incident. Finding evidence for an incident is also called a forensic analysis.

In our example, the Internet is not working because the victim's computer cannot be assigned an IPv6 address. Some other device is blocking the address assignment, and after several attempts, the victim's computer gives up. As illustrated in Figure 1, the reason for this is that an attacker applies a denial of service attack by misusing the *Duplicate address detection* mechanism.

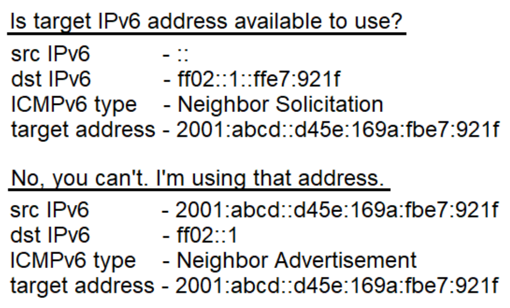


```
Neighbor Solicitation for 2001:abcd::d45e:169a:fbe7:921f
Neighbor Advertisement 2001:abcd::d45e:169a:fbe7:921f (ovr) is at 00:0c:0a:d8:1d:ab
Neighbor Solicitation for 2001:abcd::6064:dec3:35e8:3bb0
Neighbor Advertisement 2001:abcd::6064:dec3:35e8:3bb0 (ovr) is at 00:0c:41:d3:68:16
Neighbor Solicitation for 2001:abcd::800f:ddfb:3666:8f1b
Neighbor Advertisement 2001:abcd::800f:ddfb:3666:8f1b (ovr) is at 00:0c:1f:3e:35:9c
Neighbor Solicitation for 2001:abcd::1193:1c8c:691f:977e
Neighbor Advertisement 2001:abcd::1193:1c8c:691f:977e (ovr) is at 00:0c:01:91:a6:a7
Neighbor Solicitation for 2001:abcd::2c21:184:d2db:d8a1
Neighbor Advertisement 2001:abcd::2c21:184:d2db:d8a1 (ovr) is at 00:0c:f1:6d:a4:ca
```

Figure 1: The output from the Wireshark tool. The client is trying to assign a unique address, but each time the client checks the availability of the address with the Neighbor Solicitation message, the attacker blocks its assignment by sending the Neighbor Advertisement message.

The primary purpose of the gathering information about the incidents is to reduce damage to the affected company owning the network infrastructure and services [10]. The harm can be caused by interrupting normal business processes, data breach, wasting resources exploited by a botnet. After the incident is solved, it is very important to prevent similar incidents from happening again.

Each analysis should consist of several actions. It is necessary to find out who or what is the source and destination of the attack (or generally incident) [11]. The source can be a corrupted device inside the network or person on the other side of the globe. Another action is identifying exploited vulnerabilities and finding the solution for fixing them. It is also necessary to identify which parts of the network were exposed by the attack and can potentially be abused.



```
Is target IPv6 address available to use?
src IPv6        - ::
dst IPv6        - ff02::1::ffe7:921f
ICMPv6 type     - Neighbor Solicitation
target address - 2001:abcd::d45e:169a:fbe7:921f

No, you can't. I'm using that address.
src IPv6        - 2001:abcd::d45e:169a:fbe7:921f
dst IPv6        - ff02::1
ICMPv6 type     - Neighbor Advertisement
target address - 2001:abcd::d45e:169a:fbe7:921f
```

Figure 2: In our example, we can see that for each Network Solicitation (NS) message an adequate Network Advertisement (NA) message is received. All NA messages have a different source IPv6 address. The administrator should find the location of this device and continue with the investigation by further analysis of the device's activities.

From a technological point of view, the security analysis is problematic because it requires information about transferred data inside the network. One possible approach is to capture all packets and perform full stack analysis. The problem is that networks usually transfer a lot of data and it is impossible to analyze all data in necessary detail. Another approach is to ignore layer 7 information completely and use only data from lower layers, for example, Netflow data [9]. With this approach, it is possible to process a huge amount of data traffic, but the analysis is limited only to data from lower layers.

The analysis is problematic from a personal point of view too. Computer networks consist of many network devices, protocols, applications, and the analysis process requires good knowledge for all of these elements. Even if an analyst has the necessary knowledge, she must understand the analyzed network very well, because each network environment is different and has some specific properties. Another problem is that the analysis can be a very time-consuming activity even for a skilled administrator. On the other hand, the analysis should be finished as soon as possible to prevent the incident from doing more damage [10]. This puts an administrator under time pressure which can lead to mistakes.

From our point of view, the network analysis process should be split into two parts:

- **Detect possible incident** - By using less complex analysis, it is possible to constantly analyze all the transferred traffic and search for an anomaly. E.g., anomaly detection over Netflow data, IDS system over transferred data payloads or keywords detection inside the server log files [26]. Usually, most of the anomalies will not be a security incident, but just a false-positive detection. Based on the anomaly parameters, like IP addresses, a small portion of network traffic is captured and saved into a PCAP file for further analysis.

- **Execute complex analysis** - Captured PCAP file is analyzed in detail to determine if the anomaly is a network security incident or not. This process is usually done manually by an administrator, and it is very time- and labor-consuming. The result of this step is a report of a security incident with as much information as possible. Emphasis is on accurate results, not quick results.

3 RELATED WORK

Intrusion detection systems (IDS) employ different mechanisms to detect potentially harmful communication. Rule-based IDS systems use predefined patterns (signatures) that match suspicious packets in network communication. The signature is a structured set of rules that is used to identify an attack [14]. Attacks are detected by searching for these signatures in packets. Anomaly-based systems are based on a creation of long-term network traffic statistics, or the use of artificial intelligence to obtain a common network traffic model [6]. When the deviation of the actual communication from the created profile is detected a system reports an attack alert. IDS are optimized to perform a real-time detection; thus the complexity and expressive power of the detection rules are limited. While IDS produces an alert for a detected attack or abuse, the burden of further work falls on the analysts who must collect evidence within the network traffic data. This is usually a series of manual activities during which the analyst filters captured traffic, decodes the packets and collects relevant information from the packet's header and payload. Research has been done towards providing automated analysis procedures or aiding the process by visual analysis methods.

3.1 Intrusion Detection Systems

One of the most widespread IDS tools is *Snort* [1]. The system allows administrators to search by string, binary data or regular expression. The search is performed either in data within the same transport streams or on a per-packet basis. *Snort* does not deal with the structure and semantics of application protocols. With *Snort*, we are not able to find if there is an LDAP packet with the resulting code 90 that represents memory problems. If the system tries to find just a value of 90 (or hexadecimal 0x5A), the result would contain many false positives.

The*Suricata* tool [2] has support for application protocol decoding, although the list of supported protocols is quite small (around 13 protocols). Also, for the supported protocols, the tool defines only very few fields. For example, only four fields are defined for protocol ICMP, and for DNS protocol only one field is defined. On the other hand, *Wireshark* defines up to 79 ICMP fields and 298 DNS fields.

The *Bro* tool [21] (currently known as Zeek [3]) performs a deep packet inspection of the traffic to detect known security threats. Zeek performs an in-depth analysis of network communication keeping an application-layer state which enables it to perform a more advanced analysis in comparison to traditional signature-based IDSs. An event-driven scripting language makes it possible to customize the system to one's specific needs. The idea behind Bro/Zeek is similar to the proposed system, but it differs in several aspects. Firstly, extending the system with a new application protocol requires writing a new dissector, thus, similarly to *Suricata*, Bro/Zeek supports just around 50 protocols, and not all protocol fields are supported. Secondly, writing Bro scripts requires advanced programming skills often not had by average network administrators. Thirdly, while the scripting language is very powerful, to write rules for non-trivial cases is not straightforward.

[1]http://www.snort.org
[2]https://suricata-ids.org/
[3]https://www.zeek.org/

3.2 Network Traffic Analysis

Network traffic analysis corresponds to the examination of network communication for the purpose of computer security, troubleshooting and system debugging. The most commonly used tool for manual network traffic analysis is *Wireshark* [4] [2, 18, 19]. *Wireshark* supports decoding of all standardized and widely used network protocols. The shortcoming of the *Wireshark* tool is that it lacks any advanced automation [5]. Also, the tool cannot provide a big picture view of the data without cognitive overloading of the user [8]. To overcome tedious work of network traffic analysis the analysts and operators rely on automation scripts that reduce cost and cut down the time required to complete the investigation. Information visualization aids security analysts in detecting anomalous events by increasing their situational awareness through the visual representation of network flow [13]. Another tool, VisAlert, provides an extensible visualization that can accept multiple data sources, including IDS alerts and system log files [15]. Krasser et al [12] proposed a tool that uses 3D animation to provide rich visualization information on network communication.

When the traffic is encrypted the analyst cannot apply the traditional approach based on decoding and examination of individual packets. In such a situation, traffic characterization is often the only viable option. Encrypted traffic is characterized into different categories, according to the type of traffic e.g., browsing, streaming, etc. To do this, statistical or machine learning methods are often used [4]. This approach is also effective for identification of malware from the network behavior [22].

3.3 Rule-based Methods

There are also other network domains where administrators use rule-based systems, for example in SCADA networks management [25] or troubleshooting [16]. Also, these systems share the same limitations: per-flow or per-packet analysis, they are not easily extendible, or they are using a language unknown to a typical administrator. The rule-based approach is, of course, practicable also outside the domain of computer network security. The primary purpose of the tool *Yara* [5] is the detection and classification of malware samples in various sources (e.g., files, folders, processes). Analysts can create rule sets based on the textual or binary patterns that are transformed into a compiled form and then used for the search in the specified entities. *Yara* can detect predefined patterns in the files, but it does not provide a way how to more

[4]https://www.wireshark.org/
[5]https://virustotal.github.io/yara/

comprehensively describe the relations between the data, and it does not involve its structure and semantics. The Yara tool was used for file system analysis as well as for memory analysis [3].

The proposed system is similar to the signature-based IDS but has some significant differences. Signatures for IDS systems are typically generated by the system creator or the community around that system. To create a new signature, a significant effort is required [7]. Another limitation of IDS systems is that they work on the flow-level [20]. Systems process each flow independently, and therefore they usually search for events within an individual stream. However, it is very also necessary to be able to look for the data across streams, because there are threats which distribute the traffic across multiple streams, e.g., malware which uses a multi-band technique to hide from the detection [1]. Our proposed system, however, can easily detect events spread across multiple streams.

4 RULE-BASED NETWORK EVENTS FINDER

The proposed tool performs automatic analysis of packet traces of network communication. The tool is driven by the collection of rules that when evaluated are able to identify if a certain event happened. As described in Section 2, we focus primarily on network incidents, thus any identified event stands for the identified security incident. However, the tool can also be used in other areas, such as Network Diagnostics or Performance Monitoring. For this reason, instead of referring to incidents, we use the more general term *event* to describe the information we are looking for inside the captured network packets.

The overall architecture of the system is illustrated in Figure 3. With this architecture, an administrator will write event descriptions in human-readable configuration files which the tool will convert into an executable format (step 1). An administrator can easily add, modify and delete event descriptions. After the tool converts all configuration files, an external tool is used to convert captured packets into the format suitable for further processing and searching in them (step 2). Lastly, the tool takes converted event descriptions one by one and tries to search for them inside the converted packets (step 3). Based on the findings, the tool generates an appropriate report.

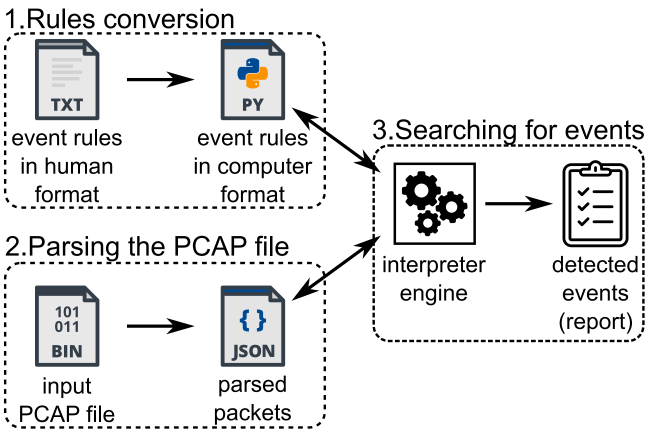


Figure 3: An architecture and workflow of the proposed tool.

The rest of this section (i) describes the format and language of the event description files, (ii) discusses parsing of the input PCAP files, and (iii) explains how the tool searches for events inside the input data.

4.1 Rules language

The rule language is used by users to specify the matching conditions and actions related to identified events. The rule language represents a compromise between easy to use representation and expressivity. Also, for expressions, the *Wireshark*'s display filter language is utilized [6]. It has two fold reasons. Firstly, the display filter language is expressive enough to represent simple queries to a collection of packets. Secondly, language is well-known and easy to understand among network administrators.

Each event rule consists of a name, description, and the body consisting of five attributes: *group*, *packets*, *asserts*, *threshold* and *report*. Figure 4 shows an example of a rule, which describes how to detect the SLAAC Duplicate Address Detection (DAD) attack. The details and explanation of the meaning for each attribute is given in the next subsection.

```
name: SLAAC DAD
description: Possible Duplicate Address
    Detection attack , find attacker from
    NA source MAC address .
group:
  - icmpv6.nd.ns.target_address icmpv6.nd.
    na.target_address
packets:
  -NS: icmpv6.type==135 and ipv6.src=="::"
  -NA: icmpv6.type==136 and ipv6.dst=="::"
asserts:
  - count(NS) > 0 and count(NA) > 0
threshold: 5
report: packets
```

Figure 4: Event description for detection of the SLAAC DAD attack.

4.1.1 *Name and description.*

Name is an identification of the rule which needs to be unique. A description is used to describe the event in a human-understandable format. Both the name and the description are part of the created report.

4.1.2 *Groups.*

The section *group* specify how the packets are split into several disjunctive groups. The tool assigns packets into groups based on the value of specified protocol fields. For example, when the administrator specifies the source IP address (ip.src) as a protocol field, each IP address will have a separated group containing packets which that IP address sent.

Figure 5 shows the format of the rule part *group*. If more than one protocol field is specified on a single line, the tool takes protocol fields one by one in the same order as specified and tries to find the values for these fields. This process stops when the value is found or when the packet does not contain any protocol value from the line. If more than one line is specified, a packet must have

[6]https://www.wireshark.org/docs/dfref/

at least one value for each line, and in that case, the tool marks the group as a list of values (one line = one list element). Otherwise, the packet will be ignored in future processing, because it does not belong to any group. One exception is when the rule does not contain any line and any protocol field. In that case, the tool assigns all packets into a single “default“ group.

```
group:
 - field_1 field_2 ... field_N
 - ...
--------------------------------------------

# field_X - any Wireshark's protocol field
    name
```

Figure 5: Format of the event's group section

4.1.3 Packets. Inside each group of packets, the tool tries to find predefined packets. These predefined packets have a special meaning for the event detection (e.g., special protocol field value or amount of these packets). Each packet has its name and filter definition. The result is a list of packets fulfilling the filter.

Figure 6 shows the format of the rule part *packets*. Each packet is defined by its unique name and by the *Wireshark*'s display filter language. This is the same filter which a network administrator would use during a manual analysis using the tool *Wireshark*. Therefore, these filters can be copied from *Wireshark* or vice versa.

```
packets:
 - name_1: filter_1
 - ...
--------------------------------------------

# name_X - the name will be used as a label
     for packets in section 'asserts'.
    Possible values: [a-zA-Z0-9_-]+
# filter_X - Wireshark's display filter
```

Figure 6: Format of the event's packets section.

4.1.4 Asserts. After the tool finished the search for the specified packets, the tool evaluates the assert conditions. These conditions define if the expected state (e.g., too many packets) was detected inside the group of packets. The assert language is based on the *Wireshark*'s display filter language, but contains three improvements:

(1) possibility to get the number of detected packets. This is implemented with function *count*() which takes the name of the specified packet (from the Section 4.1.3) as the parameter. E.g., $count(DNS) > 100$;
(2) it is possible to use basic mathematical operations (+, -, *, /). E.g., $count(DNS)/count(all) > 0.5$;
(3) searching for a specific field name can be limited only on specified packets from the section packets. E.g., $DNS[udp.port]$ == 53 will try to find value udp.port only inside detected packets with name DNS. If no packet name is specified, the search is focused on all packets in the group.

Figure 7 shows the format of the event's asserts part together with the three improvements. Name 'count' is the name of the function, *packets_name* is the name of the defined packets from the event packets section, and $field\_name$ is the name of the protocol field from the *Wireshark*'s display filter language.

```
asserts:
 - condition_1
 - ...
--------------------------------------------

# condition_X - Wireshark's display filter
    with possible extensions:
# 1) count(packets_name) - used as a
    constant
# 2) math_sign - used as a operator
# 3) packets_name[field] - used as a
    variable
# count() - name of the function.
# packet_name - name from part 'packets'.
# math_sign - one of the following
    mathematical signs: +, -, *, /
# field - Wireshark's protocol field name
```

Figure 7: Format of the event's asserts section.

4.1.5 Threshold. Each group which fulfills all assert conditions increases a counter designated to count all fulfilling groups. This counter is compared with a threshold and based on whether it is equal or greater than the threshold, the tool generates a report.

Figure 8 shows a very simple format of the threshold. Only one integer value is specified.

```
threshold: value
--------------------------------------------

# value - Numeric contant. Possible values:
     [0-9]+
```

Figure 8: Format of the event's threshold section.

4.1.6 Report. Section *report* specifies how detailed the generated report will be. For example, it is not useful to export all packets which are part of a large DDoS attack. Three levels are defined: *event* - only data from the section threshold are used, *groups* - all group values which contribute to the final report are added and *packets* - each group reported will also contain a list of all the packets inside that group.

Figure 9 shows the format of the report section. Only one string with one of the three predefined values is defined.

```
report: level
--------------------------------------------

# level - Specifies the detail of generated
     report. Possible values: event, groups,
     packets
```

Figure 9: Format of the event's report section.

4.2 Packets parser

The external tool called *TShark* [7] (command line version of the *Wireshark* tool) is used to parse captured network communications. We use an existing tool to avoid the necessity of creating parsers (or dissectors) for individual protocols. As *TShark* provides the same set of dissectors as *Wireshark*, we can immediately support almost all current network protocols in the rules. Using *TShark* approach comes with several pros and cons:

+ using an external well-maintained tool decreases the requirements for the created tool;
+ tunnel data can also be processed as if the tunnels were not used at all;
+ supports a large number of protocols (over 3000) as well as the number of fields that can be obtained from the protocols (over 227000) - valid for version 2.6.3.
+ processed data can be exported into the JSON format so that later processing can directly access the values of the protocol fields;
+ the fields of the individual protocols are labeled in the same way as in *Wireshark*, so the administrator does not need any special documentation to find the names of the fields;
+ *TShark* does not just parse the data, it also analyses it. For example, it detects packet retransmission or calculates application statistics;
- if some protocol is not supported by *TShark* (usually a proprietary protocol), adding support for it requires very good programming skills;
- *TShark* is not fast at analyzing large PCAP files (gigabytes and more).

The description and options of *TShark* can be found in its documentation. Just for clarification of how the JSON from the *TShark* looks like, Figure 10 shows an example of the output. The JSON contains protocol field values from all network protocols in a key-value data format. Hierarchy of the JSON data attributes directly represents the structure of the packet protocols.

```
...
"eth": {
  "eth.dst": "f0:79:59:72:7c:30",
  "eth.src": "00:1f:33:d9:81:60",
  "eth.type": "0x00000800",
},
...
"udp": {
  "udp.dstport": "53",
  ...
},
...
"dns": {
  "dns.id": "0x00007956",
  "dns.flags.response": "0",
  "dns.qry.name": "mail.patriots.in",
  ...
},
...
```

Figure 10: Excerpt from the TShark example output.

[7] https://www.wireshark.org/docs/man-pages/tshark.html

4.3 Rule checker

When the tool converts PCAP file and all rules into a format suitable for further processing, the tool starts searching for the events with the last part of the architecture - interpreter engine. The interpreter engine takes the rules one by one and tries to find whether some rule is matched inside the data or not. Based on the rule definition, the engine creates an adequate report. The engine consists of five parts as shown in Figure 11. Each part corresponds to one section from the event rule definition (Section 4.1). This is illustrated using boxes in the Figure together with rule lines which are copied from the rule definition example in Figure 4. Following subsections describe these five engine parts.

4.3.1 Grouping packets. The goal of the packet grouping part is to separate packets into groups according to the specified attributes (see Section 4.1.2). If the packet has multiple values for the defined attribute, the same packet will be placed into multiple groups. This separation into groups allows processing of packets from one group independently from packets in other groups. The attribute can be any field name which *Wireshark* defines. Example of such an attribute is a source IP address ($ip.src$) or TCP stream index ($tcp.stream$). In the case the packet does not contain a specified attribute, the tool ignores it during further processing. There is one exception to this rule, the situation when an administrator does not specify any attribute at all. In that situation, all packets are part of the same "empty" group.

It is possible to specify more than one attribute for dividing packets into groups. There are two modes in which an administrator can specify multiple attributes - AND and OR and these modes can be combined. In the OR mode, the tool is trying to find the first attribute inside the data and continues looking for other attributes only if there is no match. To continue the processing of the packet, the tool must find at least one attribute. With the AND mode, the tool must find all specified attributes. Another difference is that the tool will identify these groups using a list of values instead of one value.

Figure 11 shows an example with two attributes in OR mode (both are specified in the single row as can be seen in Figure 4): $icmpv6.nd.ns.target\_address$ and $icmpv6.nd.na.target\_address$. These attributes group SLAAC *Neighbor Solicitation* (NS) messages with *Neighbor Advertisement* (NA) messages according to the requested address. Example of group identification is value $2001 : abcd :: 6064 : dec3 : 35e8 : 3bb0$.

4.3.2 Searching specific packets inside groups. After the tool assigns each packet into the appropriate group, the tool tries to search for specific packets inside them. All packets fulfilling the search condition are returned. As described in Section 4.1.3, packets are specified using the *Wireshark*'s display filter language, and their evaluation is the same as in *Wireshark*.

In the example from Figure 11, the tool tries to find two sets of packets (NS and NA) specified by conditions $NS = \ldots$ and $NA = \ldots$, For the group $2001 : abcd :: 6064 : dec3 : 35e8 : 3bb0$, the NS will contain all NS messages which check the availability of address $2001 : abcd :: 6064 : dec3 : 35e8 : 3bb0$ and NS will contain replies for NS messages.

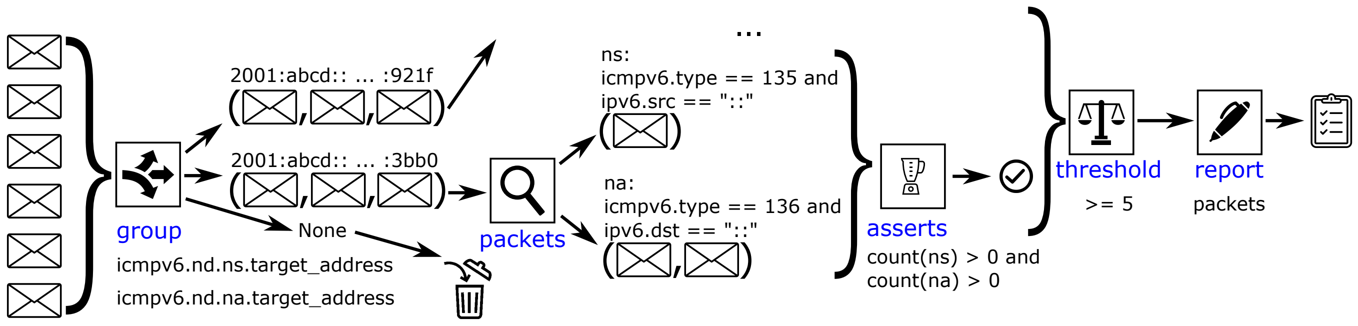


Figure 11: The rules evaluation engine which consists of five parts. On the left side are parsed packets which are the input of the engine and the output is report placed on the right side. Bellow each part are lines from the example code in Figure 4.

4.3.3 Checking asserts for each group. The event can contain several assert rules. All of the assert rules must be valid to consider a group as the group which fulfills the assert rules. Checking whether assert rules are valid or not is very similar to searching packets in groups. As was described in Section 4.1.4, asserts use an extended the *Wireshark*'s display filter language. This language extension modifies how the tool evaluates the assert filter. There are three language extensions:

(1) It is possible to work with information about how many packets the tool detected inside a group. This information is calculated using the function $count(packet\_name)$. Before the evaluation of the assert conditions, all these functions are executed and replaced by their results (numeric constant).
(2) When looking for a specific field inside the packets, *Wireshark* uses just the name of the protocol field (e.g., $ip.src =$ 8.8.8.8). When this code is placed in the assert condition, the tool will check only packets from the group. However, it is also possible to limit these searches for values only to packets which the tool found in the packets section. For example, during the asserts evaluation with the code $query[ip.src] =$ 8.8.8.8, the tool will supplement only packets fulfilling the query filter.
(3) An administrator can use basic mathematical operations with the *Wireshark* fields, functions and constants. For example, an administrator can use them for calculating the ratio between two sets of packets. With the code $count(X)/count(Y)$ $>$ 2, the tool evaluates the condition as true only if the amount of packets fulfilling the packet filter X is at least twice as many as packets fulfilling the packet filter Y.

In the example in Figure 11, the assert rule is $count(NS) > 0$ *and* $count(NA) > 0$, which specifies, that there must be at least one NS message and one NA message for the same queried IPv6 address.

4.3.4 Counting the group matches. After the tool evaluates each asserts condition for each group, the tool checks the amount of the fulfilling groups. The result is a single numeric value which the tool compares with a defined threshold. If the threshold condition is met, the tool located the event in the input data and starts the generating report process.

In the example from Figure 11, we are counting how many verified addresses ended as duplicated. If this number is equal to 5 or higher, the tool detects the DAD attack.

4.3.5 Generating report. If the tool detects an event, it generates an appropriate report. The report can have three levels of detail (as described in Section 4.1.6): *event* (minimum details), *group*, and *packets* (maximum details).

Figure 12 shows a report from our DAD attack example with three vertical lines which illustrate, which lines the report would contain with a different level of detail. This is also a result of the example introduced in Figure 1; the tool detected SLAAC DAD attack.

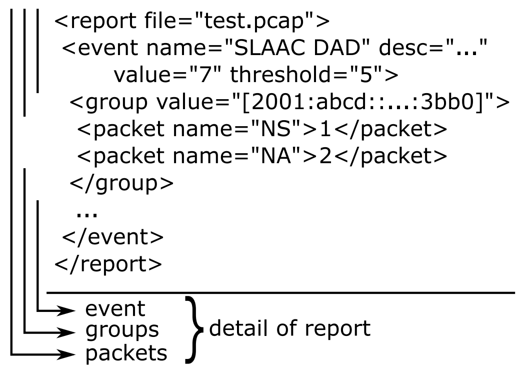

```
<report file="test.pcap">
 <event name="SLAAC DAD" desc="..."
        value="7" threshold="5">
  <group value="[2001:abcd::...:3bb0]">
   <packet name="NS">1</packet>
   <packet name="NA">2</packet>
  </group>
  ...
 </event>
</report>
```
event
groups
packets
detail of report

Figure 12: The output of the tool which contains a detected DAD attack. The vertical lines represent how the report would look with different requested detail.

5 CONCLUSION

Packet-level network analysis is a very time-consuming activity requiring good knowledge of network protocols. An administrator usually uses the packet analyzer which performs capturing network data, dissecting network packets and providing other related information. This paper describes an approach to automating this process.

Signature-based IDS tools provide similar functionality to our tool. The problems with IDS systems are that they usually require programming knowledge for extending their capabilities, they support only a limited number of protocols and their detection capabilities are limited to flow-scope or packet-scope analysis.

We have proposed an interpreter architecture, which uses a rule-based approach for defining security events. We are using *TShark* to parse the input data, which removes the necessity of writing custom protocol parsers. The tool uses a format which was inspired by the *Wireshark*'s display filter language. This format allows administrators to create or modify a rule without any programming skills with the knowledge they already have. The tool is supposed to fill in a missing spot for automation framework for security analysts that must analyze packet traces related to the incident based on alerts generated by IDS tools. The presented tool automatically performs the analysis using defined rules and identifies the location using evidentiary material.

Most of the currently used tools search for data on a per-packet or per-flow basis. Our tool is capable of searching for events across multiple flows without any limitations which makes the searching process more flexible. Because searching without these limitations is very expensive, the tool is not designed to fully replace already existing IDS solutions which are less flexible, but capable of processing much more data.

We have created a proof-of-concept implementation, which covers the entire analysis process. The event rule files are transformed into a format suitable for further processing. After that, the tool uses the external tool *TShark* (command line version of the *Wireshark* tool) to parse the input PCAP file and to save them into the JSON format. The final stage of the tool takes this JSON file and applies available rules to identify the security threats in the input data. At the end of the analysis, the tool generates a report.

To demonstrate the functionality of the implemented prototype, we have created a specification of 35 different security events, e.g., MitM ARP attack, HSRP protocol configuration with nonoptimal configuration, network scanning, using old-unsecured TLS version and so on.

The future work will focus on enriching generated output. For example, the output should contain a *Wireshark*'s display filter expression which can be used by an administrator for manually confirming the detected event in *Wireshark*. Also, the language should be enriched with new features, for example, usual statistical and aggregation functions. To be practically usable, optimizing the performance of the tool is necessary.

ACKNOWLEDGMENTS

This work was partially supported by the BUT FIT grant FIT-S-17-3964, "ICT tools, methods and technologies for smart cities".

REFERENCES

[1] MITRE ATT&CK. 2019. Technique: Multiband Communication. https://attack.mitre.org/techniques/T1026/
[2] Laura Chappell. 2017. *Wireshark 101: Essential Skills for Network Analysis-Wireshark Solution Series.* Laura Chappell University, USA.
[3] Michael Cohen. 2017. Scanning memory with Yara. *Digital Investigation* (2017). https://doi.org/10.1016/j.diin.2017.02.005
[4] Gerard Draper-Gil, Arash Habibi Lashkari, Mohammad Saiful Islam Mamun, and Ali A. Ghorbani. 2016. Characterization of Encrypted and VPN Traffic using Time-related Features. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy - Volume 1: ICISSP.* 407–414. https://doi.org/10.5220/0005740704070414
[5] Alia Yahia El Sheikh. 2018. Evaluation of the capabilities of Wireshark as network intrusion system. *Journal of Global Research in Computer Science* 9, 8 (2018), 01–08.
[6] Pedro Garcia-Teodoro, Jesus Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. 2009. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security* 28, 1-2 (2009), 18–28.
[7] Ibrahim Ghafir, Vaclav Prenosil, Jakub Svoboda, and Mohammad Hammoudeh. 2016. A survey on network security monitoring systems. In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW).* IEEE, 77–82.
[8] John R Goodall, Wayne G Lutters, Penny Rheingans, and Anita Komlodi. 2006. Focusing on Context in Network. *Security* April (2006), 72–80.
[9] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. 2014. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials* 16, 4 (2014), 2037–2064.
[10] Computer Economics Inc. 2007. 2007 malware report: The economic impact of viruses, spyware, adware, botnets, and other malicious code. http://www.computereconomics.com
[11] Karen Kent, Suzanne Chevalier, Tim Grance, and Hung Dang. 2006. Guide to integrating forensic techniques into incident response. *NIST Special Publication* 10, 14 (2006), 800–86.
[12] Sven Krasser, Gregory Conti, Julian Grizzard, Jeff Gribschaw, and Henry Owen. 2005. Real-time and forensic network data analysis using animated and coordinated visualization. In *Proceedings from the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop, SMC 2005.* https://doi.org/10.1109/IAW.2005.1495932
[13] Kiran Lakkaraju, William Yurcik, and Adam J Lee. 2004. NVisionIP: NetFlow Visualizations of System State for Security Situational Awareness. *Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security - VizSEC/DMSEC '04* (2004). https://doi.org/10.1145/1029208.1029219
[14] Hao Li, Guangjie Liu, Weiwei Jiang, and Yuewei Dai. 2015. Designing snort rules to detect abnormal dnp3 network data. In *2015 International Conference on Control, Automation and Information Sciences (ICCAIS).* IEEE, 343–348.
[15] Yarden Livnat, Jim Agutter, Shaun Moon, Robert F. Erbacher, and Stefano Foresti. 2005. A visualization paradigm for network intrusion detection. In *Proceedings from the 6th Annual IEEE System, Man and Cybernetics Information Assurance Workshop, SMC 2005.* https://doi.org/10.1109/IAW.2005.1495939
[16] GeokHong Phua Lihui Chen Ming Luo, Danhong Zhang. 2011. An interactive rule based event management system for effective equipment troubleshooting. *Proceedings of the IEEE Conference on Decision and Control* 8, 3 (2011), 2329–2334. https://doi.org/10.1007/s10489-005-4605-0
[17] Srinivas Mukkamala and Andrew H Sung. 2003. Identifying significant features for network forensic analysis using artificial intelligent techniques. *International Journal of digital evidence* 1, 4 (2003), 1–17.
[18] Vivens Ndatinya, Zhifeng Xiao, Vasudeva Rao Manepalli, Ke Meng, and Yang Xiao. 2015. Network forensics analysis using Wireshark. *International Journal of Security and Networks* 10, 2 (2015), 91–106.
[19] Yoram Orzach. 2013. *Network Analysis Using Wireshark Cookbook.* Packt Publishing Ltd.
[20] Samuel Patton, William Yurcik, and David Doss. 2001. An AchillesâĂŽ heel in signature-based IDS: Squealing false positives in SNORT. In *Proceedings of RAID,* Vol. 2001. Citeseer.
[21] Vern Paxson. 1999. Bro: a system for detecting network intruders in real-time. *Computer networks* 31, 23-24 (1999), 2435–2463.
[22] Christian Rossow, Cj Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten Van Steen, Felix C. Freiling, and Norbert Pohlmann. 2011. Sandnet: Network Traffic Analysis of Malicious Software. *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS '11)* (2011), 78–88. https://doi.org/10.1145/1978672.1978682
[23] Sankardas Roy, Charles Ellis, Sajjan Shiva, Dipankar Dasgupta, Vivek Shandilya, and Qishi Wu. 2010. A survey of game theory as applied to network security. In *2010 43rd Hawaii International Conference on System Sciences.* IEEE, 1–10.
[24] Anna Cinzia Squicciarini, Giuseppe Petracca, William G Horne, and Aurnob Nath. 2014. Situational awareness through reasoning on network incidents. In *Proceedings of the 4th ACM conference on Data and application security and privacy.* ACM, 111–122.
[25] Yi Yang, Keiran McLaughlin, Tim Littler, Sakir Sezer, and HF Wang. 2013. Rule-based intrusion detection system for SCADA networks. (2013).
[26] Wang Zhenqi and Wang Xinyu. 2008. Netflow based intrusion detection system. In *2008 International conference on multimedia and information technology.* IEEE, 825–828.