How to Make an Intrusion Detection System Aware of Steganographic Transmission

Tomasz Koziak*
ttomasz.koziak@gmail.com
Warsaw University of Technology
Warsaw, Poland

Katarzyna Wasielewska
k.wasielewska@gmail.com
Warsaw University of Technology
Warsaw, Poland

Artur Janicki
A.Janicki@tele.pw.edu.pl
Warsaw University of Technology
Warsaw, Poland

ABSTRACT

Information hiding techniques are becoming a major threat in network communication. This paper describes how to modify an intrusion detection system (IDS) to detect certain types of steganography. As a sample IDS we use open-source Zeek software. We show how to adapt it for the purpose of steganalysis. Additionally, we propose a set of validation tests that are suitable for detecting steganography and describe how they were applied to different types of covert channels. We also suggest how to build a steganography detection system by integrating Zeek with a security information and event management system with log and alert support. The scripts are freely available for download.

CCS CONCEPTS

• **Security and privacy → Network security**.

KEYWORDS

hidden transmission, intrusion detection system, malware detection, steganalysis, Zeek

ACM Reference Format:
Tomasz Koziak, Katarzyna Wasielewska, and Artur Janicki. 2021. How to Make an Intrusion Detection System Aware of Steganographic Transmission. In *DETONATOR 2021 – European Interdisciplinary Cybersecurity Conference – EICC 2021, November 10–11, 2021, Târgu Mureş, Romania.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Steganography is a process of hiding data in text, audio, image or video data, and also in computer network traffic [13, 23]. Due to the complexity of computer systems, it is easier to construct a hidden, steganographic channel than it is to detect it. Therefore, while a variety of steganographic tools exist, fewer tools are capable of steganalysis. The Internet, at its origin, was not designed with safety as the leading focus, so basic Internet protocols are vulnerable to numerous threats. Some protocols can be used as a carrier for sending hidden data that can be used by malicious software, so-called stegomalware [2]. This type of malware is particularly difficult to detect, thus increasing the need for high-quality, easily implemented steganalysis tools. Examples of such malware include: Pingback [21], which uses the Internet Control Message Protocol (ICMP) sequence number for hidden communication, Convert-TCP [19], that modifies IP identification numbers and TCP initial sequence numbers (ISN), or the TCP/IP network steganography framework [10] which embeds information into the ICMP identifier, ICMP sequence, IP identification field, and IP Differentiated services fields.

In this paper, we present selected open-source intrusion detection systems that could potentially, after some modifications, be used to detect hidden channels. One such system (Zeek) was selected and adapted to detect various popular steganographic methods used in network communication. We used several test scenarios to see if the system was able to detect typical steganographic transmission and we present the results of our experiments. The malicious traffic was generated using an external tool as well as our own internal steganography generator. The paper also proposes a method for integrating the detection system with an external logging system, to allow for alerting the network administrator to dangerous or undesirable activity.

The main contributions of our publication are as follows:

- Adapting Zeek, a popular IDS system, to be able to detect steganographic transmission, using a set of scripts.
- Proposing a light-weight method of increasing network security by updating the currently used IDS system (in our case Zeek), without the need to deploy additional products.
- Proposing a method for integration of the Zeek-based detection system with an external logging and alerting system.
- Proposing the potential direction of future improvements to the presented solution.

The paper is structured as follows. Section 2 presents aspects of network steganography and steganalysis that are important for understanding the problem. In addition, this section briefly introduces intrusion detection systems which can potentially be used to detect steganography. In Section 3, we present our case study, showing how to modify the Zeek IDS system to detect steganographic transmission. Section 4 presents the results of our experiments. Section 5 presents discussion on the work performed and results achieved, while the last section contains conclusions and future plans.

2 BACKGROUND

2.1 Network steganography

Generally, in steganography two parties communicate with each other using a standard communication channel. However, they also create a hidden (covert) channel that is embedded inside the 'official' (overt) communication, that is imperceptible to external observers.

*Corresponding author.

DETONATOR 2021, November 10–11, 2021, Târgu Mureş, Romania

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

Any potential third-party will not know where the message is hidden or if such hidden communication has taken place at all.

Table 1: Comparison of IDS systems considered.

IDS	Open Source	Basis	Parallel	Language	IPv6
Snort	Yes	Rules	No	C	Yes
Suricata	Yes	Rules, scripts	Yes	C, Rust	Yes
Zeek	Yes	Scripts	No	C++	Yes

In this paper we concentrate on network steganography. The distinction between network steganography and steganography is that the former focuses on utilising features of network protocols (e.g., TCP) to create the hidden channel. The Internet, at its origin, was not designed with safety as the leading focus. For this reason, some basic Internet protocols are far from perfect when it comes to data protection, and therefore certain protocols can be used as a carrier for sending data.

Network steganography can be divided into three main techniques [12]. The first group concentrates on modifying the protocol data units, which may include sequence bits, identification bits, etc. The second method is based on modifying the time relationship between received packets. The last group of techniques involves modification of the payload field. There are also hybrid techniques, which are a combination of the above mentioned methods. For example, hybrid methods assume that some data will be hidden in the protocol header and some data will be conveyed by manipulation of the interarrival packet time values. In steganography, it is important to transport a large amount of information along one channel without attracting too much attention. Therefore, knowing how to effectively hide data and how much information can be sent through the hidden channel without being detected is important. Examples of network steganography tools are described in [3].

Steganalysis deals with the detection of hidden channels. Several sophisticated methods have been developed to try to detect steganographic transmissions [8], e.g., by analysis of patterns in the behaviour of delivered packets. Another approach focuses on measuring the number of requests that can be observed during the call [14]. In [16] the authors proposed two types of steganalysis: signature and statistical. Signature steganalysis deals with the detection of known hidden channels, e.g., by checking if hidden information was sent in reserved bits in the TCP header. In contrast, statistical steganalysis looks at a bigger picture than just well-known covert channels and is able to check, for example, whether TCP sequence numbers were decreasing during the session.

2.2 Intrusion detection systems

An intrusion detection system (IDS) is an application that monitors network traffic or system behaviour in order to detect malicious or unusual activities. IDSs are classified based on location and information source into the categories of host-based IDS (HIDS) and network-based IDS (NIDS). A HIDS is installed on an individual computer and is responsible for traffic and registry monitoring, rootkit detection, log analysis, compliance checks, and other functions. On the other hand, a NIDS is located in the network and focuses on detecting abnormal behaviour in the subnet. Additionally, IDSs can be classified on the basis of detection methods into signature-based and anomaly-based. IDSs are commonly used and a number of techniques have been integrated with IDSs, such as machine (incl. deep) learning, and blockchain technologies [11]. NIDS appears to be a better choice for detection of network steganography because it allows tracking of network traffic in a network segment, and not just on a specific device [7].

Several popular NIDS tools are commonly in use [1]. We considered three open-source tools: Suricata[1], Snort[2] and Zeek[3], formerly known as Bro. The tools' main features are compared in Table 1. Snort is a classic rule-based IDS system. Suricata, apart from rules, has recently introduced support for the Lua scripts. Zeek is more a network traffic analyzer that can be used as a security monitor. Snort, supported by Cisco, is the most popular and can act as both a signature-based and an anomaly-based detection system. Writing Snort rules is relatively simple, but when it comes to detecting more complex traffic, it is difficult to define a sufficient Snort condition [7]. Adding a new hidden channel detection feature may require modifying the Snort core, and writing steganalysis rules can be a challenge.

Suricata is a solution similar to Snort in terms of construction. It introduced more flexibility in the rule creation process. However, there remains a problem with the definition of hidden channels. Adding new plugins and modifying software functions is a challenge. Suricata and Snort are technologies well suited to detecting malicious activity, although they seem to have problems with detecting unusual behaviour of a higher degree of complexity, such as steganography.

Zeek has a different approach when it comes to detecting malicious traffic – it uses scripts instead of rules. It allows for easier writing of complex conditions. In addition, Zeek is adaptable and flexible since it does not rely on traditional signatures. It allows for writing a code (a script) that will define what steps should be taken after certain types of actions. Zeek's operation is based on network events, such as arrival of a new TCP packet, ICMP echo reply, Message Queuing Telemetry Transport (MQTT) connect etc. It is quite simple to add a new event or modify the existing functions. From the other side, the packet inspection provided by Zeek is quite resource intensive.

3 CASE STUDY

As a case study we consider a small size LAN in an office with ca. 10 workstations, running various operating systems, and exchanging data with external servers, i.e., generating a remarkable volume of data over an Internet link. This LAN network is protected by the Zeek IDS system. Such a generic IDS system is capable of protecting this network from intruders, who try to attack it using, e.g., UDP or TCP flooding attacks or other, typical forms of intrusion.

However, such a network is vulnerable to infiltration using steganographic methods. Hidden channels can be used to allow leakage of data or command&control traffic; this leakage could

[1] www.suricata-ids.org
[2] www.snort.org
[3] www.zeek.org

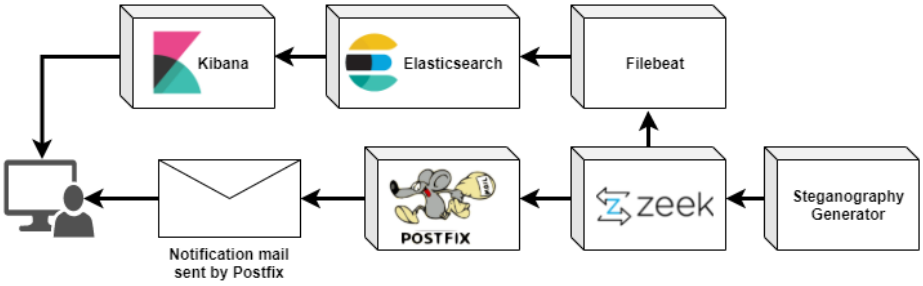

Figure 1: Proposed solution in test environment with Kibana [9], Elasticsearch [5], Postfix [18] and Filebeat [4]

facilitate the setting up and control of botnets within the organization, spreading malicious software or preparing for future attacks. Our aim is to make the Zeek-based IDS aware of such threats, to enable it to detect hidden transmission. To be able to efficiently handle intensive network traffic, we plan to search for a lightweight solution.

Therefore we propose to make Zeek sensitive to hidden transmission by adding several scripts to its configuration. Zeek was previously implemented in the steganalysis area in the Bro Covert Channel Detection (BroCCaDe) project [6], where the authors extended Zeek with plugins capable of detecting covert channels. In contrast, we have proposed a much simpler approach, with plugins not needed, just some Zeek scripts. Such a setup is much lighter and more suitable for higher traffic volumes, as in our case study.

Zeek offers many data types designed especially for handling network traffic parameters such as IP addresses, subnets, port numbers and timestamps, as well as key-to-value tables, indexed vectors and records with collection of the values. For steganography detection, we mainly used record type tables to associate addresses used as keys with certain values. We created records called Value Time Count (VTC) consisting of four parameters: value of investigated data for last arrived packet for particular IP address; timestamp of last packet that arrived for particular IP address; count, which kept track of the number of packets that were classified as suspicious for particual IP address; and global count, which was storing the absolute number of all investigate packets for particular IP. With the help of VTCs, we constructed a table that uses the each unique IP address as a key and VTC as a value. It allowed us to store a parameter value from the originator and to check if that value changed over time. This is useful in detecting steganography, which is characterized by frequent changes of values. What is more thanks to count parameter and global count the ratio of suspicious packet to all packets was investigated. Interarrival Time (IAT) was another data structure that was defined in our scripts for detection purposes. This record was used to store IAT values for unique IP addresses. In detail, the IAT structure was composed of three parameters: (1) count – used to remember how many elements can be found inside a vector, (2) vector of intervals – with actual time intervals between packets, (3) time – which stored the time value when the last interval value was measured.

Next, we defined several types of tests to detect network protocol anomalies. The *existence test* focused on detecting optional elements that were not generally used or that should take on a specific value. The *frequency test* was designed to examine whether a header field (which, by design, should not change frequently) changed often over a certain period of time. The test returned TRUE if the field value changed more often than θ times within a T seconds window. During experiments, we set $\theta = 10$ and $T = 60$. The *trend test* was used to check if the header value increased or if it returned unexpected values. The *consistency test* was useful in situations where the expected value of the next header field value could be simply estimated (for example, subsequent ICMP sequence numbers). The *out-of-context test* was able to inspect whether the received value met the requirements of the defined value. This condition was not always straightforward; therefore we decided to employ entropy metrics, as a high level of entropy may indicate a malicious event.

Lastly, the *interval gap test* was created to investigate delays between packets to find out whether the delays were manipulated by the originator, as happens in timing covert channels. This test analyzed IAT values over a certain window (in our case, set heuristically to 5 s). These values were then sorted and extreme values were stripped. Next, the relative difference between the adjacent values was calculated. An increased difference value meant that distribution of IAT values was irregular, which might indicate the existence of a timing hidden channel. During experiments we heuristically set the decision threshold to 50%, above which an alarm was triggered. The outputs of all the above-described tests were logically summed, so a TRUE value for any of the tests raised a steganography alarm.

Fig. 2 shows the data flow within the proposed system. It also depicts which network protocols were analyzed by which tests. Two data structures were used: the IAT data structure was used for the ICMP protocol and VTC records were employed for the IP, TCP, SIP and MQTT protocols. In the picture, we also included a JavaScript Object Notation (JSON) module, which was used for storing the alerts produced by Zeek code. The JSON file was used by Filebeat read pointer, which monitors that file and queues original log lines to Elasticsearch.

4 EXPERIMENTS

4.1 Experimental environment

We set up a test environment based on two physical machines (Fig. 1). One machine (OS: MS Win7) was responsible for generating the steganography. For this purpose, dedicated software was

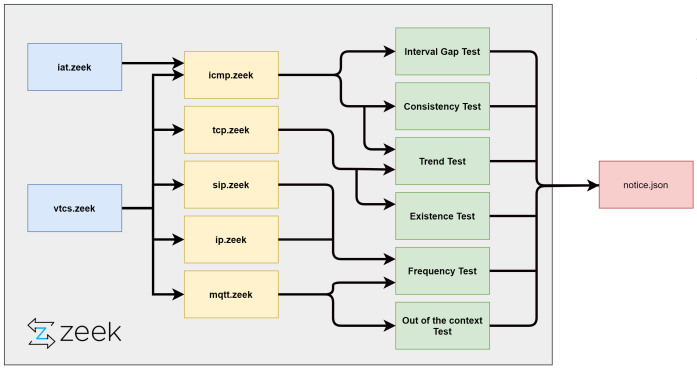

Figure 2: Data flow in proposed detection system

developed, but off-the-shelf tools were also used, such as Stego-Tester[4] and the TCP/IP network steganography framework[5]. The second device (OS: Ubuntu 20.04) was acting as an active warden with the Zeek instance. An analytics environment including Elasticsearch, Kibana, Filebeat, and Postfix was installed on the Linux machine. Elasticsearch was used as a database for data storage and Kibana for visual representation of data stored in Elasticsearch. Filebeat was employed to upload data to the database, while Postfix was used to send alerts. Such a setup allowed integration of Zeek with a security information and event management (SIEM) system with log and alert support.

Table 2: PCAP traces of normal traffic used for tests.

	CR Monday
ICMP packets	1688
IP packets	2256230
TCP packets	1892915
MQTT packets	10244
SIP packets	35264

The emulated hidden traffic was superimposed on real data, taken from the Intrusion Detection Evaluation Dataset (CICIDS 2017) [20]. We used $1h10'$ of traffic from Monday, which contained only normal traffic (no attacks) and did not contain any hidden transmission (statistics are shown in Table 2). The benign data contained traffic captured in a network consisting of eight machines, running various OS systems (Ubuntu 14.4, Ubuntu 16.4, MS Win 8.1, MS Win 10, iOS). The ratio of steganographic to benign traffic did not exceed 5%. The Stego-aware Zeek IDS ran on a machine with Intel® Core™ i7-8565U CPU @ 1.80GHz with 16GB RAM.

4.2 Testing scenarios

To verify the detection efficacy of the solution described, we prepared 23 testing scenarios, each emulating one of network steganographic methods known from literature [15, 17, 22, 24]. They mostly explore various weaknesses or features of popular network protocols: ICMP, IP, TCP, MQTT and SIP, from four different layers of the OSI model. Such a choice was made to verify if our solution would be able to work with various protocols from different layers. Attackers can control various hidden communication channels in protocol data units. E.g., it is possible to modify some fields in IP headers that are responsible for fragmentation and are not always used. Also, the Time to Live (TTL) field may contain a hidden message if we assume that a small number of hops will be encoded on this 8-bit field. The TCP protocol also has weaknesses and its header is even easier to control, as it does not change along the end-to-end path. For example, a sequence number, window size or padding fields are used as a steganography carrier. Moreover, it is possible to hide information by encoding it through manipulation of TCP timers.

Considering the above, to verify the efficacy of our system we used the below test cases:

(1) **ICMP Identification Modulation** – changing the value of the ICMP identification number so that consecutive ID numbers form ASCII code messages;
(2) **ICMP Sequence Number Modulation** – changing the value of ICMP sequence number in such a way that consecutive sequence numbers form ASCII code messages;
(3) **ICMP Interarrival Packet Time Manipulation** – sending ICMP messages so that the user is capable of defining the time gaps between the consecutive packets;
(4) **IP Identification** – changing the values of IP ID numbers so that each new packet carries a value that when divided by a constant gives the ASCII code element;
(5) **IP Time To Live** – changing TTL values, where the higher values indicate logical 1 and lower values indicate 0;
(6) **IP ESCP ECN** – changing the values of ESCP ECN bits so that each new packet carries an ASCII code;
(7) **TCP Source Port** – modifying the values of the TCP source port so that they represent the ASCII codes;
(8) **TCP Sequence Number** – changing the values of the TCP sequence number in the IP/TCP packets so that when divided by a constant they return an ASCII code;
(9) **TCP Reserved Bits** – setting the reserved bits to custom values in the consecutive IP/TCP packets in such a way that each packet carries a part of the message;
(10) **TCP URG Flag / Urgent Pointer** – unsetting the URG flag and sending the message inside of the Urgent Pointer;
(11) **TCP Reset Flag / Data** – setting the RST flag and sending the payload data at the same time with Stego-Tester;
(12) **TCP Window** – setting the window value of the TCP/IP packets to values representing the ASCII code;
(13) **MQTT Publish Payload** – setting the hashed text to the payload of the message;
(14) **MQTT Subscribe Topic** – setting the hashed text to the topic of the message;
(15) **MQTT Client ID** – setting the Client ID for each new publish message;
(16) **MQTT Client Username and Password** – modifying the Client username and password for each publish message;
(17) **MQTT Keep Alive Field in the CONNECT Packet** – setting a custom value for each new publish message so that each Keep Alive value represents an ASCII code;

[4]https://github.com/indianatoms/Stego_Tester
[5]https://github.com/microkost/Steganography-for-IP-networks

Table 3: Summary of test results.

Test ID	Steganography Type	Alert Fired	*TPR* [%]	*FPR* [%]
1	ICMP Identification Modulation	Trend Test	100.00	0.00
2	ICMP Sequence Number Modulation	Consistency	100.00	0.00
3	ICMP Interarrival Packet Time Manipulation	Interval Gap Test	100.00	0.12
4	IP Identification	Frequency Test	87.50	3.87
5	IP Time To Live	Frequency Test	90.48	0.01
6	IP ESCP ECN	Frequency Test	90.00	0.19
7	TCP Source Port	Frequency Test	89.29	6.64
8	TCP Sequence Number	Trend Test	88.46	11.14
9	TCP Reserved Bits	Existence Test	100.00	0.00
10	TCP URG	Existence Test	100.00	0.00
11	TCP RST	Existence Test	100.00	0.00
12	TCP Window	Frequency Test	90.48	5.16
13	MQTT Publish Payload	Out of the context Test	94.74	2.22
14	MQTT Subscribe Topic	Out of the context Test	90.00	1.20
15	MQTT Client ID	Frequency Test	94.74	2.05
16	MQTT Client Username and Password	Frequency Test	95.45	1.95
17	MQTT Keep Alive Field in the CONNECT Packet	Frequency Test	94.74	2.03
18	MQTT RETAIN	Frequency Test	95.65	9.03
19	MQTT CLEAN	Frequency Test	95.45	6.64
20	SIP CALL-ID	Frequency Test	90.00	0.52
21	SIP MAX FORWARDS	Frequency Test	90.00	0.57
22	SIP CONTACT	Frequency Test	90.00	0.57
23	SIP CSEQ	Frequency Test	90.00	0.54

(18) **MQTT RETAIN** – setting a custom value for each new publish message so that the Retain message indicates logical 1 and its lack indicates logical 0;
(19) **MQTT CLEAN** – setting a custom value for each new subscribe message so that the Clean Session flag indicates logical 1 and its lack indicates logical 0;
(20) **SIP CALL-ID** – sending encrypted elements of a message inside of the Call-ID parameters in consecutive SIP elements;
(21) **SIP MAX FORWARDS** – sending encrypted elements of a message inside of the MAX-FORWARDS parameters in consecutive SIP elements;
(22) **SIP CONTACT** – sending encrypted elements of a message inside of the CONTACT prmts in consecutive SIP elements;
(23) **SIP CSEQ** – sending encrypted elements of a message inside of the CSEQ prmts of consecutive SIP Call-ID elements.

For each of the above described cases, $20 - 22$ hidden channels were created with various parameters (e.g., length of the hidden transmission). For evaluation, we used the True Positive Rate (TPR) and False Positive Rate (FPR) metrics.

4.3 Results

A summary of the test results is presented in Table 3. The "Alert Fired" column contains information about the tests which raised an alert about potential steganographic transmission. The next columns present the obtained TPR and FPR values. They show that detection capabilities of the proposed solution are high – in all test cases, TPR reached almost 90% or more. In five test cases the detection was perfect. As for the false positives, in many test cases the FPR values were either zero or pretty low. However, in a few test cases (e.g., 7,8,18,19) we also encountered a significant number of false positive alarms. This will be discussed in the next section.

The PCAP file of 1.4GB was processed in $2'31.3''$, i.e., 0.036 of real time traffic, consuming about 160 MB of memory. It means that the traffic data was processed at the speed of about 72 Mbps.

5 DISCUSSION

In this paper, we presented the project, implementation and test results of a lightweight network steganography detection system. The work involved designing data structures and the data flow process, developing detection methods, and integrating several system components. Despite the low complexity of our solution, we have achieved positive results. However, work is still ongoing and our base system needs further development.

Detection failures were caused, e.g., by the fact that the heuristics for TCP window size and source port distribution turned out to be insufficient. They need to be improved in the future, e.g., by using an ML-based approach. More analytical focus on the heuristics for setting up parameters, such as θ or T in the frequency test, is also foreseen.

The same refers to the FPR values as some tests yielded unacceptably high values, i.e., 5% or even more. They can be lowered by moving a decision threshold, but this would likely impact detection capabilities. We plan to look for better models of the TCP sequence numbers and the MQTT packet parameters to find better heuristics. For now, the use of the TCP Sequence Number test, the MQTT CLEAN or MQTT RETAIN tests may lead to undesirable behavior.

We also noticed that if Zeek does not support a specific protocol (e.g., RTP), adding support for that protocol, i.e., writing an

additional analytical module (called 'analyzer' in the Zeek nomenclature), can be quite complicated. Development of a new analyzer requires the use of the Spicy[6] protocol analysis framework. It may also require the creation of new event structure or data types, which may require the support of the Zeek developers. In contrast, adding detection of hidden transmission in other protocols (such as UDP) can be realized analogously to the presented examples – it simply requires adding dedicated scripts. As for scalability, we checked that adding support for a new protocol or steganographic method increased the processor load by 0.2% relative, which means an increase much slower than linear.

ML-based techniques can also be helpful in improving the precision of steganography detection. An ML-based anomaly detection module is also worth considering, due to the availability of the Python library called Zeek Analysis Tools[7] (ZAT), which supports converting raw Zeek data to formats readable by ML packages such as Pandas, scikit-learn, and Spark. It is worth noting, however, that such an ML-based solution would not be lightweight anymore.

Data visualization still needs to be improved as it may be useful for a more accurate interpretation of the results. We are also considering adding a new panel in Kibana with ZAT-based graphs created in matplotlib. Moreover, statistical operations can be further enhanced with SumStats framework[8]. Postfix can also be considered a system element to be updated. It could be replaced with an API-equipped communication system, which would make our system easier to use.

6 CONCLUSIONS AND FUTURE WORK

Detecting network steganography is still difficult and there are no fully effective methods. This is becoming a vital issue due to the existence of fairly easy ways to hide information in network traffic and the associated risk. A significant and growing share of malware uses steganography during malicious activities. In this paper, we presented how to adapt an open-source IDS system (in our case Zeek) to detect steganography transmission. We showed that the proposed system, based on a set of detection rules, was able to detect the vast majority of steganographic test cases. Our solution may also be considered "lightweight" because, in contrast to the competing approach, it requires no computation-heavy plugins, so it can work well under heavy traffic load.

Zeek turned out to be a responsive and flexible solution increasing the security of network transmission, including the detection of steganographic methods. Extending it with dedicated scripts and data structures was sufficient to implement a steganalytic solution. Zeek has proven to be a solid base to build on, eliminating the need to design a tool from scratch. The proposed solution is scalable and can be further developed. We plan to improve its detection capabilities in areas where it still needs improvement (e.g., the detection of TCP Sequence Number-based steganography). The scripts used for the experiments are freely available at https://github.com/indianatoms/Stego-Aware-NIDS, so researchers are welcome to develop our solution further or challenge it with other types of steganographic attacks.

[6]https://github.com/zeek/spicy
[7]https://github.com/SuperCowPowers/zat
[8]https://docs.zeek.org/en/current/frameworks/sumstats.html

ACKNOWLEDGMENTS

This work has been supported by the SIMARGL Project – Secure Intelligent Methods for Advanced RecoGnition of malware and stegomalware, with the support of the European Commission and the Horizon 2020 Program, under Grant Agreement No. 833042.

REFERENCES

[1] Bhoopesh S. Bhati and Chandra S. Rai. 2019. A Survey on Intrusion Detection Tools. In *2019 6th International Conference on Computing for Sustainable Global Development (INDIACom)*. 806–810.
[2] Luca Caviglione, Michał Choraś, Igino Corona, Artur Janicki, Wojciech Mazurczyk, Michał Pawlicki, and Katarzyna Wasielewska. 2021. Tight Arms Race: Overview of Current Malware Threats and Trends in Their Detection. *IEEE Access* 9 (2021), 5371–5396. https://doi.org/10.1109/ACCESS.2020.3048319
[3] James Collins and Sos Agaian. 2016. Trends Toward Real-Time Network Data Steganography. *International Journal of Network Security & Its Applications* 8 (03 2016), 01–21. https://doi.org/10.5121/ijnsa.2016.8201
[4] Elastic.Co. [n.d.]. Zeek (Bro) Module | Filebeat Reference [7.7] | Elastic. https://www.elastic.co/guide/en/beats/filebeat/current/filebeat-module-zeek.html.
[5] Elasticsearch. [n.d.]. https://www.elastic.co/elasticsearch/. Accessed: 24.05.2021.
[6] Hendra Gunadi and Sebastian Zander. 2017. *Bro Covert Channel Detection (BroCCaDe) Framework: Scope and Background.* Technical Report. Murdoch University.
[7] Hendra Gunadi and Sebastian Zander. 2017. *Comparison of IDS suitability for covert channels detection.* Technical Report. Murdoch University. https://www.semanticscholar.org/paper/Comparison-of-IDS-Suitability-for-Covert-Channels-Gunadi/
[8] Ki-Hyun Jung. 2019. A Study on Machine Learning for Steganalysis. In *Proceedings of the 3rd International Conference on Machine Learning and Soft Computing* (Da Lat, Viet Nam) *(ICMLSC 2019)*. Association for Computing Machinery, New York, NY, USA, 12–15. https://doi.org/10.1145/3310986.3311000
[9] Kibana. [n.d.]. https://www.elastic.co/kibana. Accessed: 24.05.2021.
[10] Ivo Kostecky. 2018. ICMPStegano. https://github.com/microkost/Steganography-for-IP-networks.
[11] Deepthi H. Lakshminarayana, James Philips, and Nasseh Tabrizi. 2019. A Survey of Intrusion Detection Techniques. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. 1122–1129. https://doi.org/10.1109/ICMLA.2019.00187
[12] Józef Lubacz, Wojciech Mazurczyk, and Krzysztof Szczypiorski. 2014. Principles and overview of network steganography. *IEEE Communications Magazine* 52, 5 (2014), 225–229. https://doi.org/10.1109/MCOM.2014.6815916
[13] Wojciech Mazurczyk, Steffen Wendzel, Sebastian Zander, Amir Houmansadr, and Krzysztof Szczypiorski. 2016. *Information hiding in communication networks: fundamentals, mechanisms, applications, and countermeasures.* John Wiley & Sons.
[14] Miralem Mehić, Jiri Šlachta, and Miroslav Voznak. 2015. Hiding data in SIP session. In *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*. 1–5. https://doi.org/10.1109/TSP.2015.7296445
[15] Aleksandra Mileva and Boris Panajotov. 2014. Covert channels in TCP/IP protocol stack - Extended version. *Central European Journal of Computer Science* 4 (2014), 45–66. https://doi.org/10.2478/s13537-014-0205-6
[16] Arooj Nissar and Ajaz H. Mir. 2010. Classification of steganalysis techniques: A study. *Digital Signal Processing* 20, 6 (2010), 1758 – 1770.
[17] Anton Noskov and Frantisek Jakab. 2016. Analysis of network protocols: the ability of concealing the information. In *2016 International Conference on Emerging eLearning Technologies and Applications (ICETA)*. 245–249. https://doi.org/10.1109/ICETA.2016.7802098
[18] Postfix. [n.d.]. http://www.postfix.org/. Accessed: 12.05.2021.
[19] Craig H. Rowland. 1997. View of Covert channels in the TCP/IP protocol suite. https://firstmonday.org/ojs/index.php/fm/article/view/528/449. (Accessed on 04/16/2021).
[20] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani. 2018. Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In *Proc. 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*.
[21] Trustwave. [n.d.]. Pingback: Backdoor At The End Of The ICMP Tunnel. https://www.trustwave.com/en-us/resources/blogs/spiderlabs-blog/backdoor-at-the-end-of-the-icmp-tunnel/. (Accessed on 06/13/2021).
[22] Aleksandar Velinov, Aleksandra Mileva, Steffen Wendzel, and Wojciech Mazurczyk. 2019. Covert Channels in the MQTT-Based Internet of Things. *IEEE Access* 7 (2019), 161899–161915. https://doi.org/10.1109/ACCESS.2019.2951425
[23] Elżbieta Zielińska, Wojciech Mazurczyk, and Krzysztof Szczypiorski. 2014. Trends in steganography. *Commun. ACM* 57, 3 (2014), 86–95.
[24] Tanja Zseby, Felix Iglesias Vázquez, Valentin Bernhardt, Davor Frkat, and Robert Annessi. 2016. A Network Steganography Lab on Detecting TCP/IP Covert Channels. *IEEE Transactions on Education* 59, 3 (2016), 224–232.