Anti-forensic = Suspicious: Detection of Stealthy Malware that Hides Its Network Traffic

Mayank Agarwal, Rami Puzis, Jawad Haj-Yahya, Polina Zilberman, Yuval Elovici

Anti-forensic = Suspicious: Detection of Stealthy Malware that Hides its Network Traffic

Mayank Agarwal[1][0000−0002−6374−6737], Rami Puzis[1][0000−0002−7229−3899], Jawad Haj-Yahya[2][0000−0003−2911−0329], Polina Zilberman[1][0000−0003−3593−7330], and Yuval Elovici[1]

[1] Telekom Innovation Labs, Ben-Gurion University of the Negev, Israel. agarwalm@post.bgu.ac.il, puzis@bgu.ac.il, polinaz@post.bgu.ac.il, elovici@bgu.ac.il

[2] School of Computer Science and Engineering, Nanyang Technological University, Singapore. jawad@ntu.edu.sg

Abstract. Stealthy malware hides its presence from the users of a system by hooking the relevant libraries, drivers, system calls or manipulating the services commonly used to monitor system behaviour. Tampering the network sensors of host-based intrusion detection systems (HIDS) may impair their ability to detect malware and significantly hinders subsequent forensic investigations. Nevertheless, the mere attempt to hide the traffic indicates malicious intentions. In this paper we show how comparison of the data collected by multiple sensors at different levels of resilience may reveal these intentions. At the lowest level of resilience, information from untrusted sensors such as netstat and process lists are used. At the highest resilience level, we analyse mirrored traffic using a secured hardware device. This technique can be considered as fully trusted. The detection of a discrepancy between what is reported by these common tools and what is observed on a trusted system operating at a different level is a good way to force a dilemma on malware writers: either apply hiding techniques, with the risk that the discrepancy is detected, or keep the status of network connections untouched, with a greater ability for the administrator to recognize the presence and to understand the behaviour of malware. The proposed method was implemented on an evaluation testbed and is able to detect stealthy malware that hides its communication from the HIDS. The false positive rate is 0.01% of the total traffic analysed, and barring a few exceptions that can easily be white-listed, there are no legitimate processes which raise false alerts.

Keywords: Stealthy, Malware, Command & Control, Trusted Network Monitor, Security

1 Introduction

In the recent years, there has been an increase in the amount of malware related security incidents resulting in leakage of personal and corporate information, DDoS attacks, data loss, etc. [19,3,5]. Malware poses a serious security

threat to organizations, computer networks, and end users. As the sophistication of malware has advanced, its detection has become more complex. Malware developers employ various anti-forensic techniques in order to evade the detection mechanisms deployed on the targeted systems. Prominent anti-forensic techniques include: file and directory hiding, process hiding, anti-VM, anti-debug, hiding sockets and connections, etc. Examples of malwares that employ anti-forensic techniques include: vlany, azazel, enyelkm [11,2,6] etc.

A host-based intrusion detection system (HIDS) usually consists of various sensors (e.g., the process sensor, CPU sensor, memory sensor, network sensor, etc.) in order to monitor the various components of the host. There are stealthy malwares[3] that employ network hiding[4] anti-forensic feature [11,2]. Malware equipped with such anti-forensic features are harder to detect using HIDS, because their network communication data is not available. A network intrusion detection system (NIDS) may be able to flag network packets if they are sent to suspicious domains or if the traffic patterns resemble an anomalous behavior (e.g., ping sweep). However, neither rules nor machine learning algorithms commonly employed by NIDS are 100% accurate. Irrespective of whether the NIDS flags such packets or not, a NIDS cannot determine if a malware tried to hide its network communication from the HIDS. Furthermore, the technique used by the malware to hide its network communication from the HIDS cannot be ascertained. Notwithstanding any other malicious activities performed by a malware, the fact that a stealthy malware attempts to hide its network communication is itself an indication that the target system is compromised. However, this information (hiding network communication) is crucial for efficient forensic investigation.

The use of HIDS or NIDS alone are insufficient to detect the presence of stealthy malwares that hide their network communication. In order to detect stealthy malware, we employ a detection philosophy that combines from both the observations from host and network sensors in order to ascertain whether an adversary attempts to hide its traffic from HIDS.

The main contributions of this research are: (1) We propose a multi-level monitoring method which combines resilient trusted network monitor with multiple untrusted host-based sensors. By untrusted we mean that they can be circumvented by attackers with various skill levels. (2) The proposed multi-level monitoring method detects stealthy malware that attempts to hide its network traffic and also determines the technique adopted by the attacker. (3) The proposed method has been evaluated using stealthy malwares obtained from Internet sources. The evaluation results confirm that all the stealthy malwares that were

[3] In this manuscript, the term stealthy malware is used to refer to malware that has network hiding anti-forensic capability. A stealthy malware can possess other anti-forensic mechanisms in addition to network hiding.

[4] By network hiding, we refer to communication in which the malware hides its network communication from network sensors (e.g., Wireshark, dumpcap, netstat, etc.) that a HIDS may employ in order to hide its network activity.

tested are detected successfully while the false positive rate stands at a mere 0.01% of the total packets analyzed.

The rest of this paper is organized as follows. Section 2 describes the multi-level monitoring method which is the core of the detection method proposed for the identification of stealthy malware. The experimental setup, information regarding malware dataset, and the results obtained using the proposed method are presented in Section 3. In Section 4 we present the existing approaches used for stealthy malware detection, and Section 5 concludes the work.

2 Proposed Method

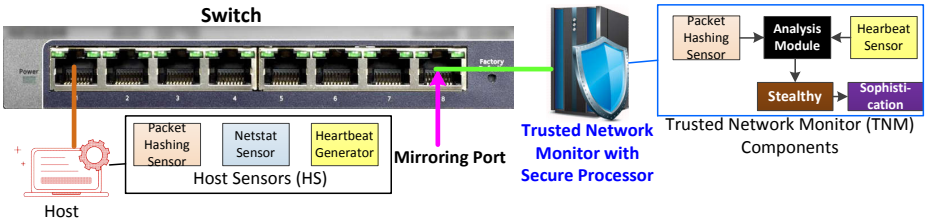

Fig. 1. Multi-level monitoring method: Uses host sensors (HS) and the Trusted Network Monitor (TNM)

The proposed multi-level monitoring method consists of the host sensors (HS) and a trusted network monitor (TNM) as shown in Fig. 1. The HS are deployed on the host and consist of packet hashing and netstat sensors. In addition to these, there is a heartbeat generator that runs on the host which informs about the running status of the HS. The HS are considered untrusted as they can be subverted by a malware. The TNM is deployed on a dedicated machine and is connected to the network via the port mirroring interface. Due to the port mirror configuration, the TNM receives a copy of all of the packets sent by the host which traverse the switch.

Ideally, in the absence of stealthy malware, for a fixed time interval in which the network communication is measured using HS and TNM, there should be no discrepancy between the measurements obtained by them. A benign application will never hide its network communication from the host sensors, as it does not have any malicious intentions. However, if a discrepancy arises when the traffic measured by the host sensors and the TNM is compared, it provides an evidence of suspicious activity on the host. Not all discrepancies are considered instances of stealthy malwares as explained below:

1. *Discrepancy arising out of localhost traffic:* Localhost traffic generated on the host is captured by the host sensors, but since the traffic is destined for localhost, it does not reach the switch interface. Consequently, it is not

mirrored on the TNM. Such a discrepancy is not considered malicious in the current research and is ignored.

2. *Discrepancy arising from packet losses:* Packets that are observed on the host sensors but are not seen on the TNM are ignored. Such a scenario might occur as packets seen on host sensors might not be mirrored correctly possibly due to too many packets being mirrored on the TNM at a given time. Also, because the host sensors observe the network communication, the malware is not hidden from the host sensors and does not fall into the category of the stealthy malware being considered.
3. *Discrepancies arising due to the presence of malware:* Packets sent from the host that were observed on the TNM but were hidden from the host sensors). Such communication definitely points out the presence of malware on the host which evades the host sensors. A malware can hide its network communication in various ways like patching the HS deployed, exploiting bugs that may exist in the libraries used by the HS deployed, detecting their execution and disabling them etc. The goal of a malware developer is to eventually hide its traces in order to escape detection.

Thus, by capturing the network communication at different levels of resilience (by HS and the TNM in our case), and comparing this data, different types of discrepancies can be identified, and the presence of stealthy malware can be ascertained. As the stealthy malware has no control over the TNM, it cannot hide its network communication from the TNM. The various components of the host sensors and TNM are shown in Fig. 1 and described below.

Packet Hashing Sensor: This sensor is a part of host sensors as well as the TNM. The packet hashing sensor is a network packet sniffer deployed in promiscuous mode. It must be noted that host sensors only capture the traffic pertaining to the host on which they are deployed, while the TNM can capture traffic from multiple hosts simultaneously. The packet hashing sensor ignores packets having broadcast and multicast address. The reason for ignoring broadcast and multicast communication is that a stealthy malware aims to minimize its footprints in order to evade detection mechanisms. If it sends broadcast or multicast communication, the respective frame is received by many hosts in the network. This leads to an increase in the footprint of the malicious communication which might eventually aid the detection mechanisms.

The packet hashing sensor only records information relating to transmitted (Tx) packets from the host and does not record the packets received (Rx) by the host. The motivation behind this philosophy is twofold: a) the stealthy malware on the host usually initiates the connection (Tx traffic) with the C&C (or some other host) and would attempt to hide this communication from the host sensors, and b) monitoring Rx leads to additional load on the TNM.

The packet hashing sensor saves the following information for every transmitted packet (src-ip, dst-ip, src-port, dst-port, seq-num, and ack-num), where src-ip (dst-ip) is the source (destination) IP address of the connection, src-port (dst-port) is the source (destination) port address of the connection, and seq-num (ack-num) represents the sequence (acknowledgment) number of a TCP

connection. For UDP packets, the seq-num and ack-num fields are set to null. In addition, this module computes and stores the hash of the above tuple for every transmitted packet which is later used for comparison purposes.

Netstat Sensor: This sensor is only a host sensor component. The netstat sensor records all of the information obtained from the netstat utility. It stores the unique connections (src-ip, dst-ip, src-port, dst-port) as seen on the host.

Analysis Module: The analysis module is only present on the TNM. The packet hashing sensor on the host and the TNM, along with the netstat sensor on the host send their information to the analysis module. On examining the information received from these sensors, the analysis module determines if there exists a stealthy malware that is trying to communicate with its C&C server or any other compromised machine in the network.

Heartbeat Generator and Heartbeat Sensor: In addition to the host sensors the multi-level monitoring method also makes use of a heartbeat generator on the host and a heartbeat sensor on the TNM. The main task of the heartbeat generator is to ensure that the host sensors are running and have not been terminated or disabled by the stealthy malware. In order to accomplish this, the heartbeat generator running on the host sends periodic keep-alive messages to the heartbeat sensor running in the TNM.

2.1 Customized Secure Processor

As described above, the TNM may need to monitor multiple hosts which may be infected by stealthy malware. It is imperative in such cases to ensure that the TNM's operations are carried out in a trusted environment. Specifically, a malware should not be able to manipulate operations of the TNM and its communication with the switch. To ensure this, the TNM software is running as a trusted application on a customized secure processor. The secure processor implements various features to protect against known attacks on the computing system hardware and the software that is running on it. The features include:

1. **Secure IO and Secure Debug** - protect against various hardware threats such as key extraction, illicit debugging, probing, and side-channel at tacks (SCA).
2. **Secure Boot** - protects against attacks such as: image hacking, botnet enrolling, and cold boot attacks.
3. **Trusted Execution Environment (TEE)** - guarantees an isolated execution environment for the trusted application. This feature is essential for protecting against attacks such as: software exploitation, privilege escalation, and botnet enrolling.
4. **Secure Storage** - an important feature that exists in secure processors in order to protect against SCAs, probing, and key extraction. In addition, it is used by the TEE to load and execute the trusted applications, while protecting the code and data of the applications running.
5. **Secure Disk Storage** - for recording and analyzing network packets and sending statistics to the Splunk server.

6. **Customizable Crypto Primitives** - these include ECC, ECDSA, AES.
7. **Dual Ethernet Ports** - the first of these ports is used for the local area network (LAN), and the second is used for streaming the mirrored data from the switch to the TNM.

All of the features above ensure that the TNM is unaffected by the presence of malware running on a host monitored by the TNM. Hence, the network communication performed by the stealthy malware eventually appears on the TNM (if they are destined for the switch), even if the malware succeeds in hiding itself from the host sensors.

3 Evaluation of the Proposed Method

In this section, we provide a detailed explanation of the experimental setup and attack model. The description of the attack model covers the various sophistication levels of the malware and their detection likelihood under distinct detection approaches. This is followed by subsections that discuss the malware dataset characteristics chosen for this research, the false positives obtained, and the runtime performance of the host sensors and the analysis module.

3.1 Experimental Setup

The experimental setup is similar to the one shown in Fig. 1 except that we used five machines to execute the malware. On each of the five machines executing the malware, different operating systems (OSs) were installed depending on malware requirements; the host sensors are deployed on each of the five machines. The switch port on which the TNM is connected is configured as a mirror port, so that it can sniff the traffic sent by the five machines.

3.2 Attack Model

We assume that a stealthy malware possesses at least one the following features (ordered based on the level of sophistication, from the least to the highest) as shown in Table 1. It shows the features detectable by different detection mechanisms for the malwares that possesses feature A, B, and C. A good detection system is the one which can detect all of the features possessed by the stealthy malware. For explanation, we consider a malware having feature C (malware hiding from netstat and packet hiding sensor) and look at the features detectable by the different detection mechanisms shown in Table 1: i) **Host sensor with only the netstat sensor deployed:** Since the malware hides itself from netstat, 'only netstat' monitoring would not help to detect the feature C. ii) **Host sensor with only the packet hashing sensor deployed:** As the malware hides itself from the packet hashing sensor, it fails to detect feature C. iii) **Host sensor with both the netstat and packet hashing sensors deployed:** The malware still remains undetected as it hides itself from both of them. iv)

Only NIDS deployed: The malware traffic may be flagged depending on the traffic generated by the malware and the NIDS configuration, however NIDS alone cannot determine if the malware hides its network communication from host sensors. v) **Both host sensors and the TNM are deployed (proposed approach):** The proposed multi-level monitoring method not only detects the stealthy malware, but it also identifies the network hiding anti-forensic techniques it adopted.

Table 1. Anti-forensic mechanisms of stealthy malware and features detectable by the different detection mechanisms

		Detection Mechanism				
		Host Sensor (Only Netstat Sensor)	Host,Sensor (Only Packet Hashing Sensor)	Host,Sensor (Netstat + Packet Hashing Sensor)	Only NIDS	Host Sensors + TNM (Proposed Method)
Feature	**Sophistication Level**	**Features Detected**				
A	**Hidden**: Netstat sensor **Visible**: Packet hashing sensor	None	None	A	None, but may flag suspicious communication	A
B	**Hidden**: Packet hashing sensor **Visible**: Netstat sensor	None	None	B	None, but may flag suspicious communication	B
C	**Hidden**:netstat and packet hashing sensor	None	None	None	None, but may flag suspicious communication	C

3.3 Malware Dataset Characteristics and Evaluation

We used the virussign.com dataset[10] for evaluating the malware samples. A total of five machines were used to execute malwares. A set of sixty malwares were executed on a single machine with the host sensors deployed. Thus five machines could execute 300 malware samples. The network activities generated by these (300) malwares is collected over a period of twenty minutes and the statistics from the various sensors are collected and analyzed. The whole setup of executing 300 malwares and recording the activities for a period of twenty minutes constitutes a single run. We execute 10 such runs.

Ideally, without the presence of malwares, the number of connections observed on the host sensors and the TNM must be identical. A connection implies a tuple consisting of (src-ip, dst-ip, src-port, dst-port). Figure 2 depicts the connections made by the network hiding stealthy malware that were observed by netstat, packet hashing sensor (PHS), or TNM while the Table 2 shows the individual statistics per run. It can be concluded from Fig. 2 that circumventing connections from netsat is the easiest. Netstat observed the least number of hidden connections in comparison to the other senors. There are a large number of connections which are observed by the packet hashing sensor which are missed

Table 2. Connections observed by netstat, packet hashing sensor (PHS), or TNM.

Run #	Seen on netstat but hidden from PHS	Seen on PHS but hidden from netstat	Seen on TNM but not on PHS	Seen on TNM but neither on PHS & netstat
1	16	147	1317	1305
2	148	259	264	127
3	3	4350	1588	1588
4	120	453	10	10
5	472	1039	1281	811
6	1174	2619	2102	1666
7	170	3368	55	55
8	291	8881	261	221
9	355	10687	5407	5407
10	6	1850	0	0
Total	2755	33653	12285	11190

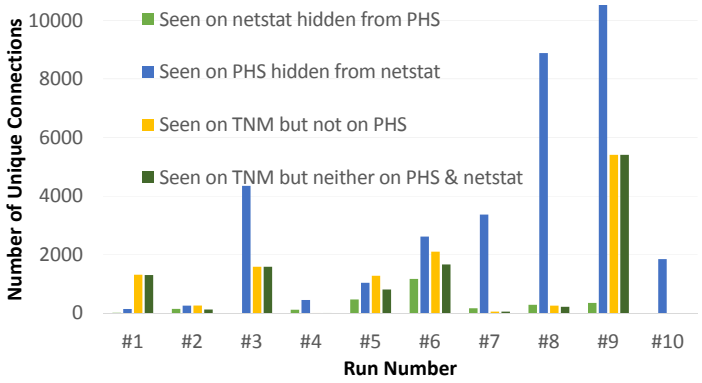

Fig. 2. Statistical observations of the connections made by the network hiding stealthy malware that were observed by netstat, packet hashing sensor (PHS), or TNM.

by netstat. The netstat utility being a part of the operating system is widely used for checking network connection management. Hiding network communication from netstat is one of the most basic anti-forensics mechanism an attacker would employ while developing a stealthy malware that performs network communication. There are connections which are invisible to the packet hashing sensor but are recorded in netstat. However, there are relatively few connections of such type as can be observed in Fig. 2.

Next, we observe that a large number of connections were hidden from the packet hashing sensor that were eventually captured by the TNM. If we compare the connections that were hidden from netstat but visible in packet hashing sensor to those that were hidden from packet hashing sensor but seen on TNM, it can be concluded that hiding network communication from packet hashing

sensors is relatively more complicated as compared to hiding from netstat. But, seeing the number of connections that were hidden from the packet hashing sensor, which were visible on the TNM, shows that many malwares are equipped with network hiding anti-forensic mechanism.

Finally, we inspect the number of connections that were hidden from both netstat and packet hashing sensor but were visible on the TNM. Such malwares possess the most sophisticated anti-forensic mechanism that cannot be detected by an HIDS deployed on the host as the network data is unavailable. We see that the number of such connections are almost the same as the number of connections that were hidden from packet hashing sensor but seen on TNM. A malware developer who writes sophisticated malware to hide from the packet hashing sensor would mostly include the intelligence to hide itself from netstat.

In Fig. 2, some runs (e.g., Run 9) have a large number of connections that were hidden from the host sensors. As we mentioned earlier, malwares employ various mechanism to hide their presence. Some attackers write sophisticated programs to circumvent the host sensors while others just terminate or disable the host sensors. As a result, the host sensors are unable to capture statistics and in the meantime the malware performs its desired activity which eventually gets logged on the TNM. On the other hand, the run 10 of Fig. 2 does not have any packets seen on TNM that were hidden from the host sensors. This shows that the said run did not include malware with network hiding anti-forensic mechanism. So, depending on the nature of the anti-forensic mechanism built into the malware, different sets of observations are obtained.

Irrespective of whether the malware hides its communication from netstat, packet hashing sensor or both, its communication is eventually captured by the TNM due to multi-level monitoring. As the TNM is unaffected by the presence of malwares, it identifies all those connections that the malware attempted to hide from the host sensors. The proposed method not only detects the IPs and ports contacted by stealthy malware, but also determines the level of stealthiness adopted by the malware.

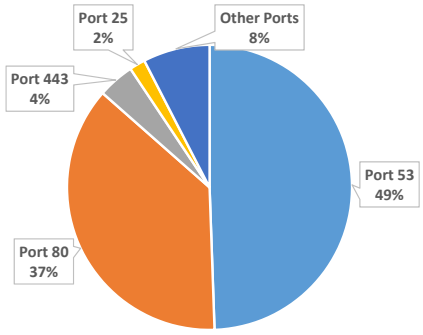

Fig. 3. Distribution of ports that were contacted by the stealthy malware

Figure 3 shows the distribution of top 5 port numbers that were used by the stealthy malware for communication. The ports 53, 80, 443, 25 constitutes a total of the total 93% of the ports communicated by stealthy malwares. The port 53 is used by the DNS server and constitutes a majority 49% of the ports communicate by the stealthy malware. It is obvious that a stealthy malware would want to hide its DNS resolution in order to conceal its activity. Similarly ports 80, 443 are http and https ports that may be used by the malware in order to exfiltrate information to C&Cs.

Table 3. Top 5 IPs contacted by stealthy malwares.

IP	Unique Hits	Threat Intelligence Reports [1,4,7,8,9]
68.178.213.61	224	Malware, Phishing, Spam, Botnet, Ransomware.
194.58.56.172	97	Malware, Phishing.
23.253.126.58	49	Malware, Phishing, Botnet.
104.239.157.210	46	Malware, Phishing, Botnet, Ransomware.
199.2.137.20	44	Malware, Botnet, Blacklisted.

Table 3 shows the top five IP addresses contacted by the stealthy malwares. In total, the proposed method observed connections to 899 unique IPs that were not captured by the HS but seen on the TNM due to multi-level monitoring approach. The threat intelligence websites [1,4,7,8,9] reports all these IPs into various categories like malware, phishing, botnet, ransomware, blacklisted, malicious etc. Not of the threat intelligence websites report all of the IPs as suspicious. As most of the threat intelligence websites are community driven, the threat information depends on its update frequency. We believe that the proposed method can also be used in order to enhance the information available on the threat intelligence websites. The proposed method is capable to detect the newer IPs that a stealthy malware may contact, and also report its stealthiness sophistication. Additionally the proposed scheme can also report on those malwares that subvert the HIDS in-order to contact the C&Cs.

3.4 False Positives and Runtime Performance

As the proposed system involves deep packet inspection of the packets seen on the host sensors and the TNM, the system must be resistant to false positives (FPs). In order to test for the FPs, we deployed our host sensors on various systems that did not contain any malware and monitored them by the TNM. The users were instructed to perform their routine activities. On a few of the systems, some dedicated tasks were assigned like opening random websites, HD video playback, downloading large files, rate limited large file download and uploading data. The goal was to obtain a good mix of the traffic patterns seen during a normal user's day-to-day browsing. Table 4 summarizes the the false positive results obtained for the various users.

Table 4. False positives observed.

Total Users	Packets Captured by TNM	FPs	FP %
11	6,904,567	905	$\frac{905}{6,904,567} = 0.0131$ %

We observed few false positives (905 packets out of 6.9M packets analyzed, representing 0.01311% of all of the traffic). The 905 packets were distributed in the following fashion: *AS15169 Google:* 489, *Ben Gurion University Internal traffic*: 399, *AS36351 SoftLayer Technologies:* 12, *AS16625 Akamai Technologies:* 2, *AS54113 Fastly:* 2, *AS8068 Microsoft Corporation:* 1. A major chunk of the FPs observed belonged to Google and internal Ben Gurion University traffic(the place where we conducted our experiments) and they could easily be avoided by putting them in a whitelist. By configuring a whitelist of the genuine hosts, the FPs can be further reduced. Now we look into the performance metrics of the packet hashing sensor and the analysis module.

We use the dumpcap utility for the packet hashing sensor. The packet hashing sensor consumes an average of 6.329 Mb per run and has a standard deviation of 0.0164 Mb. The analysis module consumes an average of 45.46 Mb per run and has a standard deviation of 13.73 Mb. Both the packet hashing sensor and analysis modules have plausible memory consumption and would not impact the performance of a system on which they are running. Our assumption is that a subset of the machines need to be monitored at a given instance for the presence of stealthy malwares. This helps us to address the scalability issues associated with our approach. Although we use per packet capture analysis for identifying stealthy malwares, we envision to use aggregation based analysis (like traffic flow statistics etc.) as a part of our future work to make it more cost efficient in terms of large scale deployment.

4 Related Work

Analysis of malware can either be static, dynamic or hybrid[16]. In static analysis, the malware is not executed at all. Information regarding the malware is gathered via the file hashes, file type, header details, embedded sources etc. Under dynamic analysis, the malware is executed in a controlled environment; e.g., sandbox, Virtualbox, or an isolated environment etc. In hybrid analysis, a combination of static and dynamic analysis is used. We describe the major approaches currently used to detect stealthy malware.

Intrusion Detection Systems: As mentioned earlier, the use of an HIDS or NIDS alone would be insufficient for detecting the presence of stealthy malware. Malware developers usually employ various techniques, including polymorphic/metamorphic code mutation, entry point obscuration [12,22] etc., which leads to the generation of equivalent code (e.g., the addition of NOPs) or the distribution of malicious code in a benign program in order to evade detection

from signature based IDSs. Anomaly based IDSs have the ability to detect newer threats, however building the normal network profile required by these systems is a challenge. In addition, stealthy malware may not leave significant traces that would enable a signature or anomaly-based IDS to detect its presence.

Emulation based techniques: These became popular because they provided the means for analyzing the behavior of a malicious program in an emulated environment, allowing anti-malware developers to study the behavior of malware and use the observations from the emulated environment to detect malware in real systems. However, malware developers created techniques that enabled them to determine whether they were operating under emulated or real environments, e.g., volume identifiers, network interfaces, special strings [20] etc. Since we deal with stealthy malware, we assume that it has built-in anti-emulation forensics, so that it can evade analysis in an emulated environment.

System hook detection methods: Many stealthy malwares use process hijacking, divert the normal code flow, or modify the function pointers in order to execute the malicious code prior to or after the execution of system calls. Such techniques are known as hooking and they can be performed in a variety of ways. Various malware detection techniques are used to identify the presence of malware by detecting the hooks [18,13,21], however not all hooks are inherently malicious, and as a result, malware detection techniques that simply reply on the detection of hooks can lead to a large number of false positives.

Visualization based techniques: These techniques employ various visualization techniques in order to depict the various connections between the clients and servers [14]. Visualization methods often require manual intervention once its tags anything suspicious, making the process cumbersome.

Noninvasive Techniques: Authors in [17] propose a noninvasive method to detect the malwares possessing anti-forensic mechanisms. Their method requires tracing flow of instructions (Opcode) and the flow of input-output operations (IO) of a malware under multiple execution environments (forensic vs non-forensic) and comparing the traces for suspected vulnerabilities. Although it is a promising approach being noninvasive, executing malwares under multiple execution environments and recording traces is challenging.

Cross view detection techniques: These techniques employ a number of measures to determine the system state, for example, by obtaining the number of files in a directory using system API calls and via actual traversal using a non-API call. Under normal conditions, the results of such operations should yield the same result. If a discrepancy is found, it indicates the presence of malware that hides files from the system. Several detection tools, including Rootkit [15], Strider Ghostbuster etc. make use of cross view detection technique in order to detect the presence of stealthy malware. Cross view detection methods often offer high detection rates. For the detection of stealthy malware that hides its traffic from HIDS, our method makes use of the cross-view detection principle via the host sensors and trusted network monitoring.

Existing techniques make use of a variety of methods in order to detect stealthy malware and often require non-trivial changes at the operating system

level or are limited in terms of detecting the sophistication level of the malware hiding technique etc., making malware detection relatively complex and prone to false positives. In contrast, the proposed multi-level monitoring method is not limited by such constraints and has the potential to effectively detect even newer stealthy malware.

5 Conclusion & Discussion

In this paper, we propose a method to detect the presence of stealthy malware that hides its traffic from HIDSs. In order to escape detection by HIDSs, state of the art malware is equipped with various anti-forensic mechanisms like anti-debug, anti-VM, hiding network connections and network communication. Our proposed method aims to enhance the security of the existing malware detection infrastructure, and we believe it should be integrated with threat detection mechanisms in place in order to increase the robustness and resilience of the existing threat detection mechanisms. If the proposed multi-level monitoring technique will be widely adopted, malware developers would have to identify other means of hiding their network traces. The proposed method not only overcomes the drawbacks associated with the existing approaches for detecting stealthy malware, but also possesses the ability to detect newer malware with network hiding anti-forensic capabilities.

The approach has been tested with real malwares and was shown to be effective against anti-forensic techniques that circumvent host-based traffic sensors. Evaluation was carried out on a testbed with hardware PCs, as opposed to a Virtualbox, sandbox, etc. Thus, we can be confident that stealthy malware that has a built-in anti-VM or anti-sandboxing or anti-emulation technologies, etc. is also executed and detected. Had this research been conducted on a VM or in a sandbox or hypervisor environment, stealthy malware that is able to detect its execution environment would not have been detected.

Detecting malware that performs stealthy communication is crucial as such malware can effectively hide itself from HIDSs. The proposed method not only detects the traffic that was hidden but also identifies the technique used by the attacker to hide its network traffic. Employment of the proposed method leads to a catch 22 situation for the malware developers that need to conceal outgoing traffic – If they do, their presence will be detected using our method. If they don't, their detection using standard measures will be easier. As a result of the proposed method, a malware developer is left with a choice either not to hide its communication from HIDS or rely on steganography, covert channels etc. to conceal their communication. The latter requires much more advanced attack infrastructure (e.g. presence on the organization gateway, man in the middle, etc.) than hooking some libraries on the host machine. Many organizations employ traffic monitoring on their premises in order to detect the presence of malware or identify suspicious traffic. For such organizations, the proposed method can easily be augmented in order to detect malware that hides its traffic from HIDS.

References

1. AbuseIPDB - IP address abuse reports, https://www.abuseipdb.com/
2. Azazel is a userland rootkit., https://github.com/chokepoint/azazel
3. Cloudflare Bug - Cloudbleed May Have Leaked Data From Millions of Sites., https://www.wired.com/2017/02/crazy-cloudflare-bug-jeopardized-millions-sites/
4. Data Mining for Threat Intelligence, https://www.threatminer.org
5. DDoS attacks in Q4 2017 - Securelist, https://securelist.com/ddos-attacks-in-q4-2017/83729/
6. enyelkm - LKM rootkit for Linux x86 with the 2.6 kernel., https://github.com/David-Reguera-Garcia-Dreg/enyelkm
7. Open Threat Exchange, https://otx.alienvault.com
8. Open Threat Intelligence, https://cymon.io/
9. Ransomware Tracker , https://ransomwaretracker.abuse.ch
10. VirusSign — Malware Research & Data Center, Virus Free Downloads, http://samples.virussign.com/samples/
11. vlany is a Linux LD-PRELOAD rootkit., https://github.com/mempodippy/vlany
12. Alam, S., Horspool, R.N., Traore, I., Sogukpinar, I.: A framework for metamorphic malware analysis and real-time detection. Computers & Security **48**, 212–233 (2015)
13. Butler, J., Hoglund, G.: VICE–catch the hookers. Black Hat USA **61**, 17–35 (2004)
14. Chen, S., Guo, C., Yuan, X., Merkle, F., Schaefer, H., Ertl, T.: Oceans: Online collaborative explorative analysis on network security. In: Proceedings of the Eleventh Workshop on Visualization for Cyber Security. pp. 1–8. ACM (2014)
15. Cogswell, B., Russinovich, M.: Rootkitrevealer v1. 71. Rootkit detection tool by Microsoft (2006)
16. Damodaran, A., Troia, F.D., Visaggio, C.A., Austin, T.H., Stamp, M.: A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques **13**(1), 1–12 (Feb 2017)
17. Guri, M., Kedma, G., Sela, T., Carmeli, B., Rosner, A., Elovici, Y.: Noninvasive detection of anti-forensic malware. In: Malicious and Unwanted Software:" The Americas"(MALWARE), 2013 8th International Conference on. pp. 1–10. IEEE (2013)
18. Hoglund, G., Butler, J.: Rootkits: subverting the Windows kernel. Addison-Wesley Professional (2006)
19. Kalita, E.: WannaCry Ransomware Attack: Protect Yourself from WannaCry Ransomware Cyber Risk and Cyber War. Independently published (2017)
20. Musavi, S.A., Kharrazi, M.: Back to Static Analysis for Kernel-Level Rootkit Detection. IEEE Transactions on Information Forensics and Security **9**(9), 1465–1476 (Sept 2014)
21. Rutkowska, J.: Detecting windows server compromises with patchfinder 2. Personal Communication, January (2004)
22. Szor, P.: The art of computer virus research and defense. Pearson Education (2005)