Host Based Intrusion Detection System For Small Network

Hamid Raza, Syed Sohaib Shah, Shayan Ahmed

May 2024

1 Abstract

This paper provides an extensive explanation of the development and implementation of a Host-Based Intrusion Detection System (HIDS) which focuses on enhancing system security via different steps and measures. It includes "Manual and Automatic Port Filtering", "Packet sniffing", "Comparing against YARA Rules", and "ML for the detection of Malicious EXE files". The main purpose of the system is to block ports which are not necessary for our organization by doing so we will eventually mitigate any threat from those ports and securing those ports can tackle several security threats then we will monitor network traffic and in real time test against YARA rules. If the traffic is HTTPS then there is going to be some encryption so we will keep monitoring our download folder and our ML algorithm will try and see if there is any EXE file there or not if it detects an EXE file it extracts the PE information and then inform the user if that file is malicious or not.

2 Introduction

In this age of Modernization and information technology the use of internet is quite wide spread. Every person has at least somewhat exposure to the Internet. "The Internet is a global network of billions of computers and other electronic devices." The availability of Internet to the general public has placed a strain on the security concerns. There are numerous technologies and methods to safeguard one's system/device from being exposed to vulnerable softwares/files. There are two ways to detect Intrusion on one's system. First one is Network Based Intrusion Detection System and the second one is Host Based Intrusion Detection System. Intrusion Prevention systems are used to mitigate the intrusion which is beyond the scope of our project.

Host-Based Intrusion Detection Systems (HIDS) are crucial when we are attempting to secure individual systems from unauthorized access and misuse of critical data. Different from network-based intrusion detection systems (NIDS), which keeps track of the network traffic for suspicious activity, HIDS prioritizes the monitoring and analysis of the events occurring inside a certain host/system. This gives us additional knowledge and control over the security and maintenance of the system. This paper outlines the architecture, design, and implementation of a HIDS that improves and enhances the security of the system through different techniques such as port filtering, monitoring network traffic and Machine Learning.

3 Port Filtering

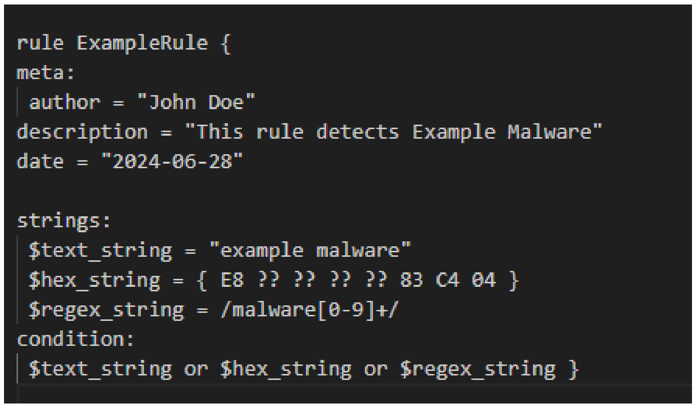
A port is a virtual point where network connections start and end. Each port has a number assigned to it called a "Port Number". It is a 16-bit unsigned integer that ranges from 0 to 65535.[2] They are managed by OS. On a 7 layer OSI model the concept of Port is found on the fourth layer. The fourth layer is the transport layer. TCP and UDP on the transport layer indicate what port should the packet be assigned with.

Each port has its own designated service that they carry out and perform. [2] Port 0 to 1023 are referred to as well known ports. There are two states of a port. Either a port is open or close. An Open Port refers to a TCP or UDP port number that is configured to accept packets. A Close Port is a port that rejects connections or ignores all packets is a closed port. Each port has its vulnerability. A port vulnerability is a weakness or flaw in a network port, which can be exploited by attackers to gain unauthorized access to systems or data. For instance, "Port 135" if left open may allow the attackers to execute arbitrary code, gain unauthorized access to sensitive data or launch DDoS attacks. Port 21 used by FTP, File Transfer Protocol. An unrestricted inbound access to this port can allow attackers to connect to the FTP server and potentially exploit vulnerabilities, leading to unauthorized access or data ex filtration.

In order to mitigate Port vulnerability we have to apply port filtering. When we restrict a port from communicating online it is called port filtering. The ports filtered are categorized as filtered ports. We selected a list of common ports and restricted the rest of the ports. Common ports include port 80, port 443, port 22 and so on. There is also a feature included via which one can block/unblock any port. The study indicates that though a number of open ports still remain in most desktop based user computers, it clarifies that systems can be made really tight by ensuring that no unnecessary ports are left open. [2]

4 Packet Sniffer

After the implementation of port filtering now our task is to monitor the rest of the ports. For this purpose we are going to build a packet sniffer. A packet sniffer can intercept data packets during transmission across a network to capture and then extract useful information from each packet. What are data packets? A data packet is a unit of data transferred over a network. It has some information attached to it. For instance the IP address, port number, checksum, payload



```
rule ExampleRule {
meta:
 author = "John Doe"
description = "This rule detects Example Malware"
date = "2024-06-28"

strings:
 $text_string = "example malware"
 $hex_string = { E8 ?? ?? ?? ?? 83 C4 04 }
 $regex_string = /malware[0-9]+/
condition:
 $text_string or $hex_string or $regex_string }
```

Figure 1: Example YARA Rule

and much more. In our packet tracer we are extracting the Source/Destination IP and Port and the payload. There are two types of packets we are mainly going to work with. TCP packets and UDP packets. We are using the .NET library ”Sharp pcap” to capture packets and extract useful information from it. The most important part is the payload. We are going to compare the payloads against the YARA rules. Rest of the information, related to the IP/port is saved in a log file. This can be used for future forensics. It may give us useful insights about the attacker.

5 YARA Rules

YARA (Yet Another Recursive Acronym) is used for malware research and detection. It forms a way for researchers and security experts to describe malware families using textual or binary patterns. As such, rules are then developed that help in the identification and classification of malware by scanning files, processes, or network traffic for matching patterns. There exist three major constituents within a YARA rule: metadata, strings, and conditions. The metadata section contains authors, descriptions, and creation dates for a rule. The strings section enumerates what patterns in the target data to look for; these may be strings or hexadecimal, or regular expressions. The condition section defines what logic controls whether a rule triggers on a target dataset and can contain logical operators or the count of occurrences, or even more complex expressions. Figure 1 shows an example of a YARA rule.

This rule comes with a metadata section that provides information on the author, description, and date. Then it has a strings section that lists three patterns: a text string ”example malware”, a hexadecimal pattern, and a regular expression that matches the word ”malware” followed by one or more digits. The condition section at the end simply states that the rule will match whenever any of the three patterns are found in the target data.

The libyara.net library is a .NET wrapper of the YARA library. It enables a .NET application to access a subset of YARA features. Using this, a .NET developer can include the compilation of YARA rules, scanning, and result processing in his .NET projects and hence use the power of YARA pattern matching in a .NET environment. This is therefore very useful in developing security tools or applications directed toward the detection and analysis of malware or other kinds of patterns in disparate data sources.

6 Machine Learning

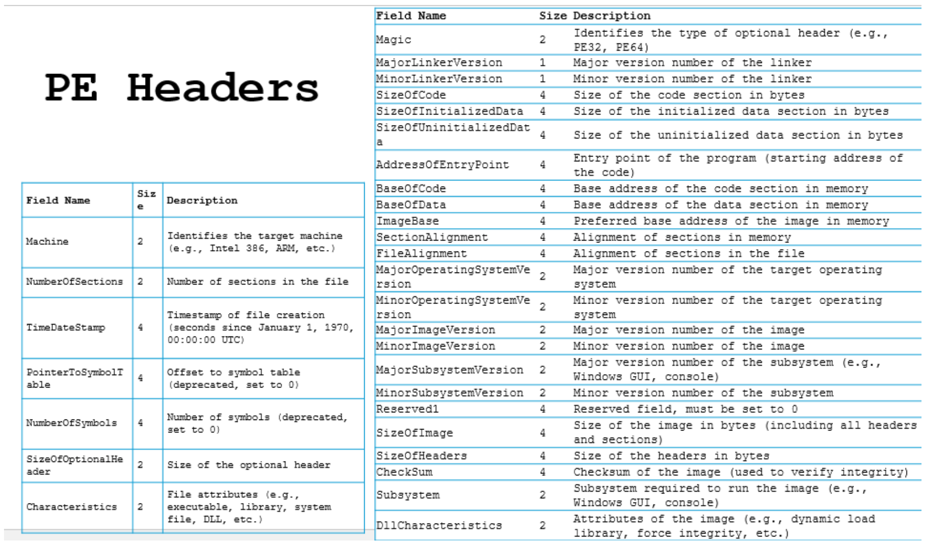
Until now we monitored the packets and the content of the packets. Now is the part when we monitor the files on the system. For example, We keep monitoring our Folder, Downloads, and the moment we detect a PE file we extract the features and check if malicious or not. For this purpose we are going to use Machine Learning.

Machine learning is a branch of Artificial Engineering which is defined as the capability of a machine to imitate intelligent human behavior. There are four types of learning. Supervised learning, Unsupervised learning, Semi-supervised learning, Reinforced learning. Supervised learning is when the present input data is used to reach the result set. Classification and Regression are the types of supervised learning. Classification is distributing the data into the categories defined on the data set according to their specific features. Regression is predicting or concluding the other features of the data based on its some available features.[3]

The portable executable (PE) file format is a windows based file format for executable files based on the UNIX Common Object File Format. The PE format contains instructions for the windows dynamic linker on loading the code stored in the file and necessary libraries to execute the code into memory. [4] There is a brief description of the features of a PE file in figure 2.

A security expert, Prateek Lalwani created a dataset with features extracted from the following: 41,323 Windows binaries (.exe and .dlls), as legitimate files. 96,724 malware files downloaded from the Virus Share website. So, the dataset contains 138,048 lines, in total. We later on downloaded several thousand virus and windows bin files. From the PE files we are extracting 57 features. From the 57 features we selected Twelve (12) important features. Then the Dataset was split into four(4) parts. We selected 3 classifiers, Random Forest Classifier, Gradient Boosting Classifier, and AdaBoost Classifier. The accuracy for Random Forest was 99.4

The model is trained and the model is then saved and mounted onto our Application. We are monitoring the Downloads folder. If there is any EXE file present then the features are extracted and tested against the trained model. The results are then logged.


PE Headers

Field Name	Size	Description
Machine	2	Identifies the target machine (e.g., Intel 386, ARM, etc.)
NumberOfSections	2	Number of sections in the file
TimeDateStamp	4	Timestamp of file creation (seconds since January 1, 1970, 00:00:00 UTC)
PointerToSymbolTable	4	Offset to symbol table (deprecated, set to 0)
NumberOfSymbols	4	Number of symbols (deprecated, set to 0)
SizeOfOptionalHeader	2	Size of the optional header
Characteristics	2	File attributes (e.g., executable, library, system file, DLL, etc.)

Field Name	Size	Description
Magic	2	Identifies the type of optional header (e.g., PE32, PE64)
MajorLinkerVersion	1	Major version number of the linker
MinorLinkerVersion	1	Minor version number of the linker
SizeOfCode	4	Size of the code section in bytes
SizeOfInitializedData	4	Size of the initialized data section in bytes
SizeOfUninitializedData	4	Size of the uninitialized data section in bytes
AddressOfEntryPoint	4	Entry point of the program (starting address of the code)
BaseOfCode	4	Base address of the code section in memory
BaseOfData	4	Base address of the data section in memory
ImageBase	4	Preferred base address of the image in memory
SectionAlignment	4	Alignment of sections in memory
FileAlignment	4	Alignment of sections in the file
MajorOperatingSystemVersion	2	Major version number of the target operating system
MinorOperatingSystemVersion	2	Minor version number of the target operating system
MajorImageVersion	2	Major version number of the image
MinorImageVersion	2	Minor version number of the image
MajorSubsystemVersion	2	Major version number of the subsystem (e.g., Windows GUI, console)
MinorSubsystemVersion	2	Minor version number of the subsystem
Reserved1	4	Reserved field, must be set to 0
SizeOfImage	4	Size of the image in bytes (including all headers and sections)
SizeOfHeaders	4	Size of the headers in bytes
CheckSum	4	Checksum of the image (used to verify integrity)
Subsystem	2	Subsystem required to run the image (e.g., Windows GUI, console)
DllCharacteristics	2	Attributes of the image (e.g., dynamic load library, force integrity, etc.)

Figure 2: Features of a PE file

7 Conclusion and Future work

In summary we have used the technique of port filtering to filter out all the unnecessary ports. This step will totally mitigate any risk from various open ports. After this first step we implemented a packet sniffer. From the data extracted from the payload of packets captured via the packet sniffer, we made a comparison against the YARA rules. Then we used Machine Learning to monitor a specific part of our system in our case the downloads folder. We used different classifiers among which random forest was the best. Our model was capable of detecting malicious PE files. This will prevent our system from executing malicious PE files from execution.

In this paper, we showed the implementation of basic technology and mitigation techniques, Port filtering, Packet sniffing/monitoring, YARA rules, Machine Learning. In future more work can be done on the machine learning algorithms. Deep Learning can be used for better detection of anomalies. HIDS can be combined to with an NIDS, this will provide better synergy. Improvements in GUI can be made. Instead of only monitoring PE files other files like PDF and other documents, database files, zip files etc.

8 References

[1] What is a port computer port? https://www.cloudflare.com/learning/network-layer/what-is-a-computer-port/

[2] A Study Of Open Ports As Security Vulnerabilities In Common User Computers. 2014. DOI:10.13140/2.1.1807.2324. Conference: International

Conference on Computation Science and Technology 2014 (ICCST'14)At: Kota Kinabalo, Malaysia.

[3] A Research on Machine Learning Methods and Its Applications. September 2018 Journal of Educational Technology and Online Learning September 2018, DOI:10.31681/jetol.457046.

[4] Machine Learning for Detecting Malware in PE Files. Collin Connors and Dillip Sarkar. Link, https://arxiv.org/pdf/2212.13988.