Host-based Intrusion Detection Systems

dwBruijn – BS in Computer Science

Contents

1 Introduction

1.1 Intrusion Detection Systems

Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of intrusion. An intrusion is defined as attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network. An intrusion detection system (IDS) is a device or software application that monitors a system or a network for malicious activity or policy violations. Any malicious activity or violation is typically reported or collected centrally using an event management system. Some IDS's are capable of responding to detected intrusion upon discovery, these are classified as intrusion prevention systems (IPS).

Intrusion detection research began in 1972, after James Anderson published a report discussing the need to detect security breaches of computing systems. In the early age of IT technology, system administrators used to rely on manual investigation of logs and audit data to detect intrusions, but to accommodate the rapidly developing computing technology, automated intrusion detection systems became necessary. That's why, during the early 1980s, intrusion detection research started with a focus on automation in order to detect anomalies in system audit data. This lead to the development of the first real-time IDS based on expert-written rules in 1985 by Denning and Neumann. The early research laid the groundwork for modern intrusion detection, comprised of manual techniques, algorithms, commercial and open source products all geared towards one goal: the continuous monitoring of computer systems for signs of compromise.

In the past 30 years, networked computing systems have increasingly emerged as critical assets on which state, personal, and industrial infrastructure critically depend. In parallel, the threat of security breaches has risen, especially with the rise of the profitable underground cybercrime economy and nation-state ambitions. Intrusion detection in practice started by focusing on signature and rule-based detection at the network and file levels. These rule-based IDSs are good at detecting known attacks, but cannot identify novel or polymorphic threats. In addition, their computational overhead (i.e. time, CPU, memory costs) is usually high [1]. This has motivated researchers to shift from signature-based detection towards sophisticated statistical learning algorithms and anomaly detection which has shown better results especially against novel threats.

1.2 Intrusion Detection Approaches

The Two main approaches to intrusion detection are signature-based and anomaly based detection. While signature-based detection is used to detect known threats, anomaly-based detection is used to detect changes in behavior and thus potentially novel attacks.

1.2.1 Signature-based Detection

Signature-based detection, also known as misuse detection, uses predefined attack patterns or indicators of compromise (i.e. known malware signatures, file hashes, known packet signatures and byte sequences, known malicious IPs/domains…) to flag matching events. Thus attacks exploiting previously unknown vulnerabilities (zero-day exploits) or using unusual attack vectors generally bypass signature-based detection.

Misuse detection dominates IDS use in practice as it has been the main focus of commercial detection products, generally it consists the first line of defense. Misuse detection relies on a large database of attack signatures which can be cumbersome to use efficiently and needs regular updates, on the other hand, misuse detection generally yields a relatively low rate of false positives and a high rate of false negatives because it looks for something known [2].

The increasing rate of zero-day attacks has rendered signature-based detection progressively less effective because no prior signature exists for such attacks. Polymorphic variants of malware and the rising number of targeted and sophisticated attacks have further undermined the adequacy of this traditional paradigm [3].

1.2.2 Anomaly-based Detection

Anomaly-based detection overcomes the limitation of misuse detection. In anomaly-based detection, a description of normal behavior is learned by observing normal system behavior and a normal profile (baseline) is built, based on which any deviation is flagged as a potential attack, thus the detection of novel and zero-day attacks is possible.

The development of an anomaly-based IDS comprises two phases: the training phase and the testing phase. In the training phase, the normal system is observed to learn a model of normal behavior, and then in the testing phase, a new dataset is used to establish the system's capacity to detect previously unseen attack patterns.

The assumption made here is that malicious behavior differs from typical system and user behavior, so this technique suffers from a high rate of false positives. Moreover, training datasets with large variance can make it easier for an attacker to masquerade their behavior as normal user behavior in order to evade detection.

Many modern IDSs use a combination of signature-based and anomaly-based detection.

1.3 Types of Intrusion Detection Systems

The two most commonly used types of intrusion detection systems are host-based and network-based IDS. HIDS and NIDS complement each other in order to provide the best possible security to an enterprise network [1].

1.3.1 Host-based Intrusion Detection System (HIDS)

A host-based IDS is an IDS which monitors the characteristics of a single host and events occurring within that host for suspicious activities. In practice, a HIDS usually monitors in real-time the behavior of running processes, enforces the integrity of critical system files and registry keys, performs complex log and system call analysis...

A HIDS has excellent visibility into the system state, but poor isolation from the system, meaning that an attacker with access to that system can either mislead or disable the HIDS sensor on that system.

1.3.2 Network-based Intrusion Detection System (NIDS)

Network-based intrusion detection systems are placed at a strategic point(s) within the network to monitor traffic to and from all devices. They use sensors to monitor and analyze passing traffic on

one or more network segments. The IDS checks for attacks or irregular behavior by inspecting the contents and header information of packets moving across the network.

NIDSs rely heavily on signature-based detection, they match the traffic that's passing on the network to a database of known attack signatures. Once an attack is identified, or abnormal behavior is sensed, an alert is generated.

Encryption is one of the major limitations that face network-based intrusion detection. If the network traffic is encrypted, a sensor cannot scan the protocols or the content of the captured packets, so performing analysis and signature matching is almost impossible. To overcome this shortcoming, researchers developed and are still developing behavior-based detection mechanisms that rely on similarity measurements among other techniques to detect intrusions as well as insider activities such as data exfiltration in encrypted environments.

2 Host-based Intrusion Detection System

2.1 HIDS Implementation

HIDSs are generally implemented following either server-agent or local model.

2.1.1 Server-Agent Model

Each host is monitored by a lightweight agent installed on it, all agents report back to the central management server which is responsible of analyzing the events and generating notifications as appropriate. This model works well for fixed infrastructure, however it has some additional overhead such as central server management overhead, and adding/removing agents overhead especially in dynamic environments which involves frequent scaling (adding and removing hosts).

2.1.2 Local Model

Each host monitors itself; actions like monitoring, alerting, and logging take place on the host itself where the HIDS sensor is installed. This eliminates maintenance overhead for the central management server.

The server-agent model is preferred over the local model when dealing with a large number of hosts, in which case centralized management and logging becomes vital [4].

2.2 HIDS Features

Some of the most common features present in most HIDSs are File integrity and access monitoring, log monitoring and analysis, rootkit detection, system call monitoring and analysis, and registry monitoring…

2.2.1 File Integrity and Access Monitoring

File integrity monitoring (FIM) refers to the act of checking sensitive files (operating system, database, application software files, log files...). to determine whether or not they have been tampered with or corrupted. FIM, which is a type of change auditing, verifies and validates these files by comparing the latest versions of them to a known, trusted "baseline." If FIM detects that files have been altered, the incident is logged and usually an alert is generated to ensure further investigation [5].

A file integrity monitoring software examines various aspects of a file to create a fingerprint. It then compares this fingerprint to a known good baseline fingerprint. To create a file fingerprint an FIM software usually examines stuff like: created, modified, and accessed file settings, security and privilege settings, content of the file, core attributes such as file size and creation/modification date…

Files and file systems are often targets for attackers since this is where sensitive information is usually kept. Also, the file system is an integral part of any host system and thus attackers will need to interact with it in most cases in order to achieve anything. Malicious actions against files and file systems often involve modifying or adding new files or metadata to allow unauthorized future access or remove evidence of previous access, that's why FIM is a crucial part of any HIDS.

Sometimes, just monitoring for file changes is not enough, especially when dealing with files that are meant to be updated, that's why a built-in intelligence is needed to detect if changes to certain files that are meant to be changed are positive or negative, such as detecting if a log file was appended to or deleted from.

HIDS with FIM gives admins the ability to see exactly what file changes have been made, by whom, and when. This is the quickest way to detect and limit a breach in real-time before any lateral movement can be made inside the network

2.2.2 Rootkit Detection

A rootkit is a special type of malware that actually replaces or makes changes to existing operating system components in order to alter the behavior of the system, usually with the intent of hiding itself and other processes, files, network connections or other operations that might expose the attacker's activities, Attackers use rootkits to reduce the chance of being detected once they have gained access to a system.

2.2.2.1 User-mode Rootkits

User mode rootkits modify executable files or libraries that interact with the kernel on the user's behalf in order to provide a stealthy environment for the attacker by hiding his activities. This is usually accomplished by adding filtering capability to executable files so that users, including system administrators, receive only the output that the attacker wants them to receive. For example, if the attacker wants to open a port that they will later use as a backdoor into the system, and bind a socket to it, he can add functionality to programs like *netstat* or *lsof* that report information about open ports, the added functionality will filter the output that is returned to the user, showing for instance that the port is closed, while in fact there is a socket listening on that port for incoming connections. Attackers add functionalities to executables by modifying the source code for those programs then compiling them in case they were open source (*netstat*, *ps*, *sshd*… on UNIX) or by patching the binaries and injecting custom code into them in case they were pre-compiled, and then re-installing them on the target system.

For a user mode rootkit to be successful, the system administrator must use these patched executables (utilities) to administer and monitor the system. A system administrator who does not check the integrity of the executable files he is utilizing may very well be fooled by such a rootkit. This stealth may help the attacker maintain access to a system for a long time.

When operating systems components and binaries are regularly checked using file integrity checking tools which are included in most HIDSs, detection of user mode rootkits is quite successful. When an attacker replaces or modifies a binary file its cryptographic hash will change. If a HIDS watches for changes in these cryptographic hashes on a regular basis, comparing the hashes with a table of known good hashes, the rootkit can be easily discovered.

HIDSs can also detect user mode rootkits using behavioral analysis (anomaly-based detection) by calling different system calls and commands/utilities at specific intervals and then comparing the results in order to detect any changes. For instance, a typical backdoor binds a socket to a port and listens for incoming connections from the attacker, rootkits will commonly attempt to hide those ports/connections to help maintain stealth, on Linux, a HIDS can use the *netstat* utility to log all the open ports and inboud/outbound connections on the system at a certain moment in time. Then it uses the bind() system call to attempt to bind a socket to every TCP and UDP port. If it is unable to bind to a port, then that port must be in use and should also be found in the *netstat* logged output. If that port in not included in the logged *netstat* output, then *netstat* is flagged as a possible rootkit because the it may have been modified to hide that specific port.

Because detection is so straightforward, user mode rootkits are no longer common especially in high-profile attacks.

2.2.2.2 Kernel-mode Rootkits

A kernel mode rootkit makes modifications to the kernel itself. The kernel is the interface between the software and the hardware. For instance, Programs that a user or system administrator runs on a Linux system run in user mode and query the kernel for information about files, processes, network connections, etc... When the user wants a listing of files in the current directory he would issue the "*ls*" command, which is a user mode utility. The "*ls*" command would then query the kernel which will return the listing of files to the "*ls*" utility and "*ls*" would then return that information to the user. A kernel mode rootkit would manipulate the information passed back from the kernel to the user mode utility. After performing a file integrity check of the "*ls*" binary confirming that it has not been tampered with, the system administrator trusts the "*ls*" command and thus trusts the information it provides. In other words, the system administrator is not aware that he is not being told "the truth" about the directory listing he received. Kernel mode rootkits provide a stealthier environment than user mode rootkits in which an attacker can operate because they are harder to detect and they are more thorough. It is trivial to determine that a user mode binary file has been tampered with. This is done by regularly comparing the cryptographic hashes on the binaries against a known good database, as noted above. However, detecting that the kernel has been tampered with is more difficult because so many of the tools used for detection are user mode tools, which can be fed inaccurate information by an undermined kernel. Kernel mode rootkits manipulate the information sent back from the kernel to user mode programs by interfering with the system call table. System calls are the fundamental interface between and application and the kernel, and the system call table is a kernel data structure that maintains pointers to the locations in kernel memory where these system calls reside. The rootkit can modify addresses in the system call table so that the address points not to the original function but instead to the attacker's malicious injected function. Or the attacker can modify the base location of the system call table itself, basically replacing the entire legitimate call table with his own table. In

whichever way the system calls are interfered with, the goal of the rootkit is to modify information sent from the kernel to user mode processes in order to hide the attacker's processes, files and network activities. A kernel mode rootkit can consistently "lie" to any user mode process that issues those system calls. For example, in order for a user mode rootkit to hide a file it would have to alter and install malicious copies of all binaries that show files on the system, but since all of those binaries are using the same system call to query the file system, a kernel mode rootkit can fool them all without even knowing who is asking for the information.

A kernel mode rootkit can do much more than just filtering and tampering with data passing from the kernel to the user mode, it can control mostly all aspects of the compromised system. SmartService which became prominent in 2017, is an excellent example of a kernel mode rootkit, it disables and prevents many AVs and security products from running, thereby acting as a bodyguard for other malicious binaries deployed by the attackers.

HIDSs can detect rootkits by conducting kernel memory scans looking for memory-resident rootkits, signature based scans to check the file system for patterns of known rootkit files, log file analysis which might reveal in certain cases a high number of process/daemon crashes which in turn might indicate in certain cases the presence of a kernel mode rootkit, and finding anomalies in the results of different system calls which might be caused by system call hooks installed by a rootkit. For instance, rootkits often attempt to hide running processes to maintain stealth, on Linux a HIDS will look for processes hidden by a kernel mode rootkit by querying the system using the getsid(), getpgid() and kill() system calls and comparing their results [6], every process on Linux has both a session ID and a process ID, sending a zero as the signal to a process using kill(pid, 0) will not kill a process but instead will return a 0 if the process exists, If the process does not exist the return value from kill() will be -1, a HIDS kernel mode detection module cycles through all possible process IDs, issuing all three system calls and comparing the results: If getsid() finds the process but getpgid() or kill() does not, this may indicate that a kernel level rootkit is intercepting the getpgid() and kill() system calls in order to hide a process. In this case, the HIDS will generate an alert and identify the "hidden" process.

2.2.3 Log Monitoring and Analysis

A log file is a file that records events that occur in an operating system or in other running applications. Almost every software running on the system generates logs, mostly for debugging purposes. A log is generated whenever an activity occurs in the software/system. These logs can be used for different purposes including serving as forensic evidence, audit trails, or to understand how the system behave after certain input. Current generation log analyzers work well with logs from known applications. They are able to parse and interpret logs from known formats but fail to parse logs from a new application or a different and unknown source.

Log files provide vital information about systems and the activities occurring on them. For instance, database logs can help trace how data was added and modified, while web server logs reveal patterns of how resources are accessed from the web.

Large networks consist of a variety of hosts running multiple types of software, including security software, running on multiple hardware devices generating multiple log files. The logs generated

daily by these host are massive and not easy to keep track of. It's even more difficult to keep track of all log file formats present in heterogeneous networks in which new software and hardware are often added, each generating a log file with a unique schema.

Attacks these days are fairly sophisticated affecting different parts and applications as well as several systems simultaneously and are carried out over a period of several days and even months. Detecting and preventing such low and slow vectors of attack would require the generation a holistic view of the attack and all the involved systems and their log files rather than analyzing each log individually. Threat detection in such scenarios requires integrating data from multiple sources.

Log Analysis for intrusion detection is the process or techniques used to detect attacks on a specific environment using logs as the primary source of information [5]. A common implementation of generic log analysis in HIDSs that follow the server-agent architecture relies on rule-based or signature-based detection:

A. Monitoring and Collection of log files:
   The acquisition of host log file data mainly includes two categories: the system-level logs, and the application layer logs. On each host, a log monitoring process monitors log files for changes and notifies the log collector. The collector sends log files to the server for analysis.
B. Pre-decoding of log files:
   The purpose of the log file pre-decoding is to extract general information from the log. For example, suppose an "*sshd"* log contains the following entry:

> *APR 2 22:13:06 host sshd [1024]: Accepted password for root from 172.16.24.100 port 4444 ssh2*

Figure 1 SSHD successful login log entry

   After pre-decoding this entry, the date and time: "APR 2 22:13:06", the hostname: "host", and the process name: "*sshd*" will be recorded.
C. Decoding of log files:
   Log file decoding is the process of extracting key information from logs. Usually, regular expressions are used to identify certain keywords. After decoding the log entry above, the message "Accepted password for root from 172.16.24.100", the source IP address "172.16.24.100" and the user name "root" are all extracted and recorded.
D. Analysis of log file:
   The extracted information is matched with a set of predefined rules (signatures) and based on the result the analyzer arrives at a conclusion.
E. Event reporting:
   If a log entry or a combination of log entries matched a signature, the event is logged and an alert is generated.

2.2.4 System call Monitoring and Analysis

Application programs cannot access hardware such as disk, memory or CPU directly, that's why the operating system provides an API that must be used by all programs in order to access the hardware through the kernel. A running process cannot accomplish much without going into kernel mode, therefore system call monitoring and analysis enable one to monitor interaction between application layer programs and the kernel and thus detect abnormal behavior.

The most common techniques used to detect anomalies in system call traces are short sequence-based, and frequency-based detection [7].

The short sequence-based technique creates a profile of normal behavior by extracting subsequences of system call traces of running processes, a system call trace is a sequence of system calls called by various processes during their execution time. An anomaly is detected by looking for a significant deviation of an intercepted system call sequence from the extracted normal traces or the normal profile, a pre-determined threshold decides whether a deviation is significant or not.

In order to monitor and analyze system calls, three main components are needed; the data source or dataset used for training and testing, the feature extraction component, and the classification algorithm.
In the training phase, the classification algorithm takes as input a part of the dataset which contains only normal behavior of processes. The output of the training phase is a detection model. Based on this model, in the evaluation phase it is decided whether the input system call trace is abnormal or not.

In the short sequence based technique, the trace is divided into short sequences of system calls based on window size, an algorithm takes a trace as input and produces a set of distinct or unique sequences of size "w" which is the window size. By applying the short sequence feature extraction technique to all normal traces, a normal profile is built, it contains distinct sequences for each normal trace. Consider the following normal trace of system calls recorded during the normal execution of a certain program: open(), mmap(), read(), socket(), mmap(), execve(), open(), read(), close(), brk().

With a window size of 5, the following set of subsequences will be generated:

open(), mmap(), read(), socket(), mmap()

mmap(), read(), socket(), mmap(), execve()

read(), socket(), mmap(), execve(), open()

socket(), mmap(), execve(), open(), read()

mmap(), execve(), open(), read(), close()

execve(), open(), read(), close(), brk()

This set of subsequences will be recorded in the database as part of the normal profile of this program. Profiles usually consists of thousands of these short sequences. The size of a profile is not proportional to the size of the program, there might be a program which is smaller in size but

generates more sequences than some other larger program because the sequences generated are related to the complexity of different execution paths within the program.

Now suppose during the monitoring phase of this program the following pattern was observed:

socket(), mmap(), execve(), open(), write()

This would generate mismatches with the sequences in the normal profile of that program, this can be caused by the hijacking of the control flow of that process due to the exploitation of a vulnerability in the program itself and the injection of a certain payload which got executed in the context of that program and caused the deviation from normal behavior.

The use of syscall sequences for generating normal signatures can be justified by the fact that security violations are likely to produce abnormal system call invocations.

Other techniques to monitor system calls and program behavior also exist, such as using neural networks to predict abnormal system call sequences after being trained using malicious binaries that issue a combination of abnormal system call sequences. Monitoring and analyzing the arguments passed to system calls along with system call sequences can also be used in detection. Most HIDSs, especially open-source ones, don't rely on system call monitoring to detect intrusions, since implementing such a mechanism is not easy, and the results are not always satisfying due to the hurdles that come with most system call analysis techniques.

Separating malicious system call sequences from normal system call sequences is not always easy, that's why most of the system call analysis techniques give a high false positive rate most of the time [7]. Also, a knowledgeable attacker can slip under the radar by trying to mimic normal user behavior (mimicry attacks), in order to do something like this an attacker will need to have some knowledge of the applications running on the targeted system and the deployed HIDS, this is unavoidable if that HIDS is popular, in which case it's safe to assume that its source code has been leaked or it was reverse engineered at least. So the HIDS must detect intrusion with high accuracy assuming that the attackers inspected and studied its detection mechanisms and inner workings, since security through obscurity is rarely a reliable defense.

2.2.5 Registry Monitoring (Windows)

The registry is a hierarchical database that contains data that is critical for the operation of Windows and the applications and services running on it. The data is structured in a tree format. Each node in the tree is called a key. Each key can contain both sub keys and data entries called values. The registry contains the information required to boot and configure the system, such as system wide software settings, the security database, and per-user configuration settings.

The Windows registry is an effective data source to detect attacks because many attacks rely on the registry to succeed and thus might show up as anomalous registry behavior.

Most Windows programs access a certain set of registry keys during their normal execution, and most users use a certain set of programs routinely while using their machines, so the registry activity tends to be regular over time on a clean system, this makes the registry a good place to look for irregular and anomalous activity. A lot of attacks involve launching programs that have

never been launched before and changing keys that have not been changed before, so if a model of normal registry behavior is computed over a clean system, then malicious registry activity will most likely deviate from it.

Traditional registry monitoring techniques relies on signature-based detection by monitoring a predefined set of "sensitive" keys for any accesses or alterations. A better technique for real-time registry monitoring relies on the RAD system (registry anomaly detection system) or some upgraded variant of it [8]. The RAD system has three basic components: an audit sensor, a model generator, and an anomaly detector. The sensor stores each registry activity in a database to be used for training or it forwards the registry activity to the detector to run analysis on it. The model generator reads data from the database and creates a model of normal behavior, the created model is then used by the anomaly detector to decide if a new registry activity should be considered anomalous or not.

The generated normal behavior data model is based on features extracted from registry accesses. A number of checks is done on any new registry access to determine if this access is consistent with the generated model or not. Some of these features might include: the name of the process responsible for the activity, the type of query being sent to the registry (CreateKey, SetValue, QueryValue…), the accessed key and its value… An anomaly detection algorithm is responsible of performing the consistency checks on new registry activity and coming up with a decision.

An example of common malicious registry behavior:

A lot of malware attempt to add themselves in the auto-run section of the registry under many keys including: *HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run*, this is an easy technique to achieve persistence and survive reboot/shutdown. A program adding itself under auto-run is not malicious in nature since a newly installed program that needs to run whenever Windows starts might also use this technique, but this is a rare event that can definitely be used as a hint that a malware is running on the system, not to mention that program installation should not be considered normal behavior and thus such registry activity warrants investigation.

By monitoring the registry, HIDSs can detect alterations in system configurations, user groups, and installed software. Policy violations and misuses can also be detected using the information acquired from the registry. For instance, unauthorized usage of software like instant messaging or plugging hardware such as USB drives can be detected by HIDSs when the registry is altered.

3 Comparison Between Two Open-source HIDSs

Wazuh and Samhain are two of the most popular open-source HIDS being used today. We examined the components and features of each one of them and noticed both similarities and differences in the way each HIDS goes about detecting an intrusion using common and different implemented detection mechanisms and features.

3.1 Wazuh

Wazuh is a scalable, cross-platform, open source HIDS, it has a powerful correlation and analysis engine and can perform log monitoring and analysis, file integrity monitoring, Windows registry

monitoring, centralized policy enforcement, malware detection, vulnerability detection, real-time alerting and active response…. Wazuh runs on most operating systems including Linux, Windows, OpenBSD, FreeBSD, Mac OS X, and Solaris. Many ISPs, universities, governments, and large corporate data centers are using Wazuh as their main HIDS solution [9].

Wazuh is based on the server-agent model. Agents are deployed on multiple hosts within a network, this allows centralizing the data gathered from each monitored host on a central server where analysis, logging, and alerting can take place. All deployed agents can be centrally managed from the server, since agents are deployed across multiple hosts, making global changes from a central server is critical. The deployed agent is light, and analysis mostly occurs on the server which means very little resource usage on each host [10].

The main components of Wazuh are the Wazuh agent, which runs on all monitored hosts, the Wazuh server (or manager) which collects, analyses the data sent by agents, and forwards this data to Elastic stack where it gets indexed and stored for user-level logical analysis.

Elastic stack (ELK stack) is a combination of open source products designed to help users take data from any type of source and in any format and index, search, analyze, and visualize that data in real-time. Wazuh uses ELK stack to manage logs of events and incidents, it's composed of:

- Logstash/Filebeat: receives data from agents, parses, transforms, and forwards it to ElasticSearch.
- ElasticSearch: Transformed data from logstash or filebeat is indexed, and stored in order to be searched and accessed later.
- Kibana: a flexible and intuitive web interface for data monitoring and visualization.

3.1.1 Wazuh Features

Here are some of Wazuh's most useful features.

3.1.1.1 Log Monitoring and Analysis

When Wazuh gets installed on a host, a number of log files typically existing on the specified operating system are automatically monitored. The administrator can configure Wazuh to monitor other log files if needed.

Wazuh provides the ability to centralize events and logs from deployed agents. Each agent can be configured to send log events to the an Wazuh server for further analysis. In order to integrate network devices such as routers, firewalls… the server can be configured to receive log events using the Syslog protocol over TCP and UDP. Some of the received events might require long-term storage, that's why Wazuh can be configured to log to a database. In addition to logging for every event that matches specific rules, Wazuh can be configured to log everything received, but being able to log everything doesn't necessarily mean that everything must be logged since this requires a considerable amount of resources. Certain compliance regulations and industry standards, such as the Payment Card industry data security standard (PCI DSS), Health Insurance Portability and Accountability Act (HIPAA) among others have specific requirements surrounding log collection and retention.

Log monitoring and analysis inside Wazuh is done by the logcollector and analysisd processes. The logcollector runs on the host, it monitors and collects in real-time the logged events and sends them to the server where the analysisd process analyzes them (decoding, filtering, and classifying).

Wazuh uses rule-based detection to detect anomalies in log events. It has predefined rules (official rules) for common operating systems and applications such as SSHD, Apache web server, MS-SQL database server, common firewalls… It also allows administrators to create custom rules (local rules) to be used for matching events coming from specific applications or devices. Wazuh gives administrators full control over all the rules, so an administrator can update any rule and its severity to accommodate their environment.

This is the event analysis flow diagram:

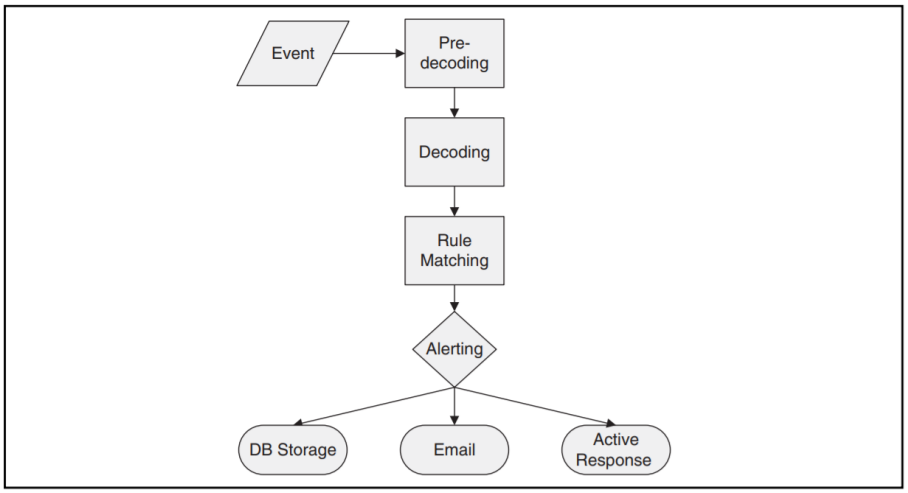

Figure 2 Wazuh event analysis diagram

When an event is received, Wazuh tries to decode and extract any relevant information from it. The event is decoded in two parts, pre-decoding extracts static information from well-known fields of an event such as time, dates, hostnames, program names, and specific log messages… Decoding extracts non-static information from events such as IP address information, usernames, URLs, and port information… After pre-decoding and decoding, the rule-matching engine is called to test if the received event matches a defined rule or group of rules, in which case the event is escalated to an alert or part of a composite alert (with multiple other events). Wazuh supports both atomic and composite rules, atomic rules are based on single events without any correlation between events, while composite rules are based on multiple events that need to occur in order for that rule to be matched.

Some information is not available in log files but it still needs to be monitored, to solve this, Wazuh can monitor the output of certain commands and treat it as if it was an entry in a log file by decoding, analyzing, then matching it against defined rules, and generating alerts or taking action when the output changes. For example, Wazuh can run the *netstat* command regularly to detect whenever there's a new port open in listening mode on the host.

3.1.1.2 File integrity Monitoring and Rootkit Detection

File integrity checking is an essential part of all HIDS technologies and is typically performed by comparing the previously calculated cryptographic checksum of a known good file against a recently calculated checksum to determine if the file was tampered with. Wazuh looks for changes in the MD5, SHA-1, or SHA-256 checksums of sensitive files on the system and registry keys within the Windows registry. The Wazuh agent scans the file system and registry of a host every few hours or at an interval specified by the administrator, and it sends the checksums of monitored files (usually sensitive files) to the Wazuh server. The server looks for modifications by comparing the newly received checksums against the stored historical checksums of those files and registry keys. Like log monitoring, the integrity monitoring engine uses rule-based detection to analyze events involving monitored directories, files (and their parameters), and registry keys in order to decide whether an alert should be generated and how severe the event is.

The integrity monitoring component in Wazuh is called syscheck. By default, syscheck monitors sensitive system directories such as /etc, /usr/bin, /usr/sbin on Linux and C:\Widnows\System32 on Windows. Syscheck is very flexible, it allows the administrator to add files, directories, registry keys, and network mounted file systems to be monitored, specify which file parameters must be monitored (checksum, file size, ownership, permission changes, or all available parameters…), specify when the checks should take place (date, time, frequency), specify which files, directories and registry keys must be ignored, increase the severity of alerts for important files… Files that change often such as log files are ignored automatically by syscheck. The administrator must tune syscheck's configuration to fit his needs and the system that it will be monitoring in order to get better results and to avoid false positives and unnecessary alerts.

Wazuh rootcheck process performs rootkit detection and policy monitoring/enforcement. On Linux, UNIX, and BSD, rootcheck finds user mode rootkits through signature definitions and kernel mode rootkits using anomaly-based detection techniques like system call comparisons. Anomaly-based checks are also performed to ensure that the system is operating as anticipated.

Here are some of the actions performed by rootcheck to detect rootkits:

- Using a database of known rootkits and files commonly used by them, it monitor those files using system calls like stats(), fopen(), and opendir() on Unix to detect if these files were hidden by a kernel mode rootkit. The database needs to be updated constantly, the chances of false positives are small, but false negatives can occur easily when dealing with new rootkits.
- Scans the /dev directory on Unix for anomalies, this directory is heavily used by rootkits to hide their files.
- Scans the whole file system looking for unusual files and permission problems, for example, files owned by root with write permission enabled to others are especially dangerous. Also, it looks for SUID executables which execute as root and are proved to be a significant security problem.
- Scans for hidden processes using system calls like getsid() and kill() and the *ps* utility on Unix which can reveal the presence of a kernel mode rootkit that's hiding processes by hooking system calls.

- Scans for the presence of hidden ports or open ports having their state masqueraded by a rootkit, using system calls like bind() and the netstat utility on Unix.

Wazuh also allows an administrator to define rules and to use rule-based detection with rootcheck in order to detect specific events that match defined rules, the process of event analysis and rule matching follows the one used by log analysis and integrity monitoring.

Policy monitoring/enforcement is the process of verifying that all systems conform to a set of pre-defined policies surrounding configuration, settings, and approved application usage. Rootcheck is also responsible of monitoring and enforcing policy, it can check if a process is running or not, if a file is present, if the content of a file contains a pattern, if a Windows registry key contains a certain value… Rule-based detection is also used here; the administrator can define a set of rules or use a set of predefined rules that reflect the policies that must be followed. Wazuh generates an alert when a policy violation is detected. For instance, Wazuh can detect applications that violate the organizational acceptance use policy such as detecting Skype running on a host while the policy explicitly forbids it.

3.1.1.3 Vulnerability Detection

To be able to detect vulnerabilities, the Wazuh agent pulls software inventory data from the monitored host and sends it to the server where it's stored in local SQLite databases (one per agent). Also, the server builds a global vulnerability database from publicly available CVE (common vulnerabilities and exposures) repositories, using it later to cross-correlate its content with the agent's software inventory data.

An alert is generated when a CVE affects a software/package that is known to be installed on one of the monitored hosts. A software/package is labeled vulnerable when its version is contained within the affected range of a CVE.

The server can be configured to check periodically for new CVEs in order to ensure that the global database remains up to date.

3.1.1.4 Logging Incidents and Alerting

Wazuh provides powerful email alerting capabilities with very granular control of the alert types generated. Every generated alert has a severity level from 0 to 15, with 15 being the highest, by default Wazuh logs every alert with severity level of 1 to 15 and displays it on the manager's dashboard in real-time. In addition, it generates email messages for every alert above a 7 severity level. The handling of severity levels and alert generation can be configured by the administrator.

3.1.1.5 Active Response

Automated remediation to security violations and threats is a powerful means for containing damage and reducing risk. Remediation is used to block specific hosts and/or services so that a detected violation can be contained and slowed down or stopped. The violation does not necessarily have to take the form of an attack; it might be a policy violation. Active response techniques vary, but the common theme is that the network must react reflexively to perceived attacks. By slowing down and containing attacks, security personnel can analyze the threat and determine the course of action.

Remedial actions will often be one of the following:

- Firewall block or drop in order to deny access to a host or service.
- Quarantine in order to restrict a host network access to an isolated network segment to remedy the violation.
- Traffic shaping or throttling to deal with denial of service attacks without blocking a service entirely.
- Account lockout in order to revoke access to services from an account that a violation was attributed to.

In Wazuh, active response allows a scripted action to be performed whenever an event matches a rule, has a certain severity, or is considered a violation. Wazuh has a collection of active response scripts that can be used, by default it comes with scripts that can disable logins for accounts on Unix-like systems, adding an IP to a blackhole list, firewall-drop response… Also, Wazuh allows the administrator to write scripts and bind them to commands and then bind these commands to specific rules, whenever a rule is matched, the commands bound to it will be executed which in turn will provide arguments to the attached response scripts and will execute them, thus creating an active response.

3.1.1.6 Containers Security

A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. It wraps up an application in such a way that its deployment and runtime issues —how to expose it on a network, how to manage its use of storage, memory, and I/O, how to control access permissions— are all handled outside of the application itself. Containers eliminate the problems of deploying software on "bare metal" or in a VM. Containers work a bit like VMs, but in a far more specific and granular way. They isolate a single application and its dependencies from the underlying OS and from other containers. Containers enable more efficient use of system resources, faster software delivery cycles (easier to update and to roll back if needed), better application portability since the container encapsulates the application and everything it needs so it can be shuttled easily between different environments.

Container technology like Docker is becoming the new normal to deploy software applications. Logically, container security is now more vital than ever.

Wazuh supports monitoring for signs of security incidents across containers and alerting in real-time. It protects container workloads at two different levels:

- Infrastructure level: integration with Docker engine and Kubernates API in order to analyze events and detect potential security issues, such as installation of new applications in the Docker host, creation of new Docker containers, a user-issued command or shell running inside a container, changes in Kubernates configuration...
- Container level: the Wazuh agent can be deployed into a container in order to perform file integrity checks, log analysis, process monitoring, and vulnerability detection within the container itself

3.2 Samhain

Samhain is an open source host-based intrusion detection system developed by Samhain labs. It provides file integrity checking, log file monitoring and analysis, rootkit detection, port monitoring, Windows registry checking, detection of rogue SUID executables, detection of hidden processes… Samhain runs on Mac OS, Linux, Windows (Cygwin), and Solaris. It can be deployed following the centralized client/server implementation thus allowing centralized logging and maintenance, or the standalone single-host implementation [11].

A Samhain client/server system consists of the following components:

- The Samhain agent installed on the host, it's designed to run as a daemon (background process), it monitors the host and performs file and host integrity checks.
- The Yule server which collects and logs reports from deployed agents, it allows agents to download baseline databases and configuration files, and it keeps track of the status of all deployed agents, so it's basically used for agent management.
- A relational database used to store reports sent from the agents to the server, Oracle, MySQL, and PostgreSQL are supported.
- The Beltane web-based console used to pull reports from the database and present them for review, it also allows the administrator to update the agent's baseline database and configuration stored on the server. Basically, it's used to monitor agents and get results.
- A deployment system used to facilitate the deployment of the Samhain agent on multiple hosts.

3.2.1 Samhain Features

Here are some of Samhain's most useful features.

3.2.1.1 File Integrity Monitoring

Just like Wazuh, Samhain monitors files and file metadata in order to detect any unauthorized access or tampering.

On Linux, Samhain uses the i-notify mechanism to monitor file system events. This allows it to receive immediate notifications about changes to files thus eliminating the need for frequent file system scans which may cause a high I/O load. This allows checking the file's checksum (TIGER192, SHA-256, SHA-1, or MD5), size, mode/permission, owner, group, timestamp (creation/modification/access), inode, number of file hardlinks, major/minor device number (for Linux device files), linked path (symbolic links only), number of directory hardlinks to determine if a kernel mode rootkit is hiding any subdirectories…. In addition, Samhain can leverage the Linux kernel audit system to determine which user has accessed and modified a file [12].

Samhain also supports file integrity scans, which can be performed at user-defined intervals, it's also possible to configure two different file integrity checking schedules to check some files and directories more frequently than others. The checks can also be started anytime by sending a signal from the server to the deployed agents.

3.2.1.2 Windows Registry Monitoring

On windows, it's possible to check the integrity of individual keys or complete trees/hierarchies of keys) in the Windows registry. The checking is done by storing the baseline data (normal data)

of keys to be monitored in the baseline database, then the deployed agent can scan the registry and check if any monitored key has been modified.

The Windows registry contains a large amount of keys, storing baseline data for all of them is impractical since that would blow up the size of the baseline database significantly, that's why it's better to monitor only sensitive keys and keys that are usually used in attacks.

The configuration of registry scans is flexible, the administrator can specify things like the frequency of scans by specifying the time interval between consecutive scans, single keys or key hierarchy (beginning and ending at specific keys) to be monitored, keys that must be ignored within a hierarchy, the severity for reports on modification to the registry…

3.2.1.3 Log File Monitoring and Analysis

It's not as sophisticated as Wazuh's log monitoring and analysis capabilities, but Samhain supports many log file formats including UNIX Syslog, Apache access and error logs, pact BSD-style process accounting logs (also available on Linux) …

Unlike Wazuh, Samhain doesn't monitor logs in real-time, it relies on a set time interval for log file checking and analysis. A pointer stored persistently on the disk for each monitored log file will keep track of the log entry where the last log file analysis ended so that another analysis can start from there.

In order to detect anomalies in log files, log file entries can be filtered with Perl-style regular expressions (filter rules). Both whitelisting and blacklisting are supported in order to detect if an entry is anomalous or not, by default whitelisting is used: if a log file entry does not match any filtering rule it is reported (known-good entries). Blacklisting can be configured by adding known-bad filtering rules in order to catch known anomalies in log file entries.

Samhain supports checking for correlated events by marking individual entries under a user-defined label and storing them for a user-defined time, then entries marked under two or more different labels can be correlated using regular expressions that matches those labels, this allows us to extract information and draw conclusions based on data from the correlated log file entries.

Samhain can automatically detect and report bursts of similar, repeated events in a monitored log file. Here similar repeated events refer to events that differ only in details that can be expected to change between entries of the same kind such as IP addresses, emails… The event history goes back 12 minutes by default, and thus an alert is triggered if the number of similar events within the last 12 minutes exceeds a given threshold.

3.2.1.4 Checking the file system for SUID/SGID binaries (UNIX)

Samhain can check the entire file system hierarchy for SUID/SGID files at user-defined intervals, and report on any that are not included in the baseline database (which are added upon database initialization).
When a suspicious SUID/SGID binary is located, first Samhain will report the event, then it will strip SUID/SGID permissions from it, quarantine, or delete it entirely.

3.2.1.5 Detecting Open Ports

Samhain can check for open ports and running services on the host, and report ports that are open but not listed in the configuration database. By default, all TCP and UDP ports are checked.

3.2.1.6 Detecting Hidden/Fake/Missing Processes

Samhain can check for processes that are hidden (i.e. running processes that are not listed in the output of the "*ps*" utility on UNIX), fake (i.e. listed by "*ps*" although they don't exist), or missing (i.e. user-defined processes that must be running but are actually not). The entire range of possible PIDs is checked to create a list of running processes and then comparing it to the output of the "*ps*" utility.

3.2.1.7 Logging Facilities

Samhain's central log server (Yule server) is used to store messages (reports) sent by clients (agents). These messages are encrypted by the agent using AES and sent over TCP to the server. Before sending any messages, the agent must be authenticated and provided with a session key. The Yule server keeps track of all connected agents and their session keys. Also, all messages are signed by the agent. On receipt, Yule will check the signature, if it's verified it will log the message and the agent's hostname to the console and to the log file, then the log file entry is signed to prevent unauthorized modifications of existing log records.

Yule can be configured to send any received reports by email in order to notify security personnel of an incident. Also, it can be configured to log those reports to a database for long-term storage.

3.2.1.8 Integrity of the Samhain System

Since the baseline database and the configuration file are downloaded on each host, maintaining their integrity is crucial. That's why Samhain signs the database and configuration file using GnuPG in order to detect any tampering.

The agent runs on the monitored host as a daemon, any stop/restart of the process will leave a recognizable mark, it's not possible to "slip in" a rogue executable in order to replace the agent's executable as long as the daemon is running.

If an intruder doesn't know that Samhain is running, they will not make any attempts to subvert it. That's why the agent can be configured to run in "stealth" mode, meaning that the agent can run without any obvious trace of its presence on the disk. Also, the baseline database can be appended to some binary file (i.e. executable, image…) and its strings obfuscated, and the configuration file can be hidden in an image using steganography.

3.3 A Quick Observation

We noticed that Wazuh is way more flexible, easier to deploy, and its features are better documented than Samhain. When it comes to configuration and functionality, Wazuh gives the user more control over detection options and configurations.

Both of them have a lot of features in common, but Wazuh has some distinctive and very useful features that Samhain lacks, such as vulnerability detection, security configuration assessment, policy monitoring, active response…

Another major difference between Wazuh and Samhain is that in Wazuh most analysis and checks are performed on the server, the agent's job is to monitor the host and forward data to the server. In Samhain on the other hand, analysis and checks are performed on the host itself, the agent downloads the baseline database which contains baseline data (such as signatures) to be used in checks/scans, and it's the agent that must perform those checks. So the server is only used for centralized logging and agent management. It's clear that Samhain puts a bigger load on the host than Wazuh does, which can affect the host's performance in some instances.

4 Testing Wazuh HIDS

4.1 Testing Environment

The testing environment consists of the Wazuh server (Linux), the monitored host (Ubuntu 20.04 LTS) with the Wazuh agent deployed on it, and a third host used as the attacker's machine, all deployed on the same network.

4.2 CIS Benchmark Checks

After installing the Wazuh agent on a host, registering it with the server, and restarting it, the first thing it does is run CIS benchmark checks.

CIS (Center for Internet Security) is an entity dedicated to safeguard private and public organizations against cyber threats. This entity provides CIS benchmark guidelines, which are a recognized global standard and best practices for securing IT systems and data against cyber-attacks. There are more than 80 CIS benchmarks that cover nearly all OSs.

The agent performs the CIS benchmark checks specific to the host's OS and reports the result of each check back to the admin's dashboard. For Debian/Linux, the CIS benchmark checks include things like: checking for accounts other than root that have a UID of 0 (to make sure that root is the only account with superuser privileges), checking */etc/shadow* and */etc/passwd* to ensure that their permissions are configured correctly, checking that the *sudo* log file exists in order to track all *sudo* incidents…

Wazuh supports scheduling CIS benchmark checks to run at specific times. Also, specific checks will run after the installation of a new package or program, for instance, SSH related checks will run automatically after the installation of openSSH server (SSHD) on a monitored host.

4.3 Detecting Basic System Events

Wazuh can detect in real-time logged system events that match one of its defined rules. For example, it can detect events like the creation of a new user, the successful execution of the *sudo* command, opening/closing a user session…

4.3.1 Detecting the creation of a user

After issuing the "*useradd*" command on the monitored host, a bunch of alerts are generated on the Wazuh dashboard, each with its level of severity and the rule that was matched by the event for this alert to be generated:

Tactic(s)	Description	Level
	PAM: Login session closed.	3
Credential Access, Persistence	Information from the user was changed.	8
	PAM: User changed password.	3
Persistence	New user added to the system.	8
	New group added to the system.	8

Figure 3 User creation alerts

The alert can be expanded to display further details associated with this event, such as the log file that the event was logged to, and the log entry appended to that file. In this case, we can see the username of the newly created user, his UID and GID, his home directory, and the exact time of the event…

rule.id	5902
rule.gpg13	4.13
decoder.parent	useradd
decoder.name	useradd
full_log	Apr 24 13:05:07 Ubuntu64 useradd[3490]: new user: name=testuser, UID=1002, GID=1002, home=/home/testuser, shell=/bin/bash
location	/var/log/auth.log

Figure 4 User creation alert details

4.4 Detecting Malware

4.4.1 Detecting a Malicious Binary

Wazuh uses File integrity monitoring and VirusTotal integration to detect in real time malicious binaries dropped on the disk of the monitored host.

The Wazuh agent can be configured to monitor specific directories in real time and generate FIM alerts instantly when a file in a monitored directory gets added, deleted, or modified.

VirusTotal aggregates many antivirus products and online scan engines, offering an API that can be queried by using either URLs, IPs, domains, or file hashes.

After an FIM alert is triggered, Wazuh can automatically perform a request to the VT API with the hashes of the files added or altered in any monitored directory in order to scan and determine whether any of those files is detected by VT as malicious. If VT's response is positive, Wazuh will generate an alert about the detected file.

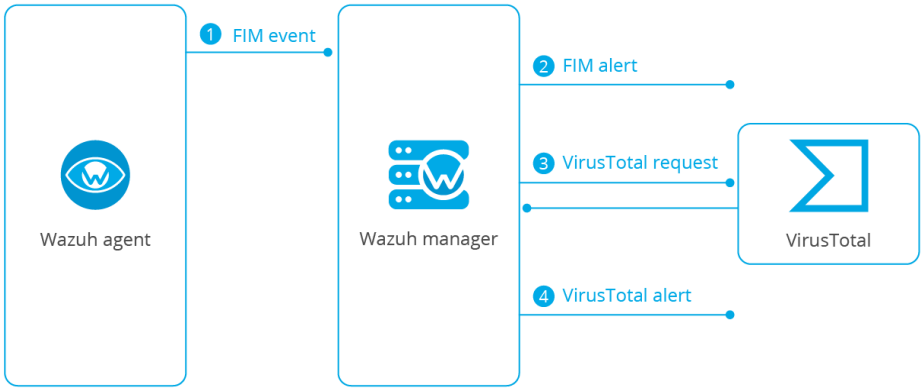

Figure 5 Malicious binary detection diagram

To test this feature, we used an Emotet malware sample. Emotet was one of the most professional, well-maintained, and long lasting cybercrime service ever created, so much so that law enforcement and judicial authorities worldwide launched in February 2021 a massive operation to disrupt the Emotet botnet and its infrastructure. Emotet's main goal was to establish a foothold on computer systems and networks on a global scale, once these accesses were established, they were sold to top-level groups to conduct many activities including data theft and extortion. Emotet was mostly distributed through automated email campaigns which relied on a variety of social engineering tricks and lures to fool unsuspecting users into opening malicious attachments.

We configured Wazuh to monitor a directory and use VirusTotal to check any newly created file.

As soon as the Emotet malware sample was downloaded into the monitored directory from an email as part of the initial attack vector, wazuh generated 2 alerts:

Tactic(s)	Description	Level
Execution	VirusTotal: Alert - /home/rijndael/monitored/emotet_sample.dll - 57 engines detected this file	12
	File added to the system.	5

Figure 6 Malicious binary alerts

The first is the FIM alert about a file being added to the monitored directory, and the second is the VT alert reporting that 57 engines detected the downloaded file as malicious.

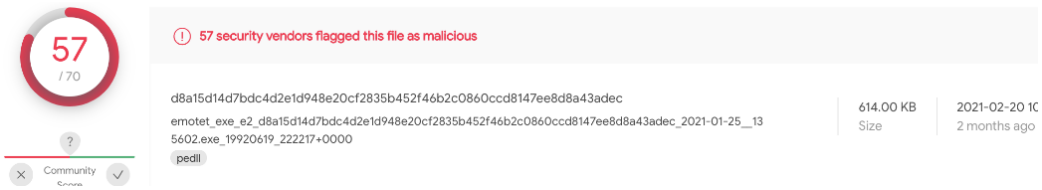

Figure 7 Malicious binary scan results

4.4.2 Detecting a Rootkit Hiding Processes

To test Wazuh's rootkit detection capabilities, we used a modified version of an LKM rootkit, the rootkit has been modified to hide a selected process on demand by intercepting a specific signal (*SIGSYS*) sent to that process and then hiding it from "*ps*" and other utilities that list running processes by manipulating the system call table and the /proc directory. The rootkit also hides itself from the kernel module list.

We modified Wazuh's rootcheck module, responsible of detecting rootkits and policy violations, to run every 300 seconds, then the rootkit kernel module was installed on the host.

The process that will be hidden by the rootkit is the *rsyslogd* process present on all Linux systems.

```
rijndael@Ubuntu64:~$ ps auxw | grep rsyslogd | grep -v grep
syslog     506  0.0  0.1 263032  4188 ?        Ssl  11:42   0:00 /usr/sbin/rsyslogd -n
rijndael@Ubuntu64:~$ kill -31 $(pidof rsyslogd)
rijndael@Ubuntu64:~$ ps auxw | grep rsyslog | grep -v grep
rijndael@Ubuntu64:~$
```

Figure 8 rsyslogd process hidden by rootkit

Here we can see that the process is running with PID 506. The kill command is used to send signal 31 *SIGSYS* to the *rsyslogd* process, which will be intercepted by the rootkit, and this process will be hidden.

When rootcheck runs, the rootcheck kernel module iterates over the entire range of PIDs and uses the *getsid()*, *getpgid()* and *kill()* system calls to check for hidden processes (discussed earlier).

On the dashboard, an alert is generated, signaling that rootcheck has detected something using anomaly based detection.

Description	Level	Rule ID
Host-based anomaly detection event (rootcheck).	7	510

Figure 9 Hidden process alert

The details of the generated alert include the PID of the *rsyslogd* process that was hidden by the rootkit.

rule.id	510
rule.gdpr	IV_35.7.d
decoder.name	rootcheck
full_log	Process '506' hidden from /proc. Possible kernel level rootkit.

Figure 10 Hidden process alert details

4.4.3 Detecting a Ransomware Attack in Real-time

Ransomware, in simple words, is a malicious executable that mainly iterates over files in the file system and encrypts them, thus denying the user access to their data, then it asks the user to pay a ransom in order to get the decryption key to recover their data. Ransomware became a very prominent and effective threat in the last few years, especially with the rise of ransomware as a service (RaaS). RaaS allows malware developers to write and sell or lease ransomware mostly to unexperienced attackers which will distribute them and manage ransomware campaigns with the developers behind the ransomware receiving a cut of each ransom victim's pay for the decryption key. This affiliate scheme is very lucrative and very effective, well organized groups are earning millions per year (i.e. REvil).

Ransomware spreads typically through email phishing and spam campaigns. Besides, some ransomware families (like WannaCry and notPetya) take advantage of exploits (like EternalBlue) to infect other systems in the enterprise network, meaning that they do not require user interaction to propagate.

Ransomware can cause significant damage to a corporate's network, detecting a ransomware attack in real-time can be very helpful.

Wazuh, using its file integrity monitoring module can detect ransomware attacks in real-time. During an attack, a ransomware performs the following actions; read the file content, encrypt the content and write it into a new file, and then remove the original file. Wazuh's file integrity monitoring module is able to monitor addition, changes, and deletion of files in monitored directories, thus an unlikely high number of file creation and deletion alerts might indicate that a ransomware is active on the host.

To test Wazuh's ransomware detection capabilities we used a ransomware that encrypts every file in a specific directory and its subdirectories. We instructed Wazuh to monitor this directory and its content in real-time. Also, we created two monitors: "fim-massive-add" which monitors the "file added" event and fires a trigger when the number of occurrences of this event exceeds a specified threshold (100 in our case) within a specific time interval (1 minute in our case), and "fim-massive-delete" which monitors "file deleted" event and behaves like the "fim-massive-add" monitor.

As soon as the ransomware attack starts, Wazuh detects and reports in real-time a bunch of "file added" and "file deleted" events.

File added to the system.	5	554
File deleted.	7	553
File added to the system.	5	554
File deleted.	7	553
File added to the system.	5	554
File deleted.	7	553

Figure 11 File add/delete alerts

After exceeding the specified threshold for a monitored event, the monitor's state changes to "active" and the associated trigger is fired.

Monitor name ↓	Last updated by	Latest alert	State	Last notification time	Active
fim-massive-delete	admin	fim-massive-delete-trigger	Enabled	04/26/21 6:21 pm	1
fim-massive-add	admin	fim-massive-add-trigger	Enabled	04/26/21 6:21 pm	1

Figure 12 File add & delete monitors

Each fired trigger generates an alert on the dashboard, seeing these two alerts generated at the same time is an indication that a ransomware is active on the host.

Alert start time ↓	Alert end time	Monitor name	Trigger name	Severity	State
04/26/21 6:21 pm	-	fim-massive-delete	fim-massive-delete-trigger	5	Active
04/26/21 6:20 pm	-	fim-massive-add	fim-massive-add-trigger	5	Active

Figure 13 Alerts generated by file add & delete triggers

4.5 Detecting a Simple SSH Brute Force Attack

SSH brute force has been around for many years, it's used to get remote shell access to a target host. Nowadays, it's mostly used to attack IoT devices. Attackers often employ malware and other tools to automate the process of brute force attacks by distributing the attack across a variety of source locations (botnets). Usually attackers target a range of IP addresses using common SSH credentials in hopes of gaining access to some of them.

OpenSSH server is running and listening on port 22 on the monitored host. Using a simple SSH brute force script on our attacker's machine we started brute forcing the SSH credentials.

Wazuh directly reports the failed login attempts, after reaching a specific threshold it generates an alert about the SSH brute force attack in progress.

T1110	Credential Access	sshd: brute force trying to get access to the system.	10
T1110	Credential Access	sshd: Attempt to login using a non-existent user	5
T1110	Credential Access	sshd: Attempt to login using a non-existent user	5

Figure 14 SSH brute force alerts

The details of the SSH brute force alert can be expanded to get more information such as the username used and the source IP address.

rule.id	5712
decoder.parent	sshd
decoder.name	sshd
full_log	Apr 24 13:50:41 Ubuntu64 sshd[3490]: Failed password for invalid user bob from 192.168.56.1 port 54245 ssh2
location	/var/log/auth.log

Figure 15 SSH brute force alert details

4.6 Tracking Down Vulnerable Packages and Applications

Wazuh's vulnerability detector module can detect if software applications and packages installed on a monitored host have flaws that may affect the host's security.

First the vulnerability detector module downloads and stores all vulnerabilities data (CVEs) starting from a specified date and from specified sources. Then it analyzes the list of software packages installed on the monitored host which is gathered by the Wazuh agent installed on that host and updated regularly to reflect any changes. The analysis correlates between downloaded CVE information and installed software packages: for each installed software package, the detector module looks for CVEs with the same package name, if the package name is affected, it checks if the package version is also reported as vulnerable in the CVE report. When both the software package name and its version are reported as vulnerable, an alert is generated.

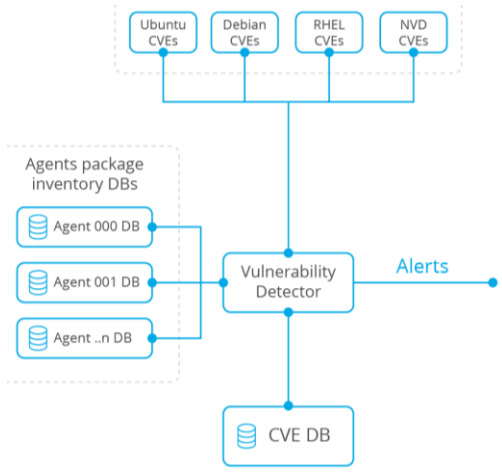

Figure 16 Vulnerability detection diagram

To test this feature, we configured the vulnerability detector module to detect and report vulnerabilities affecting software packages installed on our monitored host. After populating the CVE database, gathering the installed software packages, and analyzing them, Wazuh's vulnerabilities dashboard displayed the top affected packages and the number of vulnerabilities detected in each of them:

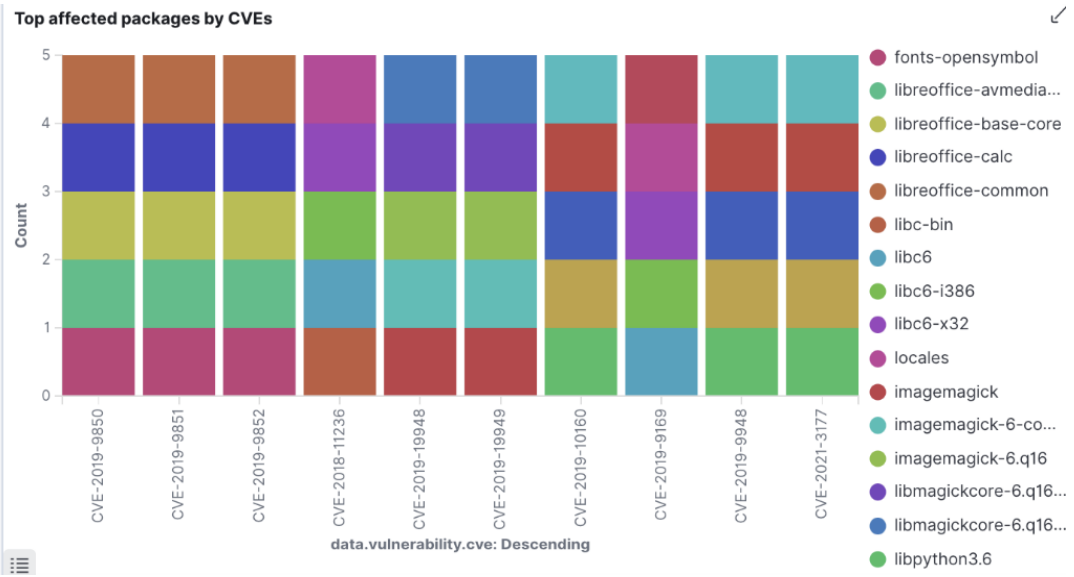

Figure 17 Affected packages/applications shown on the dashboard

Wazuh also generated an alert about each discovered vulnerability specifying the CVE and the vulnerability's severity level.

data.vulnerability.package.name	data.vulnerability.cve	data.vulnerability.severity
libwebkit2gtk-4.0-37	CVE-2020-9850	Critical
libjavascriptcoregtk-4.0-18	CVE-2020-9850	Critical
gir1.2-webkit2-4.0	CVE-2020-9850	Critical
gir1.2-javascriptcoregtk-4.0	CVE-2020-9850	Critical
multiarch-support	CVE-2018-11236	Critical

Figure 18 Detected vulnerabilities alerts

Details including the vulnerable package version, the CVE publishing date, a description about the vulnerability, and the patch release date are also provided for each detected vulnerability. Here are some of the details provided about a detected vulnerability in the installed version of Firefox:

t data.vulnerability.package.name	firefox
t data.vulnerability.package.version	68.0.1+build1-0ubuntu0.18.04.1
data.vulnerability.published	May 26, 2020 @ 03:00:00.000
t data.vulnerability.rationale	Mozilla developers and community members reported memory safety bugs present in Firefox 75. Some of these bugs showed evidence of memory corruption and we presume that with enough effort some of these could have been exploited to run arbitrary code. This vulnerability affects Firefox < 76.
data.vulnerability.references	https://www.mozilla.org/security/advisories/mfsa2020-16/, https://bugzilla.mozilla.org/buglist.cgi?bug_id=1339601%2C1611938%2C1620488%2C1622291%2C1627644, https://security.gentoo.org/glsa/202005-04, https://nvd.nist.gov/vuln/detail/CVE-2020-12396, http://people.canonical.com/~ubuntu-security/cve/2020/CVE-2020-12396.html, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-12396, https://usn.ubuntu.com/usn/usn-4353-1, https://www.mozilla.org/en-US/security/advisories/mfsa2020-16/#CVE-2020-12396
t data.vulnerability.severity	Critical
t data.vulnerability.title	CVE-2020-12396 affects firefox

Figure 19 Detected vulnerability details

4.7 Detecting Newly Opened/Closed Ports

Sometimes we might want to monitor things that are not in the logs. To address this, Wazuh has the ability to monitor output of specific commands and treat it as if it came from a log file. This can be helpful in monitoring things like running processes, disk utilization, ports and listening sockets…

To test Wazuh's command monitoring capabilities, we configured Wazuh to monitor port statuses by defining the following command:

```
<localfile>
  <log_format>full_command</log_format>
  <command>netstat -tulpn | sed 's/\([[:alnum:]]\+\)\ \+[[:digit:]]\+\ \+[[:digit:]]\+\
\+\(.*\):\([[:digit:]]*\)\ \+\([0-9\.\:\*]\+\).\+\ \([[:digit:]]*\/[[:alnum:]\-]*\).*/\1 \2 == \3 == \4 \5/' | sort
-k 4 -g | sed 's/ == \(.*\) ==/:\1/' | sed 1,2d</command>
  <alias>netstat listening ports</alias>
  <frequency>10</frequency>
</localfile>
```

Figure 20 Command to monitor netstat's output

Netstat is a cross-platform tool used to evaluate network activity and configuration such as displaying open ports and the processes listening on those ports. Here we instructed Wazuh to monitor the output of *netstat* by executing the command every 10 seconds and looking for differences in the output.

On the monitored host, we executed a bind shell which opens port 4444, binds a socket to it, and listens for incoming connections. This caused a change in the output of the *netstat* command.

Within 10 seconds, Wazuh's command monitoring module detected the newly opened port and the following alert was generated:

rule.description	rule.level
Listened ports status (netstat) changed (new port opened or closed).	7

Figure 21 Newly opened port alert

4.8 Detecting Web Attacks

To test Wazuh web attacks detection capabilities, we deployed Apache web server along with a vulnerable web application on our monitored host, and we configured Wazuh to monitor Apache logs in real-time.

4.8.1 Detecting Directory/File Fuzzing

Directory brute forcing or fuzzing is part of the recon phase when attacking web applications. It's used to find hidden directories and configuration files which might give attackers more information about the server and the web application, and sometimes widen the attack surface.

On the attacker's machine, we started fuzzing the web application running on the monitored host using a wordlist of commonly used web directory and file names:

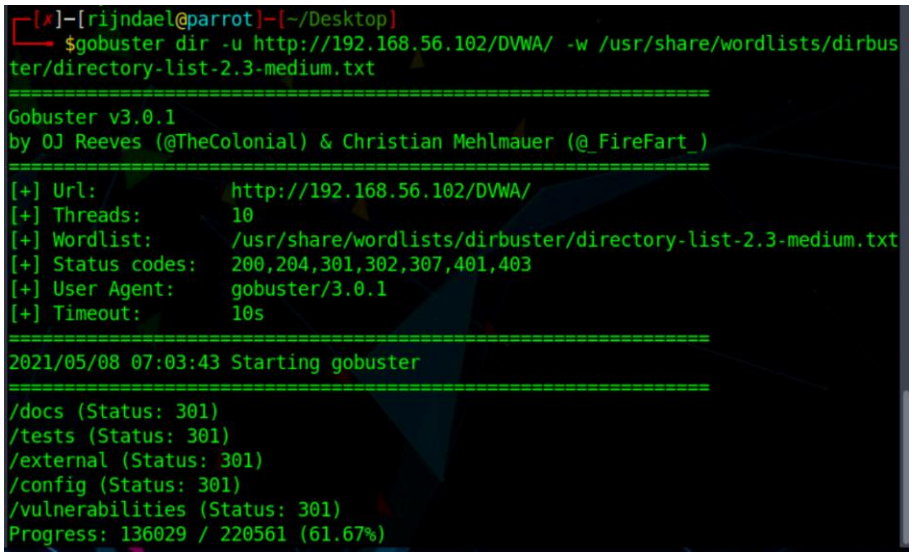

Figure 22 Directory/File brute forcing

As soon as the fuzzing began, Wazuh started genrating an alert for each non-existent URL tried, and after exceeding a specified threshold, Wazuh generated a high-severity alert indicating that directory fuzzing is in action:

Description	Level
Multiple web server 400 error codes from same source ip.	10
Web server 400 error code.	5
Web server 400 error code.	5

Figure 23 Directory/File brute force alerts

4.8.2 Detecting SQL Injection Attacks

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data stored in the database that they are not normally able to retrieve and in many cases, modify or delete this data. This is done by injecting SQL statements in an input entry point such as a form field or URL parameter, in an attempt to get the web application to pass the newly formed rogue SQL query to the database where it will be executed. In some situations, an attacker can escalate an SQLi attack to compromise the underlying database server and other back-end infrastructure.

To test if Wazuh is capable of detecting SQLi attacks, we submitted the following input to a field in the web application:

Figure 24 Basic SQLi test

This is one of the simplest manual tests that an attacker will perform in order to detect the presence of some types of SQL injection.

Wazuh detected this and generated the following alert:

Tactic(s)	Description	Level
Defense Evasion, Privilege Escalation, Initial Access	SQL injection attempt.	6

Figure 25 SQLi detection alert

Wazuh also detected our attempts at dumping data from tables, and generated the following alerts:

Tactic(s)	Description	Level
Initial Access	A web attack returned code 200 (success).	6
Initial Access	A web attack returned code 200 (success).	6

Figure 26 Alerts generated when dumping tables using SQLi

In the details of each alert, Wazuh provided the malicious GET request that was matched to the SQL injection and web attack rules.

4.9 Detecting Metasploit Attacks Using Custom Rules and SCA

The Security Configuration Assessment module (SCA) provides the ability to run out-of-the-box and custom checks that are used for system hardening. An SCA policy is a group of configuration checks that rely on a rule or combination of rules to verify the state of the monitored host. The SCA module can, among other things, run custom commands, inspect configuration files, monitor

running processes and registry keys… in order to ensure that the defined SCA policies are not violated, in case a policy violation is detected, an alert is generated.

Our monitored host has a vulnerable version of Drupal CMS running on top of an Apache web server, and the "*find"* Unix system utility with the SUID bit set, which means that *"find"* will run with root privileges every time it's executed by any user.

To test Wazuh's custom rules and Security Configuration Assessment capabilities we simulated a real attack where an attacker uses metasploit to exploit vulnerabilities in the monitored host and gain root access. The conducted attack started by exploiting Drupal's CVE-2018-7600 also known as "Drupalgeddon 2 forms API property injection" in order to get a meterpreter shell on the system, and then the *"find"* utility was used to escalate privileges to root.

To detect and stop such an attack, we used two methods, a proactive method and a reactive method.

The proactive method relies on SCA policy scans. We defined two SCA policies, one to detect the vulnerable version of Drupal, and another to detect dangerous binaries with the SUID bit set and generate appropriate alerts.

Here's an example of a defined SCA policy, this one was used to detect binaries with the SUID bit set:

```
policy:
  id: "system-files"
  file: "system-files.yml"
  name: "Security checks for system files"
  description: "Scan the file system for SUID binaries"
checks:
  - id: 100002
    title: "Dangerous binaries with SUID bit set found"
    description: "Binaries with SUID bit set can result in a root shell."
    condition: none
    rules:
      - 'c:find /usr/bin -perm -u=s -type f -printf "%y:%p\n" -> !r:arping|bwrap|…|chfn|chrome^"$'
```

Figure 27 Policy to detect SUID binaries

After defining the two policies, the SCA module was able to detect and generate alerts about the vulnerable version of Drupal and the *"find"* utility with the SUID bit set.

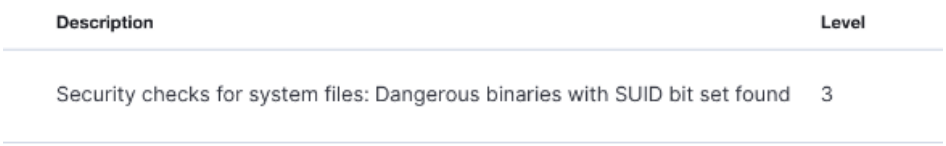

Figure 28 SUID binary detected

	SCA summary: Security checks for Drupal: Score less than 30% (0)	9
	Security checks for Drupal: Drupal Drupalgeddon 2 Forms API Property Injection (CVE-2018-7600)	3

Figure 29 Vulnerable version of Drupal detected

Using SCA allows security personnel to be proactive by fixing the detected issues before the attack takes place.

The reactive method implemented relies on custom rules to detect the execution of the meterpreter shell after the successful exploitation of Drupal's CVE-2018-7600.

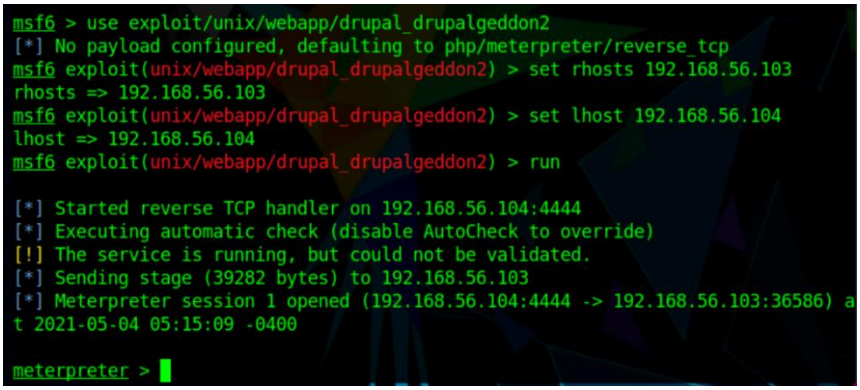

Figure 30 Successful exploitation of CVE-2018-7600

After the successful exploitation and the execution of the meterpreter shell, we noticed that a meterpreter-related process which decodes base64 encoded payloads started running on the targeted host:

Figure 31 Detected process on the victim's host

The presence of this process is a good indication that a meterpreter shell is running. We relied on this observation to craft a custom command which monitors running processes:

```
<wodle name="command">
  <disabled>no</disabled>
  <tag>ps-list</tag>
  <command>ps -eo user,pid,cmd</command>
  <interval>10s</interval>
  <ignore_output>no</ignore_output>
  <run_on_start>yes</run_on_start>
  <timeout>5</timeout>
</wodle>
```

Figure 32 Command to monitor running processes

We also created a custom group of rules that inspect the output of the command above and generate an alert if any running process' command matches the string *"eval(base64_decode"*:

```
<group name="wazuh,">
  <rule id="100001" level="0">
    <location>command_ps-list</location>
    <description>List of running process.</description>
    <group>process_monitor,</group>
  </rule>
  <rule id="100002" level="10">
    <if_sid>100001</if_sid>
    <match>eval(base64_decode</match>
    <description>Reverse shell detected.</description>
    <group>process_monitor,attacks</group>
  </rule>
</group>
```

Figure 33 Rule group to detect the meterpreter-related process

Then we configured Wazuh to use the newly defined custom group of rules and associated command.

After exploiting CVE-2018-7600 again, Wazuh detected that a meterpreter shell is running on the host and generated an alert:

Description	Level
Reverse shell detected.	10

Figure 34 Meterpreter detection alert

4.10 A Quick Observation

Wazuh is very well documented, easy to install and configure, and its out-of-the-box features and defined detection ruleset detect mostly all common attacks.

When it comes to web attacks detection, we noticed that Wazuh's defined web attacks ruleset is limited, it doesn't support the detection of a lot of attacks such as XSS and command injection. While the user can write custom rules to detect those attacks, we think that an attack like XSS is way too common not to be included in the out-of-the-box detection ruleset.

We must note that Wazuh has a gradual learning curve, anyone with moderate security knowledge can deploy it and get it running without any problems, but configuring its advanced features especially when it comes to writing custom rules, commands, and policies might take some time to master even for someone who's experienced, since that would require a good understanding of both Wazuh and the monitored operating system(s).

5 Most Popular HIDSs – A Quick Comparison

Table 1 Most Popular HIDSs and Their Features

SolarWinds Security Event Manager 	• Commercial product.
	• Cross-platform.
	• Anomaly-based and Signature-based detection.
	• Features:
	  ○ Integrated audit reporting to verify compliance with a wide range of data security and integration standards (PCI DSS, SOX, HIPAA…).
	  ○ File integrity monitoring.
	  ○ Automated Log monitoring and protection: real-time centralized log file monitoring and analysis to detect potential APT activity, encryption of log files in transit and storage.
	  ○ Analysis tool for manual log inspection.
	  ○ Real-time automated alerting.
	  ○ Forensic analysis.
	  ○ USB monitoring
	  ○ Automated threat remediation: blocking IPs, modifying privileges, blocking USBs and hardware devices, killing processes…

OSSEC 	• Open source product.
	• Cross-platform.
	• Anomaly-based and signature-based detection.
	• Features:
	  ○ Log file monitoring and analysis.
	  ○ File integrity monitoring.
	  ○ Compliance auditing.
	  ○ Windows registry monitoring.
	  ○ Rootkit detection.
	  ○ Real-time alerting.
	  ○ Predefined threat intelligence rule sets provided by the community.
	  ○ Active response.

SolarWinds Papertrail
 	
papertrail
	• Commercial product.
	• Cross-platform.
	• Anomaly-based and signature-based detection.
	• Features:
	  ○ Cloud-based centralized log management service.
	  ○ Easy access and rapid search functionalities for the stored log archives.
	  ○ Real-time log monitoring and analysis.
	  ○ Log files encryption in transit and in storage.
	  ○ The ability to add and modify your own detection rules and policies.

Wazuh
 	
WAZUH
	• Open source product.
	• Cross-platform.
	• Anomaly-based and signature-based detection
	• It's an upgraded fork of OSSEC.
	• Features:
	  ○ Log monitoring and analysis.
	  ○ File integrity monitoring.
	  ○ Command monitoring
	  ○ CVE database to detect known vulnerabilities.
	  ○ Windows registry monitoring.
	  ○ Rootkit and malware detection.
	  ○ Cloud and container security.
	  ○ Regulatory compliance (SOX, HIPAA, PCI DSS…).
	  ○ Configuration assessment.
	  ○ Alerting and active response.

Samhain 	• Open source product.
	• UNIX & Windows (Cygwin)
	• Anomaly-based and Signature-based detection.
	• Features:
	  ○ File integrity monitoring.
	  ○ Log file monitoring and analysis.
	  ○ Centralized encrypted logging repository.
	  ○ Rootkit detection.
	  ○ Rogue SUID executables detection.
	  ○ Port monitoring.
	  ○ Process monitoring.
	  ○ Windows registry checks.
	  ○ PCI DSS compliance checks.

6 Conclusion

Intrusion detection systems are a vital component of any enterprise network. A combination of both HIDS and NIDS is necessary to provide the maximum security to a network.

A host-based intrusion detection system monitors hosts for intrusion by collecting host-related data in real-time, analyzing it, and then using signature-based and/or anomaly-based detection to conclude whether a detected event is an intrusion or not.

Wazuh is an open-source HIDS supported by a great community. It provides basic and advanced features for host monitoring and intrusion detection, and it's easy to install and configure. Wazuh demonstrated that it's capable of detecting a lot of common attacks that were and still are being used heavily today.

HIDS research today is mostly focused on anomaly-based detection and machine learning in hopes of detecting novel and sophisticated attacks which are becoming more and more prevalent.

7 Table of Figures

8 Bibliography

[1] H. Debar, "An Introduction to Intrusion-Detection Systems," 2009.

[2] R. Kumar, "Signature-Anomaly based Intrusion Detection Algorithm," 2018.

[3] V. Bukac, P. Tucek and M. Deutsch, "Advances and Challenges in Standalone Host-based Intrusion Detection Systems," 2012.

[4] L. Ying and Y. Zhang, "The Design and Implementation of Host-based Intrusion Detection System," 2010.

[5] R. Bridges, M. Vincent and Q. Chen, "A Survey of Intrusion Detection Systems Leveraging Host Data," 2019.

[6] S. Vandeven, "Rootkit Detection with OSSEC (SANS Institute)," 2014.

[7] M. Grimmer, M. Röhling, M. Kricke and B. Franczyk, "Intrusion Detection on System Call Graphs," 2018.

[8] F. Apap, A. Honig and S. Hershkop, "Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses," 2012.

[9] "Wazuh documentation," [Online]. Available: https://documentation.wazuh.com/current/.

[10] S. Tirumala and H. Sathu, "Free and Open source intrusion detection systems: A study," 2015.

[11] B. Wotring, Host integrity monitoring using Osiris & Samhain, Syngress.

[12] "Samhain documentation," [Online]. Available: https://www.la-samhna.de/samhain/s_documentation.html.