Ali Anafcheh

Intrusion Detection with OSSEC

Bachelor's thesis
Information Technology

2018

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Ali Anafcheh	Bachelor of Engineering	May 2018
Thesis title Intrusion Detection with OSSEC		40 pages
Commissioned by		
Supervisor Reijo Vuohelainen		

Abstract

The purpose of this thesis was to study the way of intrusion detection with OSSEC. The first chapter was the theoretical part where my understanding of OSSEC and its components was introduced. The chapter was divided to multiple sections explaining OSSEC's fork Wazuh and how it can be used with Elastic Stack to enhance monitoring and add features to OSSEC. The second chapter started by setting up testing machines using Google Cloud and an Infrastructure as a Code tool called Terraform. Next, Wazuh installation was done automatically using Ansible as a configuration management tool. In the final section of Chapter 2, Wazuh's important features were evaluated on two virtual machines.

The motivation to write this thesis was derived from being in a position to monitor many servers for any security issues. Therefore host-based intrusion detection was the best choice to comply to security policies specifically. This study is useful for companies interested in monitoring every single activity on a host and taking actions accordingly.

Keywords

OSSEC, Wazuh, intrusion detection, security

CONTENTS

1 INTRODUCTION

Nowadays, most companies are using SaaS services where the infrastructure and the end point software are already provided. This helps businesses to focus on their main business without worrying about hardware, software, or security maintenance. In most cases, these smaller companies use a security solution to protect their employees' laptops from malware and viruses. However, that is only a small part of the security world. Big companies that host their own servers physically or in the cloud need a solution to monitor their servers for various logs, file integrity or any abnormal behavior to identify or prevent intrusions. This is where a host-based intrusion detection and prevention system (HIDPS) or generally an intrusion detection and prevention system (IDPS) is needed.

HIDPS is normally a package installed on a computer which analyses different aspects of the operating system and reports any abnormalities to the HIDPS administrator. The most distinctive feature of an HIDPS is that it can be configured to detect behaviors that the administrator thinks are abnormal. So, it's a customizable system that can be configured according to the security policies of a company. This is what makes it a must-have tool compared to regular home security solutions or antivirus software.

In this thesis, I will go through OSSEC(Open Source HIDS SEcurity) which is one of the most popular tools in this area and mainly its fork Wazuh which extends OSSEC's main functionality with added features and integration with Elastic Stack for visualization and better data processing and management. I will also study and test the main features of OSSEC which according to Wazuh's official website (2018) are the following:

- File integrity monitoring
- Intrusion and anomaly detection
- Automated log analysis
- Policy and compliance monitoring

The first part of the thesis will be the theoretical part in which I explain OSSEC, Wazuh and Elastic Stack and how all of these tools come together to create a powerful security system with real time monitoring and alerts. In the second part, I will implement a practical solution where I use all of these tools to monitor a few test machines for any suspicious activities. I will also briefly explain the tools that I will use to automate this whole process to make it easily re-usable in the future using configuration management tools like Ansible and Terraform.

2 INTRUSION DETECTION SYSTEM (IDS)

This chapter explains the tools needed to have a useful and convenient IDS. The description of the tools introduced is based on the official documentation of OSSEC (2018), Wazuh (2018) and Elastic Stack (2018).

2.1 OSSEC

OSSEC is an open source HIDS developed by Daniel B. Cid whom sold the project to Trend Micro in 2008 (Daniel Cid, 2018) but the project continued to be free and open source. Its current stable release is 2.9.3. It consists of many services and modules that each provide their own unique features in terms of intrusion detection. HIDS has many aspects and OSSEC mixes them all which in return provides some fundamental benefits as explained in the following subsections.

2.1.1 Benefits

OSSEC fulfills security compliance requirements. Many customers, mostly business customers require the business they work with to have certain security compliances such as Payment Card Industry Data Security Standard (PCI DSS), Health Insurance Portability and Accountability (HIPAA) etc. Monitoring logs and analyzing them for suspicious activities is one of the ways how OSSEC allows businesses to comply by different security requirements.

It is multi-platform and flexible. OSSEC is a multi-platform solution that enables companies to collect and inspect logs on and from different operating systems and devices, which is a vital feature compared to its competitors. By using custom rule and log decoders collecting almost every type of log is possible. The OSSEC agent which does the job of collecting information and sending them to the OSSEC manager is supported on GNU/Linux, Windows, MacOS, FreeBSD etc. It is also able to collect information via SSH or Syslog which extends its support to almost all network devices as well. Many companies might have certain policies or requirements that would prevent them from installing software on their hosts, and this feature helps to comply to such requirements.

OSSEC provides real-time monitoring and alerts. It has a very customizable and configurable system. The administrator can configure which specific issues to be alerted about and through which communications channels. It is easily integrable to most traditional and modern methods such as SMS, email, Slack etc. OSSEC also supports custom rules and scripts to be triggered when a certain suspicious activity is detected. For example, certain IPs can be blocked when a certain rule with a high alert lever is triggered, or emails can be sent with specific information to a defined address. These are called active responses and the possibilities are limitless using custom scripts.

It is integrable with many tools to give it additional functionalities. OSSEC fits in anywhere and is able to send logs to a designated log management system. This is one of the important features that allows it to be integrated with solutions like Elastic Stack which is going to be tested in this thesis.

It can be centrally managed which facilitates administration. OSSEC normally consists of one server and several agents. However, this architecture is scalable in case high availability is required. In any case, the information is all accessible through one management server.

2.1.2 Features

OSSEC is capable of monitoring and analyzing logs. It uses its agents or other agentless methods to collect logs and it will notify the administrator of any abnormal activities such as authentication failure, package installation, privilege escalation or any other specific activity that was configured by the administrator to be detected by OSSEC. For example, let's assume a user does not have privileges to run a specific command. In this case, the user would normally try using "sudo" or an equivalent command to escalate privileges. OSSEC will immediately write this activity to the alert log, if there is already an alert rule configured for this case.

OSSEC does frequent file Integrity checks (Syscheck). The OSSEC agent runs an initial scan on the specified files or directories and sends the checksum information to the OSSEC server. As the server stores the information, the administrator will be alerted in the future in case of any modification in the checksum of these files. This is specifically necessary in a situation where additions or deletion on specific files or directories are not allowed and are important. In most attacks, there will be some modifications on specific files or directories, and the goal of this feature is to detect such modifications.

OSSEC can detect harmful software such as malware. In case of a smart attack, hackers would normally try to hide or fake their activities and delete important logs. Using this feature, OSSEC will alert the administrator if there are any changes that might be suspiciously done by malware. This detection is performed using certain database files containing information regarding different malware and Trojans.

OSSEC can be configured to actively respond to certain activities. Active responses are like automated attack encounters. This feature is perfect for situations when the administrator is not immediately available to counter an attack. So, this is basically a set of pre-configured scripts to be triggered in case

of a corresponding attack to prevent it from getting bigger. This is a very powerful feature that could be used creatively to reduce different kind of attacks and risks.

2.1.3 Architecture

OSSEC can have a single or multiple managers architecture. OSSEC is consisted of many modules and pieces. As Figure 1 shows below, there is a OSSEC manager that collects all the information centrally from OSSEC agents and other devices monitored in other methods. The manager stores everything just to facilitate the administration of the whole system and the agents. The main way the agents communicate with the manager is through the 1514 UDP/TCP port which is used for the main communication and 1515 port which is only used during the registration process to send a registration request to the OSSEC manager. The agent itself is a very small package running on its own isolated environment and has very little effect on the system's performance.

Figure 1. OSSEC single manager architecture (OSSEC, 2018)

OSSEC can be deployed in a cluster mode as well. It means there can be a few managers and many agents. This is a perfect architecture for big companies that require high availability and fail-over to guarantee strong communication among the agents and the managers. So, the agents send information to many managers, and the load is distributed allowing more events to be processed compared to a single manager architecture. For precise load distribution, a load-balancer must be used and its IP address will be used as the manager address for the agents. In addition, no data is lost in fail-over mode. In case any manager goes offline, the requests will be automatically redirected to another manager.

2.1.4 Modules

I wanted to understand how OSSEC works in more detail, I also studied each of its modules and commands and included their brief description in Table 1. Some modules are Linux daemons or services that are on the background and do their job, the rest are tools that can be used as commands in the terminal.

Table 1.OSSEC modules (OSSEC, 2018)

Module	Description
ossec-authd	Daemon that adds agents to the manager
ossec-agentlessd	Daemon that handles agent-less communications
ossec-analysisd	Daemon that creates alerts by analyzing logs
ossec-csyslogd	Daemon that forwards alerts using syslog
ossec-dbd	Daemon that stores alert logs in a configured database
ossec-execd	Daemon that runs the active response scripts
ossec-maild	Daemon that sends email alerts
ossec-monitored	Daemon that monitors agents' connectivity
ossec-remoted	Daemon that handles agents' communication
ossec-reportd	Daemon that creates alert logs
ossec-syscheckd	Daemon that checks files for any changes
agent-auth	Tool used with ossec-authd to add agents to the manager
ossec-control	Tool to control all of OSSEC's services

ossec-logcollector	Tool collect specified logs
ossec-logtest	Tool to test logs to help with troubleshooting
ossec-makelists	Tool to recompile the outdated ones
ossec-regex	Tool to read regex expressions
verify-agent-conf	Tool to verify agents configuration file
clear_stats	Tool to clear the events stats
list_agents	Tool to list available agents connected to the manager
agent_control	Tool to control agents and get their information
manage_agents	Tool to manage authentication keys of the agents
syscheck_control	Tool to manage the integrity checking database
syscheck_update	Tool to update the integrity checking database
rootcheck_control	Tool to manage the auditing database
util.sh	Tool to add files to be monitored by ossec-logcollector

2.2 Wazuh

This section explains how Wazuh extends OSSEC's functionality and makes it much easier to use. Wazuh integrates three features to OSSEC which are explained as follows.

Briefly starting with Elastic Stack, it is a combination of tools, mainly Elastic Search, Logstash and Kibana. That's where the name ELK Stack comes from. This stack works together to show the received data from tools like OSSEC to be viewed in a user friendly way. Wazuh created their own plugin for Kibana which is a data visualization tool running in a browser. The plugin allows the user to communicate with the OSSEC manager through a browser and view important statistics about the agents. Some of the most important features of these integration are:

- Sophisticated visualization based on alerts, file changes etc
- A search engine to help find specific information
- Long-term data storage

The second feature is the Wazuh Ruleset. OSSEC itself has its own default rules to detect attacks and suspicious activities. Wazuh extends those rules by adding several rules of their own such as a few rules for Amazon Web Services(AWS), docker, firewall, openvpn etc. On top of that, they also maintain their own version of OSSEC's default rules to increase precision.

In my opinion, the other great feature over OSSEC is that the documentation that Wazuh provides is great and always up to date. It walks you through all the features and possibilities and customization that you can have on OSSEC, and also provides great tutorials on how to implement the whole Wazuh system using configuration management tools like Ansible and Puppet or virtualization tools Docker and Virtual Box.

2.3 Elastic Stack

Elastic Stack and its tools and the way they are related to OSSEC are explained in more details In this section.

Elastic Search is an open-source search and analytics engine which can store big amounts of data and add features like search, filters and other advanced search features. This tool can be used in almost any case where search is needed, like a simple search for a food delivery service, or in our case to search the aggregated logs sent by our Wazuh agents. All data is searchable by Elastic Search by one second latency. The data is indexed then becomes available in that second which makes Elastic Search near real time(NRT).

Elastic Search runs in a cluster which may contain one or more nodes(servers). Each cluster has a unique name as nodes join a cluster based on its name. Nodes may join the wrong cluster if the same name is reused. A node in terms of Elastic Search is the server running the tool itself which also stores and analyzed all the data. All nodes have a unique identifier by default and automatically would

try to join a cluster named ”elasticsearch”. In the most basic case, Elastic Search runs as a single-node cluster.

Each piece of indexed data in Elastic Search is a called a ”document”. You may have a document for a different items like different kind of products which are stored in JSON. Each index can store many number of documents depending on the hardware limits. If an index stores way too much data that takes too much space to be on a single node, that index can can be divided into ”shards”. Each shard is an index on its own as well. Shards give the flexibility of distributing data on multiple nodes which would increase performance. Elastic Search carries out aggregating the shards and forming them to a searchable form to the user. Besides shards, it is also possible to replicate indexes to provide high availability. The number of shards and replicas can be configured before index creation.

Kibana is another tool provided by elastic.co which is used to add a visual interface for all the other tools of the company especially Elasticsearch. It allows the user to view and interact with the data and it also helps in presenting the data in a more readable way using charts, tables, maps etc. As it is a browser application with an appealing interface, the tool facilitates the process of looking at logs and other settings. It runs on port 5601 by default, and it is powered by Node.js. The other tool and the middle letter of ELK is Logstash. Logstash is mainly a data parser which works based on rules to receive, parse, index and send logs to Elasticsearch.

3 PRACTICAL IMPLEMENTATION

In this section, I will implement a working intrusion detection system in Google Cloud using configuration and automation management tools Terraform and Ansible. Once my Wazuh installation is up and ready, I will go through the different features that it provides and analyze the data gathered from each one of the hosts.

3.1 Infrastructure

I need three machines to implement my installation. One of them will be the OSSEC manager which must be a small to average machine with the minimum of 4 GiB of RAM and 2 CPU cores, because it will be running Elastic Stack and Wazuh. Elastic Stack is the tool that will consume much of the resources as it will be doing the data calculations and analysis.

The other two machines will be used as test hosts to be monitored and to detect any intrusion on them. I will describe how I setup this infrastructure using Terraform on Google Cloud based on the official documentation of Terraform (2018) and Google (2018).

3.1.1 Google Cloud

Nowadays, big companies use cloud services for any of their heavy services that would need to be highly available and easily scalable. Running Elastic Stack in the cloud would be a great practice, because services like Google Cloud or AWS offer much flexibility such as easily creating backup images of their virtual machines, adding resources or simply running multiple machines and setting up a load-balancer. All of these services help decrease the need for more human resources and maintenance. Personally, I am an AWS user but I have decided to take the opportunity and use Google Cloud for this study to familiarize myself with a another cloud service provider. Before I describe the type of resources I will use

in Google Cloud, I will integrate it with my current environment first to be able to interact with it through the command line which will make it much easier and simpler to understand instead of using the GUI which could lead to human errors and miss clicks. The other reason for this integration is to be able to write my infrastructure as a code and then use Terraform to lunch my specific infrastructure on Google Cloud. I will describe the advantages of this practice in the section 3.1.2.

First, we need to install Google Cloud Tools which is the command line tool to allow us to programmatically interact with our Google Cloud account. On my personal computer, I am using Arch Linux. The google-cloud-sdk is available in the Arch Linux User Repositories(AUR) which can be installed using tools like ”yaourt”. Once the package is installed, the ”gcloud” command will be available to be used to interact with our account as follows:

```
[alian@aanafcheh ~]$ gcloud --version
Google Cloud SDK 201.0.0
app-engine-go
app-engine-python 1.9.69
beta 2017.09.15
bq 2.0.33
core 2018.05.11
gsutil 4.31
```

Google makes the authentication and integration very easy. We simply run ”google init” and an interactive shell will open up asking us to authenticate in a browser . After authentication, I will create my first project and make it my default in the following way:

```
[alian@aanafcheh ~]$ gcloud projects create ali-anafcheh-oalan002
Create in progress for
[https://cloudresourcemanager.googleapis.com/v1/projects/ali-anafcheh-
oalan002].
Waiting for [operations/cp.8829041521380186492] to finish...done.
```

```
[alian@aanafcheh ~]$ gcloud config set project ali-anafcheh-oalan002
Updated property [core/project].
```

As we created a new project, we need to link our billing account to this new project. The following command must be run to get our billing account ID. In this case and in future cases where personal data is exposed, I randomly replace the data to keep the process as natural and real as possible.

```
[alian@aanafcheh ~]$ gcloud beta billing accounts list
ACCOUNT_ID            NAME                OPEN  MASTER_ACCOUNT_ID
123abc-123abc-123abc  My Billing Account  True
```

Next, we execute the following command to link our project to the billing account above:

```
[alian@aanafcheh ~]$ gcloud alpha billing projects link ali-anafcheh-
oalan002 -billing-account=123abc-123abc-123abc
billingAccountName: billingAccounts/123abc-123abc-123abc
billingEnabled: true
name: projects/ali-anafcheh-oalan002/billingInfo
projectId: ali-anafcheh-oalan002
```

Google cloud has a big list of virtual machines available that fit different cased depending on the requirements. Some machines are designed for high CPU usage and some are designed for high memory or I/O usage. However, for this study a standard machine would be enough. The details of the machine are as follows:

```
[alian@aanafcheh ~]$ gcloud compute machine-types list | grep n1-
standard
n1-standard-1    us-central1-f              1     3.75
n1-standard-2    us-central1-f              2     7.50
```

The n1-standard-1 is a good choice for our agent hosts to be monitored and the n1-standard-2 is good enough to be the manager in a test environment.

3.1.2 Terraform

Now that we have integrated our environment with Google Cloud, we can use a infrastructure as code software to create our infrastructure in the cloud. The main reason I chose to do this is to keep things easier to manage. Terraform allows us to write each of our resources and our configuration as a code using their easy to learn and understand configuration language. The best part of this is that we can create different files for different resources which allow us to manage our infrastructure conveniently and smartly.

As an example, we can create one Terraform file for our Wazuh manager virtual machine that includes the code about all the machine's configuration. This will allow us to be able to see all the relevant configurations of a certain machine in one file and change things easily in the future. This way we can also keep track of our infrastructure in a version control system like Git and revert changes in case of any issues.

How does Terraform run the code? In this section we will go through the steps to allow Terraform to access our project programmatically through a command line. Once Terraform has access permissions to our project and our infrastructure code is ready, running a simple ”terraform plan” command will give us the whole information of what would be created. Once we accept the plan, all the changes will be saved in a file called Terraform state which can be hosted remotely in a storage node. This allows us to have a unified state file which will contain the updated version of the infrastructure and prevents people from making unwanted changes. Therefore, in the future, if there is a certain change in the infrastructure code, Terraform will evaluate how that change affects the whole infrastructure and report the result back to us.

After installing Terraform, let's start by connecting it to our Google Cloud project. We need to create a service account specifically for Terraform which would

generate credentials specifically for Terraform to be able to communicate with our project as follows:

```
[alian@aanafcheh ~]$ gcloud iam service-accounts create terraform --
display-name "Terraform"
Created service account [terraform].
```

Now let's create the keys that will allow Terraform to access our "ali-anafceh-oalan002" project as follows:

```
[alian@aanafcheh ~]$ gcloud iam service-accounts keys create
~/.config/gcloud/terraform.json --iam-account terraform@ali-anafcheh-
oalan002.iam.gserviceaccount.com
created key [76439639grefdfv9e869864983fdvd939r8v6fd98v63] of type
[json] as [/home/alian/.config/gcloud/terraform.json] for [terraform@ali-
anafcheh-oalan002.iam.gserviceaccount.com]
```

Next, we give Terraform permissions to make any changes in our project. Google Cloud already has default roles like "viewer", "editor" etc which can be used. We will use the "editor" role in this case:

```
[alian@aanafcheh ~]$ gcloud projects add-iam-policy-binding ali-
anafcheh-oalan002 --member serviceAccount:terraform@ali-anafcheh-
oalan002.iam.gserviceaccount.com --role roles/editor
bindings:
- members:
 - serviceAccount:service-33458730939827@compute-
system.iam.gserviceaccount.com
 role: roles/compute.serviceAgent
- members:
 - serviceAccount:33458730939827-compute@developer.gserviceaccount.com
 - serviceAccount:33458730939827@cloudservices.gserviceaccount.com
 - serviceAccount:terraform@ali-anafcheh-
oalan002.iam.gserviceaccount.com
 role: roles/editor
```

```
- members:
 - user:aanafcheh@protonmail.com
 role: roles/owner
etag: Tcidfv09tB=
version: 1
```

Terraform is all ready to be used, but let's do one final step to make things even easier. As mentioned before in this section, it's better to store the Terraform state file remotely rather than locally, accordingly let's store our state file in the same Google Cloud account. We will use the "gsutil" tool which was already installed with the "gcloud" package. This tool allows us to interact with Google Storage to create files, buckets etc. Hence, we will create a bucket named "terraform-statefile" that will host our statefile:

```
[alian@aanafcheh ~]$ gsutil mb -p ali-anafcheh-oalan002 gs://terraform-statefile
Creating gs://terraform-statefile/…
```

If we would like to enable versioning on this bucket to keep old versions of our objects, run the following command:

```
[alian@aanafcheh wazuh-terraform-ansible]$ gsutil versioning set on gs://terraform-statefile
Enabling versioning for gs://terraform-statefile/...
```

We can start creating our Terraform files now. I created a repository below that will include all of the Terraform configuration code that I used for this practical implementation.

```
[alian@aanafcheh repos]$ git clone https://github.com/aanafcheh/wazuh-terraform-ansible.git
Cloning into 'wazuh-terraform-ansible'...
warning: You appear to have cloned an empty repository.
```

Terraform reads any files with the extension ”tf” and ”tf.json”. Firstly, let’s create our first Terraform file which connects it to our Google Cloud. Create a file with any name such as ”backend.tf” that includes this configuration:

```
// Terraform state file stored in Google cloud Storage
terraform {
 backend "gcs" {
   bucket  = "terraform-statefile"
   path    = "/terraform.tfstate"
   project = "ali-anafcheh-oalan002"
 }
}

// Google Cloud provider
provider "google" {
  credentials = "${file("/home/alian/.config/gcloud/terraform.json")}"
  project     = "ali-anafcheh-oalan002"
}
```

As the code above shows, in the first part I configured Terraform to store a state file called ”terraform.tfstate” in my “terraform-statefile” bucket that I have created already. In the second part, it is configured to read the credentials files that were specifically created for Terraform to access our project. Before we initialize our Terraform, we also need to export the following variable to be able to connect Google Cloud Storage:

```
[alian@aanafcheh wazuh-terraform-ansible]$ export
GOOGLE_APPLICATION_CREDENTIALS=/home/alian/.config/gcloud/terraform.json
[alian@aanafcheh wazuh-terraform-ansible]$ export GOOGLE_PROJECT=ali-
anafcheh-oalan002
```

Of course, the environment variables above could be placed in our user’s bash profile in case we will be using the same configuration often.

It’s finally time to initialize our Terraform backend:

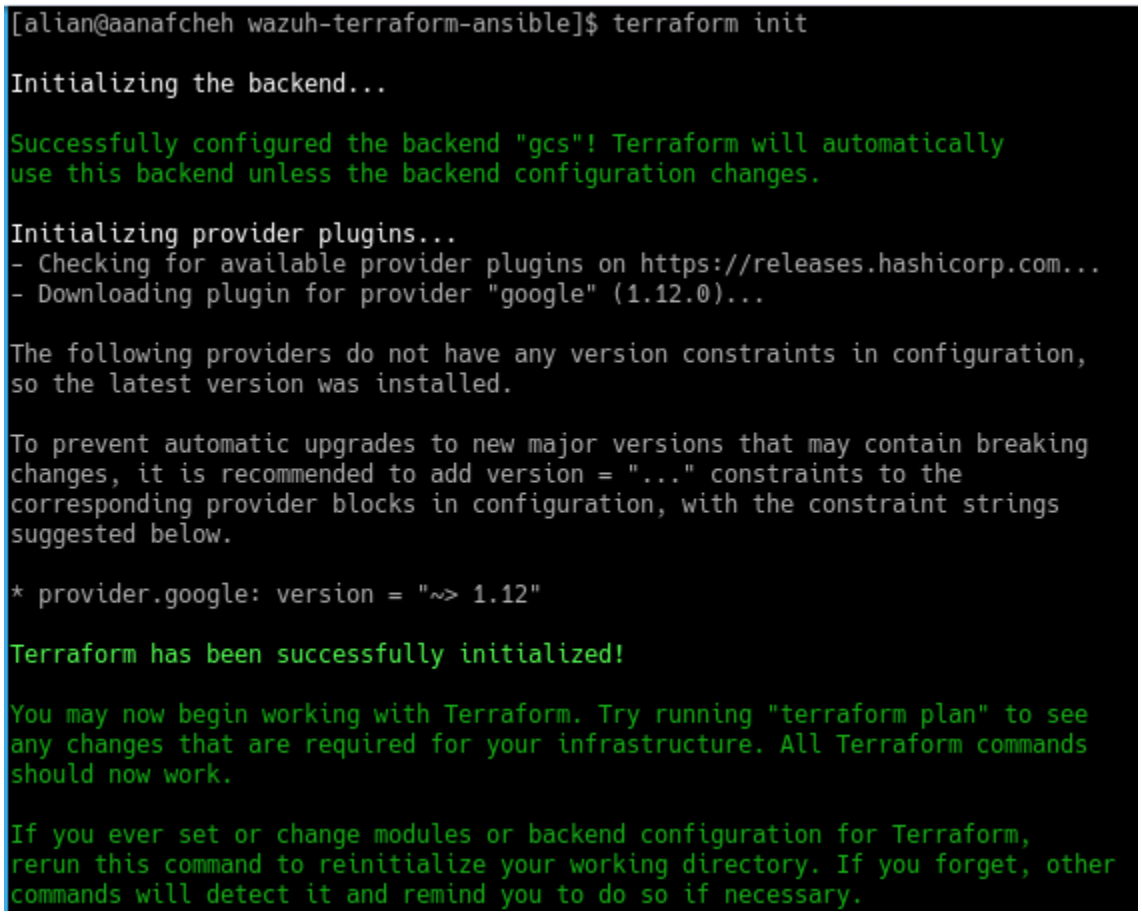


Figure 2. Terraform initialization succeeded, stage 1

As we didn't apply anything into our Terraform state yet, running "terraform plan" will report that our infrastructure is up to date:

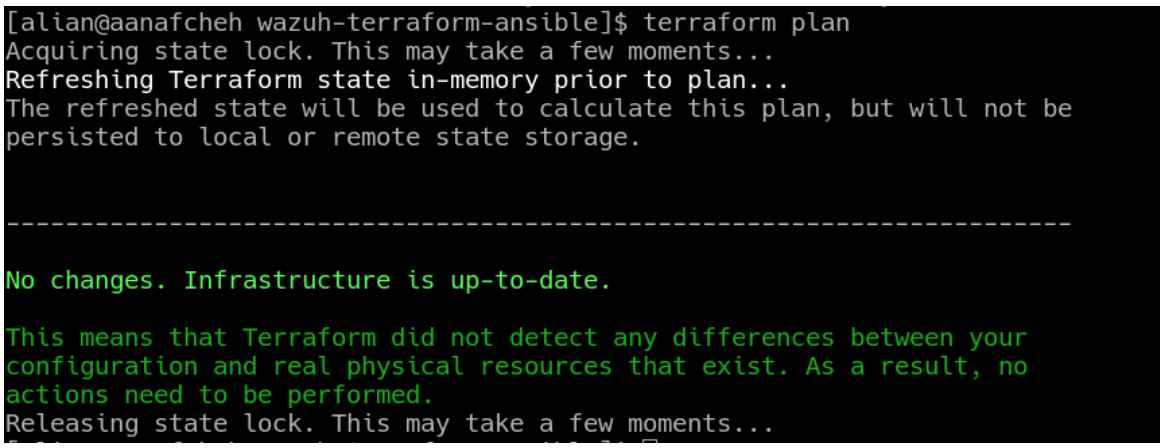


Figure 3. Terraform infrastructure up-to-date stage 2

Terraform is all ready now, and we will create three “tf” files for each of our machines. Here’s the Wazuh manager:

```
# wazuh-manager.tf
# Create a new virtual machine
resource "google_compute_instance" "wazuh_manager" {
   name = "wazuh-manager"
   machine_type = "n1-standard-2"
   zone = "europe-west3-a"  // Frankfurt region

   tags = ["wazuh-manager", "staging"]

   boot_disk {
      initialize_params {
      image = "centos-7"
   }
  }

  metadata {
    sshKeys = "${var.gc_ssh_user}:${file(var.gc_ssh_pub_key_file)}"
  }

network_interface {
   network = "default"
   access_config {}
}
}

// show the virtual machine's public IP address
output "wazuh_manager_address" {
  value = "$
{google_compute_instance.wazuh_manager.network_interface.0.access_config.
0.nat_ip}"
}
```

The two files for the two virtual machines used as agents are as follows:

```
# wazuh-agent-1.tf
# Create a new virtual machine
resource "google_compute_instance" "wazuh_agent_1" {
   name = "wazuh-agent-1"
   machine_type = "n1-standard-2"
   zone = "europe-west3-a"  // Frankfurt region

   tags = ["wazuh-agent-1", "staging"]

   boot_disk {
      initialize_params {
      image = "centos-7"
   }
  }

  metadata {
    sshKeys = "${var.gc_ssh_user}:${file(var.gc_ssh_pub_key_file)}"
  }

network_interface {
   network = "default"
   access_config {}
}
}

// show the virtual machine's public IP address
output "wazuh_agent_1_address" {
  value = "$
{google_compute_instance.wazuh_agent_1.network_interface.0.access_config.
0.nat_ip}"
}

# wazuh-agent-2.tf
# Create a new virtual machine
resource "google_compute_instance" "wazuh_agent_2" {
   name = "wazuh-agent-2"
```

```
   machine_type = "n1-standard-2"
   zone = "europe-west3-a"  // Frankfurt region

   tags = ["wazuh-agent-2", "staging"]

   boot_disk {
      initialize_params {
      image = "centos-7"
   }
  }

  metadata {
    sshKeys = "${var.gc_ssh_user}:${file(var.gc_ssh_pub_key_file)}"
  }

network_interface {
   network = "default"
   access_config {}
}
}

// show the virtual machine's public IP address
output "wazuh_agent_2_address" {
  value = "$
{google_compute_instance.wazuh_agent_2.network_interface.0.access_config.
0.nat_ip}"
}
```

One of the other strengths of Terraform is that it is easily readable and understandable. All you need to do is to provide it with the right information. The code above shows, we created three instances that will run CentOs 7. We created a common network called “default” for all the three machines to be able to communicate with each other and also assigned them public IPs to be able to access them from my local computer. In Terraform, we can create variables for repeated values. For example, I added my public SSH key in a file called ”variables.tf” and named my variables as ”${var.gc_ssh_user}” and $

{var.gc_ssh_pub_key_file}". The purpose of the SSH key is to be able to easily log in to my machine with no password, and for future use with Ansible.
Last but not least, I also added the relevant security rules to allow traffic to Kibana and master-agent communication in a file called "firewall.tf" which is as follows:

```
#firewall.tf
// Allow https traffic and pings from everywhere
resource "google_compute_firewall" "wazuh" {
    name = "wazuh"
    network = "default"

    allow {
      protocol = "icmp"
    }

    allow {
        protocol = "tcp"
        ports = ["5601"]
    }

    allow {
        protocol = "tcp"
        ports = ["1515"]
    }

    allow {
        protocol = "tcp"
        ports = ["1514"]
    }

    source_ranges = ["0.0.0.0/0"]
}
```

Everything is ready and we can simply create our infrastructure by running ”terraform apply”. We review the changes that will take affect and type ”yes”, if everything looks fine. Briefly, Terraform will make the following changes:

+ google_compute_firewall.wazuh
+ google_compute_instance.wazuh_agent_1
+ google_compute_instance.wazuh_agent_2
+ google_compute_instance.wazuh_manager

Green color in Terraform means that new resources will be created. Red color means that the specified resources will be destroyed and yellow mean that the specified resource will change. Figure 4 shows the result after confirming Terraform’s change:

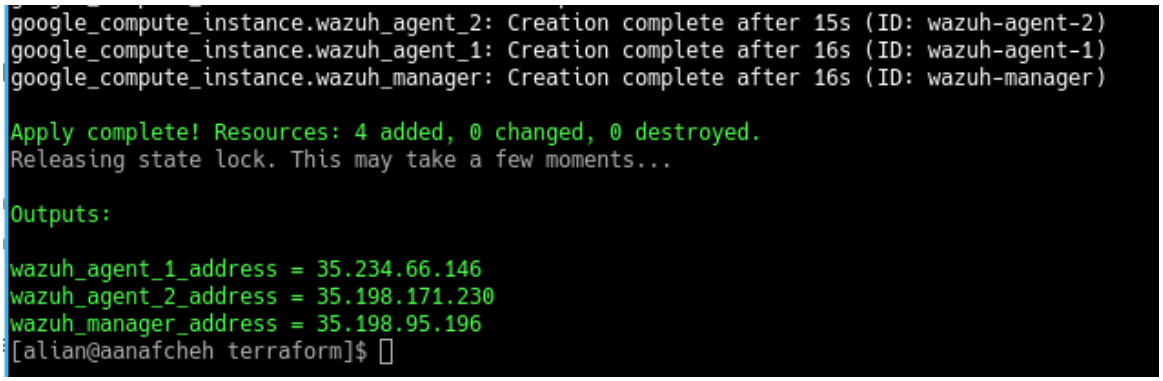


Figure 4. Terraform added resources, stage 3

Another important feature that you see in Figure 4 above is ”output variables”. In our Terraform code, the public IP of each machine was specified as an output variable so that we can save it immediately after instances creation without the need to make further checks.

I have specified ”aanafcheh” as my username for the instances, and they automatically use the ”name” part of the Terraform code as their host name. Let’s make an attempt to test SSH to our manager as follows:

```
[alian@aanafcheh ~]$ ssh aanafcheh@35.198.95.196
Enter passphrase for key '/home/alian/.ssh/id_ed25519':
Last login: Mon May 21 21:33:10 2018 from dsl-hkibng11-50dc4c-
22.dhcp.inet.fi
[aanafcheh@wazuh-manager ~]$
```

As the code above shows, SSH was successful to our virtual machine which was named “wazuh-manager”.

3.2 Wazuh Installation

This section explains Wazuh installation using an IT automation tool called Ansible. Subsection 3.2.1 introduces Ansible and how Wazuh can be installed using the tool according to Ansible (2018) and Wazuh (2018). Subsection 3.2.2 explains the process of installing Wazuh using Ansible.

3.2.1 Ansible

Ansible is an IT automation and configuration management tool. It is perfect for running generic pieces of code or commands on many machines. It is used by big companies to manage and maintain applications and configurations on their infrastructure. It is agentless and runs everything through SSH. In addition, it does not need any extra software to be installed on the remote hosts.

Ansible describes their configuration and deployment language as Playbooks. In this case, I will create a Playbook for Elastic Stack, the Wazuh manager and Wazuh agent. Each Playbook has roles which include tasks to be executed, variables, templates, dependencies etc. For the installation, I will fork the official Ansible Playbooks provided by Wazuh and modify them to fit my requirements

and infrastructure. I will describe the important parts of Ansible and the Playbooks to keep this section short and stay on the topic. The first step is to fork or clone the following repository locally:

```
[alian@aanafcheh ansible]$ git clone https://github.com/wazuh/wazuh-ansible
```

Ansible uses YAML as its configuration language. In the repository's root directory, we can see different files such as "wazuh-elastic_stack-single.yml" which is the Playbook that that will setup a single manager architecture system. As an example, here is the Playbook's content:

```
- hosts: 35.198.95.196
  roles:
      - { role: ansible-wazuh-manager }
      - { role: ansible-role-elasticsearch, elasticsearch_network_host: 'localhost' }
      - { role: ansible-role-logstash, elasticsearch_network_host: 'localhost' }
      - { role: ansible-role-kibana, elasticsearch_network_host: 'localhost' }
```

As the YAML code above shows, the Playbook starts with specifying a host. I specified the IP address of the manager here as I will run this Playbook on it. Ansible is very flexible and a separate host file with different groups and variables can be created for bigger environments. However, I will keep the Ansible part of this study simple and straightforward to continue with the installation and have the intrusion system installed. In the rest of the Playbook, we can see the roles specified in which each of them are in their own directory at the root of the repository.

The core of a role is its "tasks" directory. It includes at least one YAML file or in other words, task files that include a set of Ansible modules specifically programmed to carry out certain tasks. There are a lot of Ansible modules

available already for different kind of services and operating system. For example, the tasks that add the Wazuh repository and install the Wazuh manager are as follows:

```
- name: RedHat/CentOS/Fedora | Install Wazuh repo
  yum_repository:
    name: wazuh_repo
    description: Wazuh repository
    baseurl: https://packages.wazuh.com/3.x/yum/
    gpgkey: https://packages.wazuh.com/key/GPG-KEY-WAZUH
    gpgcheck: yes
  when:
    - ansible_distribution_major_version|int > 5

- name: Install wazuh-manager and expect (EL5)
  package: pkg={{ item }} state=latest
  with_items:
    - wazuh-manager
    - expect
  when:
    - ( ansible_distribution == 'CentOS' or ansible_distribution ==
'RedHat' ) and  ansible_distribution_major_version|int < 6
```

Therefore, a role includes tasks like above to carry out certain tasks and copy certain configuration files to meet a goal such as fully installing and configuring a Wazuh manager.

3.2.2 Installation Process

Each role may have its own default variables located under ”role-name/defaults/main.yml”. I will go through all of these variables to make sure they match my preferences and my installation. One of the important variables is the log files to be monitored. For now, I will go with the defaults which are the following:

- /var/log/messages
- /var/log/secure
- /var/log/maillog
- /var/log/httpd/error_log
- /var/log/httpd/access_log
- /var/ossec/logs/active-responses.log

Also, in the “wazuh-agent” Playbook we have to specify the address of the manager so that authentication requests are sent there. I will not be using any https certificates in this setup as this is a demo environment with no DNS names, in other words there is no point in setting up self-signed certificates.

One of the last variables to modify is to specify an “authd”(explained in Table 1) password in both the manager and the agent playbooks to be able to authenticate and register agents on the manager. After running the manager playbook first, we will run the agent playbooks on each host afterwards which will do the authentication part described earlier.

After modifying a few other values, we run the manager playbook on our manager host as follows:

```
[alian@aanafcheh wazuh-ansible]$ ansible-playbook wazuh-elastic_stack-
single.yml -u aanafcheh -i 35.198.95.196,

PLAY [all]
*********************************************************************
TASK [Gathering Facts]
*********************************************************************
Enter passphrase for key '/home/alian/.ssh/id_ed25519':
ok: [35.198.95.196]
TASK [ansible-wazuh-manager : RedHat/CentOS | Install Nodejs repo]
*********************************************************************
changed: [35.198.95.196]
```

```
TASK [ansible-wazuh-manager : Fedora | Install Nodejs repo]
*********************************************************************
skipping: [35.198.95.196]

TASK [ansible-wazuh-manager : RedHat/CentOS/Fedora | Install Wazuh repo]
*********************************************************************
changed: [35.198.95.196]
.
.
PLAY RECAP
*********************************************************************
35.198.95.196              : ok=58   changed=11   unreachable=0
   failed=0
```

The above Ansible run was summarized. As the code above shows, like Terraform, each single task has a color and it can be either ”ok”,”changed”,”skipping” or ”failed. In this case, I only specified one host. We can easily run this on a huge group of hosts, and allow ansible handle everything. That is the power of Ansible. The Kibana web page should now be available at ”35.198.95.196:5601” as shown in Figure 5:

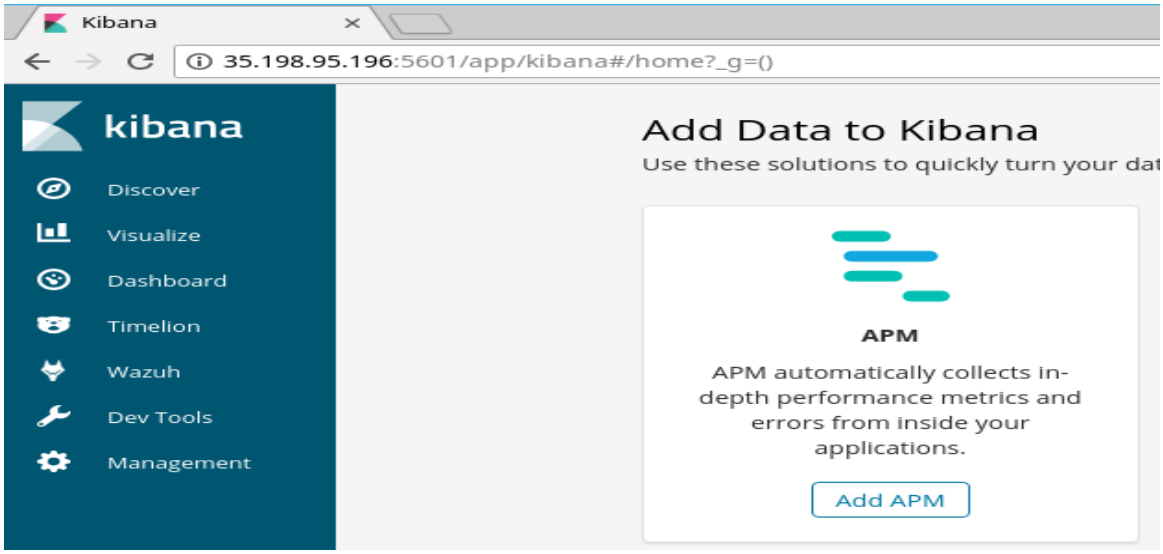


Figure 5. Kibana and Wazuh plugin, stage 1

We can also see the Wazuh plugin on the left menu as shown in Figure 5. That's one of the important Wazuh added features on top of OSSEC which will make administrating the agents easier. Let's start by clicking on the Wazuh plugin and setup the rest of the agents. Once we click on the plugin, we will be presented with a page to connect to the Wazuh manager API as all the information collection for the plugin will be done through the API. I already added a username "aanafcheh" and a password as my credentials and authenticated to the API as shown in Figure 6:

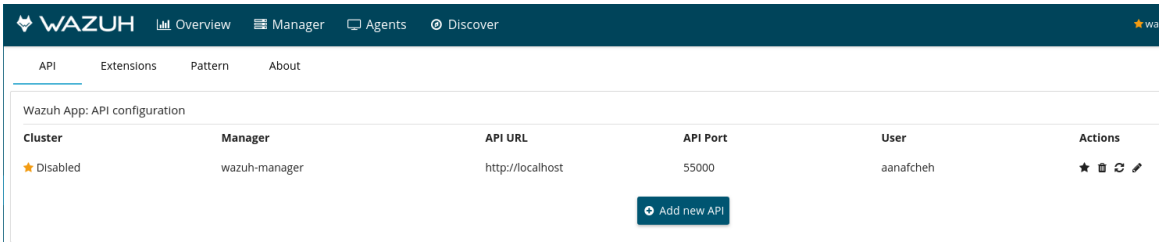


Figure 6. Wazuh Manager API, stage 2

After successful authentication we are presented with the interface of the Wazuh plugin as shown in Figure 6. We will go through the plugin in section 3.3. Let's continue the installation process by adding agents. This is done through Ansible as well and by running the following command:

```
ansible-playbook wazuh-agent.yml -u aanafcheh -i 35.234.66.146, -vvv
```

This task "TASK [ansible-wazuh-agent : Linux | Register agent (via authd)] " must be executed and be labeled as "changed. Once the Ansible run is done with no fails, the agent should show up in Kibana in the Wazuh plugin as follows in Figure 7:

ID	Name	IP	Status	Group	OS platform
000	wazuh-manager	127.0.0.1	Active	---	CentOS Linux
003	wazuh-agent-1	35.234.66.146	Active	default	CentOS Linux

Figure 7. Available Wazuh agents, stage 3

We will do the same steps as above with the second host and add it as an agent. Finally, we will configure our Elasticsearch template for analyzing data in the discover page of Kibana. There are two templates and one has to be chosen as default. The wazuh-monitoring template is used to monitor the manager as it highlights specific values from its logs like monitoring a cluster. The other template is the wazuh-alerts template which analyses the logs and highlights information as shown in Figure 8.

Figure 8. Elasticsearch log analysis, stage 3

3.3 Wazuh Capabilities Evaluation

In this section, I will test the important features of Wazuh like file integrity, log monitoring etc. that I introduced earlier in sections 2.1.2 and 2.2.

3.3.1 Log Monitoring

This feature is already working out of the box as one of Wazuh's most import features is their enhanced OSSEC rules that are deployed by default during the installation. I'll test this feature by installing a "yum" package as I am monitoring the log file of the package manager. I installed "nano" text editor on wazuh-agent-1, and this event was instantly reported to the manager. I could easily find this log by simply searching in Kibana which is powered by the Elasticsearch engine. Figure 9 shows the results as follows:

@timestamp		May 24th 2018, 04:48:59.000
_id		j3jVj2MBiGZfqYq5vz4n
_index		wazuh-alerts-3.x-2018.05.24
_score		-
_type		wazuh
agent.id		003
agent.ip		35.234.66.146
agent.name		wazuh-agent-1
full_log		May 24 01:48:58 wazuh-agent-1 yum[15393]: Installed: nano-2.3.1-10.el7.x86_64
host		wazuh-manager
id		1527126539.709582
location		/var/log/messages
manager.name		wazuh-manager
path		/var/ossec/logs/alerts/alerts.json
predecoder.hostname		wazuh-agent-1
predecoder.program_name		yum
predecoder.timestamp		May 24 01:48:58
rule.description		New Yum package installed.
rule.firedtimes		1
rule.groups		syslog, yum, config_changed, gpg13_4.10
rule.id		2932
rule.level		7
rule.mail		false
rule.pci_dss		10.6.1, 10.2.7

Figure 9. Yum package installation alert

Figure 9 also shows a lot of other important information and values such as the source IP address, the agent name, the command that was run, the location of the log etc. The more interesting part is that as the installation command was run with ”sudo” another alert was received regarding this event as well indicating which user escalated privileges and whether it was successful or not.

3.3.2 File Integrity

This feature will inform us of any changes in the directories that we specified. The following is my current configuration:

```
syscheck:
  frequency: 43200
  scan_on_start: 'yes'
  auto_ignore: 'no'
  alert_new_files: 'yes'
  ignore:
    - /etc/mtab
    - /etc/mnttab
```

```
    - /etc/hosts.deny
    - /etc/mail/statistics
    - /etc/random-seed
    - /etc/random.seed
    - /etc/adjtime
    - /etc/httpd/logs
    - /etc/utmpx
    - /etc/wtmpx
    - /etc/cups/certs
    - /etc/dumpdates
    - /etc/svc/volatile
  no_diff:
    - /etc/ssl/private.key
  directories:
    - dirs: /etc,/usr/bin,/usr/sbin
      checks: 'check_all="yes"'
    - dirs: /bin,/sbin
      checks: 'check_all="yes"'
```

In the configuration above, I have file integrity enabled on a directory such as ”/etc/”, meaning that I will be alerted of any new files or changes in that directory. We can also ignore files that change often. I will make a change on my agent’s OSSEC configuration and also add a file to the “/etc” directory. Wazuh plugin should report that as shown in Figure 10:

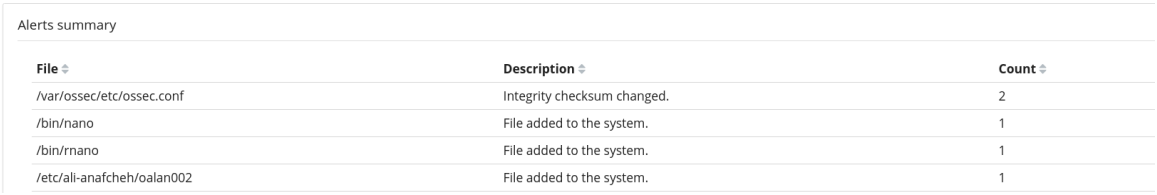


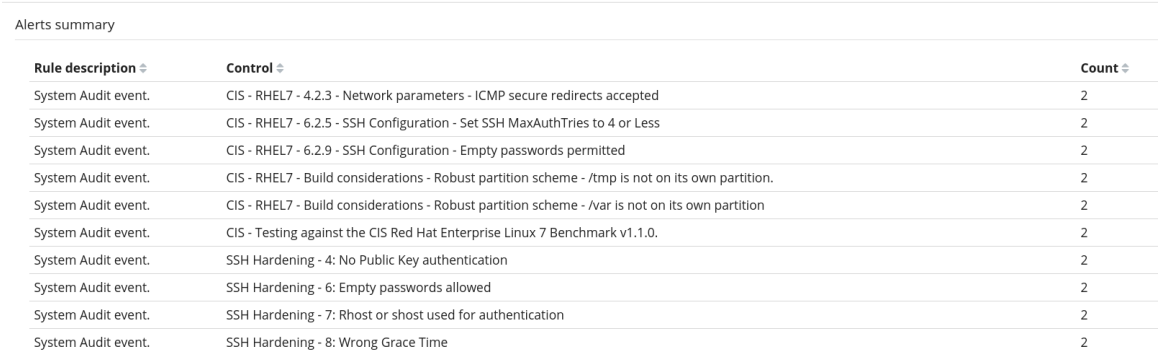
Figure 10. Wazuh file integrity check

3.3.3 Policy Monitoring and Malware Detection

This part is done using both syscheck and rootcheck. The current configuration of rootcheck is as follows:

```
 <rootcheck>
   <disabled>no</disabled>
   <check_unixaudit>yes</check_unixaudit>
   <check_files>yes</check_files>
   <check_trojans>yes</check_trojans>
   <check_dev>yes</check_dev>
   <check_sys>yes</check_sys>
   <check_pids>yes</check_pids>
   <check_ports>yes</check_ports>
   <check_if>yes</check_if>

   <!-- Frequency that rootcheck is executed - every 12 hours -->
   <frequency>3600</frequency>




   <rootkit_files>/var/ossec/etc/shared/rootkit_files.txt</rootkit_files>
   <rootkit_trojans>/var/ossec/etc/shared/rootkit_trojans.txt</rootkit_t
rojans>
   <system_audit>/var/ossec/etc/shared/system_audit_rcl.txt</system_audi
t>
   <system_audit>/var/ossec/etc/shared/system_audit_ssh.txt</system_audi
t>

   <system_audit>/var/ossec/etc/shared/cis_rhel7_linux_rcl.txt</system_a
udit>

   <skip_nfs>yes</skip_nfs>
 </rootcheck>
```

As you can see, it is instructed to audit the system and also scan for Trojans or malware every hour on wazuh-agent-2. It uses the defined standard databases listed in the code above to make the scans, compare and suggest corrections. As an example, Figure 11 shows the security flaws that were detected on wazuh-agent-2:



Alerts summary

Rule description	Control	Count
System Audit event.	CIS - RHEL7 - 4.2.3 - Network parameters - ICMP secure redirects accepted	2
System Audit event.	CIS - RHEL7 - 6.2.5 - SSH Configuration - Set SSH MaxAuthTries to 4 or Less	2
System Audit event.	CIS - RHEL7 - 6.2.9 - SSH Configuration - Empty passwords permitted	2
System Audit event.	CIS - RHEL7 - Build considerations - Robust partition scheme - /tmp is not on its own partition.	2
System Audit event.	CIS - RHEL7 - Build considerations - Robust partition scheme - /var is not on its own partition	2
System Audit event.	CIS - Testing against the CIS Red Hat Enterprise Linux 7 Benchmark v1.1.0.	2
System Audit event.	SSH Hardening - 4: No Public Key authentication	2
System Audit event.	SSH Hardening - 6: Empty passwords allowed	2
System Audit event.	SSH Hardening - 7: Rhost or shost used for authentication	2
System Audit event.	SSH Hardening - 8: Wrong Grace Time	2

Figure 11. Policy monitoring

3.3.4 Active Response

This is one of the interesting features in which Wazuh manager takes automatic actions to counter certain attacks or activities. Wazuh has a few default scripts located at ”/var/ossec/active-response/bin” which can be configured to be used as an active response. I decided to try the firewall-block active response which will be fired once a user tries to make invalid ssh connections for eight times. The following has to be in the configuration file of the manager:

```
<command>
  <name>firewall-drop</command>
  <executable>firewall-drop.sh</executable>
  <expect>srcip</expect>
</command>

<active-response>
  <command>firewall-block</command>
  <location>all</location>
  <rules_group>authentication_failed,authentication_failures</rules_group>
  <timeout>700</timeout>
</active-response>
```

The first part of the configuration defines the script to be used for the active response and the second part of the configuration is the active response itself which refers to the name of the command to be used as well. The user will be

blocked for 700 minutes when this active response is provoked. Let's try and test the connection to our wazuh-agent-2 first as follows:

```
[alian@aanafcheh ~]$ ping 35.198.171.230
PING 35.198.171.230 (35.198.171.230) 56(84) bytes of data.
64 bytes from 35.198.171.230: icmp_seq=1 ttl=57 time=38.7 ms
64 bytes from 35.198.171.230: icmp_seq=2 ttl=57 time=40.3 ms
^C
--- 35.198.171.230 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
```

Now we will make many invalid SSH connections with an invalid user as follows:

```
[alian@aanafcheh ~]$ ssh ali@35.198.171.230
ali@35.198.171.230: Permission denied (publickey,gssapi-keyex,gssapi-
with-mic).
```

After eight retries the active response is invoked in the manager, as can be seen in the following logs as well:

```
** Alert 1527131910.778856: -
ossec,active_response,pci_dss_11.4,gpg13_4.13,
2018 May 24 03:18:30 (wazuh-agent-2)
35.198.171.230->/var/ossec/logs/active-responses.log
Rule: 601 (level 3) -> 'Host Blocked by firewall-drop.sh Active Response'

Src IP: 80.244.70.11
Thu May 24 03:18:28 UTC 2018 /var/ossec/active-response/bin/firewall-
drop.sh add - 80.220.76.22 1527131908.777026 5712
script: firewall-drop.sh
type: add
```

The connection to the agent is blocked for 700 minutes from the source IP above:

```
[alian@aanafcheh ~]$ ping 35.198.171.230
PING 35.198.171.230 (35.198.171.230) 56(84) bytes of data.
^C
--- 35.198.171.230 ping statistics ---
5 packets transmitted, 0 received, 100% packet loss, time 4060ms
```

4 CONCLUSION

The aim of this study was to study intrusion detection in detail and learn about the process of monitoring activities on individual hosts in various ways. I find Wazuh to be very interesting and full of great features with great flexibility to monitor different systems. For smaller companies, this solution might not be suitable, as it would require maintenance and human resources to watch and acknowledge the alerts and activities and to develop different ways to encounter them. However, this is a great tool for companies with many personal servers that host sensitive information of different customers.

I find Wazuh’s integration with Elastic Stack to be its greatest feature as it provides great log monitoring and analysis in combination with Elastic search specifically which makes reading, classifying and filtering huge number of logs much easier.

REFERENCES

OSSEC. 2018. Welcome to OSSEC's documentation! WWW document. Available at: https://www.ossec.net/docs/ [Accessed 12 May 2018].

Wazuh. 2018. Welcome to Wazuh. WWW document. Available at: https://documentation.wazuh.com/current/index.html [Accessed 12 May 2018].

Elastic Stack. 2018. Elastic Stack and Product Documentation. WWW document. Available at: https://www.elastic.co/guide/index.html [Accessed 14 May 2018].

Google. 2018. Google Cloud Platform Documentation. WWW document. Updated 9 May 2018. Available at: https://cloud.google.com/docs/ [Accessed 16 May 2018].

Terraform. 2018. Terraform Documentation. WWW document. Available at: https://www.terraform.io/docs/index.html [Accessed 20 May 2018].

Isla, D. 2017. Managing GCP Projects with Terraform. WWW document. Available at: https://cloud.google.com/community/tutorials/managing-gcp-projects-with-terraform [Accessed 20 May 2018].

Ansible. 2018. Ansible Documentation. WWW document. Available at: http://docs.ansible.com/ansible/latest/index.html [Accessed 20 May 2018]