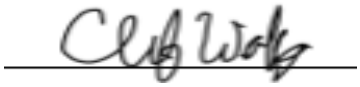
SOC-in-a-Box

by

Clifton Wolfe & Jai Singh

Submitted to
the Faculty of the School of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

Jai Singh and _____ 04/20/2020
Date



Clifton Wolfe _____ 04/20/2020
Date

Tony Iacobelli _____ 04/20/2020
Tony Iacobelli, Faculty Advisor Date

University of Cincinnati
College of
Education, Criminal Justice, and Human Services

April 2020

Table of Contents

Section **Page**

List of Illustrations

Figures

ABSTRACT

Today everyone is a target for cyber-attacks, hackers are only getting more sophisticated and persistent. According to Verizon’s 2018 annual breach report, Cyber-attacks against small businesses make up 58% of targeted attacks, and 76% of all attacks are financially motivated[1]. This makes small financial institutions a prime target for cyber-attacks. These smaller businesses also lack proper cyber defense, only 27% of businesses say they are prepared for a cyber-attack. SOC-In-A-Box is a cyber security solution for businesses that brings the best of on-site and cloud-based architectures together to help solve the growing problem of cyber-attacks and data breaches. SOC-In-A-Box's cutting-edge system is modularly designed to quickly scale and easily integrate with existing solutions, allowing businesses to maintain visibility over their IT and OT infrastructure and trust that their data and business operations are always safe and secure.

1. INTRODUCTION

1.1. Introduction

Cyber security has increasingly become an important aspect of a business to protect its own digital data. Being able to have one hundred percent network visibility is a key aspect to understand what data is being set into and out of a companies' network. In order to fully protect a company, we must have a functional Security Operation Center (SOC) that can provide intrusion detection, endpoint detection and malware analysis. These tools are then being tied together with our backend correlation engine that generate tickets for an analyst to investigate.

1.2. Problem

Cyber-attacks have been on the rise in recent years, and it is quickly becoming apparent that cyber security is necessary for even the smallest of businesses. A Verizon study from earlier in 2019 showed that around 43% of cyber-attacks focus on small businesses[2]. A common issue in this day and age for small companies is that security operations centers are necessary to effectively monitor enterprise networks and respond to security events as they occur, but are incredibly expensive to set up, as they often include licensing for servers and specialized security software on top of the cost of the physical servers. In addition to the aforementioned expenses, the cost of maintaining a SOC and hiring people specifically for that purpose is just too high for some businesses to afford, so many businesses go without a security operations center and instead just accept the risk of being completely unprotected in the event of a cyber security incident.

1.3. Solution

Though many security solutions are available on the market, most only provide a single product or service such as (Cisco's IDS and IPS systems) and most do not supply integration support with other products that would typically be used in a SOC. Our solution, SOC-in-a-box, will be an enterprise security operations center solution that is capable of installing and managing a full suite of free open source software that fulfills the general needs of a security operations center on a single machine or across multiple machines on a network, with working and usable integration with other SOC services already configured at install time. SOC in a box will provide different services including a database of indicators of compromise, a full packet capture service, a central Intrusion detection system, endpoint detection, and a central ticketing and reporting engine that receive alerts from all of the other services to report to in one interface.

1.4. Project Goals/Brief Methodology

Develop an installation configuration that can be deployed on customer networks via ansible with minimal modifications required for each setup. SOC in a box will provide the following services using open source software:

- A database of indicators of compromise (MISP)
- A full packet capture service (Moloch)
- A central Intrusion detection system (Surricata)
- Endpoint detection (Wazuh)
- A central security incident response engine (TheHive)

1.5. Overview

The remainder of this final report outlines in detail how the project was completed. The report includes the following sections: project concept, design objectives, technical approach, user profiles, use case diagram, timeline, and problems encountered.

2. DISCUSSION

2.1. Project Concept

The Security Operations Center (SOC) is tailored towards small to medium sized businesses who need an easily customizable, affordable, and consistently available cyber security solution. The SOC provides this by using strictly free open-source software, which means if a business wants another component added onto their SOC it can easily be added with a simple download, and the software inside the SOC are built as services.

2.2. Design Objectives

A list of the main design objectives along with the explanation is provided below:

2.2.1. Scalable

2.2.1.1. Can handle a small business and up to a larger business with no hiccups or errors.

2.2.2. Customizable

2.2.2.1. A business can take add or remove components from the SOC to fit their custom needs.

2.2.3. Affordable

2.2.3.1. Built in strictly free open-source software so there is no cost for the software.

2.2.4. Available 24/7

2.2.4.1. Needs to be monitoring what always goes in and out of each machine and the network in order to catch nefarious users.

2.2.5. Responsive

2.2.5.1. Needs to catch an attack within 15 minutes for it to be useful.

2.2.6. Protects users

2.2.6.1. Monitors a user's actions and if a user does something to harm a system or network, it will be able to stop it.

2.2.7. Detects and prevents intrusions

2.2.7.1. Monitors everything that enters and exits the internal network and will catch bad traffic.

2.2.8. Easily construct and destroy Virtual Machines

2.2.8.1. Makes the SOC customizable and used for research topics.

2.3. Methodology/Technical Approach

2.3.1. Design Requirements

Our design for our SOC-in-a-box will consist of open source software that will reduce the cost of deployment for our end users/companies and their required budget constraints. We will make the integration process as easy as possible to allow ease of transition for their security operation center. For companies that already have existing solutions, we will take those outputs from those systems and feed them into our back-end database so we can analyze that as well. Our SOC-in-a-box will have none to very minimal configuration due to the way we have written our automation deployment code. Our configuration relies on Ansible that will use shell scripts so that the end product will be able to sit on top of their network. We will be offering an on-prim and cloud-based solutions. For companies that would like to keep their confidential data on site, we will be building a hardware box that will be stored on their site so that they could have an air gapped network. For companies that have multiple data centers and would like to keep it in the cloud, we will be offering network base deploy ability as well since we want to be able to configure everything over the network in a repeatable manner.

2.3.1.1. SOC-in-A-Box Data-Flow Model

Figure 1 shows a diagram of our data flow model.

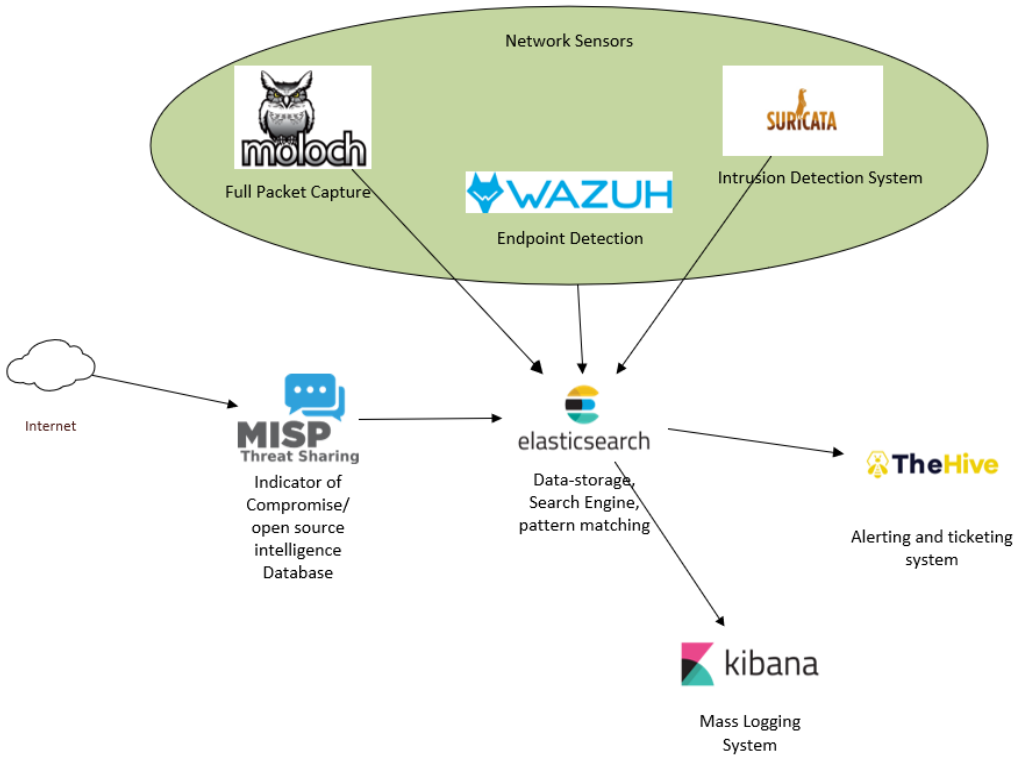


Figure 1. SOC Data Flow Diagram

Like any Security Operations Center, ours relies on network sensors to collect data from which we can detect breaches or potentially malicious traffic. Moloch serves as our full packet capture system, which will help us to keep an exact record of all of the network traffic that our SOC is exposed to so that we can monitor the exact path that an attacker takes from within the network. To compliment Moloch's FPC service, Suricata will serve as an intrusion detection system. Suricata will aid our soc by both monitoring traffic for known malicious intrusion patterns as well as by keeping a log of all the connections that are made in the network. Finally, Wazuh will serve as our platform

agnostic endpoint security solution. Wazuh will monitor all the endpoints of the network to make sure that if malicious programs are downloaded or run, we can neutralize them as fast as possible and can begin to assess the situation.

In addition to our network and endpoint sensors, we will also be utilizing MISP to gather information about current malware trends and to keep a database of indicators of compromise on site. MISP will update our database daily so that we can search the traffic that our network and endpoint monitors record to further ensure that we are always looking for even the most bleeding edge malicious traffic.

Elasticsearch and the ELK stack serve as the backbone for our whole project. All our network and endpoint sensors will relay their data to Elasticsearch instances which will correlate the data to identify any malicious activity and help to track how widespread a breach based off what systems report back specific behaviors. To pair with Elasticsearch, Kibana will be used to aggregate logging from the massive amount of data that Elasticsearch will be processing. By using Kibana we can make sure that we are always able to view our gathered data in a manageable and comprehendible interface. Finally, we will be using theHive to serve as both an alerting interface and a ticketing system. By utilizing the Elasticsearch API and theHive's priority system, we can automatically alert security analysts of any behavior that appears to be malicious or any traffic that is outside the norm within that specific network.

Figure 1 shows these relationships and the ELK stack serves as the centerpiece of data flow in SOC-in-a-box.

2.3.2. Procedures

We are using git as our source control management solution to ensure that all our work is saved and available to everyone. We will use ansible to handle our network-based deployments and automated configuration.

2.4. User Profile

2.4.1. Project Title

SOC-in-a-Box

2.4.2. Potential Users

2.4.2.1. Small Business Information-Security departments

2.4.2.2. Medium sized Business Information-Security departments

2.4.2.3. Financial institutions, logistics, cyber insurance, medical

2.4.3. Software, Interface, and Related Experience

The target demographic of our product will be IT/Information-Security departments, managers for those departments, C level executives, and business owners. All our features will be targeted towards individuals in these roles. Our assumption is that the company will have an IT infrastructure in place already with IT employees with basic knowledge, but the IT staff will need to have a basic knowledge of security operation center tasks or get formal training to use our product.

2.4.4. Experience with Similar Applications

Users in Information-Security roles in both small and medium size businesses will most likely have experience with similar applications and may even have experience with some of the applications used in our product. However, these applications are often custom made by Information-Security departments.

2.4.5. Task Experience

We will provide a ticketing-based security operation center that will integrate multiple components that will be tied together through our correlation engine. The user will be able to gather information which will be displayed on our ticketing system. The analyst can then pursue the incident.

2.4.6. Frequency of Use

The product is intended for daily use to facilitate a workflow for a SOC-analyst.

2.4.7. Key Project Design Requirements that the Profile Suggests

2.4.7.1. Easy to use web interface

2.4.7.2. Ability to integrate with user's current tools

2.5. Use Case Diagram

This diagram visualizes the standard use case for SOC-in-a-Box. End users will have minimal interactions with both endpoint systems and the backend servers. Users will update and manage tickets and alerts through Web interfaces and can shrink or grow the scope of the information that is available through each interface as well.

2.6. Technical Discussion

See Figure 2 for User Case Diagram below shows how the SOC-in-a-Box is set up on the server and how potential users can interact with the ticketing system.

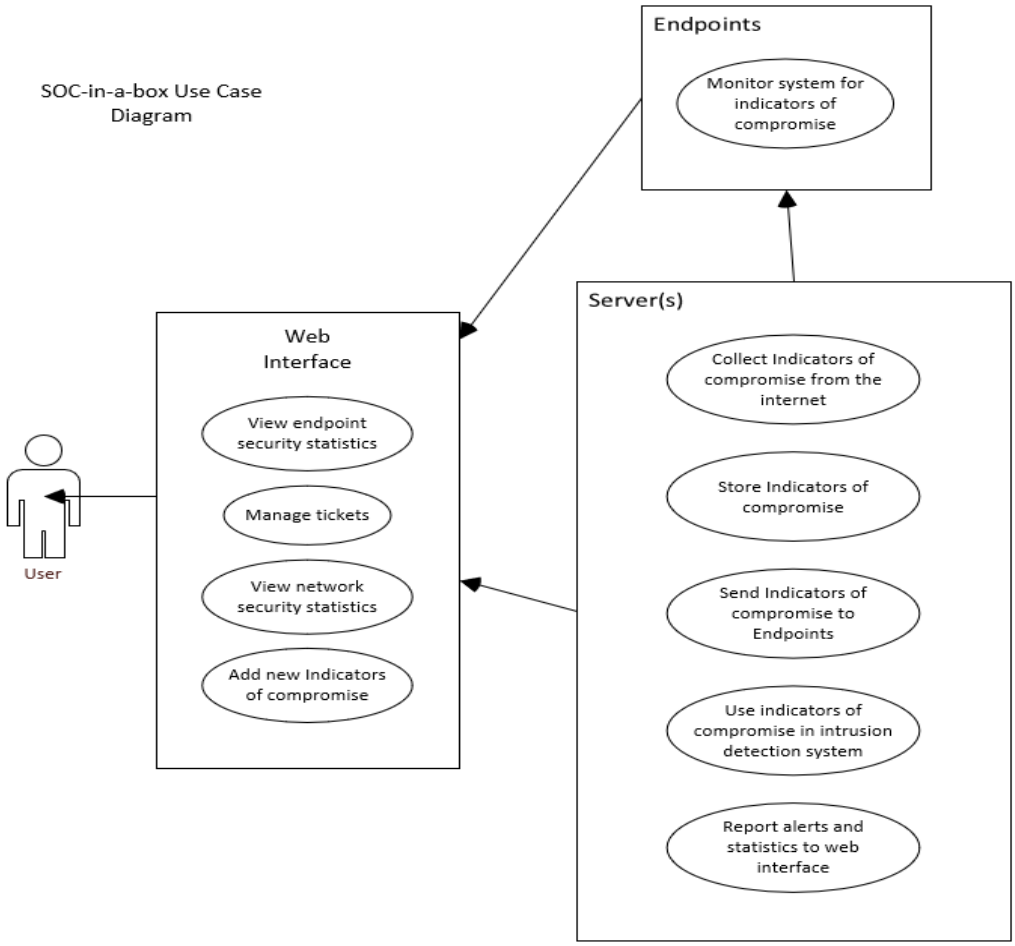


Figure 2. Use Case Diagram

Figure 2 shows the ideal way that a user would interact with SOC-in-a-box.

2.6.1. Network Overview and Discussion

One of the core elements of our design was the use of Ansible to deploy services automatically across a network. By utilizing some of ansible's built-in modules, specifically the Jinja2 templating module, we were able to reduce the requirements of the network hosting our device and configure each service to use the IP addresses of the other services automatically.

After Ansible handled most of the network configuration, there are only three requirements left. First, the devices must have access to the internet, mostly to receive updates for MISP's indicator of compromise (IOC) database and Suricata's rule list. Second, the devices must be able to reach each other, because most of the services are dependent on at least one other service. Because the default configuration for SOC-in-A-Box is to have all of the services on the same machine, this requirement is always met as long as the machine is on some sort of network. Finally, configuring port mirroring on a switch, router, or firewall is necessary for Suricata to best detect network-based attacks and anomalies.

2.6.2. Application Overview and Discussion

SOC-in-A-Box can be broken down into three core sections: Sensor devices, a backend supporting structure, and frontend web interfaces. The sensor devices include devices that have Wazuh endpoint installed on them, Suricata, and Moloch. These devices are primarily meant to detect indicators of compromise

and suspicious activity. When any of these devices detect something worthy of an alert, they pass the alert off to the backend of SOC-in-A-Box.

SOC-in-A-Box's backend is mostly comprised of the ELK stack, which includes ElasticSearch, Logstash, and Kibana. This backend serves many purposes: First, it provides an interface for logs from the sensor devices to be sent and stored for later analysis. As logs pour in throughout the day, this backend layer can be queried to watch for severe alerts that need to be sent up to end users on the front end.

The last layer is comprised of multiple frontend web apps, including TheHive, Cortex, and MISP. While Cortex and MISP are both useful, end users only must interact with TheHive, as integrations between all three provide analysis options from Cortex and MISP from theHive's web app.

2.6.3. Security Elements Deployed

When designing SOC-in-A-Box, we chose to utilize security features made available to us by both our choice in operating system and by the web applications that we used. For security's sake, we run each service under its own user account. In addition, we used Security-Enhanced Linux, which is a Linux kernel module meant to provide more control over access controls. On our front-end applications, we utilized password-based authentication for user logins. We also made use of API keys to allow for more secure interactions between services.

2.6.4. Technical Architecture Diagram

See Figure 1. SOC-Dataflow Diagram for Architecture

2.6.5. User Interface

Figure 3. Shows TheHive's dashboard, which was our primary means of end user interaction.

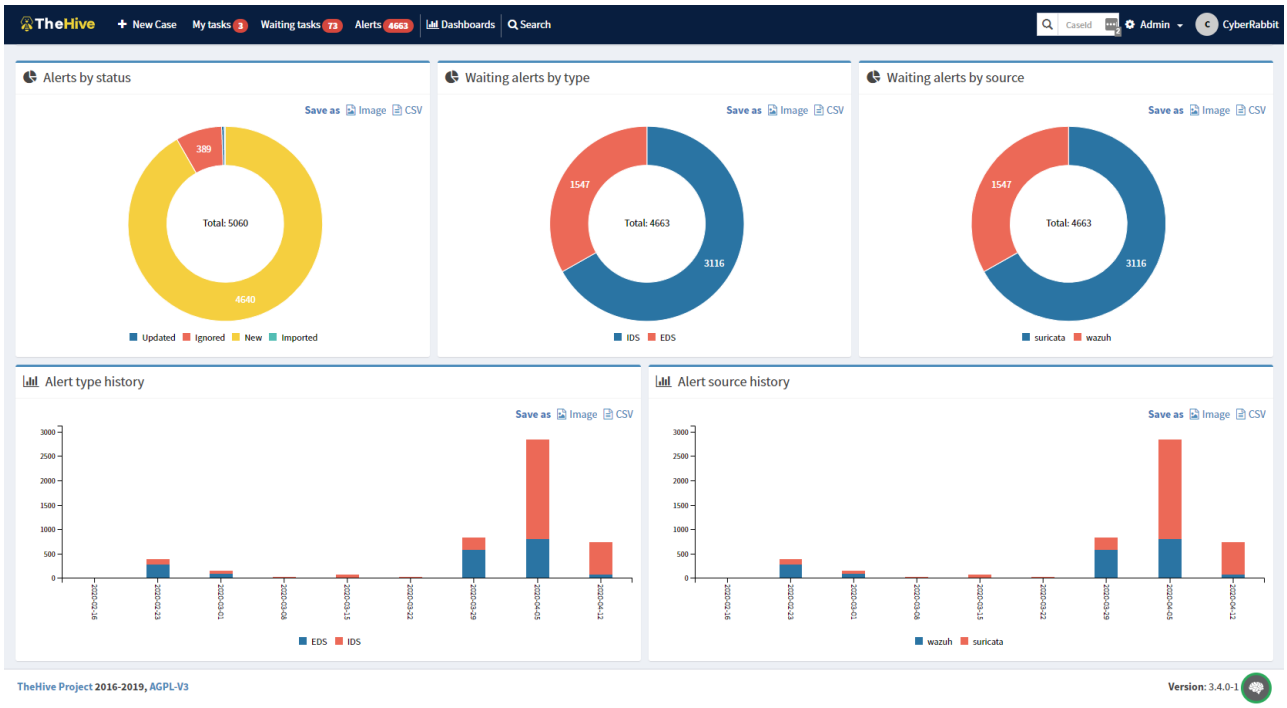


Figure 3. TheHive Dashboard

Figure 3 shows some basic statistics for the alerts that our test instance of SOC-in-a-box generated.

2.7. Testing

This section explains the testing process that was targeted towards the development of SOC-in-a-Box and each core component that ties together to create our fully automated deployment and configuration of each tool (Endpoint Detection, Intrusion Detection, Malware Analysis, Correlation Engine and Ticketing System). Each test plan is to ensure that each service is properly deployed onto a server and works as intended.

2.7.1. Testing Methodology

Because SOC-in-a-Box is built on an on-premise server or using a platform such as AWS, all the infrastructure is code. We used a hybrid approach of Agile and Waterfall methodology during development but focused more on sprints to complete each task. It was important to develop some tools in a specific order because some of the tools are dependent on others. Additionally, some of the tools can be tested individually which allows us to verify each tool is running the way it should as we automate the deployment. Taking that into account, our goal with this methodology is to test each tool at the lowest level (unit testing) and then using integration testing to ensure that all the services are working together in a functional state.

2.7.2. Testing Scope

The scope of this testing encompasses basic deployments, api functionality, and interactions of the services that are deployed. This ensures that our tools

are communicating properly. TheHive is supposed to ingest data from Suricata, Wazuh and MISP using ElasticSearch as a passthrough, so we can confirm the desired flow of data through our services by examining service statuses, log data, and alerts received. We treat single detections of indicators of compromise as test suites in this way, and each test case in that testing suite is just a stop along the data's path. An example is when a Suricata alert triggers, the data is sent to logstash. When the data is seen by cortex, it will be able to identify the pcap file from moloch that was recording at the time of the alert. Cortex can then create a ticket in TheHive that contains a link to that pcap file along with the Suricata alert.

2.7.3. Testing Objective

The objective of SOC-in-a-Box testing is to ensure that each service is up and running when the automated deployment scripts run. Due to the complexity of the infrastructure that the code is built on, the test plans must properly stress the system under heavy load to safeguard the system from failing. In addition, the SOC must be capable of reporting indicators of comprises accurately and in a timely manner. Furthermore, SOC-in-a-Box must remain stable after the automated deployment and configuration scripts are deployed.

2.7.4. Automated Deployment Testing

The purpose of these tests is to ensure that each service is properly deployed onto a fresh CentOS 7 install and that each service is running at a stable version which allows each service to properly integrate with the other deployed services. The quickest way to test the deployment is to use Docker containers which will allow each service to be tested individually. This prevents us from needing to install a whole new server every time we want to test a configuration change. Ansible reports error codes for deployments that fail by default, so these error codes can be used to test whether deployments are successful.

This deployment script will check to see if each service is deployed and running. It will return an error alert if the deployment fails.

2.7.5. Tool Testing

Suricata:

By sending a known IOC to suricata's interface, we were able to test its rules, and were able to view the results by looking through suricata's output file. A basic test was triggering an alert by sending a curl request to TestMyIDS.com, which responds with a standard indicator of compromise.

Wazuh:

We were able to test Wazuh in a similar manner to Suricata. We introduced a known indicator of compromise to a system that a Wazuh agent was installed on, then looked to see if an alert was sent to elasticsearch.

ElasticSearch:

Because interactions between end users and elasticsearch are meant to be kept to a minimum, the most effective way for us to test elasticsearch is to query its API. We sent an http request to elasticsearch and parsed the response for the status of the elasticsearch node. Then we were able to further check the status of each individual index on that node for its status of green, yellow, or red. Ideally we would want only green as the status, but by parsing for all three we can identify some potential issues that might affect our cluster in the future.

TheHive:

We were able to test TheHive by querying its api, authenticating against its api, and attempting to create a ticket in its api.

This test will ensure that each component of the SOC-in-a-Box integrates as intended with proper data propagation into each system. The tests will start with our intrusion detection system capturing packets into a log file that will be stored in an ElasticSearch back-end database.

2.7.6. Automated Configuration and Integration Testing

The configuration for each service is tested on a basic level just by deploying the service and checking the status of the service, but more in depth testing was needed to make sure that services are working together and interacting with each other's APIs.

These tests often focused on making sure that data was flowing from different input channels (like Suricata, moloch, and wazuh agents) into the pipelines and datastores where the data could be effectively utilized in a production environment. To accomplish this our integration testcases send known indicators of compromise to these input channels and watch different APIs for the expected population of data or alerts based upon those inputs. Some of the code we used to create test cases can be found in Figure 4 and in Figure 5.

t/t/capturetest.sh

```
#!/bin/bash
SERVICESTATUS=$(docker exec -it test_capture_1 systemctl show suricata --property=SubState | grep --color=never running)
SERVICE="suricata"
if [ ! -z "$SERVICESTATUS" ];then
    echo "$SERVICE Running: PASS"
else
    echo "$SERVICE Running: FAIL"
fi

docker exec -it test_capture_1 curl 'testmyids.com' -s >/dev/null
OUTPUT=$(curl -XGET "http://localhost:9201/_search" -H 'Content-Type: application/json' -s -d'
{
  "version": true,
  "size": 500,
  "sort": [
    {
      "_score": {
        "order": "desc"
      }
    }
  ],
  "_source": {
    "excludes": []
  },
  "stored_fields": [
    "*"
  ],
  "script_fields": {},
  "docvalue_fields": [
    {
      "field": "@timestamp",
      "format": "date_time"
    },
    {
      "field": "flow.end",
      "format": "date_time"
    },
    {
      "field": "flow.start",
      "format": "date_time"
    },
    {
      "field": "timestamp",
      "format": "date_time"
    }
  ],
  "query": {
    "bool": {
      "must": [],
      "filter": [
```

Figure 4. Test Case Code #1

```
#!/bin/bash
SERVICESTATUS=$(docker exec -it test_elastic_1 systemctl show elasticsearch --property=SubState | grep --color=never running)
SERVICE="elasticsearch"
if [ ! -z "$SERVICESTATUS" ];then
    echo "$SERVICE Running: PASS"
else
    echo "$SERVICE Running: FAIL"
fi

APIQUERYABLE=$(curl -X GET "http://localhost:9201/_cat/indices/*?v&s=index&pretty" -s)
TEST="API QUERYABLE"
if [ ! -z "$APIQUERYABLE" ]; then
    echo "$SERVICE $TEST: PASS"
else
    echo "$SERVICE $TEST: FAIL"
fi

BADSTATUS=$(curl -X GET "http://localhost:9201/_cat/indices/*?v&s=index&pretty" -s | cut -d ' ' -f1 | grep --color=never "red")
TEST="IS INDEX HEALTH RED"
if [ -z "$BADSTATUS" ]; then
    echo "$SERVICE $TEST: PASS"
else
    echo "$SERVICE $TEST: FAIL"
fi


BADSTATUS=$(curl -X GET "http://localhost:9201/_cat/indices/*?v&s=index&pretty" -s | cut -d ' ' -f1 | grep --color=never "yellow")
TEST="IS INDEX HEALTH YELLOW"
if [ -z "$BADSTATUS" ]; then
    echo "$SERVICE $TEST: PASS"
else
    echo "$SERVICE $TEST: FAIL"
fi
```

Figure 5. Test Case Code #2

Figures 4 and 5 show how we went about creating some of our tests.

2.7.7. Test Results

Initially we had quite a few test failures for both integration testing and unit testing, but eventually we were able to get all our test cases to pass.

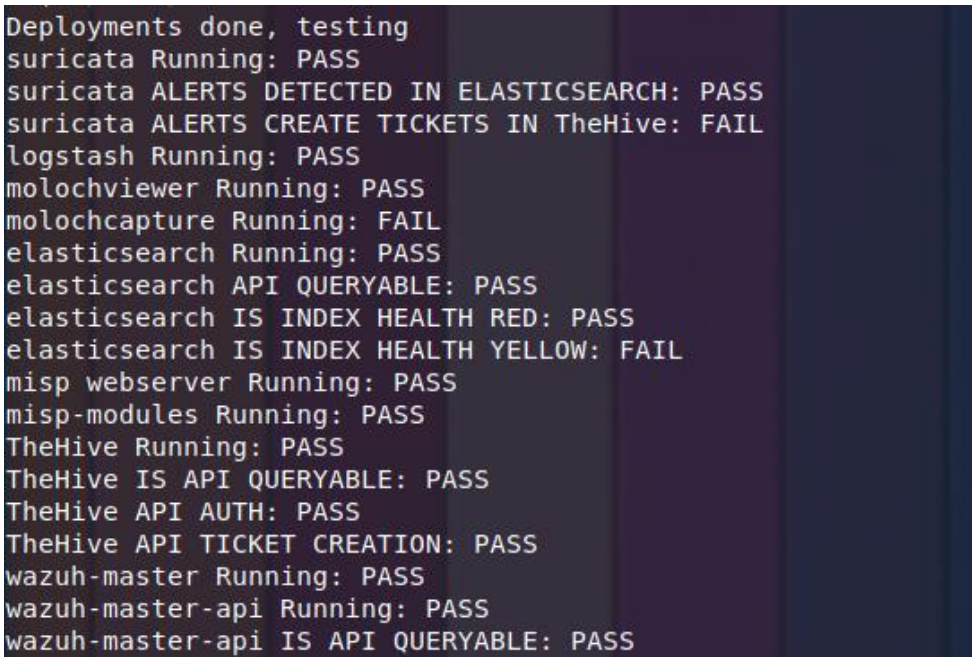


Figure 6. Example Test Results

As you can see in Figure 6, initially we had quite a few test failures for both integration testing and unit testing, but eventually we were able to get all our test cases to pass.

2.7.8. What we learned during Testing

Testing our deployments highlighted a few bugs that were not found during development, but the most valuable results that we received were from the integration tests. Our integration tests helped to identify an issue with our ability to create tickets in theHive, and an issue with index status of theHive's elasticsearch index. By running the automated tests, we were able to find and quantify a lot of bugs that were major issues in our system, and

eventually were able to fix those bugs.

2.8. Budget

Our goal from the start was to make this project using free and open source technologies that would allow us not to spend any money on the development of our tool. Due to this, the budget remained constant throughout the entire duration of our development.

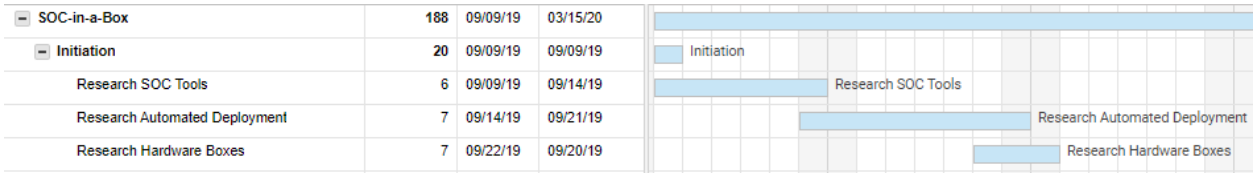
Original budget: $0

Final budget: $0

2.9. Gantt Chart

Given the grand scale of this project our time and progress must be monitored carefully if we intend to have our end product ready for market. As shown in Figure 7, our Gantt chart has a detailed timeline of the development of SOC-in-a-Box. Overall, we expect the entire project to take roughly 8 months before we are ready to sell to our first client. See Figure 8 for the Work Breakdown Structure.

2.9.1. Initiation Phase



2.9.2. Design Phase

2.9.3. Development Phase

2.9.4. Testing Phase

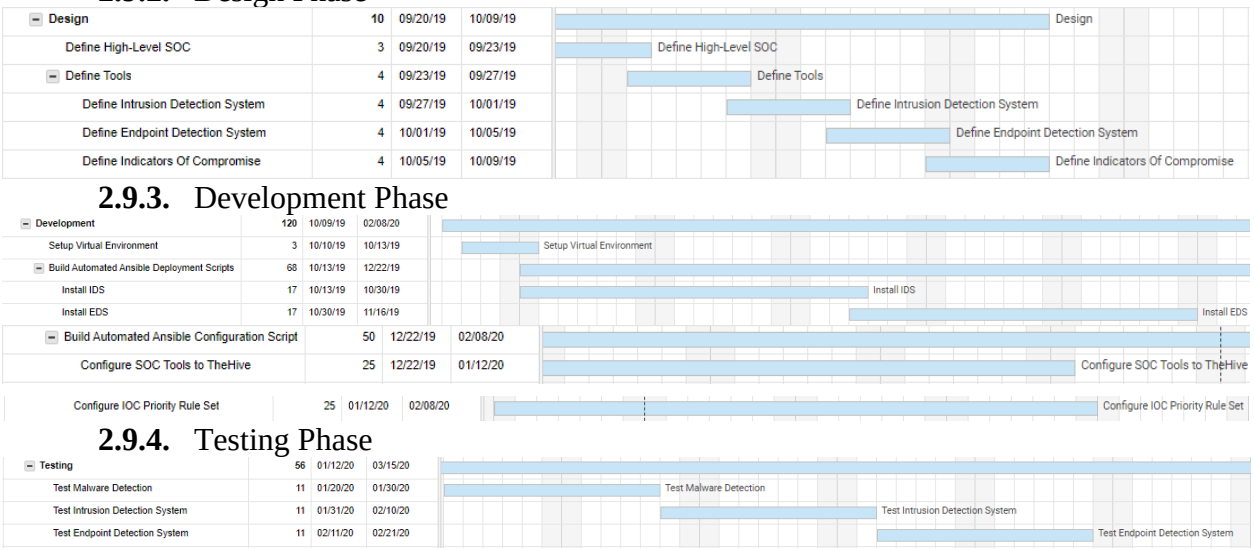


Figure 7. Gantt Chart

We were able to hit all our project plan deliverables that we set out on Figure 7.

2.10. Work Breakdown Structure

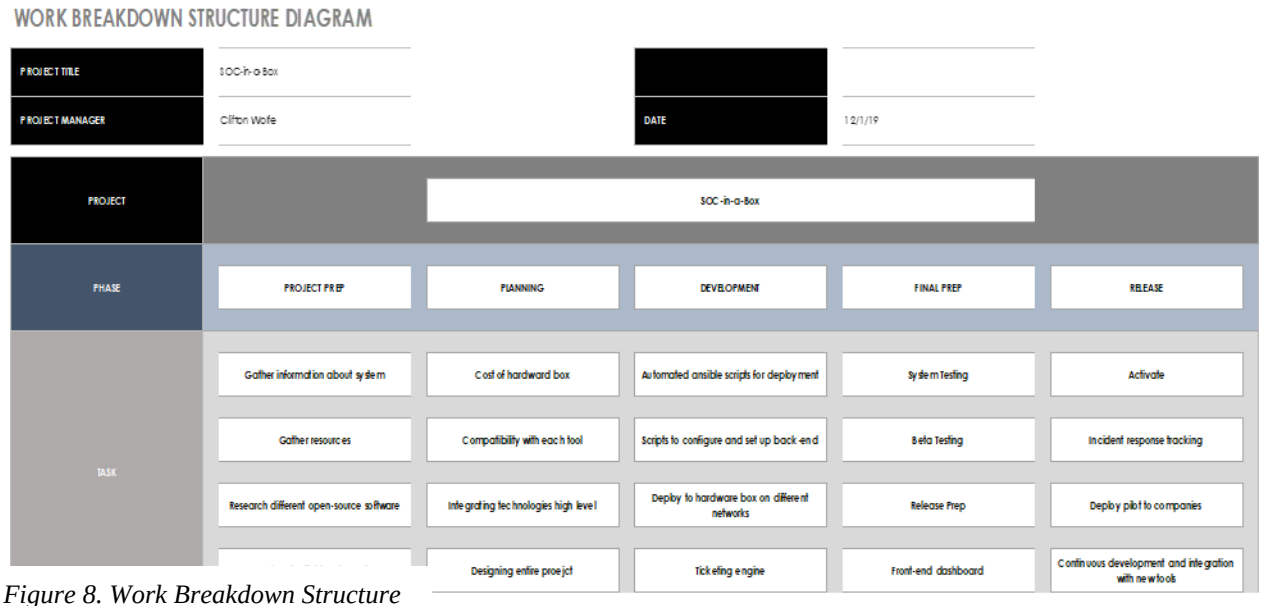


Figure 8. Work Breakdown Structure

The Work Breakdown Structure, Figure 8, helped us divide each section of our project to allow us to focus on each portion of the development.

2.11. Problems Encountered

Tool Breakage - We encountered an issue where one of the tools that we were using to install an ansible role made a breaking change that made that specific role completely unusable. We resolved this by manually installing the role from the specific release that we want.

Package availability - We chose to use CentOS7 as our operating system to make our deployed environment as stable as possible. Unfortunately, some of the software that we are using is not available in a pre-packaged rpm format for this operating system. We resolved this issue by building some of the software from source on the CentOS7 machine.

2.12. Future Recommendations

Our focus was to develop a core Security Operation Center that can easily deploy onto a small server box and scale to be able to monitor small to medium size businesses. We were able to develop the core SOC but had issues along the way with the automated configuration scripts that would tie each tool together.

3. CONCLUSION

3.1. Fall Semester 2019

Overall, the Security Operations Center was a huge learning opportunity. With our progress so far, we have learned how to get enterprise-level hardware up and running, how to effectively utilize open-source technologies like Ansible, and about creating a tool that businesses can easily integrate into their already existing IT infrastructure. Our team has put in a lot of work and man-hours into this project. We are planning on trying to put this on a single machine and are attempting to start a business selling this to small to medium businesses who can't afford a cost-effective cyber security solution.

3.2. Spring Semester 2020

Overall, our goal of developing an enterprise level Security Operations Center has been successful. We were able to deliver on every component we set out to develop. There were many challenges that we faced during development, but we were able to successfully automate our SOC deployment. Moving forward, we have been working with a local data center in Cincinnati, OH to test our project out on a real-world environment. This project has taught us many different skills, from developing a complex product to meeting with customers to identifying and solving real world problems that impact the security industry.

APPENDIX A. ADDITIONAL INFORMATION

ACRONYMS AND ABBREVIATIONS

IT Information Technology
SOC Security Operation Center
VPN Virtual Private Network

PROGRAMMING LANGUAGES USED

Python 3
Ansible's playbook/roles based on YAML
Bash and POSIX compliant shell scripting languages

APPENDIX B. REFERENCES

[1] “2018 Data Breach Investigations Report”, Verizon, April 2018. https://enterprise.verizon.com/resources/reports/DBIR_2018_Report.pdf

[2] “2019 Data Breach Investigations Report”, Verizon, April 2019. https://enterprise.verizon.com/resources/executivebriefs/2019-dbir-executive-brief.pdf

See Figure 9 for our senior design final poster.

Poster:

Figure 9. Poster

We were able to successfully display a simple explanation of our project on a poster,

Figure 9.

SOC-in-a-Box

Jai Singh and Clif Wolfe Team 48

Agenda

- Cyber Attack Story
- Problem
- Solution
- Technologies used
- Demo
- Benefits
- Conclusion

The Story of a Cyber Attack

SMB Owner

Receive an Invoice

Open Attachment

You've Been Hacked!

Problem

- Cyber Attacks are becoming more common
- Hackers target small businesses

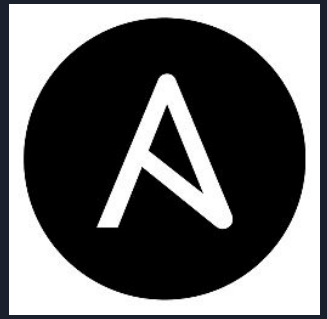
- **78 Days** Median time to detect a breach[1]
- **43%** of breaches involved small business victims[2]
- **$2.4 M** Average cost of malware attack for companies[3]

1. FireEye M-Trends 2019; 2. Verizon Data Breach Investigation Report 2019; 3. Ponemon 2017 Cost of Cyber Crime Study

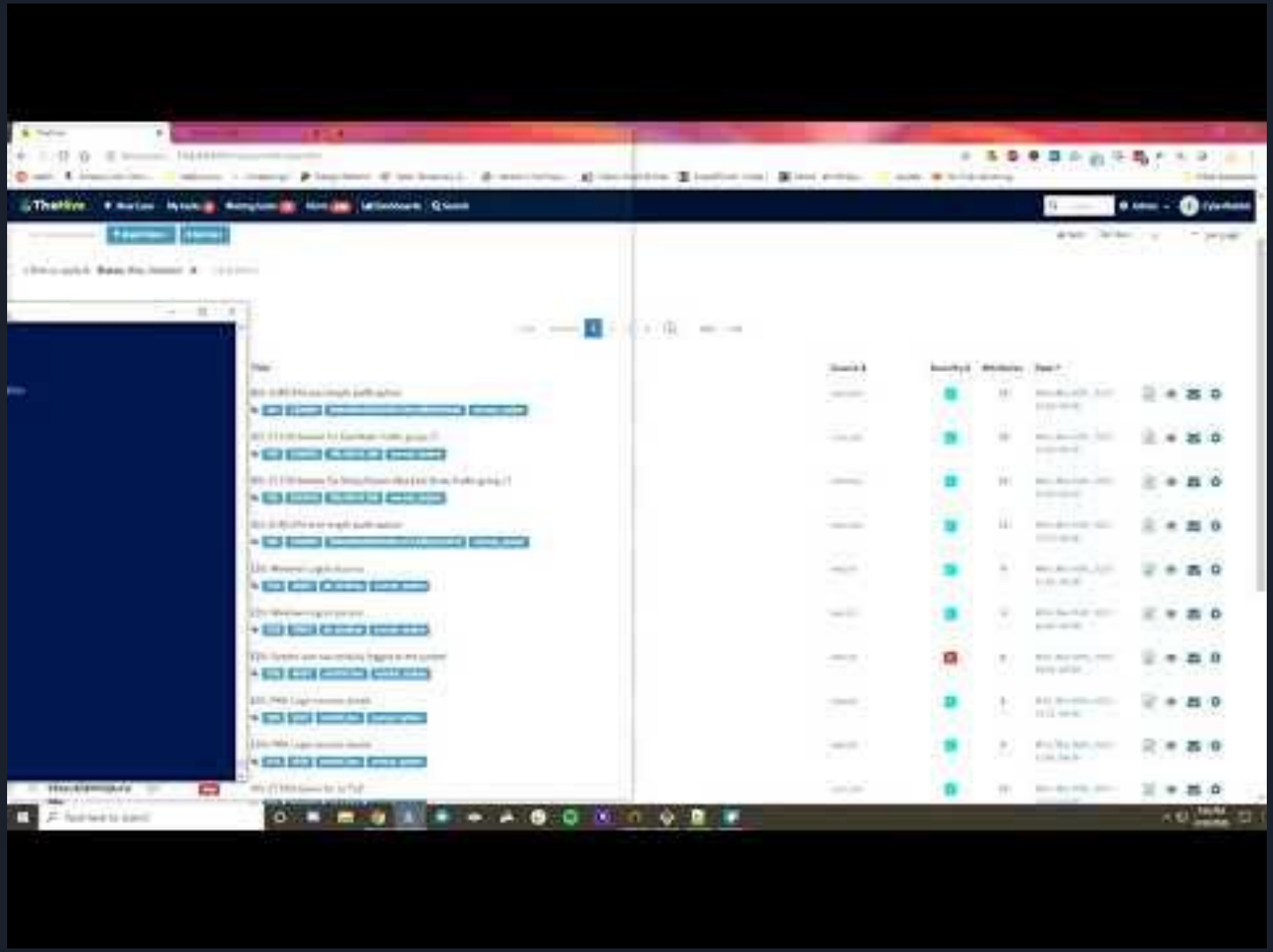
Solution

- Provide small businesses with the tools they need to track and defend against attackers
- Constantly perform network and endpoint monitoring to reduce the time it takes to detect a breach
- Gather Threat Intelligence so that the small business has the same resources as larger businesses

Our Technology



- Easy deployment of apps/services via ansible playbooks
  - Uses ssh/winrm to connect to machines remotely for install/configuration
- Easily searchable data via ELK
  - API is used to facilitate a lot of the inter-app interactions
- Endpoint detection system Wazuh
- Intrusion Detection system Suricata
  - Custom rules
- Full packet capture system Moloch
- Open source threat intelligence MISP
- Alert system TheHive



Benefits

- Easily deployable and scalable Security Operations Center
- Almost no costs to deploy code onto a server
- Cheaper than enterprise level tools yet provides the same benefits

In Conclusion

- Deployment across almost any network with services already integrated
- Services can be integrated with existing solutions with ease
- Automatically pushes the latest malware signatures and indicators of compromise to endpoint antivirus
- Monitors network for known indicators of compromise
- Alerts security analysts if there are indicators of compromise found on your network on on one of your machines