CYBR410 Group Project: Big Deployment

Paul Aguilar
Front-End/Services

Nicholas Burlakov
Services/Orchestration

Nehemiah Fiedls
Logging

Eric Leachman
Orchestration

Thomas Longwell
Defense

Robert Rutherford
Defense

Spencer West
Services

Abstract

CSCD412, Cyber Defense, tasks students with learning a variety of tools and techniques to help defend systems against cyber attacks. The goal of the Big Deployment project is for students to work as a group with each member implementing a unique portion of a larger deployment. The end goal is a secure and easily accessible web server with an aesthetic front end, useful services such as IP to physical location and an IP to weather converters, a robust defense to deter attackers, and comprehensive logging to track all traffic on the server.

1 Introduction

This project was made to show our understanding of various cybersecurity related concepts. By deploying a complex web server, we were given an opportunity to display the knowledge we gleaned from the course, and to give our Cyber-410, Cyber Operations class a chance to demonstrate their own skills by attempting to break it. This project was designed to have groupmates work together to design and implement a full deployment. Each teammate was assigned a specific piece of the deployment as their responsibility and each section of this paper relates to a section of the deployment. Front-End is in charge of creating the user interface of the deployment as well as the web server that will host the services. The Services team then implemented a URL to physical location feature as well as a URL to the current weather forecast converter. Behind the scenes the Defense team integrated an IP blocking honeypot, an easy to access fake deployment with false services [8] to deter attacks, while Logging used a Sysdig script to keep a record of all traffic. This was handed to Orchestration, who spun up the servers on the designated IP assigned by the professor.

2 Related Work

In a basic connection between a client and a host, one computer would send a request that would get relayed by a server to the destination computer. In a proxy environment, the proxy sits in the middle between the client and the server, relaying the requests to the server on the client's behalf. One defense implementation used in this project is a reverse proxy. Unlike a forwarding proxy, which forwards network traffic on your behalf, a reverse proxy forwards traffic to a specific destination, usually you. They are positioned in front of web servers and forward browser requests to those servers. This is typically done for a few reasons, including load balancing, cyber defense, caching, and even SSL encryption [6]. A load balancer is used to distribute traffic among several servers, preventing direct traffic from overloading any given one, like a mail room. In terms of defense, it obfuscates a client's actual IP address, making it a more difficult target; and ideally, the reverse proxy is a third-party that has the infrastructure to defend against attacks.

For this project, rather than use a third-party company to act as a reverse proxy, a software tool called NGINX was employed. NGINX is a free, open-sourced web server released in 2004 that can be used as either an HTTP, reverse proxy, mail proxy, and generic TCP/UDP proxy server. It is popular and dependable enough to be used by major companies like Netflix and Dropbox. In fact, it is one of the top four most used services, and incredibly popular in the Docker community as the most commonly deployed technology in docker containers [2]. For this reason, it made it an easy choice when looking for a technology to act as the project's reverse proxy that could be easily incorporated into our Docker implementation.

Honeypots are widely used tools that can provide invaluable data when in a defensive position. In general, they're decoys that lure attackers in to allow a defensive team to study the incoming attacks. In this project specifically, the network service daemon OpenCanary has been implemented [3]. Major benefits of this implementation are its extremely low resource requirements and the easy extensibility. It can be a good way to get IPs scanning networks, as the only case it should be discovered is if someone is scanning the entire network searching for vulnerabilities. Use of this stemmed from a want to diversify the implemented defensive techniques.

By incorporating OpenCanary it has allowed greater surface from which logs can be collected which is extremely valuable for an evolving defense. Additionally, malicious IPs can be identified more regularly, presenting the opportunity to create IP blocking rules in the future.

Due to the influx of Ransomeware attacks in recent years [1] we opted to focus on increasing the speed of our system's deployment while also reducing the technical knowledge required to establish a working system from scratch. To this end we elected to use Docker Containers and Docker Compose. This gives us the benefit of having preconfigured machines that will deploy in a preconfigured manner, drastically decreasing the time from no service to live service operations.

3 Threat Model

The threat model for this group project involves simulating a real-world attack which will be conducted by students from the cyber operations class. These students will attempt to breach the security of the deployed systems. The primary aim is to generate network traffic that the defense class can log and analyze allowing for demonstration of understanding attack patterns and defensive measures. Currently, risk assets include the web server, which hosts the user interface and services, and the Flask server, which manages API requests. The primary attack surfaces are network ports, the web applications, honeypots, and user authentication. Anticipated threats include the use of Nmap to initially enumerate the server. The information gained will likely include open ports and services, as well as running services and their versions. It's also very likely that we will see brute force or credential-stuffing attacks. Each piece of the system that pertains to defense and infrastructure will be explained further below.

As mentioned in the preceding paragraph, anticipated attacks are basic Nmap scans that will discover ports and service information. It is also expected that attackers will search for weak passwords for SSH and possibly attempt password brute forcing through a tool like Hydra. We also expect our web server to be searched using tools like dirbuster or gobuster and manually searched and attempted to be exploited. A detailed description of testing performed on defenses described in the defense section 8 as well as other expected attacks above are included in the Evaluation 9 and Results 10 section below.

4 Orchestration

The orchestration for this deployment concerns itself with three key areas. The first area, services, have been designed utilizing docker containers and docker compose for simple build up and tear down. Our docker compose solution utilizes two docker images. The first container utilizes Docker's NGINX image. NGINX was chosen as our webserver for its portability and ease of configuration as a reverse proxy. Our second container runs a flask server. The flask server allows us to run web services locally, which are then accessed by NGINX to serve to clients.

The second area, defense, utilizes OpenCanary and iptables. OpenCanary serves as a honeypot, a target designed to waste the time and enumerate the techniques utilized by would-be aggressors. The command-line program, iptables, is a series of rules and conditions used to control traffic control into and out of a machine. OpenCanary is deployed as a docker container, though the complexity in building it warrants its own separate orchestration. To accomplish this, we utilize some basic bash scripting to automate the necessary commands to run OpenCanary with the configuration and services required. To supplement this, we issue some iptables commands as part of this bash script. These rules filter ports and drop traffic that attempts to enumerate OpenCanary and its ports.

Our final area of focus is automation. By automating the setup of our deployment, our services and defenses can be deployed quickly and by a greater number of individuals, even those not intimately familiar with its inner workings. The use of docker containers and docker compose also allow our deployment to be mobile, only requiring prospective users to install a small number of programs (i.e., docker, docker compose, git). We further simplify this setup for users by automating the installation of these programs from within a bash script.

5 Front end

The front end is designed around a few simple HTML template pages, stylized with a CSS style guide. There are three main templates and a variation on two – the first template is the index, a home page that is repeatedly linked on some blank links in the navigation bar. There are also two pages dedicated to the address and weather API's. These pages have two versions, a search and an output template. The search template uses a simple form for the user to input a URL. The output template displays the results from the API call. These files are hosted via a flask server. [7]

The flask server uses the render_template and request objects to function seamlessly with the HTML and CSS. Following traditional directory organizations, a templates and static directory are used to hold those files for flask to access.

```
/app
     - fServer.py
     /templates
        - index.html
     /static
logo png's
/fonts
       /styles
             - style.css
```

Flask's render_template is used to display the correct HTML template in any given scenario, either from the hostWeather method or directly in the HTML links. The request object is used to handle the form submissions so that the user provided URL's can be used by the API methods to generate the correct output. Both hostAddress and hostWeather use a similar method to pass the URL correctly depending on whether a POST request was made from a form submission, via requests `request.form` method or directly from the URL query string. From there the API methods are used to gather the data, which is the outputted using the render_template method and displayed with in the output template. While the host methods route to the correct output page, the remaining routes direct a user to the index or the search pages.

6 Services

This web application incorporates two API's in a flask server. One service takes a URL and outputs the address of the registered owner based on the whois [5] registry. The other uses that address and another weather API [10] to output the weather of that given location. It also uses a list to cache the data of visited sites, to improve performance. To do all this, the API is broken up in to multiple methods, which ultimately called by the hostWeather and hostAddress methods when a form submission is made on the webpage.

The first method that is used is `getIP()`, which first strips the incoming URL of any whitespace and then uses the socket library's `gethostbyname()` method to save the IP. This method is then used by both `getWeather()` and `getAddress()` in order to find the related information. `getAddress()` uses the subprocess' `getstatusoutput()` method to run the `whois` commandline tool to search for the registrar information based on the provided IP. It then splits the incoming tuple by a newline and parses out the different address fields, combines them and returns a whole address [4].

The getWeather method then passes that address in to a geocoding API to find the latitude and longitude of the address, which is required by the weather API that is used. The geocoding call returns a json file, that needs to be parsed and split between x and y coordinates correctly. The request's get method is then used to make an API call to weather.gov [10] via the command line again. This outputs another json file that is then parsed to find the correct URL that contains the weather information. The get method is used once again on that forecast URL, which outputs the final json file, this is correctly parsed and returned as the weather for the given IP.

The remaining methods handle the caching, either directly adding to it based on the correct list or traversing it to search for the given key which returns true or false. The host methods use this to first check the cache and then output the value based on the given key or add the key/value pair to the given cache if it isn't found. To summarize, this script creates a Flask web server that provides endpoints for IP address lookup and weather information retrieval, caching the results to improve performance. It also demonstrates how to integrate external APIs and system commands into a Flask application.

7 Logging

The bash script our group made automates the monitoring of a system by capturing system activity using the sysdig tool every 30 minutes. Each session stops when the data file reaches about 20 MB or after 4 hours of total runtime. When the bash script is done logging we of course need to go over the logs and make sure everything is okay and that no one has tried anything to breach our resources (website, security, etc). The script checks if each generated file exceeds a size limit of 20MB and, if so, runs a specific program designed to print the scaps for the allotted memory during that time period so we can check them. This can be useful for system monitoring, security analysis, and troubleshooting.

The script helps manage data capture efficiently but requires careful consideration regarding storage, privacy, and the potential impact on system performance. Another layer to add to logging is the infamous trouble with obtaining multiple scaps. When testing the bash script it came to my attention that you can only have one of the same scap, regardless of how many times the bash script runs. This means that if you were to manually delete a scap file and leave it in the trash all that data is still being accounted for, which in turn makes it so you cannot make multiple copies of a pcap if you were to restart the bash script/service. Now the nice part about all of this is that we can simply just delete it once it is done and the bash script can go about doing its job. Looking back at it all of this makes more sense in term of making sure scaps arent something you can take lightly because the task they have is important and very useful. In conclusion, logging has its ups and downs but the pros outweigh the cons in a majority of ways

8 Defense

In this section, we will discuss the implementation of defense for our group project in detail. Our group's approach to defense includes simple defensive measures and a two part system, consisting of a honeypot and iptable rules to control traffic which will be explained below. The simple defensive measures used are only allowing users with ssh keys to connect, other than a single account that has sudo permissions and uses a strong password. The goal of the honeypot is to provide an avenue for investigation that distracts from the actual critical infrastructure. Once this unauthorized traffic occurs, the rules that we created drop the connection. This slows down or may entirely stop unauthorized users who have the intent to scan the server and try to connect or exploit it. With fake services on the honey pot, their attention will be

diverted to multiple distractions. This defensive approach assumes authorized users will know what is in scope and not try to connect to or scan the honeypot.

The honeypot we chose for this project is opencanary [9]. Opencanary was selected because of our familiarity with it, as well as its simplicity and interoperability with the rest of the group's implementation. Opencanary integrates into the system easily because it's built with docker, and allows our group to integrate it with our other dockerfiles seamlessly. This allows for quick configuration, and once the configuration file is set, building and running the dockerfile is quick with simple integration. The ease of deployment works perfectly for the attacks that we anticipated that were covered in the Threat Model in section 3. This will allow transferring of the honeypot to a new server to be easily handled with a single script. The ports we opened using opencanary are `8444, 8881, 33061, 2201, 63791, 3391, 50601, 1162, 1124, 1070, 8002, 2301, 14331, and 5901`

Traffic rules set by iptables include a very basic approach to discouraging traffic on the machine. The rules we have set up drop incoming traffic, preventing any connection on them. It was made easily extensible as well, with new ports simply needing to be added to a block list. All ports listed in the preceding paragraph have these rules applied to them. Opencanary has enabled a large volume of ports that seem promising to exploit. After an initial scan, the attacker would recognize these and begin to press on these attack surfaces. Unbeknownst to them, all traffic going to these ports will be dropped with no reason or message given to the attacker. Using DROP instead of REJECT provides the client no insight into why their connection is unsuccessful. On their side, it just appears that their request is hanging forever. This simple yet effective measure is a major piece of our defensive strategy. Opencanary's large volume of unnecessary ports in tandem with these rules will cause major disruptions to any penetration attempts, guaranteeing a strong level of security for the system.

9 Evaluation

To evaluate our services, we opted to focus on the usability of our platform and its services as well as its security. Focusing on the usability aspect allows us to approach from a user-perspective. We are concerned with bugs, ensuring products are running appropriately, and ensuring that the interface is not broken. While focusing on security related aspects, we focus our efforts based on our threat model.

For usability and service evaluation, we approached our deployment from the front end. Input fields were tested for usability and for improper input. All links were tested on all page redundantly to ensure consistent usability. All services were tested with proper and improper input.

Security evaluation of our deployment was done by engaging with our threat model to determine what knowledge and tools are available to our adversaries. Knowing these key points allow us to accurately emulate the types of enumeration and attacks we might see on our systems, allowing us to patch security vulnerabilities. We evaluated our systems from all points that an aggressor will have access to, including the front end and any services revealed via enumeration.

We first enumerate the IP address the deployment is located at, as we know that the aggressors in our threat model will likely initially approach the system via this method. After establishing that the machine is online utilizing Internet Control Message Protocol (ICMP) pings, we then enumerate services and ports using the network mapper Nmap. After recording the results of Nmap, we then perform enumeration with another CLI program, Gobuster. Gobuster will map out the directory structure of machine giving us insight into what attack vectors might be exposed to any aggressors.

With enumeration completed we know open ports, services offered, service versions, and directory structure, allowing us to probe more deeply into individual components for vulnerabilities. This included probing SSH for vulnerabilities and attempting access to our SQL server.

Investigation into some of the ports causes lots of confusion. Any non conventional port access is met with hanging requests when using curl or wget to access the web page. Further attempts to access these in a browser are met with the same results. While Nmap does show that those ports are open, it's unable to detect exactly if there's something blocking traffic.

Our group also evaluated our logging by purging old logs that aren't necessary, as we wanted to clear the logs to clarify what traffic is produced by the expected attack referenced in the threat modeling section 3. Once the logs were purged our group ran a Nmap scan to generate traffic that would be captured using our logging. The Nmap scan used was `sudo nmap -T4 -A -sC 10.102.67.18`.

10 Results

After combining all components into a final working network with a web app with multiple services, the next stage was to analyze each of the services. Analyzing the services (otherwise known as enumeration) is the first step that a malicious actor might take. Using surveillance tools such as nmap, gobuster, or a ping sweep to map the hops of the system to checking what ports are available. One thing to note is that the tools listed are not exclusively used by malicious actors but are used by admins to provide better Cybersecurity solutions which is what the researchers of this study are applying.

We utilize nmap, a command line tool to determine what ports and services are currently on offer on a system, with the `-sC` and `-sV` flags which, respectively, will gather detailed information and the version that a service is operating on the network. In the study, the services are revealed as well as what ports those services operate on. For example it can be observed that there is an OpenSSH service operating on

port 22 using tcp. A Defensive measure to protect this service would be to look at the CVE (Common Vulnerabilities and Exposures) for OpenSSH. Looking at the CVE for OpenSSH, if the service is the most updated version, then admins might try to defend the encrypted key protocol to prevent messages from being deleted.

The ports that were revealed through an nmap scan also show that there is little security in protecting routing. This was observed as typing the port number at the end of the url such as ":9000" would reveal a webpage that had a scramble of letters and characters. This vulnerability could be remedied through proper routing controls so that url redirection attacks are minimized. In addition to eliminating security threats, testing was implemented on SSH for weak passwords. Passwords that were identified as weak were changed.

The iptable rules proved relatively effective in minimizing what could be discovered on the server. While they are found to be open with an nmap scan, they do drop all traffic that attempts to connect to them. By adding this element to the server we provide more opportunity for distraction and make it significantly more difficult to find something of value to an attacker.

After testing our logging we found that former logs needed to be purged to make traffic from the expected attack clearer so we did so. After running the Nmap scan to generate traffic to test as referenced in the Evaluation section 9 we found that our logs had produced multiple logs which when analyzed had traffic meaning logging was working as intended.

11 Conclusion

This project provided us with a valuable opportunity to learn and apply various cybersecurity concepts in a real-world context. By working together to deploy a functional web server, we gained a deeper understanding of real-life deployments.

Our evaluation process focused on both usability and security. From a usability perspective, we ensured that input fields and links were tested thoroughly for proper and improper inputs, guaranteeing a smooth user experience. For security evaluation, we engaged with our threat model to emulate potential attack scenarios, allowing us to identify and patch vulnerabilities.

The threat model anticipated attacks such as port scanning, service enumeration, and brute-force attempts. Through tools like nmap and Gobuster, we were able to map open ports and services. This enabled us to understand the attack vectors and reinforce our defenses accordingly. The deployment successfully withstood these simulated attacks, confirming the robustness of our security measures.

Overall, the project demonstrated our practical application of cybersecurity principles and highlighted areas for future improvement. It provided us with a deeper understanding of the challenges involved in real-world cybersecurity deployments, and prepared us to tackle similar challenges in our future professional careers.

12 Acknowledgements

We would like to give extra thanks to Alex Moomaw and Lewis Thomas for helping us and giving advice on our implementation.

References

[1] Terrence August, Duy Dao, and Marius Florin Niculescu. Economics of ransomware attacks. *Available at SSRN*, 2019.

[2] Multiple Authors. Wikipedia - nginx, May 2024.

[3] Thinkst Canary. Opencanary docs, June 2024.

[4] United States Census Bureau. Geocoding api, Aug 2023.

[5] Leslie Daigle. WHOIS Protocol Specification. RFC 3912, September 2004.

[6] Cloud Flare. What is a reverse proxy? | proxy servers explained, Jan 2024.

[7] Miguel Grinberg. *Flask web development*. " O'Reilly Media, Inc.", 2018.

[8] Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder. A survey on honeypot software and data analysis. *arXiv preprint arXiv:1608.06249*, 2016.

[9] thnkst. Opencanary repository. *https://github.com/thinkst/opencanary*, 2024.

[10] Past Weather. National weather service. *URL: https://www. weather. gov*, 2009.