Chris Cohen

HONEYPOTS

Creation, Deployment, and Analysis

Purdue University

CS422 - Final Project

April 29, 2020

1 Problem Definition

A honeypot is a term used in the cyber security field describing a network entity that purposefully lures attackers. It is particularly useful for frustrating and distracting attackers, since it is typically a dead end with no access to any real, desirable information. Attackers are constantly scanning the internet for vulnerabilities, and can potentially shut down an entire network if given the opportunity. A honeypot is an extra layer of security for a network, and even though it may not capture every attacker, it can take a good portion of malicious traffic away. One common use case for a honeypot is to set up some sort of monitoring, and collect information on any attackers. Analyzing this information can be valuable to help improve network security in other places that need it.

2 Motivation

I chose this project in order to learn more about my field of study, cyber security. As I've been applying to internships over the past year, it's been increasingly evident to me that this field is tough to break into. When a company looks for an employee to help secure their important assets, an inexperienced student is their last choice. Employers think highly of students who have completed personal projects, so this was the perfect opportunity for me to stand out. I knew that I had to start with something simple so that I didn't overwhelm myself, and creating a honeypot seemed like the best idea. The mention of a honeypot on the project rubric intrigued me - it seemed so simple, yet effective if done right. I already had a HTTP web server from the first lab to build off of, so I wouldn't be spending much time on the basics. Additionally, in the future, I could easily convert it into a normal web server for my personal needs. Going into this project, I realized that this would be tough, as hosting a web server on your home network is almost never advised. I welcomed the challenge - I knew that I would run into problems and inevitably learn much more than I set out to.

3 Related Work

3.1 HoneyHTTPD

One particular project that inspired me is *HoneyHTTPD*. The idea of this project is that it makes it easy to set up fake web servers and record requests. It proved helpful when I wasn't sure how to efficiently structure my server code, and when I wasn't sure how to generate SSL certificates for the fake web server. Hartman [2] also used object-oriented programming to allow for multiple types of servers to be used. It's a cool concept, but I couldn't think of a situation where it would be useful, so I decided against implementing it. It also didn't seem very convincing, since there was nearly no data to access and distract an attacker. To build off of this, I decided to implement a single server that contained lots of data for a potential attacker to browse. There is also functionality to deploy multiple honeypot servers at once, each listening on different ports. Originally, I attempted to use this idea, but ultimately decided against it since my original server, a Raspberry Pi 3 B+, suffered in performance. Overall, this project was a very good start for brainstorming, and gave me multiple good leads to start testing my honeypot.

3.2 StackHoneyPot

Another project that I found very interesting is called *Stack Honeypot* - a honeypot specifically for detection and trapping of spam bots. When I originally looked up the term "honeypot", my first thought was to use one for spam detection and prevention. One of the main goals of a honeypot is to redirect malicious traffic from the real server, preventing any attacks possibly causing a break in service. This honeypot detects if a field in a response form has been altered. A bot likely wants to fill out all fields given to it, so if the dummy field (invisible to a normal user) is filled out, it will be obvious that the client is a bot. If detected, it sends the bot to a dead end blank page. Although I didn't implement this exact

idea, I ran with the idea of an automatic blacklisting functionality. I get into this more in the next section, but the basic overview is that if a client connects 10 or more times in 3 seconds, the IP associated will be banned from connecting. Admittedly, this may not be the best approach, since the attacker could move on and find the actual server, but it may frustrate an attacker enough to stop them from trying. I don't want spam requests to slow down the honeypot, either.

4 Completed Work

4.1 Creation

4.1.1 Server Response

In order to create an effective honeypot, I had to create a reasonably robust web server that looked convincing. This meant that I had to implement significantly more features on top of my original HTTP server for the first lab. The final product implements 14 different response codes, each serving a real purpose on the server. The most important of these are the 500 level response codes, informing the client of any internal server errors. I realize that this server will inevitably have bugs, so I make sure that the server cannot crash due to any exploits.

4.1.2 Browsing

Although the primary targets for this server are bots, there is always the chance that an actual person comes across the site. In this case, I wanted to make sure that it seemed like a real server, so I implemented interactive directory browsing. For each directory, the server displays HTML that give links to each other file and directory present, along with icons, each file's size, and last modified date. I cite aesthetic inspiration from Purdue's Spring 2020 CS354 directory browsing page. The server also supports 10 different file extensions,

so each file that is present is able to be opened with the click of a mouse. This functionality also made the server a lot easier to test manually.

4.1.3 SSL Certification

The unit on Computer Security that we recently learned about inspired me to try SSL certification. Following the instructions present in *SSL All The Things*, I was able to set up HTTPS, and used a self-signed certificate for testing. When the time came to deploy, I could no longer use the self-signed certificate. Luckily for me, I had already purchased a domain in the winter, so it was relatively simple to map the IP that I was serving to a subdomain. Using *Let's Encrypt* [5]'s "Certbot" script, I was able to verify ownership of my domain and receive a legitimate SSL certificate. After setting up the certificates and private keys for my server, I made sure to force a 301 redirect on any clients that attempted to connect without HTTPS.

4.1.4 Logging

In order to know anything about how successful this project was, I needed to record connections made to the server. Using my very base-level knowledge of MySQL, I was able to set up a local instance that the web server logged to. Each listing contains the timestamp, client IP, client port, host name (if IP is known), and request given to the server. This simple step allowed me to analyze incoming connections at the end, and tweak the structure of the server when needed.

4.1.5 Blacklisting

At this point, there was nothing special about the honeypot. Inspired by Hochstrasser [3]'s *Stack Honeypot*, I decided to implement blacklisting. I made a script that scanned the local MySQL database, checking for any repeat requests made. I had nowhere near enough time to analyze each individual connection for malicious activity, so I decided to just implement

spam protection. If a client requests the same URL 10+ times within 3 seconds, that client will be IP banned. Even though a honeypot is supposed to absorb malicious traffic, a DDoS attack would certainly slow it down and decrease the effectiveness. Due to this, spam protection was necessary.

4.1.6 Honeypot Structure

A honeypot has no purpose without attackers. The idea is to make it look enticing to any malicious clients, so I made it look like a misconfigured server. The root directory consists of a fake web server, README, helper functions, and SSL certs/keys. This gives the illusion that the server is serving up the parent directory of the actual server root (titled 'htdocs'). The exposed key files should be especially tempting to attackers.

4.2 Deployment

4.2.1 On Home Network

My intent was to deploy the honeypot by Saturday, April 25th, giving me just shy of a week to collect data. However, I ran into many more problems than anticipated. The source code was completed by Saturday, as planned, but deploying it took a bit of tweaking, mainly concerning SSL certification. After all issues and bugs were seemingly ironed out, the initial test run was deployed on Sunday, April 26th. Within hours, my router had been taken down. I was aware of the risks of exposing your network to the public, but had no idea that it would be this much of a pain. Three or four internet outages later, after numerous failed attempts to better secure my network, I conceded.

4.2.2 On Amazon Lightsail

I decided instead to host on Amazon Lightsail, a service that provides a VPS [1]. I chose this because it offered the first month free of charge, and gave me a full-fledged virtual Ubuntu

[1] Virtual Private Server

18.04 LTS server to work with, nearly identical to my home setup. Originally, I was skeptical. The whole reason that I attempted to set up my honeypot on my home network was because I thought that a home network would seem more enticing to attackers. Who would try to attack an Amazon server? Luckily, I got the same amount of traffic, if not more. After successfully setting up the VPS, I collected data for the rest of the week. I started out with my originally planned HTTPS server, but I noticed in the active server output that multiple clients attempted to connect with HTTP, were rejected, and never attempted to reconnect. Given the short period of time that I had to test, I wanted as many data points as possible. I saw no drawback in accepting HTTP connections as well, especially since I'm not sending anything important or hosting the website on my own network. From Wednesday, April 29th and onwards, my server defaulted to HTTPS, but any HTTP requests sent would also be served, giving me more data points to work with.

4.3 Analysis

5 Conclusion

References

[1] Chris Cohen. *HTTPS Web Server Honeypot*. 2020. URL: https://github.com/cohenchris/honeypot_webserver.

[2] Jacob Hartman. *HoneyHTTPD*. 2018. URL: https://github.com/bocajspear1/honeyhttpd.

[3] Christoph Hochstrasser. *Stack Honeypot*. 2014. URL: https://github.com/CHH/stack-honeypot.

[4] Markus Holtermann. *SSL All The Things*. SpeakerDeck. Sept. 16, 2016. URL: https://speakerdeck.com/markush/ssl-all-the-things-pycon-nz-2016?slide=18.

[5] *Let's Encrypt*. Internet Security Research Group. 2020. URL: https://letsencrypt.org/.