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 cyber security describing a network entity that purposefully lures attackers away from some more important part of a network. Since it is typically a dead end with no access to any real, desirable information, it is particularly useful for distracting and frustrating attackers. People are constantly scanning the internet for vulnerabilities, and can shut down an entire network if the opportunity is presented. A honeypot is an extra layer of security, and even though it may not capture every attacker, it may take a good portion of malicious traffic away. One common use is to deploy a bugged, seemingly important web server to collect information on visitors. Analyzing this information is useful as a test bench for identifying network vulnerabilities before they can be exploited to cause real damage elsewhere.

2 Motivation

I chose this project in order to learn more about my field of study, cyber security. As I've applied to internships over the past year, it's been increasingly evident that this field is tough to break into. When a company looks for an employee to help secure important assets, an inexperienced student is their last choice. Employers think highly of those who have completed personal projects, so this was the perfect opportunity 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 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 any difficulties I ran into would help me learn much more than I set out to.

3 Related Work

3.1 HoneyHTTPD

One particular project that inspired me is *HoneyHTTPD*, which 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 self-signed SSL certificates for testing. 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 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,

deployed on a Raspberry Pi 3 B+, suffered in performance. Overall, this project was a very good start for brainstorming, giving me multiple good leads to start testing.

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", this sort of application was my first thought. 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 exactly, 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 associated IP 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, so it adequately fits my needs.

4 Completed Work

4.1 Creation

4.1.1 Server Response

In order to create an effective honeypot, I had to craft a reasonably robust, convincing web server. This meant that I had to implement significantly more features on top of my original HTTP server. 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 or attacks.

4.1.2 Browsing

Although the primary targets for this server are bots, there is always a chance that an actual person visits. 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, file size, and last modified date. The server also supports 10 different file extensions, so each file that is present can be opened with the click of a mouse. This functionality also made the server a lot easier to manually test.

4.1.3 SSL Certification

The recent unit on Computer Security inspired me to try SSL certification. Following the instructions 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 just 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. Most real-world servers do this, so it only seemed right.

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 in the development process, 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, rather than ban, malicious traffic, a DDoS attack would certainly slow my server down. I needed it to be running in top shape at all times, so spam protection seemed 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 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. Following three or four internet outages later, and numerous failed attempts to better secure my network, I conceded.

4.2.2 On Amazon Lightsail

My next move was to host on Amazon Lightsail, a service that provides a VPS (Virtual Private Server). I chose this because it offered a month of hosting for free, and gave me a full-fledged virtual Ubuntu 18.04 LTS server to work with, nearly identical to my home setup. Originally, I was skeptical. The whole reason that I 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? Surprisingly, I got the same amount of traffic, if not more. After successfully setting up the VPS, I collected data for the following few days. I started out with my originally planned HTTPS server, but I noticed in the live 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

Going into this project, I didn't expect much data to work with, but was pleasantly surprised at the amount of traffic that my server received. In just a few days, I received around NUMTRAFFIC requests! The final server logfile (unfortunately missing some days due to restarting the server) resulted in the following:

num	timestamp	ip	request

Only use num, timestamp, ip, request (without HTTP/1.)

SELECT num,timestamp,ip,request FROM webserver-logs.logfile WHERE NOT ip='73.103.88.65' INTO OUTFILE '/var/lib/mysql-files/logs.csv' FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '/n';

sudo mv /var/lib/mysql-files/logs.csv ./

As you can see, there were quite a few instances where the connection was very clearly a bot that was scanning for common vulnerabilities.

5 Conclusion

If given the chance to do this project again, I would change a few things. After a day or two, it became clear that my method of setting up the server to look vulnerable was not extremely effective. A majority of attackers requested common authentication files, or tried to make the server unintentionally execute code. I would put in a bit more research towards the most common files that attackers are looking for. I would have given myself more time to gather data and restructure the server as needed. Hacking entities on the internet are often bots that look for common vulnerabilities - rarely real people. Less focus should have been put on the directory browsing, since I only had a few instances that made it seem like a real person was on my server. In addition to these changes, I would steer clear of hosting the server on my home network. It only caused me headaches, and gave me less data to work with. The longer that I left the server up, the more active it became, so if I avoided self-hosting from the beginning, I would have significantly more data.

This project made me realize just how many attacks occur on the internet, most of which we don't ever know of. My completely unimportant web server experienced a significant amount of attacks in a relatively short period of time. I can only imagine what a desirable target like the FBI has to deal with. Especially at higher levels, this type of work is risky if you don't know exactly what you're doing. It makes me excited for the future of my career in cyber security. It's an interesting occupation - if you do your job well, nobody knows, but if you mess up just once, significant damage could be done. I have a feeling that this project won't quite end here, maybe stemming into a hobby to beef up my network security. This project has sparked interest in me at a level that school hasn't, and I'm eager to learn more.

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/.