Analysing Attackers and Intrusions on a High-Interaction Honeypot System

1st Mandy Knöchel
Institute for Computer Science
Martin Luther University Halle-Wittenberg
Halle (Saale), Germany
mandy.knoechel@informatik.uni-halle.de

2nd Sandro Wefel
Institute for Computer Science
Martin Luther University Halle-Wittenberg
Halle (Saale), Germany
sandro.wefel@informatik.uni-halle.de

***Abstract*—Attackers and malware are a major threat to the growing number of servers and devices on the internet. Therefore, it is essential to study characteristics of malicious activities which can be used to aid future security mechanisms in finding and preventing these threats. Honeypots are a powerful tool to get insight into current attack techniques, malware and botnets. In this paper, we present our findings from observing the behaviour of attackers on a high-interaction Linux honeypot. We focused on attacks targeting the SSH service and analysed all steps of the intrusions, starting from the initial dictionary attack and leading to the final intrusion executing commands or malware on the honeypot. Further, we present our approach on how to decrypt and analyse the encrypted network traffic.**

***Index Terms*—Security, Honeypot, Malware, SSH**

I. Introduction

Attackers and malware are a major threat to the continuously growing number of devices and servers on the Internet. In order to design and improve defence mechanisms, it is essential to gain insight into current techniques and malware used by attackers. To get an overview of common threats and to research attack methods or malware, honeypots are a powerful tool used by researchers and security practitioners. Honeypots are systems that are intentionally designed to be vulnerable in order to attract attackers and observe their behaviour on the system. Based on the level of interaction possibilities, one can distinguish between low, medium and high interaction honeypots [1]. Low- and medium-interaction honeypots are typically programs that simulate specific services and offer attackers only restricted interaction options. On the other hand, high-interaction honeypots are real systems that offer the full range of capabilities of the operating system and its applications. High-interaction honeypots provide the best insight into an attack, but also pose the greatest risks, as attackers have access to a fully functional server. Thus, additional security measures must be taken to prevent the honeypot from being used to attack other systems.

Though simple in concept, brute force attacks are still one of the most common types of attacks. Here, attackers try a large number of different username/password combinations in order to gain access to a service. One reason is that many devices, especially the growing number of IoT devices, still employ weak or default login credentials [2]. Compromised systems are often used as part of a botnet for Denial-of-Service attacks (DoS), spamming or cryptocurrency mining [3]. Due to its widespread use, the SSH protocol has become a popular target for these kinds of attacks. SSH is used on many systems for remote administration and file transfer. Its encryption makes it a secure alternative to unencrypted protocols such as Telnet or FTP, but also allows attackers to bypass traditional security software like IDS that inspects incoming traffic.

In this paper, we present our findings from observing attackers on a high-interaction honeypot. For our study, we collected network traffic of attacks targeting the SSH protocol to gain insight into current attack techniques and malware. We will show statistics about dictionary attacks, the origin of attackers and which tools, commands and malware are used by the attackers. Further, we used a new approach to decrypt the collected network traffic. This allows us not only to analyse the attackers' behaviour during these attacks, but also makes it possible to use the collected data to train machine learning models in the future.

II. Honeypot Setup

To collect data from attacks we use a high-interaction Linux honeypot that runs in a virtual machine with a Debian Jessie operating system located in our university network in Germany. All incoming and outgoing connections of the honeypot are routed through a monitoring server, that records the network traffic as pcap files as well as controls and restricts outgoing connections to prevent attacks originating from the honeypot. Incoming connections are allowed to pass through unhindered. All data is further analysed on an Elasticsearch cluster. To lure attackers to the honeypot, an OpenSSH server with 5 different user accounts with weak passwords (the root account and 4 unprivileged accounts: guest, test, support, admin) was set up as an attack target. The passwords of the user accounts were changed regularly to attract different attackers to the honeypot.

Due to the encryption of SSH and HTTPS connections, it is usually not possible to inspect these connections on the monitoring server. To address this issue, most other works use modified SSH server [4], [5], modified Linux kernel [6] or Virtual Machine Introspection [7] to record login attempts and activities on the honeypot. However, these approaches make it impossible to decrypt individual network packets. In order

to preserve the correlation between the observed activities and the encrypted network traffic, we used a new approach, which allows us to decrypt each network transmission. This way the collected traffic can also be used to train machine learning models, where labelling the data is essential. For this, we have modified the OpenSSH and OpenSSL source code on the honeypot in a way that allows us to export the session key during the handshake phase of the network transmission. The session key, called shared secret in SSH and (pre-)master secret in SSL/TLS, is the final result of the handshake between the client and the server and is used to encrypt and decrypt all packets during a session. The keys are then sent to the monitoring server as UDP packets and stored inside a database. For HTTPS connections, Wireshark offers the ability to decrypt network traffic given the session key of the connection [8]. Since there is no equivalent function to decrypt SSH traffic in Wireshark, we developed a custom Ruby script that uses the session key to decrypt the SSH session.

III. Experimental Results

In this section, we will present our findings from observing attackers on the honeypot. The honeypot ran for two different time periods from May 2017 to September 2019 and from January 2021 to October 2021. There were over 2 million SSH connections on the honeypot. Only connections where at least the client banner was received were taken into account, thus excluding port scans. Overall, 92% of the connections made a login attempt on the honeypot, with the majority of those (88.7%) trying only one password and the maximum being 15 password attempts in one connection (observed in 2 attacks). 249,806 (11.1%) were connections with a successful login. Table I and II show the most frequently observed usernames and passwords.

TABLE I
Top 10 usernames

Username	Frequency	%
root	1,041,415	38.00%
support	230,245	8.40%
admin	146,564	5.35%
guest	145,355	5.30%
test	114,362	4.17%
user	60,810	2.22%
ubnt	45,152	1.65%
default	44,847	1.64%
administrator	41,529	1.52%
user1	40,769	1.49%

TABLE II
Top 10 passwords

Password	Frequency	%
<username>	558,031	21.83%
qwerty	114,712	4.49%
password	108,689	4.25%
123456	103,806	4.06%
admin	48,884	1.91%
<empty>	45,769	1.79%
12345	45,136	1.77%
1234	45,059	1.76%
123	44,933	1.76%
pass	44,190	1.73%

TABLE III
Top 10 SSH identification strings

SSH Identifier	Frequency	%
SSH-2.0-Go	793,142	32.35%
SSH-2.0-libssh-0.6.3	308,221	12.57%
SSH-2.0-libssh_0.9.6	149,589	6.10%
SSH-2.0-PUTTY	134,019	5.47%
SSH-2.0-libssh-0.1	87,758	3.58%
SSH-2.0-OpenSSH_7.3	75,530	3.08%
SSH-2.0-libssh_0.9.5	67,940	2.77%
SSH-2.0-libssh2_1.4.3	57,556	2.35%
SSH-2.0-libssh-0.5.2	40,372	1.65%
SSH-2.0-libssh_0.9.3	36,433	1.49%

A. Attack Software

During the establishment of an SSH connection, an identification string, often also referred to as banner or version string, is sent both from the client and server side. In addition to the protocol version, this identifier also contains a description of the SSH software used. Table III shows the SSH identification strings most commonly used by attackers. It is noticeable that several identifiers used by attackers do not belong to any real SSH software, e.g.:

SSH-2.0-PUTTY	SSH-2.0-OpenSSH_+3Vke
SSH-2.0-OpenSSH	SSH-2.0-OpenSSH_/q+po
SSH-2.0-OpenSSH_7	SSH-2.0-OpenSSH_1aCAJ

The identifiers shown on the right are especially notable as each consists of the string `SSH-2.0-OpenSSH_` followed by a 5-character random string. There were a total of 280 different identifiers of this format, each observed only once on the honeypot. Wu et al. [9] made a similar observation on their honeypot where attackers used a massive amount of randomized SSH identifiers. All these identifiers do not correspond to the usual naming conventions of the specified software. The original SSH client software Putty uses an identifier of the form `SSH-2.0-PuTTY_Release_0.76`, which uses both upper and lower case as well as an integrated version number. OpenSSH uses an identifier of the form `SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.10` which has an integrated major and minor version number and may be followed by a specification of the Linux distribution. We also noticed, that 79 different IP addresses iterated between multiple identifiers for each new connection. While this is a small number of IP addresses, they account for 23% of all SSH connections on the honeypot. As most of them iterated through exactly 55 different identification strings, it is likely that these IP addresses either employ the same brute force tool or originate from the same attacker. Concluding, we observed that attackers use two methods to disguise the software they use, either by specifying a fictitious

TABLE IV
MOST COMMON SSH ALGORITHM CONFIGURATIONS

SHA1 Hash	Frequency	%	Announced Software	Real Software
288c66588db389b6801571880924dd85297d2a79	643,680	26.32%	SSH-2.0-Go	SSH-2.0-Go
8578c6a37e5b097284a20d8f0519a5cb2ad026e0	613,020	25.07%	varying (62 different)	libssh2 (1.7.0 to 1.8.2)
c8ad4c355704aaba3800940bd9ed054d08e4cad8	307,264	12.57%	libssh (0.6.0 to 0.6.3)	libssh (0.6.0 to 0.6.3)
2f98915624e4915998bb5df0fd479130692fe6ea	204,283	8.35 %	libssh (0.9.3 to 0.9.5)	libssh (0.9.4 to 0.9.5)
b434178777373b47db393ca51da74db1480b14f1	132,143	5.40%	SSH-2.0-PUTTY, libssh2 (1.4.0 to 1.6.0)	libssh2 (1.2.2 to 1.6.0)
d28b3083118279bcb03c38970fd9693a439a86ed	120,715	4.94%	libssh (0.1 to 0.3.0)	libssh (0.1 to 0.4.0)
515319d6cdf4996c60308f6dd26e181d6e54baf8	96,121	3.93%	SSH-2.0-Go	SSH-2.0-Go
cdde4ab3e6f918d2c9debc0943e79a48f0380039	75,413	3.08%	OpenSSH_7.3, SSH-2.0-ssh2js0.3.6	<unknown>
65c1f4b8dfcd62b87cee2de33cd6dea6f639bc1a	64,371	2.63 %	SSH-2.0-PUTTY, libssh2 (1.8.1 to 1.9.0)	libssh2 (1.9.0)
228a732a419c331781625f127e575ad35b0431fe	40,387	1.65%	libssh (0.4.6 to 0.5.5)	libssh (0.4.1 to 0.5.5)

identification string or by enumerating through multiple valid identification strings.

Following, we will show how to get to the real software used by the attackers. One way to identify the real SSH software used by the attackers is to analyse the key exchange packet. This packet is sent during the SSH handshake by both sides to negotiate the cipher algorithms for the following encrypted connection. The offered algorithms as well as their order can vary for each SSH software and may change between different versions of the same software. The large amount of possible algorithms as well as their order makes it highly unlikely that different SSH libraries use the exact same default configuration. This makes the algorithm negotiation packet ideal for distinguishing the different SSH software from each other and detecting spoofed SSH identification strings. To automatically distinguish the different algorithm configurations from each other, we compute a SHA1 hash of the portion of the negotiation packet containing the algorithms. The hash is used as a fingerprint for that specific configuration. Following is an example of the default configuration of libssh in version 0.6.0 to 0.6.3:

Kex algorithm	curve25519-sha256@libssh.org,ecdh-sha2-nistp256,diffie-hellman-group14-sha1,diffie-hellman-group1-sha1
Host Key alg.	ecdsa-sha2-nistp256,ssh-rsa,ssh-dss
Cipher	aes256-ctr,aes192-ctr,aes128-ctr,aes256-cbc,aes192-cbc,aes128-cbc,blowfish-cbc,3des-cbc,des-cbc-ssh1
MAC algorithm	hmac-sha1
Compression alg.	none

For this key exchange packet we get the hash `c8ad4c355704aaba3800940bd9ed054d08e4cad8`.

Looking at the different hashes calculated for the connections on the honeypot, we can see that a few configurations are responsible for the majority of the connections, with the 10 most frequent configurations accounting for 94% of all SSH connections. For all configurations, we both collected the identification strings provided by the attackers and researched the SSH software that actually belongs to the configurations, which is shown in Table IV. We were unable to determine the software belonging to the configuration `cdde4ab3e6...48f0380039`, as this configuration neither conforms to the announced SSH libraries nor to any software known to us. This configuration may belong to a custom software used by the attackers. For the second most frequent configuration, a total of 62 different identifiers were employed by the attackers. However, based on the algorithms used, we were able to determine that this configuration belongs to the library libssh2. Also, the spoofed identification string SSH-2.0-PUTTY belongs to the library libssh2. The spoofed identifiers shown at the beginning of the section, which consist of the string `SSH-2.0-OpenSSH_` followed by a 5-character random string, were found to belong to the SSH package of the programming language Go. It is also noteworthy that the most frequently used configuration `288c66588d...85297d2a79` belonging to SSH-2.0-Go was first seen in 2021 and quickly took the lead among the SSH configurations.

B. Attacker Behaviour after Intrusion

To study the activities performed by attackers on the honeypot after a successful login, we decrypted each connection and analysed the SSH commands. SSH is typically used to log into a server and use the remote shell to interact with the server. In addition, it also provides the ability to tunnel connections through port forwarding as well as to upload and download files using SFTP or SCP. The SSH connection protocol, specified in RFC4254 [10], defines all services that can be executed within an SSH connection. An SSH connection is divided into one or multiple channels, each of which is started by a CHANNEL_OPEN message. Within this message, the parameter channel_type specifies the purpose of the channel. On the honeypot, only the channel types `session` for launching an interactive session and `direct-tcpip` to start port forwarding could be observed. During a session, the command CHANNEL_REQUEST can be used to start a program on the remote side. This program is specified by

TABLE V
SSH CHANNEL AND REQUEST TYPES

Type	Frequency	%
direct-tcpip	371,082	99.50%
session	1870	0.50%
exec	1587	0.43%
shell	152	0.04%
subsystem (sftp)	85	0.02%

the parameter request_type and can be a shell (request_type: `shell`), an application program (request_type: `exec`), or a subsystem such as SFTP (request_type: `subsystem`) used to upload or download files. Other request types such as `env`, `exit-status`, `window-change` or `exit-signal` were excluded from the statistics as they have no relevance for an attack. Table V shows the frequency of channel types and request types.

The majority of sessions used the request type `exec`. This type is used to start the execution of a command on the server. Unlike the request type `shell`, this does not start a terminal where the user can enter commands interactively, but sends only a single command to the server. In OpenSSH, this can be achieved by passing the command directly to the ssh program (e.g. `ssh user@server 'ls -la'`). Since this type of interaction with a server is quite uncommon for a human user, it can be assumed that these attacks were carried out by bots. While sessions are used to attack the honeypot itself by reading the data on the server or installing malware, it has become apparent that the majority of attackers use the honeypot for other purposes. Through the method of port forwarding (`direct-tcpip`), attackers can use the SSH connection to tunnel connections to other servers. This way, they can hide their real IP address and use the compromised server as a proxy to attack their actual targets. The high number of these attacks on the honeypot shows that compromised servers are often only used as proxies by attackers. Looking at the target ports of the tunnel connections, we found that the majority was web traffic, with 46% HTTP (port 80) and 30% HTTPS (port 443), followed by SMTP traffic on port 25 (19%), port 587 (3.4%) and port 465 (0.3%).

Next, we will have a look at the commands used by attackers during SSH connections. For this we consider all connections that include sessions with commands (`exec`) or shell input (`shell`). Connections that only use the honeypot as a tunnel are excluded, since they do not interact with the server itself. The commands were parsed and classified into 5 categories:

- **Recon:** Commands used to investigate the filesystem, software or hardware configuration like `ls`, `df`, `ifconfig`, `w`, `who`, `netstat`, `ps`, `top`, `uname`, `lscpu`.
- **Download:** This includes the download and installation of malware or tools using commands like `curl`, `wget`, `ftp`, `git clone`, `scp`.
- **Exec:** This involves the execution of programs or malware. This is detected based on the presence of the `./` notation, used in Linux to run a custom program, or through the use of an interpreter or shell such as `perl`, `python`, `bash`, `sh` to execute a script.
- **User:** This includes actions which affect the users of the system like changing passwords (`chpasswd`, `passwd`), adding users (`useradd`) or adding SSH public keys.
- **Cleanup:** This includes commands that attackers use to cover their tracks. This involves deleting or emptying files (`rm`, `cat /dev/null >`, `crontab -r`), deleting bash history (`history -c`) or killing processes (`kill`, `pkill`, `killall`).

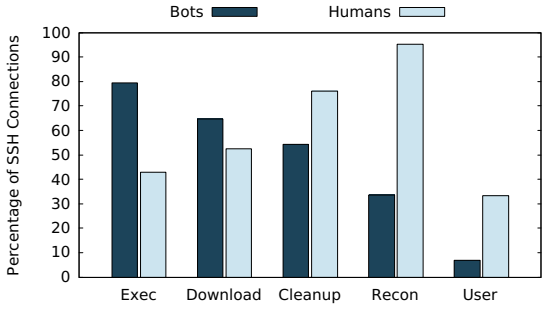

Fig. 1. Command types used by human attackers and bots

Each SSH connection was labelled based on these categories if it contained at least one command that belongs to the particular category. Thus, an SSH connection can be assigned anything between 0 and 5 categories. Fig. 1 shows the percentage of SSH connections assigned to the 5 categories. This is further divided into human attackers and bots. We classified shell connections to be executed by a human based on the sequence and timing of packets. During a shell via SSH, each character that the user types is transmitted separately to the server and is mirrored back by it afterwards. As human users tend to type commands instead of just pasting the entire command, a large part of such an SSH connection consists of sequences of small packets with the sender alternating between server and client. Also, humans type commands at a slow and steady rate and need time to look at the output while bots send commands immediately. Using these characteristics, we manually classified the 152 shell sessions as either human or bot. 125 sessions had no activity from the client side and were excluded. Of the remaining, we found 21 connections to be executed by human attackers. Even though this number is very low (1.1% of all sessions), it is consistent with the observations of other studies stating that most attacks are carried out by bots [11]. Connections made by bots are mainly used to install and execute malware, which is evident in the high number of commands belonging to the categories download and exec. Some bots even run identical commands in multiple SSH connections and also retried previously failed commands. In contrast, human attackers almost always perform reconnaissance of the system first and try to cover their tracks afterwards. Also, human attackers try to change the user password much more frequently than bots.

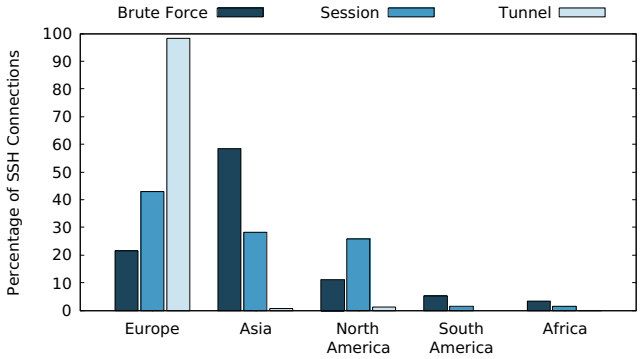

Fig. 2. Origin of attacks by attack type

C. Origin of Attacks

To get an impression of the origin of the attackers, we analysed the IP addresses used for the different types of attacks. In total, we received connections from 35,445 different IP addresses that made a login attempt on the honeypot. Looking at the group of IP addresses used for brute force attacks and the group of IP addresses that made a successful login, we can see that there is a clear separation between both groups. 97.7% of the IP addresses were only used to perform dictionary attacks. These IP addresses, even upon finding valid credentials, never executed commands on the server. 542 (1.53%) IP addresses successfully logged into the server but did not perform any dictionary attacks. 268 (0.76%) were used to perform both dictionary attacks and intrusions on the server. If we look at the groups of IP addresses that perform the different types of intrusions, we can observe the same pattern. There is a clear separation between IP addresses performing tunnelling attacks and IP addresses executing commands on the server. We found only 14 IP addresses performing both tunnelling attacks and executing commands. So, we can conclude that most IP addresses are specialized for a particular task.

For each IP address, we determined the geographic location. Fig. 2 shows the origin of the attackers, categorized into the 5 continents and divided by the type of attack performed. Brute force attacks originate mainly from Asia and Europe, with China being the largest contributor. Attackers who executed commands during sessions are spread almost equally between the 3 continents Europe, Asia and North America with the United States being the country where most attacks came from. Interestingly, the majority of tunnelling attacks originate from Europe only. The Netherlands were by far the largest contributor, accounting for 68% of the tunnelling attacks. Barron and Nikiforakis [11] made a similar observation in 2017. They also noticed that a large part of tunnelling attacks originated in the Netherlands. This indicates that this type of attack is already being carried out for several years.

D. Malware Files

Many attackers who used the server to execute commands installed additional malware. A total of 1429 file transfers were observed. Various methods were used to transfer files to the server. The most common method was HTTP (93%) through the commands wget and curl, followed by SFTP (6%) and SCP (0.8%). We could not observe the use of HTTPS. Often the same file was uploaded multiple times and through different commands. We extracted the files from the captures, obtaining a total of 302 unique files based on the SHA1 hash of the files. The files were analysed using the API of VirusTotal [12] and classified into malware categories based on the malware labels returned by the detection engines of VirusTotal. From the 302 unique files analysed, a total of 249 were detected as malware. Table VI and VII show the frequency of the most common categories and file types of the malware files. Crypto Miner account for the largest percentage of uploads. The second place belongs to malware of the botnet Tsunami. Tsunami, also known as Kaiten, is an IRC botnet and is mainly used to launch DDoS attacks [13]. Following are other DDoS malware variants, which also include malware known as XOR DDoS. Also, malware belonging to the botnets Mirai and Bashlite could be found, all of which are also DDoS-capable malware. Mirai is a botnet that started in 2016 and is still one of the most common botnet malware found. Even though Mirai prefers Telnet as an intrusion vector [14], we could find several different variants on our SSH honeypot. Furthermore, 31 different shell scripts, which we have classified as Downloader, were found. These are primarily used to download additional files using shell commands like wget or curl. Looking at the file types used, there is a predominance of executable binaries. There is a slight majority of 64-bit binary files (44%) compared to 32-bit binary files (34%) in terms of the total number of uploads. However, looking at the number of unique files shows a large majority of 32-bit binaries (56%) compared to 64-bit binaries (12%). This indicated that there are far more different 32-bit malware variants in use by attackers.

TABLE VI
MOST COMMON MALWARE TYPES

Malware Type	Uploads	Unique Files
Crypto Miner	489	32
Tsunami	331	75
Downloader	151	31
DDoS	80	34
Mirai	69	27
Shellbot	37	9
Bashlite	13	9

TABLE VII
MOST COMMON MALWARE FILE TYPES

File Type	Uploads	Unique Files
ELF 64-bit	547	29
ELF 32-bit	426	140
Shell Script	161	38
gzip	54	25
Perl Script	45	11
tar	5	4
JSON	4	2

IV. Related Work

One of the most well-known projects related to honeypots is the Honeynet project [15]. Established in 1999 as a research organization, they designed multiple generations of honeynet architectures [16] and developed open source tools like Sebek [17] to monitor networks of honeypots.

Various works address the setup, architecture and design of honeypots. Franco et al. [18] give an extensive overview of honeypots for IoT, Industrial IoT and Cyber-Physical Systems, including several SSH honeypots. Nicomette et al. [6] describe their architecture of a high-interaction SSH honeypot. Barron and Nikiforakis [11] used Cowrie honeypots to investigate how certain properties, such as the location, the difficulty to break in, or the available files, affect the attractiveness of a honeypot.

Several studies [4], [19], [20] reported basic statistics about brute force attacks like IP addresses of attackers, SSH banners or usernames and passwords. Rabadia and Valli [21] used Kippo SSH honeypots to analyse dictionaries and wordlists involved in brute force attacks. Rabadia et al. [22] analysed the timing of intrusion attempts during a 24-hour day gathered from 6 Kippo SSH honeypots. Ghiëtte et al. [23] used a fingerprinting method similar to ours based on hashes of advertised algorithms to identify SSH brute force tools. However, the period of this study was only one month. Also, even though they mentioned finding spoofed SSH identification strings, they never showed the real software used by attackers.

Contrary to our approach of using a high-interaction honeypot, most previous works relied on low- or medium-interaction honeypots like Kippo or Cowrie which are easy to maintain and avoid the risk of attackers gaining access to a fully functional server. Others prevented a login altogether, thus investigating only incoming dictionary attacks. All of these approaches make it impossible to guarantee realistic interaction of the attackers with the honeypot. Also, well-known honeypots such as Kippo or Cowrie may be detected by attackers [24] and some IP addresses used by attackers only emerge after a login has already been found.

V. Conclusion

In this paper, we presented an overview of current attacks targeting the SSH protocol observed on a high-interaction honeypot. Through the use of a high-interaction honeypot, we were able to observe the full spectrum of attacker activities and provided attackers with a fully functional server, making it possible to observe attacks that are as realistic as possible.

In some aspects, we could reconfirm the results of other studies based on new data. This includes the fact that we found that most attacks are carried out by bots and that in most cases attackers only use the server as a proxy to attack other targets. Further, we presented new insights into the origin and tools used by attackers. We showed that attackers use different methods to disguise the software they use and we are the first to present the actual software used by attackers. Also, our observations suggest that attackers organize their IP addresses depending on specific tasks, with certain IP addresses used for brute force attacks and others performing the intrusion step.

References

[1] L. Spitzner, *Honeypots: Tracking hackers*, 2002.

[2] Unit 42 - Palo Alto Networks, "2020 Unit 42 IoT threat report," Tech. Rep., 2020.

[3] H. Zhao, H. Shu, and Y. Xing, "A review on IoT botnet," in *The 2nd International Conference on Computing and Data Science (CONF-CDS 2021)*, 2021.

[4] A. Abdou, D. Barrera, and P. C. van Oorschot, "What lies beneath? Analyzing automated SSH bruteforce attacks," in *Technology and Practice of Passwords*, 2016, pp. 72–91.

[5] D. Ramsbrock, R. Berthier, and M. Cukier, "Profiling attacker behavior following SSH compromises," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN' 07)*, 2007, pp. 119–124.

[6] V. Nicomette, M. Kaâniche, E. Alata, and M. Herrb, "Set-up and deployment of a high-interaction honeypot: Experiment and lessons learned," *Journal in Computer Virology*, vol. 7, pp. 143–157, 2011.

[7] S. Sentanoe, B. Taubmann, and H. P. Reiser, "Virtual machine introspection based SSH honeypot," in *Proceedings of the 4th Workshop on Security in Highly Connected IT Systems (SHCIS '17)*, 2017, pp. 13–18.

[8] J. Bullock and J. T. Parker, *Wireshark® for security professionals: Using Wireshark and the Metasploit® framework*, 2017.

[9] Y. Wu, P. M. Cao, A. Withers, Z. T. Kalbarczyk, and R. K. Iyer, "Mining threat intelligence from billion-scale SSH brute-force attacks," in *Workshop on Decentralized IoT Systems and Security (DISS)*, 2020.

[10] T. Ylonen and C. Lonvick, "RFC4254 - The secure shell (SSH) connection protocol," 2006. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4254

[11] T. Barron and N. Nikiforakis, "Picky attackers: Quantifying the role of system properties on intruder behavior," in *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC 2017)*, 2017, pp. 387–398.

[12] (2022, Sep.) VirusTotal. [Online]. Available: https://www.virustotal.com/

[13] M. De Donno, N. Dragoni, A. Giaretta, and A. Spognardi, "DDoS-capable IoT malwares: Comparative analysis and Mirai investigation," *Security and Communication Networks*, 2018.

[14] H. Griffioen and C. Doerr, "Examining Mirai's battle over the internet of things," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCS '20)*, 2020, pp. 743–756.

[15] L. Spitzner, "The Honeynet Project: Trapping the hackers," *IEEE Security & Privacy*, vol. 1, no. 2, pp. 15–23, 2003.

[16] D. V. Silva and G. D. Rodríguez Rafael, "A review of the current state of honeynet architectures and tools," *International Journal of Security and Networks*, vol. 12, no. 4, pp. 255–272, 2017.

[17] The Honeynet Project, "Know your enemy: Sebek, a kernel based data capture tool," 2003. [Online]. Available: http://honeynet.onofri.org/papers/sebek.pdf

[18] J. Franco, A. Aris, B. Canberk, and A. S. Uluagac, "A survey of honeypots and honeynets for internet of things, industrial internet of things, and cyber-physical systems," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 4, pp. 2351–2383, 2021.

[19] G. K. Sadasivam, C. Hota, and B. Anand, "Honeynet data analysis and distributed SSH brute-force attacks," in *Towards Extensible and Adaptable Methods in Computing*, 2018, pp. 107–118.

[20] S. Z. Melese and P. Avadhani, "Honeypot system for attacks on SSH protocol," *International Journal of Computer Network and Information Security*, vol. 8, no. 9, pp. 19–26, 2016.

[21] P. Rabadia and C. Valli, "Finding evidence of wordlists being deployed against SSH honeypots - implications and impacts," in *Proceedings of 12th Australian Digital Forensics Conference (ADF 2014)*, 2014, pp. 114–120.

[22] P. Rabadia, C. Valli, A. Ibrahim, and Z. Baig, "Analysis of attempted intrusions: Intelligence gathered from SSH honeypots," in *Proceedings of the 15th Australian Digital Forensics Conference (ADF 2017)*, 2017, pp. 26–35.

[23] V. Ghiëtte, H. Griffioen, and C. Doerr, "Fingerprinting tooling used for SSH compromisation attempts," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019, pp. 61–71.

[24] A. Vetterl and R. Clayton, "Bitter harvest: Systematically fingerprinting low- and medium-interaction honeypots at internet scale," in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018.