Masterarbeit

Predicting SSH keys in Open SSH Memory dumps

A report by

Rascoussier, Florian Guillaume Pierre

Prüfer

Prof. Dr. Michael Granitzer

Christofer Fellicious

Prof. Dr. Pierre-Edouard Portier

September 6, 2023

Abstract

As the digital landscape evolves, cybersecurity has become an indispensable focus of IT systems. Its ever-escalating challenges have amplified the importance of digital forensics, particularly in the analysis of heap dumps from main memory. In this context, the Secure Shell protocol (SSH) designed for encrypted communications, serves as both a safeguard and a potential veil for malicious activities. This research project focuses on predicting SSH keys in OpenSSH memory dumps, aiming to enhance protective measures against illicit access and enable the development of advanced security frameworks or tools like honeypots.

This Masterarbeit is situated within the broader SmartVMI project, a collaborative research initiative with the objective to advance artificial intelligence-based mechanisms for attack detection and digital forensics. Specifically, this work seeks to build upon existing research on key prediction in OpenSSH heap dumps. Utilizing machine learning algorithms, the study aims to refine feature extraction techniques and explore innovative methods for effective key detection prediction. The objective is to accurately predict the presence and location of SSH keys within memory dumps. This work builds upon, and aims to enhance, the foundations laid by SSHkex [1] and SmartKex [2], enriching both the methodology and the results of the original research while exploring the untapped potential of newly proposed approaches.

This report encapsulates the progress of a year-long Master's thesis research project executed between October 2022 and October 2023. Conducted within the framework of the PhDTrack program between the University of Passau and INSA Lyon, the research has been supervised by Christofer Fellicious and Prof. Dr. Michael Granitzer from the University of Passau, as well as Prof. Dr. Pierre-Edouard Portier from INSA Lyon. It offers an in-depth discussion on the current state-of-the-art in key prediction for OpenSSH memory dumps, research questions, experimental setups, programs development as well as discussing potential future directions.

Acknowledgements

A special acknowledgment goes to Christofer Fellicious, my engaged supervisor at the University of Passau, for his guidance, support and feedback during the Masterarbeit.

I want to express my sincere gratitude to my colleague and friend, Clément Lahoche, whose human and technical skills have been a great source of inspiration and motivation throughout this project; especially considering that we have been working on closely related subjects. It has been a great pleasure to share our ideas and insights, and to collaborate on the development of several programs necessary for the experimentations.

Another acknowledgments go to my esteemed supervisors Prof. Dr. Granitzer and Prof. Dr. Portier for their support and feedback during the Masterarbeit.

I would also like to express my sincere gratitude to all the persons that have helped me, even punctually, during the Masterarbeit with their valuable help, insights, discussions and contributions as well as all the persons involved in the PhDTrack program that made this Masterarbeit possible, including but not limited to:

- Lionel Brunie, Director of CS Department at INSA Lyon, that makes this PhDTrack program possible from the French side.
- Harald Kosch, Head of the Chair of Distributed Information Systems at the University of Passau, that makes this PhDTrack program possible from the German side.
- Natalia Lucari, PhDTrack coordinator at INSA Lyon, for her support and help during the PhDTrack program.
- Ophelie Coueffe, PhDTrack coordinator at the University of Passau, for her support and help during the PhDTrack program.
- Elöd Egyed-Zsigmond, PhDTrack coordinator at the University of Passau, for the subject selection and administrative support.
- All the other PhDTrack students for the great atmosphere, mutual help and the interesting discussions during almost two years.

Finally, my last acknowledgments go to my family and friends for their support and encouragements.

Contents

1 Introduction

The digital age has brought with it an unprecedented increase in the volume and complexity of data that is being generated, stored, and processed. This data is often sensitive in nature, and its security is of paramount importance, making cybersecurity a critical focus area. This evolving landscape is fraught with challenges that continue to amplify the importance of digital forensics in IT systems. One area that stands out for its widespread use and importance is the Secure Shell protocol (SSH) and its most popular implementation, OpenSSH. SSH is a cryptographic network protocol widely used for secure remote access to systems. It is also used for secure file transfer, and as a secure tunnel for other applications. SSH is a key component of IT systems whose encryption capabilities are critical to the security of IT systems. However, it also presents a unique set of challenges, most notably the concealment of malicious activities.

A common case is when a unauthorized actor gains access to SSH keys so as to get access to a system. This can happen through a malicious human actor, but more commonly through automated processes such as malwares and botnets. This situations present a formidable and growing threat to cybersecurity, affecting a broad range of stakeholders from governments and financial institutions to individual users. In just 2019, the number of Command and Control (C&C) servers for botnets increased by 71.5%, leading to an estimated $19 billion in advertising theft [22]. Many malwares and botnets “have in common that they have used as attack vector the Secure Shell (SSH) remote access service” [22].

At the heart of the issue lies the fact that SSH veils its communications through encryption, making it difficult to detect malicious activities. To be able to detect those potential malicious actors, it is possible to replace SSH by a honeypot that enable to monitor pseudo-SSH activities. There is a range of readily available honeypots, such as Kippo or Cowrie, which are designed to emulate a vulnerable SSH system and attract attackers [31]. The problem lies that thoses honeypots are not able to mimic perfectly a real system, which makes them easy to detect by experienced attackers. As stated by „Analysis of SSH Honeypot Effectiveness“: “The ability to collect meaningful malware from attackers depends on how the attackers receive the honeypot. Most attackers fingerprint targets before they launch their attack, so it would be very beneficial for security researchers to understand how to hide honeypots from fingerprinting and trick the attackers into depositing malware. [...] What is certain is that if a cautious attacker believes they are in a honeypot, they will leave without depositing malware onto the system, which reduces the effectiveness of the honeypot” [32].

The are other approaches that allows to decrypt SSH connections without relying on a honeypot, like the *man-in-the-middle* or *binary manipulation* with their own set of challenges [1]. Instead of relying on softwares that mimics or modify a real system, it is possible to use a real unmodified system directly. The idea is to be able to decrypt SSH connection channels, which is possible if the SSH keys are known. Since SSH encryption keys are typically stored in the main memory of a system, it is possible for the administrators to extract them through the exploitation of memory dumps of a targeted system. In this context, the ability to detect SSH keys in memory dumps, and specifically OpenSSH keys, is critical to the development of effective SSH honeypot-like systems. The research introduced by the SmartVMI project with SSHKex, SmartKex, the present thesis and the future related work could be used to develop such a new type of system-monitoring tools. This new kind of tools would be very difficult to detect by attackers, increasing their effectiveness, and wouldn’t require the alteration of the system. The present report is focused on the SSH key detection in memory dumps, which is a key component allowing to decode SSH communications such that it become possible to intercept malicious communications and to detect malicious activities.

2 Research Questions

At the very beginning of this thesis, the objective was to answer the following research questions:

- RQ1: What is the state of the art in the field of security key detection in heap dump memory?
- RQ2: What are the challenges of security key detection in heap dump memory?
- RQ3: How can the existing methods for detecting SSH keys in OpenSSH heap dumps be improved?

The SmartVMI project has already made significant progress in the detection of SSH keys in OpenSSH heap dumps. An open dataset of memory dumps has been created, and a simple yet effective method for detecting SSH keys has been developed. The dataset has been used to train and test simple machine learning algorithms, and the results have been promising. The research has been published in the form of two papers, SSHkex [1] and SmartKex [2], which is the basis of this thesis.

However, there is still room for improvement, particularly in the area of machine learning algorithms. This thesis seeks to build upon the existing research by refining feature extraction techniques and exploring innovative methods for effective key detection prediction. The objective is to accurately predict the presence and location of SSH keys within memory dumps. Rooted in this context, this Masterarbeit aims to address several key research questions:

- RQ4: What features are most indicative of SSH keys in memory dumps?
- RQ5: How can these features be extracted from memory dumps and used to train machine learning algorithms?
- RQ6: How can machine learning algorithms be optimized for the prediction of SSH keys in memory dumps?

By tackling these research questions, this thesis seeks not only to advance the academic understanding of SSH key prediction and digital forensics but also to provide practical insights that could lead to the development of more secure and effective systems.

3 Structure of the Thesis

4 Background

The evolving landscape of cybersecurity necessitates robust techniques for safeguarding digital communications. OpenSSH, a pivotal element in this landscape, is a popular implementation of the Secure Shell (SSH) protocol, which enables secure communication between two networked devices. The protocol is widely used in the industry, particularly in the context of remote access to servers. Using digital forensic techniques, it is possible to extract the SSH keys from memory dumps, which can then be used to decode encrypted communications thus allowing the monitoring of controlled systems. At the crux of this Masterarbeit is the development of machine learning algorithms to predict SSH keys within these heap dumps, focusing on using graph-like-structures and vectorization for custom embeddings. With an interdisciplinary approach that fuses traditional feature engineering with graph-based methods as well as memory modelization for inductive reasoning and learning inspired by recent developments in Knownledge Graph (KG)s, this research not only leverages existing machine learning paradigms but also explores new avenues, such as Graph Convolutional Networks (GCN).

The objective of this background section is multifaceted. Since the project has seen two distinct phases, one more classical Machine Learning (ML) approach, and a second one centered aroung graph-based advanced learning methods, the background section is divided into several subsections that introduce the reader to the different concepts and techniques used in the project. First, it aims to offer an overview of SSH protocols, particularly focusing on OpenSSH key implementations subsection 4.1. Second, it delineates the dataset and prior work on key extraction techniques including SmartKey subsection 4.2. Third, it delves into the technical aspects of graphs modelization subsection 4.3, followed by feature engineering and embeddings both traditional and graph-based subsection 4.4. Finally, it addresses the machine learning models employed in the research, emphasizing their suitability for maximizing recall in key prediction subsection 4.5. By fusing these distinct but interrelated areas, this section lays the foundation for the research methodologies and hypotheses tested in this study.

4.1 SSH and OpenSSH Implementation

4.1.1 Basics of the Secure Shell Protocol (SSH)

The Secure Shell Protocol, commonly known as SSH, is designed to facilitate secure remote login and other secure network services over insecure networks. SSH has been design since its inception with security in mind, as a successor of the Telnet protocol, which is not secure, and other “unsecured remote shell protocols such as rlogin, rsh and rexec” [1]. As stated by the authors of the *SSH Annual Report 2018*, “The founder of SSH, Tatu Ylönen, designed the first version of the SSH protocol after a password-sniffing attack at his university network. Tatu released his implementation as freeware in July 1995, and the tool quickly gained in popularity. Towards the end of 1995, the SSH user base had grown to 20,000 users in fifty countries. By 2000, there were an estimated 2,000,000 users of the protocol. Today, more than 95% of the servers used to power the Internet have SSH installed in them. The SSH protocol is truly one of the cornerstones of a safe Internet.” [23].

SSH is defined in *The Secure Shell (SSH) Protocol Architecture* [15]. It is divided into three major components:

- **Transport Layer Protocol:** This provides server authentication, confidentiality, and integrity.

It can also optionally provide compression. Typically, the transport layer runs over a TCP/IP connection but can also be used on top of any other reliable data stream.

- **User Authentication Protocol:** Running over the transport layer, this protocol authenticates the client-side user to the server. Multiple methods of authentication such as password and public key are supported.
- **Connection Protocol:** This multiplexes the encrypted tunnel established by the preceding layers into several logical channels. Channels can be used for various purposes, such as setting up secure interactive shell sessions or tunneling arbitrary TCP/IP ports.

"The client sends a service request once a secure transport layer connection has been established. A second service request is sent after user authentication is complete. This allows new protocols to be defined and coexist with the protocols listed above" [15].

For the scope of this Masterarbeit, understanding SSH's key exchange and encryption mechanism is important. As Summarized in SSHKex [1], the SSH key exchange procedure results in a derived master key *K* and a hash value *h*. These are critical for client-server communication encryption and session identification.

During the key exchange, multiple session keys are computed for different purposes:

- **Initialization Vectors:** Key A and Key B are used for initialization vectors from the client to the server and vice versa.
- **Encryption Keys:** Key C and Key D serve as encryption keys for client-to-server and server-to-client communications, respectively.
- **Integrity Keys:** Key E and Key F are used to maintain the integrity of the data transmitted between the client and server.

These keys are computed using hash functions that take the master key *K* and a hash value *h*, a unique letter (A, B, C, D, E, or F), and the session ID as inputs. This is summarized in „The OpenSSH Protocol under the Hood": " The equations used for deriving the above vectors and keys are taken from RFC 4253 [16]. In the following, the $||$ symbol stands for concatenation, K is encoded as mpint, "A" as byte and $session_id$ as raw data. Any letter, such as the "A" (in quotation marks) means the single character A, or ASCII 65.

- Initial IV client to server: $HASH(K||H||"A"||session_id)$.
- Initial IV server to client: $HASH(K||H||"B"||session_id)$.
- Encryption key client to server: $HASH(K||H||"C"||session_id)$.
- Encryption key server to client: $HASH(K||H||"D"||session_id)$.
- Integrity key client to server: $HASH(K||H||"E"||session_id)$.
- Integrity key server to client: $HASH(K||H||"F"||session_id)$.

" [17]. Details about the hash function are given in the next section.

The most interesting keys are the encryption keys, as they are used to encrypt the communication between the client and the server. The other keys are used for integrity checks and initialization

vectors. Decrypting encrypted SSH communication necessitates either to retreive these session keys and variables so as to recompute the keys, or to retreive those keys directly, which is the focus of this Masterarbeit.

4.1.2 OpenSSH Implementation

OpenSSH (OpenBSD Secure Shell) is an open-source implementation written in C of the SSH protocol suite, and it is the most widely used SSH implementation [17]. It is the default SSH implementation on most Linux distributions, and it is also available for Windows. OpenSSH is used for a wide range of purposes, including remote command-line login and remote command execution. It is also used for port forwarding, tunneling, and transferring files via SCP and SFTP either manually or via automated processes, such as backup systems, configuration management tools, and automated software deployment tools.

OpenSSH is composed of several tools and daemons, including client and server components [20]:

- **ssh:** The basic client program that allows to log into and execute commands on a remote machine.
- **scp:** A program for securely copying files between machines.
- **sftp:** An interactive file transfer program that uses SSH to secure the connection.
- **sshd:** This is the SSH daemon that runs on the server. This is used for connecting to a remote machine when using the SSH client from another system.
- **ssh-agent:** The program that holds private keys in memory, so one doesn't have to enter ones passphrase every time.
- **ssh-add:** A program for adding RSA or DSA identities to the authentication agent.
- **ssh-keygen:** A utility for creating and managing SSH keys.
- **ssh-keyscan:** A utility for gathering public SSH host keys from a number of hosts.
- **ssh-keychk:** A utility for checking the validity of SSH keys.
- Several other tools to support the SSH protocol and the OpenSSH implementation.

OpenSSH employs a variety of hash functions and algorithms to secure data, most commonly using SHA1. However, SHA1 is increasingly seen as weak due to its vulnerability to collision attacks [17]. In light of this, the contemporary standard leans towards SHA512. The hash functions are used alongside cipher algorithms like "Advance Encryption Standard (AES) Cipher Block Chaining (CBC), AES Counter (AES-CTR), and ChaCha20" [2]. The Message Authentication Code (MAC) typically uses either MD5 or SHA1 hash algorithms in combination with a secret key. Since cybersecurity and cryptography are constantly evolving, so do SSH and OpenSSH. Depending on the version [17], the available hash options include:

- **ssh-dss:** *(disabled at run-time since OpenSSH 7.0 released in 2015)* SSH-1 version using Digital Signature Algorithm (DSA) from the Digital Signature Standard (DSS). Originally popular but phased out due to vulnerabilities to collision attacks for DSA Key in a 1024-bit

modulus. As stated by *Key Exchange (KEX) Method Updates and Recommendations for Secure Shell (SSH)*: “These attacks are still computationally very difficult to perform, but it is desirable that any key exchange using SHA-1 be phased out as soon as possible” [3] [10].

- **ssh-rsa:** *(disabled at run-time since OpenSSH 8.8 released in 2021)* It refers to the use of RSA (Rivest-Shamir-Adleman) encryption algorithm. In the context of SSH-1, this version had to be replaced due to the related to key size issue similar to DSS: “RSA 1024-bit keys have approximately 80 bits of security strength”... “which may not be sufficient for most users.” [3] [13].
- **ecdsa-sha2-nistp256:** *(since OpenSSH 5.7 released in 2011)* Uses the SHA-2 family for hashing and the NIST P-256 curve. It is considered secure and efficient, with an Estimated Security Strength (ESS) of 128 bits [3] [7].
- **ecdsa-sha2-nistp384:** *(since OpenSSH 5.7)* Utilizes the SHA-2 family and the larger NIST P-384 curve for additional security at the cost of performance. It has an ESS of 192 bits [3] [7].
- **ecdsa-sha2-nistp521:** *(since OpenSSH 5.7)* Employs SHA-2 and the even larger NIST P-521 curve for maximal security with an ESS of 256 bits [3]. It is less commonly used due to performance considerations [7].
- **ssh-ed25519:** *(since OpenSSH 6.5 released in 2014)* Known for high security and performance efficiency; employs the Ed25519 elliptic curve with an ESS of 128 bits [3] which is similar to $ecdsa-sha2-nistp256$, and has been more prevalent following the 2013 suspicions of NSA backdoors in NIST curves [6] following the Snowden revelations [4] [5] [9].
- **rsa-sha2-256:** *(since OpenSSH 7.2 released in 2016)* An upgrade from ssh-rsa, using SHA-256 (with ESS of 128 bits) for hashing to improve security without major performance hits [11].
- **rsa-sha2-512:** *(since OpenSSH 7.2)* Similar to $rsa-sha2-256$ but employs SHA-512 for even stronger security, albeit with some performance cost [11].
- **ecdsa-sk:** *(since OpenSSH 8.2 released in 2020)* Security Key-enabled, uses NIST curves and is geared towards modern hardware-based authentication [12].
- **ed25519-sk:** *(since OpenSSH 8.2)* Similar to ssh-ed25519 but integrates hardware-based Security Keys for an additional layer of security [12].
- **NTRU Prime-x25519:** *(since OpenSSH 9.0)* A new, highly secure algorithm focused on post-quantum cryptography, providing future-proof security [18] [14].

These hashes have fixed lengths such that key lengths range between 12 and 64 bytes [2]. Since high-quality random number generation is crucial to ensure that thoses keys are secure and difficult to predict, it can thus be assumed that thoses key have a high entropy [19]. This is a crucial assumption as it is the basis for the use of both brute force and machine learning algorithms to predict the presence and location of SSH keys in memory dumps.

The keys generated by these hash functions are pseudo-random numbers stored in the system’s RAM. Following the Kerckhoffs’ principle: that “a cryptosystem should be secure, even if everything about the system, except the key, is public knowledge”, the code for the OpenSSH implementation is open-source and available on GitHub [20]. This allows for the analysis of the code and the identification of the memory structures where the keys are stored.

4.1.3 The state of SSH security

Since its origins, SSH has been developed with cybersecurity in mind, and is generally considered a secure method for remote login and other secure network services over an insecure network. However, as with any technology, it can be exploited if not configured or managed correctly. The protocol is used by system administrators to manage remote systems, and it is also used by automated processes to transfer data and perform other tasks. This makes SSH a valuable target for attackers. In fact, SSH has been a popular target for cyber-attacks. Due to being so prevalent, it is often used by threat actors either as a vector for initial access, as a means to move laterally across a network or as a covered exit for exfiltration of sensitive data [30]. The encrypted nature of its communications makes it an attractive option for attackers, as it can be difficult to detect malicious activity.

Here are some cases where SSH can involved in cyber-attacks, although it's important to note that SSH itself is not inherently insecure:

- **SSH Brute-Force Attacks:** One of the most common types of attacks involving SSH is a brute-force attack, where an attacker tries to gain access by repeatedly attempting to login with different username-password combinations. These attacks are not sophisticated but can be effective if strong authentication measures are not in place. For instance, the botnet *Chabulo* was used to launch a large-scale brute-force attack "through compromised SSH servers and IoT devices" in 2018 [23].
- **SSH Key Theft:** In some advanced attacks, threat actors have stolen SSH keys to move laterally across a network after initial entry. This allows them to authenticate as a legitimate user and can make detection much more challenging. It can " occurs when users have their SSH password or unencrypted keys stolen through a variety of methods (sniffed via a key-logging console program, shoulder-surfed via bad security awareness, poor key management practices, etc.)." [24].
- **Man-in-the-Middle Attacks:** Although SSH is designed to be secure, it can be susceptible to man-in-the-middle attacks if proper verification of SSH keys is not done during the initial connection setup [17].
- **Misconfiguration:** As with any technology, misconfiguration can lead to security issues. For example, leaving default passwords, using weak encryption algorithms, or enabling root login can all make an SSH-enabled system vulnerable [22].

4.1.4 SSH vulnerabilities and use in cyber-attacks

In cybersecurity, it is generally considered that any system that is connected to the Internet will be attacked at some point. Similarly, it is a common saying that no system is 100% secure. This is true for SSH as well. Although it is a secure protocol, it can be exploited if not configured or managed correctly.

Some vulnerabilities have also been discovered in the protocol itself, although these are rare.

- **SSH-1 Vulnerabilities:** A series of vulnerabilities in the first implementation of SSH were discovered from 1998 to 2001, with its subsequent fixes leading to unauthorized content insertion and arbitrary code execution. SSH-1 had many design flows and is now considered obsolete. [26], [25].

- **CBC Plaintext Recovery:** A theoretical vulnerability discovered in 2008 affecting all versions of SSH, allowing the recovery of up to 32 bits of plaintext from CBC-encrypted ciphertext [27].
- **Suspected Decryption by NSA:** Leaked information in 2014 suggested that the NSA might be able to decrypt some SSH traffic, although the protocol itself was not confirmed to be compromised [28].

SSH has been used in many high-profile cyber-attacks and malwares, including the following:

- **Operation Windigo:** This was a large-scale campaign that infected over 25,000 UNIX servers. SSH was one of the vectors used for maintaining control over compromised servers. A report by ESET mentions that the OpenSSH backdoor Linux/Ebury was first discovered in 2011 as a component of the aforementioned operation. “This operation has been ongoing since at least 2011 and has affected high profile servers and companies, including cPanel - the company behind the famous web hosting control panel - and Linux Foundation’s kernel.org - the main repository of source code for the Linux kernel” [29].
- **Linux/Hydra:** Initially unleashed in 2008, this malware is a fast login cracker that targets a range of popular protocols including SSH. Hence, SSH is one of its primary vectors to gain initial access to Internet of Things (IoT) devices. Once a device is infected by Linux/Hydra, it joins an IRC channel and initiates a SYN Flood attack [31].
- **Psyb0t:** Discovered in early 2009, Psyb0t is an IRC-controlled malware specifically designed to target devices with MIPS architecture, such as routers and modems. Notably, it was responsible for orchestrating a DDoS attack against the DroneBL service, infecting up to 100,000 devices for this purpose. The malware is equipped to conduct UDP and ICMP flood attacks and employs a brute-force attack mechanism against Telnet and SSH ports. Remarkably, it uses a pre-configured list of 6,000 usernames and 13,000 passwords to perform these attacks [31].
- **Chuck Noris:** Similar to Psyb0t in its objectives and methods, Chuck Noris targets routers and DSL modems, focusing on SoHo (small office/home office) devices. However, unlike Psyb0t, which uses ICMP flood attacks, Chuck Noris deploys ACK flood attacks. The malware carries out brute-force attacks on Telnet and SSH open ports, drawing parallels to the tactics employed by Psyb0t but with the specific variation in flooding techniques [31].

It’s worth noting that in many of these cases, SSH was not the initial attack vector but was used at some stage in the attack lifecycle. Properly configured and managed SSH is still considered a secure and robust protocol for remote access and data transfer. In all those situations, a tool monitoring the SSH traffic could have detected the malicious activities and prevented the attack.

4.1.5 The Imperative of SSH Honeypots in Cybersecurity Monitoring

SH (Secure Shell) has become an indispensable protocol for secure communication but can also conceil malicous agents. This reality underscores the urgency for robust monitoring mechanisms capable of identifying suspicious activities in real-time. Among various countermeasures, SSH honeypots have emerged as a particularly effective tool for monitoring and gathering intelligence on potential threats.

An SSH honeypot is a decoy server or service that mimics legitimate SSH services. The primary aim is to attract cybercriminals and study their tactics, thereby offering an active form of surveillance and data collection. Unlike traditional intrusion detection systems, honeypots do not merely identify an attack; they engage the attacker in a controlled environment, enabling detailed observation and logging of the intruder's actions. This allows for the collection of valuable information, such as the attacker's IP address, the tools used, and the techniques employed. This data can then be used to enhance security measures and develop more robust countermeasures [31].

SSH honeypots serve as an invaluable asset in the cybersecurity arsenal, providing not just a reactive but a proactive measure against evolving cyber threats. They can collect actionable intelligence on new hacking methods, malware, and exploitation scripts. This information can be crucial for proactively securing actual production environments. The data collected can also be used to trace back to the origin of the attack, facilitating legal pursuits against the perpetrators. By diverting attackers to decoy servers, honeypots also protect real assets from being targeted, saving both computational resources and administrative effort needed for post-incident recovery.

Popular SSH honeypots include Kippo, Cowrie, and HoneySSH. Cowrie is a fork of Kippo, with additional features such as logging of attacker's keystrokes and file transfer.

- **Kippo:** Kippo is a medium-interaction honeypot that logs the attacker's shell interaction. It specializes in capturing brute force and Telnet-based attacks [31].
- **Cowrie:** Serving as Kippo's successor, Cowrie emulates various protocols including SSH, SFTP, and SCP. It logs events in JSON format, making it particularly useful for detecting brute force and Telnet-based attacks, as well as spoofing attacks [31].
- **IoTPOT:** This IoT-focused honeypot supports multiple CPU architectures and can detect a variety of attacks including brute force, DoS, and sniffing attacks on Telnet, SSH, and HTTP ports [31].
- **HoneySSH:** HoneySSH is a low-interaction honeypot that emulates an SSH server and logs the attacker's IP address, username, and password [33].
- **Sarracenia (SSHKex):** Introduced in 2018, Sarracenia is a high-interaction SSH honeypot that has been enhanced by SSHKex. Instead of "requiring the VM to be paused for every incoming or outgoing packet, which degrades the server performance" [1], SSHKex allows for the extraction of derived SSH session keys. This reduces the performance degradation significantly, as the VM is paused less frequently [1] [8].

These honeypots are useful tools for gathering intelligence on potential threats. However, they are not without their limitations. For instance, they are not able to mimic a real system, such that attackers might be able to detect them „Analysis of SSH Honeypot Effectiveness". Hence, the need for more advanced SSH honeypots that can leverage data forensic and machine learning techniques so as to be able to use directly a real server as a honeypot, without the need to emulate a system. The current master's thesis is aligned with this ongoing research, further enhancing the state of SSH honeypots.

4.2 Prior work on key extraction

4.2.1 SSHKex

SSHKex is a research project that aims to address the challenges of analyzing encrypted SSH traffic by leveraging Virtual Machine Introspection (VMI) techniques. Developed by Sentanoe and Reiser, the project focuses on extracting SSH keys and decrypting SSH network traffic in a stealthy, non-intrusive manner while maintaining evidence integrity [1]. This paper is itself a continuation of the work presented in „Sarracenia: Enhancing the Performance and Stealthiness of SSH Honeypots Using Virtual Machine Introspection“ [8], which introduced Sarracenia, a high-interaction SSH honeypot. It is also related to a range of other research projects and papers [1, section 5.6 and 6].

The SSHKex approach combines standard network traffic capturing methods with dynamic SSH session key extraction. It assumes that the SSH implementation running on the server is known, which is crucial for the key extraction process. The project employs VMI tools like LibVMI and Volatility to gain a complete and untainted view of all guest VM's state information. This allows to efficiently locate SSH session keys in the main memory of a Linux machine.

Here is a summary of the SSHKex methodology for key extraction:

1. **Data Structure Information:** The method leverages detailed knowledge about the data structures used to store the keys. Specific debugging symbols corresponding to the SSH implementation version on the target system provide essential offset values to facilitate the extraction of key material. The structures of interest include `struct ssh`, `struct session_state`, `struct newkeys`, and `struct sshenc`. These structures store a range of information such as IP addresses, ports, session states, and encryption keys.
2. **Tracing OpenSSH Functions:** Function tracing is employed to identify the precise locations of data structures and to extract keys at the right time. The focus is on two key functions: `kex_derive_keys` (which initiates key generation) and `do_authentication2` (which kicks off user authentication).
3. **Breakpoints Injection:** Software breakpoints are intentionally placed in the program execution to facilitate debugging. SSHKex utilizes Virtual Machine Introspection (VMI) to inject these breakpoints at the initial points of the two aforementioned key functions.
4. **Key Extraction:** Upon calling the *kex_derive_keys* function, SSHKex initially stores the address of the *ssh struct*. The actual keys are extracted from memory when the *do_authentication2* function is subsequently called, adhering to the known structures.
5. **Key Indexing:** OpenSSH stores client-to-server and server-to-client keys in distinct indices of the `newkeys` structure. SSHKex extracts keys based on these specific indices.
6. **Handling Multiple Connections:** To manage multiple SSH connections, OpenSSH spawns child processes. SSHKex extends its key extraction strategy to each child process by identifying them through their unique process IDs.

One of the key strengths of SSHKex is its focus on stealthiness, preservation, and evidence integrity. The approach aims to be as unobtrusive as possible, avoiding any modifications to the system under investigation. This is particularly important in forensic contexts, where the integrity of the evidence is crucial [1].

4.2.2 SmartKex

SmartKex is a direct followup project that focuses on the extraction of SSH keys from heap memory dumps. Its primary objective is to automate the process of SSH key extraction from heap memory dumps. The project introduces a machine learning-assisted methodology that significantly improves the efficiency and accuracy of key extraction compared to traditional brute-force methods. This method is also significantly more straightforward to implement compared to the previous SSHKex approach, which requires detailed knowledge of the SSH implementation and the ability to inject breakpoints into the program execution.

SmartKex discusses two distinct methods for SSH key extraction:

- **Brute-Force Baseline Method:** This is a traditional approach that scans through the heap memory to identify potential keys based on known patterns.
- **Machine Learning-Assisted Method:** This approach uses a Random Forest algorithm trained on a highly imbalanced dataset using SMOTE balancing. The machine learning model is designed to identify SSH keys with high precision and recall rates, but is not exact as compared to the brute-force method since it is based on a probabilistic model.

4.2.2.1 Baseline brute-force method

Here is a summary of SmartKex's brute-force method for SSH key extraction from heap dumps [2]:

1. **Heap Dump Generation:** Heap dump binary files of OpenSSH server process have been generated and serves as the input for the key extraction process.
2. **Data Reduction:** To minimize the heap size, the method removes memory pages that are irrelevant (empty) based on Hamming distance.
3. **Brute-force key search:** Starting from the first byte, a key length of 128 bytes is taken from the heap dump as the potential key. The algorithm iterates over the entire heap, continuously updating the potential key until the heap's end is reached.
4. **Decryption Attempt:** For every potential key, an attempt is made to decrypt network packets. If decryption fails, the process is repeated with a new potential key.

Although the brute-force approach is exact, it is computationally expensive. It performs poorly especially when keys are located at the end of the heap dump [2, section 6.2].

4.2.2.2 SmartKex machine-learning method

The real innovation of SmartKex is its machine learning-assisted methodology for SSH key extraction. At the cost fo exactness, this approach is significantly faster than the brute-force method and has a high degree of accuracy in identifying encryption keys. It also allows for the heap size to be reduced to less than 2% of its original size, further optimizing the extraction process.

Here is a summary of SmartKex's machine learning-assisted method for SSH key extraction from heap dumps [2]:

1. **Heap Dump inputs:** Similarly to the brute-force method, heap dump binary files of OpenSSH also serve as inputs for the key extraction process.

2. **Preprocessing:** The raw heap dump is resized into an $N \times 8$ matrix. High entropy parts of the heap dump, which are likely to be encryption keys, are identified using the logical AND operation on the vertical and horizontal differences of adjacent bytes. This creates an array that flags potential key locations.

3. **Training:** A Random Forest algorithm is trained on 128-byte slices of the preprocessed heap. The dataset is imbalanced, with the slices that contain keys being rare. A stacked classifier approach is used, comprising a high precision classifier and a high recall classifier.

4. **Key Identification:** The machine learning model is used to predict which 128-byte slices of the heap dump are likely to contain encryption keys. These slices are then subjected to a brute-force method to actually extract the keys.

SmartKex is significantly faster than the brute-force method alone and has a high degree of accuracy in identifying encryption keys. It also allows for the heap size to be reduced to less than 2% of its original size, further optimizing the extraction process.

SmartKex has broad applications in the field of cybersecurity, particularly in memory forensics. Its machine learning-assisted methodology can be adapted for other types of sensitive data extraction, making it a versatile tool for researchers and practitioners alike. The project is open-source, with the code available on GitHub [1].

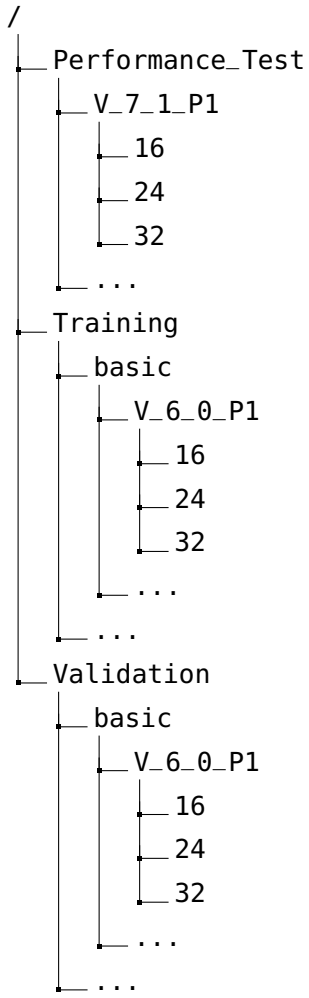
4.2.3 OpenSSH memory dumps dataset

SmartKex has contributed to the research community by generating a comprehensive annotated dataset of OpenSSH heap memory dumps [2]. The dataset is publicly available on Zenodo [2].

[1] https://github.com/smartvmi/Smart-and-Naive-SSH-Key-Extraction
[2] https://zenodo.org/record/6537904

The dataset is organized into two top-level directories: $Training$ and $Validation$ with an additional $Performance\_Test$. The first two main directories are further divided based on the SSH scenario, such as immediate exit, port-forward, secure copy, and shared connection. Each of these subdirectories is then categorized by the software version that generated the memory dump. Within these, the heaps are organized based on their key lengths, providing a multi-layered structure that aids in specific research queries.

Figure 1: Illustration of the Dataset Directory Structure



Two primary file formats are used to store the data: JSON and RAW. The JSON files contain meta-information like the encryption method, virtual memory address of the key, and the key's value in hexadecimal representation. The RAW files, on the other hand, contain the actual heap dump of the OpenSSH process.

Here is an example of content of a RAW memory dump file, displayed using *vim* and *xdd* commands:

```
00000000: 0000 0000 0000 0000 5102 0000 0000 0000 ........Q.......
00000010: 0607 0707 0707 0303 0200 0006 0401 0206 ................
00000020: 0200 0001 0100 0107 0604 0100 0000 0203 ................
00000030: 0103 0101 0000 0000 0000 0000 0000 0002 ................
00000040: 0001 0000 0000 0000 0000 0100 0000 0001 ................
00000050: 8022 1a3a 3456 0000 007f 1a3a 3456 0000 .":4V.....:4V..
00000060: f040 1a3a 3456 0000 9032 1a3a 3456 0000 .@.:4V...2.:4V..
00000070: 608b 1a3a 3456 0000 9047 1a3a 3456 0000 `..:4V...G.:4V..
```

Listing 1: Hex Dump from *Training/basic/V_7_8_P1/16/5070-1643978841-heap.raw*

The original file contains the raw byte content of the heap dump of a specific version of OpenSSH. It is a binary file, which means that it is not human-readable. However, it can be converted to a human-readable format using the *xxd* command. The first column to the left represents the offset in hexadecimal. The last column represents the actual content of the bytes, in ASCII format. The columns in between represent the content of the bytes in hexadecimal format.

Since hexadecimal is a base-16 number system, each byte is represented by two hexadecimal digits. The ASCII representation of the bytes is displayed on the right, and is only used for reference, as it is not always possible to convert the bytes to ASCII. For instance, the bytes at offset 0x10 are not printable characters, and thus cannot be converted to ASCII. Each line represents 16 bytes, and the offset is incremented by 16 for each line.

For the purpose of this thesis, it will be more interesting to visualize the content of the heap dump as 8 bytes lines. This can be achieved by using the *xxd* command with the *-c* option, as shown in the following example:

The same example as before, a memory dump file, displayed using *vim* and *xdd -c 8* commands:

```
00000000: 0000 0000 0000 0000 ........
00000008: 5102 0000 0000 0000 Q.......
00000010: 0607 0707 0707 0303 ........
00000018: 0200 0006 0401 0206 ........
00000020: 0200 0001 0100 0107 ........
00000028: 0604 0100 0000 0203 ........
00000030: 0103 0101 0000 0000 ........
00000038: 0000 0000 0000 0002 ........
00000040: 0001 0000 0000 0000 ........
00000048: 0000 0100 0000 0001 ........
00000050: 8022 1a3a 3456 0000 ."..:4V..
00000058: 007f 1a3a 3456 0000 ...:4V..
00000060: f040 1a3a 3456 0000 .@.:4V..
00000068: 9032 1a3a 3456 0000 .2.:4V..
00000070: 608b 1a3a 3456 0000 `..:4V..
00000078: 9047 1a3a 3456 0000 .G.:4V..
```

Listing 2: Hex Dump

This example shows the exact content of the preceding one.

To this RAW file is associated a JSON file, which contains its annotations.

Here is a example of content of a JSON annotation file that comes with the previous RAW file:


```
        {
           "SSH_PID": "5070",
           "SSH_STRUCT_ADDR": "56343a1a4800",
           "session_state_OFFSET": "0",
           "SESSION_STATE_ADDR": "56343a1a8d30",
           "newkeys_OFFSET": "344",
           "NEWKEYS_1_ADDR": "56343a1aaa40",
           "NEWKEYS_2_ADDR": "56343a1aab40",
           "enc_KEY_OFFSET": "0",
           "mac_KEY_OFFSET": "48",
           "name_ENCRYPTION_KEY_OFFSET": "0",
           "ENCRYPTION_KEY_1_NAME_ADDR": "56343a1a9db0",
           "ENCRYPTION_KEY_1_NAME": "aes128-gcm@openssh.com",
           "ENCRYPTION_KEY_2_NAME_ADDR": "56343a1a3fb0",
           "ENCRYPTION_KEY_2_NAME": "aes128-gcm@openssh.com",
           "key_ENCRYPTION_KEY_OFFSET": "32",
           "key_len_ENCRYPTION_KEY_OFFSET": "20",
           "iv_ENCRYPTION_KEY_OFFSET": "40",
           "iv_len_ENCRYPTION_KEY_OFFSET": "24",
           "KEY_A_ADDR": "56343a1a3170",
           "KEY_A_LEN": "12",
           "KEY_A_REAL_LEN": "12",
           "KEY_A": "feb5fd4ef0759b034d69b858",
           "KEY_B_ADDR": "56343a1a33e0",
           "KEY_B_LEN": "12",
           "KEY_B_REAL_LEN": "12",
           "KEY_B": "f50b988297fa19709445c4ee",
           "KEY_C_ADDR": "56343a1aa1b0",
           "KEY_C_LEN": "16",
           "KEY_C_REAL_LEN": "16",
           "KEY_C": "f5b53280e944db0fe196668d877cd4c0",
           "KEY_D_ADDR": "56343a1a4010",
           "KEY_D_LEN": "16",
           "KEY_D_REAL_LEN": "16",
           "KEY_D": "ac4f18a963d9e72c857497b7dc9d088d",
           "KEY_E_ADDR": "56343a1a7d90",
           "KEY_E_LEN": "0",
           "KEY_E_REAL_LEN": "0",
           "KEY_E": "",
           "KEY_F_ADDR": "56343a1a2f60",
           "KEY_F_LEN": "0",
           "KEY_F_REAL_LEN": "0",
           "KEY_F": "",
           "HEAP_START": "56343a198000"
        }
```


Listing 3: Complete JSON example, from *Training/basic/V_7_8_P1/16/5070-1643978841.json*

Those annotation files contain the meta-information about the heap dump, such as the encryption method, virtual memory address of the key, and the key's value in hexadecimal representation. Those annotations are invaluable for the development of machine learning models for key extraction.

However, it is worth noting that the dataset is not perfect, as some of the annotations are sometimes missing. This is probably due to the fact that the annotations were generated automatically, and some of the keys were not correctly identified. For instance, in *Training/basic/V_7_8_P1/16/*,

literally the first file of the dataset contains an incorrect annotation file, as some of the keys are missing. This is a limitation of the dataset that should be kept in mind when using it for research purposes.

Here is an example of content of a JSON annotation file with missing keys:

```
        {
           "ENCRYPTION_KEY_NAME": "aes128-ctr",
           "ENCRYPTION_KEY_LENGTH": "16",
           "KEY_C": "689e549a80ce4be95d8b742e36a229bf",
           "KEY_D": "76788e66a56d2b61eec294df37422fcb",
           "HEAP_START": "5589d41e0000"
        }
```

Listing 4: Missing keys in JSON annotation file *Training/basic/V_6_0_P1/16/24375-1644243522.json*

The dataset is not just limited to SSH key extraction; it also serves as a resource for identifying essential data structures that hold sensitive information. This makes it a versatile tool for various research applications, including but not limited to machine learning models for key extraction and malware detection.

4.3 Graph-based memory modelization

4.3.1 Defining memory concepts and modelization

4.3.2 Graphs and Knowledge Graphs

4.4 Data processing for Machine Learning

4.4.1 Feature engineering

4.4.2 Graph-based embeddings

4.4.3 Dataset splitting and sampling

4.5 Machine Learning and Deep Learning

4.5.1 Machine Learning

4.5.2 Machine Learning models

4.5.3 Deep Learning

4.5.4 GCN

5 Methods

Describe the method/software/tool/algorithm you have developed here

Dataset in Methods Chapter: On the other hand, if the specific features and challenges of your dataset are more related to your methodology—for example, how you cleaned, balanced, or sampled the data—then these details would fit well into the Methods chapter. This is especially relevant if the issues tackled are more about "how-to" rather than "why."

6 Results

Describe the experimental setup, the used datasets/parameters and the experimental results achieved

7 Discussion

Discuss the results. What is the outcome of your experimetns?

8 Conclusion

Summarize the thesis and provide a outlook on future work.

A Code

B Math

C Dataset

Acronyms

ESS Estimated Security Strength. 6

GCN Graph Convolutional Networks. 3

KG Knownledge Graph. 3

ML Machine Learning. 3

SMOTE Synthetic Minority Over-sampling Technique. 11

SSH Secure Shell Protocol. i, 3, 6

VMI Virtual Machine Introspection. 10

References

[1] Stewart Sentanoe and Hans P. Reiser. „SSHkex: Leveraging virtual machine introspection for extracting SSH keys and decrypting SSH network traffic“. en. In: *Forensic Science International: Digital Investigation* 40 (2022), p. 301337. doi: 10.1016/j.fsidi.2022.301337. url: https://linkinghub.elsevier.com/retrieve/pii/S2666281722000063.

[2] Christofer Fellicious et al. „SmartKex: Machine Learning Assisted SSH Keys Extraction From The Heap Dump“. In: arXiv:2209.05243 (Sept. 2022). arXiv:2209.05243 [cs]. doi: 10.48550/arXiv.2209.05243. url: http://arxiv.org/abs/2209.05243.

[3] M. Baushke. *Key Exchange (KEX) Method Updates and Recommendations for Secure Shell (SSH)*. RFC 9142. Updates: 4250, 4253, 4432, 4462; Errata exist. Internet Engineering Task Force (IETF), Jan. 2022.

[4] Nicole Perlroth, Jeff Larson, and Scott Shane. „N.S.A. Able to Foil Basic Safeguards of Privacy on Web“. In: *The New York Times* (Sept. 2013). url: https://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html.

[5] James Ball, Julian Borger, and Glenn Greenwald. „Revealed: how US and UK spy agencies defeat internet privacy and security“. In: *The Guardian* (Sept. 2013). Published at 11.24 BST. url: https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security.

[6] Aris Adamantiadis. *OpenSSH introduces curve25519-sha256@libssh.org key exchange !* Retrieved 2023-09-05. Nov. 2013. url: https://www.libssh.org/2013/11/03/openssh-introduces-curve25519-sha256libssh-org-key-exchange/.

[7] OpenSSH. *OpenSSH 5.7 release notes*. Retrieved 2022-11-13. Jan. 2011. url: https://www.openssh.com/txt/release-5.7.

[8] Stewart Sentanoe, Benjamin Taubmann, and Hans P. Reiser. „Sarracenia: Enhancing the Performance and Stealthiness of SSH Honeypots Using Virtual Machine Introspection“. In: *Lecture Notes in Computer Science*. Vol. 11252. Conference paper. Nov. 2018. url: https://link.springer.com/chapter/10.1007/978-3-030-03638-6_16.

[9] OpenSSH. *OpenSSH 6.5 release notes*. Retrieved 2022-11-13. Jan. 2014. url: https://www.openssh.com/txt/release-6.5.

[10] OpenSSH. *OpenSSH 7.0 release notes*. Retrieved 2022-11-13. Aug. 2015. url: https://www.openssh.com/txt/release-7.0.

[11] OpenSSH. *OpenSSH 7.2 release notes*. Retrieved 2022-11-13. Jan. 2016. url: https://www.openssh.com/txt/release-7.2.

[12] OpenSSH. *OpenSSH 8.2 release notes*. Retrieved 2022-11-13. Feb. 2020. url: https://www.openssh.com/txt/release-8.2.

[13] OpenSSH. *OpenSSH 8.8 release notes*. Retrieved 2022-11-13. Sept. 2021. url: https://www.openssh.com/txt/release-8.2.

[14] OpenSSH. *OpenSSH 9.0 release notes*. Retrieved 2022-11-13. Apr. 2022. url: https://www.openssh.com/txt/release-9.0.

[15] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251. Updated by: 8308, 9141. Network Working Group, Jan. 2006.

[16] T. Ylonen and C. Lonvick. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253. Updated by: 6668, 8268, 8308, 8332, 8709, 8758, 9142; Errata Exist. Network Working Group, Jan. 2006.

[17] Girish Venkatachalam. „The OpenSSH Protocol under the Hood“. In: *Linux Journal* 156 (Apr. 2007). url: https://www.ecb.torontomu.ca/~courses/coe518/LinuxJournal/elj2007-156-OpenSSH.pdf.

[18] Oscar M. Guillen et al. „Towards post-quantum security for IoT endpoints with NTRU“. In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. 2017, pp. 698–703. doi: 10.23919/DATE.2017.7927079.

[19] P. McLaren et al. „Decrypting live SSH traffic in virtual environments“. In: *Digital Investigation* 29 (2019), pp. 109–117. url: https://www.sciencedirect.com/science/article/abs/pii/S1742287619300647.

[20] Tatu Ylonen. *Portable OpenSSH*. Github repositor. Accessed on 25.08.2023. 1995. url: https://github.com/openssh/openssh-portable/.

[22] José Tomás Martínez Garre, Manuel Gil Pérez, and Antonio Ruiz-Martínez. „A novel Machine Learning-based approach for the detection of SSH botnet infection“. In: *Future Generation Computer Systems* 115 (Feb. 2021), pp. 387–396. doi: 10.1016/j.future.2020.09.004. url: https://www.sciencedirect.com/science/article/pii/S0167739X20303265.

[23] SSH Communications Security. *SSH Annual Report 2018*. Annual Report. Accessed: 2023-08-30. SSH Communications Security, 2018. url: https://info.ssh.com/hubfs/2021%20Investor%20documents/SSH_Annual_Report_2018_final.pdf.

[24] W. Yurcik and Chao Liu. „A first step toward detecting SSH identity theft in HPC cluster environments: discriminating masqueraders based on command behavior“. In: *CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid, 2005.* Vol. 1. May 2005, 111–120 Vol. 1. doi: 10.1109/CCGRID.2005.1558542.

[25] *Weak CRC allows packet injection into SSH sessions encrypted with block ciphers*. Accessed: 2023-08-30. Nov. 2001. url: https://www.kb.cert.org.

[26] Core Security Technologies. *SSH Insertion Attack*. https://www.coresecurity.com/core-labs/advisories/ssh-insertion-attack. Archived from the original on 2011-07-08. 2023.

[27] US CERT. *SSH CBC vulnerability. Vulnerability Note VU#958563 - SSH CBC vulnerability*. https://www.kb.cert.org/vuls/id/958563. Archived from the original on 2011-06-22. 2011.

[28] Spiegel Online. *Prying Eyes: Inside the NSA's War on Internet Security*. Spiegel Online. Archived from the original on January 24, 2015. 2014. url: https://www.spiegel.de/international/germany/inside-the-nsa-s-war-on-internet-security-a-1010361.html.

[29] Olivier Bilodeau et al. *Operation WINDIGO*. en. Mar. 2014, p. 69. url: https://web-assets.esetstatic.com/wls/2014/03/operation_windigo.pdf.

[30] Pooneh Nikkhah Bahrami* et al. „Cyber Kill Chain-Based Taxonomy of Advanced Persistent Threat Actors: Analogy of Tactics, Techniques, and Procedures“. In: *Journal of Information Processing Systems* 15.4 (Nov. 2019), pp. 865–889. doi: 10.3745/JIPS.03.0126. url: http://xml.jips-k.org/full-text/view?doi=10.3745/JIPS.03.0126.

[31] Sanjay Madan and Monika Singh. „Classification of IOT-Malware using Machine Learning“. In: *2021 International Conference on Technological Advancements and Innovations (ICTAI)*. Nov. 2021, pp. 599–605. doi: 10.1109/ICTAI53825.2021.9673185.

[32] Connor Hetzler, Zachary Chen, and Tahir M. Khan. „Analysis of SSH Honeypot Effectiveness“. en. In: *Advances in Information and Communication*. Ed. by Kohei Arai. Lecture Notes in Networks and Systems. Cham: Springer Nature Switzerland, Mar. 2023, pp. 759–782. isbn: 9783031280733. doi: 10.1007/978-3-031-28073-3_51.

[33] ppacher. *honeyssh*. https://github.com/ppacher/honeyssh. GitHub repository. 2017.

Additional bibliography

[21] Auguste Kerckhoffs. „La cryptographic militaire“. In: *Journal des sciences militaires* (1883), pp. 5–38.

[34] Dimitrios Georgoulias et al. „Botnet Business Models, Takedown Attempts, and the Darkweb Market: A Survey“. en. In: *ACM Computing Surveys* 55.11 (Nov. 2023), pp. 1–39. doi: 10.1145/3575808. url: https://dl.acm.org/doi/10.1145/3575808.

[35] Aidan Hogan et al. „Knowledge Graphs“. In: *ACM Comput. Surv.* 54.4 (July 2021). issn: 0360-0300. doi: 10.1145/3447772. url: https://doi.org/10.1145/3447772.

[36] Paul Groth et al. „Knowledge Graphs and their Role in the Knowledge Engineering of the 21st Century“. In: *Dagstuhl Reports* 12.9 (2022). Report from Dagstuhl Seminar 22372. Specific usage: pp. 60-72, Subsection "3.2 A Brief History of Knowledge Engineering: A Practitioner’s Perspective", pp. 60–120. doi: 10.4230/DagRep.12.9.60.

[37] Marvin Hofer et al. „Construction of Knowledge Graphs: State and Challenges“. In: *arXiv preprint arXiv:2302.11509* (2023). url: https://doi.org/10.48550/arXiv.2302.11509.

[38] Lisa Ehrlinger and Wolfram Wöß. „Towards a Definition of Knowledge Graphs“. In: (2016), pp. 1–4.

[39] Frederick Edward Hulme. *Proverb Lore: Many Sayings, Wise Or Otherwise, on Many Subjects, Gleaned from Many Sources*. E. Stock, 1902, p. 188.

[40] Google. „Introducing the Knowledge Graph: Things, not strings“. In: *Google Blog* (May 2012). Accessed: 2023-06-16. url: https://blog.google/products/search/introducing-knowledge-graph-things-not/.

[41] Michelle Venables. *An Introduction to Graph Theory*. Accessed: 2023-06-12. 2019. url: https://towardsdatascience.com/an-introduction-to-graph-theory-24b41746fabe.

[42] Gianluca Fiorelli. *Best of 2013: No 13 - Search in the Knowledge Graph era*. Accessed: 2023-06-12. 2013. url: https://www.stateofdigital.com/search-in-the-knowledge-graph-era/.

[43] Jackson Gilkey. *Graph Theory and Data Science*. Accessed: 2023-05-25. 2019. url: https://towardsdatascience.com/graph-theory-and-data-science-ec95fe2f31d8.

[44] M.S. Jawad et al. „Adoption of knowledge-graph best development practices for scalable and optimized manufacturing processes“. In: *MethodsX* 10 (2023), p. 102124. issn: 2215-0161. doi: https://doi.org/10.1016/j.mex.2023.102124. url: https://www.sciencedirect.com/science/article/pii/S2215016123001255.

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie, dass ich die Masterarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, September 6, 2023

_____

Rascoussier, Florian Guillaume Pierre