Masterarbeits

Structure embeddings for OpenSSH heap dump analysis

A report by

Lahoche, Clément Claude Martial

Prüfer

Prof. Dr. Michael Granitzer

Christofer Fellicious

Prof. Dr. Pierre-Edouard Portier

August 29, 2023

Abstract

Acknowledgements

Contents

List of Figures

List of Tables

1 Introduction

Digital forensics is a linchpin in cybersecurity, enabling the extraction of vital evidence from devices like PCs. This evidence is key for detecting malware and tracing intruder activities. Analyzing a device's main memory is a go-to technique in this field. The fusion of machine learning promises to amplify and streamline these analyses.

With the rising need for encrypted communication, Secure Shell (SSH) protocols are now commonplace. However, these security-focused channels can inadvertently shield malicious actions, posing challenges to standard investigative approaches. Cutting-edge research offers solutions. The work in *SmartKex: Machine Learning Assisted SSH Keys Extraction From The Heap Dump* [3] highlights how machine learning can boost the extraction of session keys from OpenSSH memory images. In a complementary vein, „SSHkex: Leveraging virtual machine introspection for extracting SSH keys and decrypting SSH network traffic“ [9] showcases the power of Virtual Machine Introspection (VMI) for direct SSH key extraction.

Inspired by *SmartKex: Machine Learning Assisted SSH Keys Extraction From The Heap Dump*, this thesis zeroes in on a central challenge: data embedding. While previous studies set the stage for key extraction, the data embedding technique, especially windowing, can be optimized. The design of data embeddings is pivotal for machine learning efficacy, especially in nuanced tasks like memory analysis. This research introduces fresh embedding strategies, aiming to refine extraction and unearth deeper memory snapshot patterns. Merging graph embeddings with advanced machine learning, the goal is to craft a sophisticated toolkit for OpenSSH heap dump studies, bridging digital forensics and machine learning.

2 Research Questions

Write down and explain your research questions (2-5)

3 Structure of the Thesis

Explain the structure of the thesis.

4 Background

In the complex world of cybersecurity and digital forensics, innovative approaches are crucial for revealing hidden or encrypted information. OpenSSH stands out as a key instrument for ensuring secure communication. The memory snapshots, or heap dumps, of OpenSSH are treasure troves of data. Through graph generation from these dumps, we can uncover the detailed connections between data structures, identified by their malloc headers, and their associated pointers.

This research delves deep into the smart embedding of these connections, aiming to use machine learning classifiers to identify structures that contain OpenSSH keys. The journey is not just about representing data through graphs but also about understanding the raw sequences of bytes in the heap dump. Classical techniques like Shannon entropy, Byte Frequency Distribution (BFD), and bigram frequencies provide foundational knowledge. However, the rapidly evolving domain of deep learning opens up a plethora of avenues. Models such as Recurrent Neural Networks (RNN) [7] (Long Short-Term Memory (LSTM)[6] and Gated Recurrent Units (GRU)[2]) and sequence-to-sequence learning [11] offer unique perspectives on raw byte embedding. Furthermore, the efficacy of convolutional approaches (CNN), both standalone[8] and in conjunction with recurrent networks, for sequence modeling is well-documented [1]. Notably, the application of neural networks in file fragment classification, especially with lossless representations, has shown promising results [5].Finally, we will dive into transformers and autoencoders.

The aim of this background section is to provide a comprehensive overview of graph creation from heap dumps, techniques for raw byte embedding, and their role in identifying OpenSSH key structures. By merging age-old techniques with modern approaches, we strive to highlight the most effective methods for analyzing OpenSSH heap dump.

4.1 Graph Generation from Heap Dumps

4.1.1 Secure Shell (SSH)

„The Secure Shell (SSH) is designed to enable encrypted communication across potentially unsecured networks, ensuring the confidentiality of data during transmission. Each SSH session utilizes a specific set of session keys, encompassing six distinct keys:

- **Key A:** Client-to-server initialization vector (IV)
- **Key B:** Server-to-client initialization vector (IV)
- **Key C:** Client-to-server encryption key (EK)
- **Key D:** Server-to-client encryption key (EK)
- **Key E:** Client-to-server integrity key
- **Key F:** Server-to-client integrity key

To decrypt the encrypted traffic within an SSH session, knowledge of the IV and EK pair (either Key A with Key C or Key B with Key D) is essential, assuming the presence of passive network monitoring tools. OpenSSH, a prevalent implementation of SSH, is the primary subject of this research, covering versions from V6_0P1 to V8_8P1. OpenSSH incorporates various encryption methodologies, including Advanced Encryption Standard (AES) Cipher Block Chaining (CBC), AES Counter (AES-CTR), and ChaCha20, with IV and EK key lengths varying between 12 and 64 bytes.“

This information is derived from the paper titled *SmartKex: Machine Learning Assisted SSH Keys Extraction From The Heap Dump* [3].

4.1.2 heap dumps of OpenSSH

„Heap memory, distinct from local stack memory, is a dynamic memory allocation mechanism. While local stack memory is responsible for storing and deallocating local variables during function calls, heap memory requires explicit memory allocation and deallocation. This is achieved using operators such as `new` in Java and C++, or `malloc/calloc` in C.

OpenSSH, which is primarily written in C, employs `calloc` for memory block allocation. These blocks are designated to store session-related data, including the cryptographic keys. By leveraging this knowledge, one can deduce that if the heap of an active OpenSSH process is dumped at an opportune moment (for instance, during an ongoing SSH session), the resulting heap dump will encompass the SSH session keys.“

This information is also derived from the paper titled *SmartKex: Machine Learning Assisted SSH Keys Extraction From The Heap Dump* [3].

4.1.3 Dataset

„We use SSHKex[9] as the primary method to extract the SSH keys from the main memory. In addition, we add two features to SSHKex: automatically dump OpenSSH's heap and add support for SSH client monitoring.

For this paper, we are using four SSH scenarios: the client connects to the server and exits immediately, port-forward, secure copy, and SSH shared connection. Two file formats, JSON and RAW, are utilized to store the generated logs. The JSON log file encompasses meta-information, including the encryption name, the virtual memory address of a key, and the key's value in hexadecimal representation (as depicted in Figure 1). Conversely, the binary file captures the heap dump of the OpenSSH process (illustrated in Figure 2 using the `xxd` command).

```
(base) [onyr@kenzael phdtrack_data]$ cat ./Training/Training/scp
/V_7_8_P1/16/1010-1644391327.json | json_pp
{
   "ENCRYPTION_KEY_1_NAME" : "aes128-ctr",
   "ENCRYPTION_KEY_1_NAME_ADDR" : "558b967f7620",
   "ENCRYPTION_KEY_2_NAME" : "aes128-ctr",
   "ENCRYPTION_KEY_2_NAME_ADDR" : "558b967fb160",
   "HEAP_START" : "558b967e9000",
   "KEY_A" : "119bd34f49d27bbbc0f9af400d4edc39",
   "KEY_A_ADDR" : "558b967fefe0",
   "KEY_A_LEN" : "16",
   "KEY_A_REAL_LEN" : "16",
   "KEY_B" : "8a77835eb2007a46a776ae0c183253b9",
   "KEY_B_ADDR" : "558b967f5ce0",
   "KEY_B_LEN" : "16",
   "KEY_B_REAL_LEN" : "16",
   "KEY_C" : "528f6dbd2907b3b4cfbd02fb32b852e7",
   "KEY_C_ADDR" : "558b967f51f0",
   "KEY_C_LEN" : "16",
   "KEY_C_REAL_LEN" : "16",
   "KEY_D" : "427f04149eed7029f031e58f3fde9844",
   "KEY_D_ADDR" : "558b967fb180",
   "KEY_D_LEN" : "16",
   "KEY_D_REAL_LEN" : "16",
   "KEY_E" : "17b6c799b5639ce5ea60c7f67cf6177f",
   "KEY_E_ADDR" : "558b967ff070",
   "KEY_E_LEN" : "16",
   "KEY_E_REAL_LEN" : "16",
   "KEY_F" : "fb75f5776184794ca92624ec6a36fd62",
   "KEY_F_ADDR" : "558b967f3d90",
   "KEY_F_LEN" : "16",
   "KEY_F_REAL_LEN" : "16",
   "NEWKEYS_1_ADDR" : "558b96800fd0",
   "NEWKEYS_2_ADDR" : "558b967fef10",
   "SESSION_STATE_ADDR" : "558b967f7f30",
   "SSH_PID" : "1010",
   "SSH_STRUCT_ADDR" : "558b967f6c20",
   "enc_KEY_OFFSET" : "0",
   "iv_ENCRYPTION_KEY_OFFSET" : "40",
   "iv_len_ENCRYPTION_KEY_OFFSET" : "24",
   "key_ENCRYPTION_KEY_OFFSET" : "32",
   "key_len_ENCRYPTION_KEY_OFFSET" : "20",
   "mac_KEY_OFFSET" : "48",
   "name_ENCRYPTION_KEY_OFFSET" : "0",
   "newkeys_OFFSET" : "344",
   "session_state_OFFSET" : "0"
}
```

Figure 1: Json exemple

```
000159d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000159e0: 0000 0008 0000 0000 0080 0000 0000 0000  ................
000159f0: 0000 0000 0000 0000 0100 0000 0000 0000  ................
00015a00: 0000 0000 0000 0000 2100 0000 0000 0000  ........!.......
00015a10: 8d08 ff65 b3bf cd8b 91ca 995a d5b7 64af  ...e.......Z..d.
00015a20: 0000 0000 0000 0000 2100 0000 0000 0000  ........!.......
00015a30: 756d 6163 2d36 342d 6574 6d40 6f70 656e  umac-64-etm@open
00015a40: 7373 682e 636f 6d00 5100 0000 0000 0000  ssh.com.Q.......
00015a50: b0d4 36d2 a655 0000 b0d4 36d2 a655 0000  ..6..U....6..U..
```

Figure 2: Xxd exemple

The dataset is structured into two primary directories: `training` and `validation`. Each of these directories is further segmented into subdirectories reflecting the specific scenario, such as OpenSSH, port-forwarding, or secure copy (SCP).

Subdirectories under OpenSSH or SCP are categorized based on the software version responsible for the memory dump. These directories are further organized by the software version that generated the memory dump. The heaps are then classified based on their key lengths, with each key length possessing its dedicated directory beneath the version directory. These version-specific directories are further divided based on the different key lengths present in a heap.

Accompanying every raw memory dump is a JSON file, distinguished by the same alphanumeric sequence, barring the "-heap" suffix. This JSON file encapsulates various encryption keys and additional metadata, such as the process ID and the offset of the heap. Consequently, the dataset's utility is not confined to extracting session keys but also extends to identifying crucial data structures harboring sensitive information. The dataset, along with the associated code and tools, is open-sourced. The dataset is accessible via a Zenodo repository[1]. The code can be found in a public GitHub repository[2]."

This data is the same as the data used in the paper titled *SmartKex: Machine Learning Assisted SSH Keys Extraction From The Heap Dump* [3].

4.1.4 Entropy's Role in SSH Key Identification

Encryption keys[3] inherently consist of predominantly random byte sequences. This characteristic stems from the foundational principle of ensuring security through transparency, which guarantees their high entropy. The paper explores the nuances of pinpointing these keys in memory dumps, underscoring the significance of entropy in this endeavor. This particularity can be used to identify the keys in the memory dump.

4.1.5 Definitions : Structures, Pointers, and the role of malloc headers

Through the use of the regular expressions (REGEX) `"[0-9a-f]{12}0{4}"`, we identified potential pointers within the dump. This heuristic approach acts as a sieve, filtering the extensive data to spotlight possible pointer candidates. Nonetheless, it's crucial to understand that while many pointers might be correctly pinpointed, some detected sequences may not be authentic pointers.

One notable characteristic of the heap dump is the *malloc header* found at the start of allocated structures. This header, often the initial non-null bytes in a series, signifies the size of the following structure. By sequentially reading the heap dump and identifying these headers, it becomes feasible to determine the dimensions and limits of every allocated structure, thereby methodically dividing the heap dump into distinct structures.

[1] `https://zenodo.org/record/6537904`
[2] Link to the GitHub repository

4.2 Traditional Statistical Embedding

Within the domain of machine learning, how data is represented significantly impacts the performance of models. Even though traditional statistical embedding techniques have been around before many contemporary methods, they continue to be vital in readying data for machine learning endeavors. Rooted in statistical foundations, these techniques provide a methodical approach to transform raw data into concise and meaningful forms. In this subsection, we'll delve into the nuances of entropy and its role in byte sequence embedding, Byte Frequency Distribution (BFD), and also highlight other classical statistical embedding methods pivotal in data representation for machine learning.

4.2.1 Entropy and its application in byte sequence embedding

Entropy, a fundamental concept in information theory, quantifies the amount of uncertainty or randomness associated with a set of data. Introduced by Claude Shannon in his groundbreaking work [10], entropy serves as a measure of the average information content one can expect to gain from observing a random variable's value.

Mathematically, the entropy $H(X)$ of a discrete random variable X with possible values $\{x_1, x_2, \ldots, x_n\}$ and probability mass function $P(X)$ is given by:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i) \tag{1}$$

Within the scope of identifying SSH keys, the significance of entropy cannot be understated. Byte sequences exhibiting high entropy typically reflect a multifaceted and varied informational content, traits that are synonymous with encryption keys, especially those in SSH. Sequences with pronounced entropy are often prime contenders for SSH keys due to their inherent randomness and lack of predictability, mirroring the attributes of robust security keys.

Fundamentally, entropy acts as a quantitative tool to evaluate the depth of information within data. When applied to SSH, it suggests that data sequences with elevated entropy levels have a heightened probability of correlating with secure keys. This positions entropy as an essential instrument for pinpointing and authenticating SSH keys.

4.2.2 Byte Frequency Distribution (BFD)

In the complex world of raw byte embedding, Byte Frequency Distribution (BFD) and n-gram embedding stand out as essential methods, each bringing unique benefits to data representation. BFD zeroes in on the distribution of individual byte values in a raw byte sequence. Analyzing these distributions allows for the identification of patterns that reflect the inherent nature of the data. This embedding technique becomes particularly relevant when assessing the randomness or structure of byte sequences, such as when detecting encrypted data or pinpointing specific file signatures.

On the other hand, n-gram embedding dives deeper into raw byte sequences. Instead of focusing solely on individual bytes, it captures patterns formed by sequences of 'n' consecutive bytes. This approach garners a wider range of contextual information from the raw byte data. For example, a trigram (3-gram) examines patterns formed by three sequential bytes, providing a richer representation than single byte values. Yet, a challenge with n-gram embedding is the potential for the output vector size to grow exponentially as 'n' increases, posing computational and storage issues, especially in real-time scenarios.

In the realm of raw byte embedding, both BFD and n-gram techniques offer invaluable perspectives. While BFD establishes a base representation centered on individual byte frequencies, n-gram embedding enhances it by spotlighting the complex relationships and patterns among consecutive bytes. Together, they form a robust arsenal for representing and analyzing raw byte data in a variety of applications.

4.2.3 Other traditional statistical embedding techniques

Mean Byte Value The Mean Byte Value represents the average value of all bytes in a given sequence. It provides an insight into the central tendency of the byte values in the sequence. Mathematically, for a byte sequence B of length n:

$$\text{Mean Byte Value} = \frac{1}{n}\sum_{i=1}^{n} B_i \tag{2}$$

Mean Absolute Deviation (MAD) MAD measures the average distance of each byte value from the mean, providing a sense of the dispersion or spread of the byte values around the mean. It is given by:

$$\text{MAD} = \frac{1}{n}\sum_{i=1}^{n} |B_i - \text{Mean Byte Value}| \tag{3}$$

Standard Deviation Standard Deviation quantifies the amount of variation or dispersion in the byte sequence. A higher value indicates greater variability in the byte values. It is defined as:

$$\text{Standard Deviation} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(B_i - \text{Mean Byte Value})^2} \tag{4}$$

Skewness Skewness measures the asymmetry of the distribution of byte values around the mean. A positive value indicates a distribution that is skewed to the right, while a negative value indicates a distribution skewed to the left. It provides insights into the shape of the distribution of byte values. It is given by:

$$\text{Skewness} = \frac{n}{(n-1)(n-2)}\sum_{i=1}^{n}\left(\frac{B_i - \text{Mean Byte Value}}{\text{Standard Deviation}}\right)^3 \tag{5}$$

Kurtosis Kurtosis measures the "tailedness" of the distribution of byte values. A higher kurtosis value indicates a distribution with heavier tails, while a lower value indicates lighter tails. It provides insights into the extremities of the distribution. It is defined as:

$$\text{Kurtosis} = \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=1}^{n} \left(\frac{B_i - \text{Mean Byte Value}}{\text{Standard Deviation}} \right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)} \tag{6}$$

n-gram on Bits When applying n-gram techniques to bits instead of bytes, we focus on sequences of 'n' consecutive bits. For example, a 2-gram on bits would consider patterns formed by two consecutive bits, resulting in four possible combinations: 00, 01, 10, and 11. This approach significantly reduces the size of the output vector compared to byte-based n-grams. By focusing on bits, we can capture more granular patterns in the data while benefiting from a more compact representation, which is computationally efficient and requires less storage.

4.3 Deep Learning Models for Raw Byte Embedding

In the area of data representation, deep learning is great for understanding raw byte sequences. Just like these models are good at understanding text, they're also good at understanding raw bytes. They can learn and show sequences on their own, which is really helpful for both text and raw bytes. In this section, we'll look at different deep learning models and how they work with raw byte embedding.

We'll start with Recurrent Neural Networks (RNN). Just like they're good with word sequences in text, Recurrent Neural Networks (RNN) are also good with raw byte sequences. Then, we'll look at Convolutional Neural Networks (CNN), which can find patterns in raw bytes, just like they find patterns in text. After that, we'll talk about Autoencoders, which can learn in a special way. To finish this section, we'll discuss Transformers. They're good at understanding data over a long time, similar to how they understand text.

4.3.1 RNNs : Understanding sequence data

Recurrent Neural Networks (RNN) are great tools for text classification. They're good at understanding the deeper meanings in text. Unlike older models that use hand-made features, RNN can learn and show sequences on their own. This makes them really useful for tasks that deal with sequences. When we think about embedding raw bytes, RNN's skill in understanding sequences is similar to how they handle word sequences in text. Here is a list of different RNN models and their advantages and disadvantages.

Recurrent Convolutional Neural Network (RCNN) for Text Classification[7]: The RCNN model, as discussed in the paper by Lai et al., is designed specifically for text classification. Unlike traditional models, RCNN do not rely on handcrafted features. Instead, they employ a recurrent structure to capture contextual information about words. This approach is believed to introduce considerably less noise compared to traditional window-based neural networks. The model's bidirectional

structure ensures that both preceding and succeeding contexts of a word are considered, enhancing its understanding of the word's semantics.

- **Advantages:**
  - No need for handcrafted features.
  - Captures richer contextual information.
  - less noisy.
- **Disadvantages:**
  - Complexity due to bidirectional structure.
  - Might require more computational resources.

;

Long Short-Term Memory (LSTM)[6]: The LSTM, introduced by Hochreiter and Schmidhuber, is a specialized form of RNN designed to combat the vanishing gradient problem inherent in traditional RNN. The vanishing gradient problem arises when gradients of the loss function, which are used to update the network's weights, become too small for effective learning. This typically happens in deep networks or when processing long sequences, causing the earlier layers or time steps to receive minimal updates. As a result, traditional RNN struggle to learn long-term dependencies in the data.

LSTM address this issue with their unique cell state and gating mechanisms. The cell state acts as a "conveyor belt" that can carry information across long sequences with minimal changes, ensuring that long-term dependencies are captured. The gating mechanisms, namely the input, forget, and output gates, regulate the flow of information into, out of, and within the cell. This design allows LSTMs to selectively remember or forget information, making them adept at learning and retaining long-term dependencies in sequences.

- **Advantages:**
  - Efficiently learns long-term dependencies; overcomes the vanishing gradient problem inherent in traditional RNN.
  - Often achieves faster and more stable learning.
- **Disadvantages:**
  - More complex architecture compared to basic RNN and even GRU.
  - Can be computationally intensive due to the multiple gating mechanisms.

Gated Recurrent Units (GRU)[2]: GRU are a variant of RNN that aim to capture long-term dependencies without the complexity of LSTM. They use a gating mechanism to control the flow of information, making them efficient in sequence modeling tasks.

- **Advantages:**
  - Simplified structure compared to LSTM.
  - Efficient in capturing long-term dependencies.
  - Sometimes outperforms LSTM.
- **Disadvantages:**
  - Still more complex than traditional RNN.
  - Might not always outperform LSTM in all tasks.

To sum it up, RNN are good at understanding sequences and context. This makes them a good choice for embedding raw bytes. Just like they understand words based on the words around them, RNN can find patterns in raw byte sequences, giving us a better understanding of the data.

4.3.2 CNNs : Pattern detection in raw bytes

Convolutional Neural Networks (CNN)[8] are a specialized category of deep learning models adept at identifying patterns. Originally designed for visual data, their prowess extends to tasks like image and document recognition. Drawing inspiration from the human visual cortex's biological processes, CNN are architected to autonomously and adaptively discern spatial feature hierarchies from inputs. This becomes particularly relevant when considering raw byte embedding, where the goal is to detect patterns in sequences of bytes. The CNN architecture boasts convolutional layers that perform operations on input data to capture localized patterns, and pooling layers that condense spatial dimensions while preserving crucial information. This layered approach enables CNN to detect intricate patterns by progressively building on simpler foundational patterns. When applied to byte sequences or document recognition, CNN excel, showcasing remarkable efficacy, especially in tasks like identifying patterns within raw byte sequences or recognizing handwritten content.

When tailored to CNN, the Sequence-to-Sequence (Seq2Seq)[4] approach emerges as a potent tool for transforming raw byte sequences into meaningful embeddings. The encoder segment of the Seq2Seq model is central to this transformation. It delves into the byte sequence, discerning intricate patterns and nuances, and distills this rich information into a concise context vector or embedding. This condensed representation captures the core essence of the byte sequence, positioning it as a valuable input for subsequent tasks, such as classification models.

At the heart of the encoder lie the convolutional layers, skilled in pinpointing specific patterns within the byte sequence. Whether it's unique byte combinations or indicative n-grams, these layers are primed to detect them. As they traverse the raw byte sequence, they employ specialized filters, honed to recognize these specific patterns. As the data flows through the encoder's layers, these identified patterns are synthesized and refined, culminating in a comprehensive embedding of the sequence.

Having explored the capabilities of CNNs and their sequence-to-sequence adaptations in raw byte embedding, it's pertinent to shift our focus to another transformative architecture in deep learning: autoencoders.

4.3.3 Autoencoders

Autoencoders stand out as neural network models primarily tailored for compressing and subsequently reconstructing data. They pivot around two central components: the encoder and the decoder. The encoder's task is to condense the input data into a more compact form, often termed as the "latent space" or "embedding." This condensed representation encapsulates the key characteristics and structures of the input, making it especially useful for tasks that demand a streamlined representation of raw byte sequences.

What sets the encoder in an autoencoder apart is its innate capability to discern and capture the most significant aspects of the input data, all without specific instructions. In the context of raw bytes, this translates to the encoder's proficiency in pinpointing inherent patterns and connections within the byte sequences, crafting a detailed and meaningful embedding in the process.

4.3.4 Transformers

4.4 Graph Embedding Methods

4.4.1 Introduction to graph embedding

4.4.2 Popular embedding techniques

Node2Vec, GraphSAGE, and others

4.4.3 Applications and significance in OpenSSH heap dump analysis

4.5 Conclusion and Transition to the Next Section

5 Methods

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

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

Acronymes

BFD Byte Frequency Distribution. 2, 6, 7

CNN Convolutional Neural Networks. 2, 8, 10

GRU Gated Recurrent Units. 2, 9

LSTM Long Short-Term Memory. 2, 9, 10

RCNN Recurrent Convolutional Neural Network. 8

REGEX regular expressions. 5

RNN Recurrent Neural Networks. 2, 8–10

SCP secure copy. 4

Seq2Seq Sequence-to-Sequence. 10

SSH Secure Shell. 1–3

VMI Virtual Machine Introspection. 1

Glossary

pointer In our study, pointers are characterized as sequences of hexadecimal numbers that reference distinct memory addresses. These sequences can be recognized using the following regular expression: `"[0-9a-f]1204"`. 5

structure In our study, structures are defined as a series of bytes that are allocated in the heap. These structures are allocated using the `calloc` function and begin everytime by a *malloc header*. 5

References

[1] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. Apr. 19, 2018. arXiv: 1803.01271[cs]. URL: http://arxiv.org/abs/1803.01271 (visited on 08/23/2023).

[2] Junyoung Chung et al. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. Dec. 11, 2014. arXiv: 1412.3555[cs]. URL: http://arxiv.org/abs/1412.3555 (visited on 08/23/2023).

[3] Christofer Fellicious et al. *SmartKex: Machine Learning Assisted SSH Keys Extraction From The Heap Dump*. Sept. 13, 2022. arXiv: 2209.05243[cs]. URL: http://arxiv.org/abs/2209.05243 (visited on 08/17/2023).

[4] Jonas Gehring et al. „Convolutional Sequence to Sequence Learning". In: *Facebook AI Research* (July 25, 2017). URL: https://arxiv.org/pdf/1705.03122.pdf.

[5] Luke Hiester. „File Fragment Classification Using Neural Networks with Lossless Representations". In: *East Tennessee State University* (May 2018). (Visited on 08/21/2023).

[6] Sepp Hochreiter and Jürgen Schmidhuber. „Long short-term memory". In: *Neural computation* 9.8 (1997). Publisher: MIT Press, pp. 1735–1780. (Visited on 08/23/2023).

[7] Siwei Lai et al. „Recurrent Convolutional Neural Networks for Text Classification". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 29.1 (Feb. 19, 2015). ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v29i1.9513. URL: https://ojs.aaai.org/index.php/AAAI/article/view/9513 (visited on 08/23/2023).

[8] Yann LeCun et al. „Gradient-Based Learning Applied to Document Recognition". In: *proc of the IEEE* (1998).

[9] Stewart Sentanoe and Hans P. Reiser. „SSHkex: Leveraging virtual machine introspection for extracting SSH keys and decrypting SSH network traffic". In: *Forensic Science International: Digital Investigation* 40 (Apr. 2022), p. 301337. ISSN: 26662817. DOI: 10.1016/j.fsidi.2022.301337. URL: https://linkinghub.elsevier.com/retrieve/pii/S2666281722000063 (visited on 08/17/2023).

[10] C E Shannon. „A Mathematical Theory of Communication". In: *The Bell System Technical Journa* 27 (Oct. 1948), pp. 379–423.

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. Dec. 14, 2014. arXiv: 1409.3215[cs]. URL: http://arxiv.org/abs/1409.3215 (visited on 08/23/2023).

Additional bibliography

[12] Vivek Gite. *How To Reuse SSH Connection To Speed Up Remote Login Process Using Multiplexing*. nixCraft. Aug. 20, 2008. URL: https://www.cyberciti.biz/faq/linux-unix-reuse-openssh-connection/ (visited on 10/21/2022).

[13] Weijie Huang and Jun Wang. *Character-level Convolutional Network for Text Classification Applied to Chinese Corpus*. Nov. 15, 2016. arXiv: 1611.04358[cs]. URL: http://arxiv.org/abs/1611.04358 (visited on 08/17/2023).

[14] Ashish Vaswani et al. „Attention Is All You Need“. In: *Advances in Neural Information Processing Systems* 30 (2017), pp. 5998–6008. (Visited on 08/23/2023).

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, August 29, 2023

Lahoche, Clément Claude Martial