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

September 5, 2023

Abstract

Acknowledgements

Contents

1 Introduction **1**

2 Research Questions **1**

3 Structure of the Thesis **1**

4 Background **2**

4.1 Graph Generation from Heap Dumps 2

4.1.1 Secure Shell (SSH) 2

4.1.2 heap dumps of OpenSSH 3

4.1.3 Dataset 3

4.1.4 Entropy's Role in SSH Key Identification 5

4.1.5 Definitions : Structures, Pointers, and the role of malloc headers 5

4.2 Traditional Statistical Embedding 6

4.2.1 Entropy and its application in byte sequence embedding 6

4.2.2 Byte Frequency Distribution (BFD) 6

4.2.3 Other traditional statistical embedding techniques 7

4.3 Deep Learning Models for Raw Byte Embedding 8

4.3.1 RNNs : Understanding sequence data 8

4.3.2 CNNs : Pattern detection in raw bytes 10

4.4 Graph Embedding Methods 11

4.5 Machine learning 11

4.5.1 Features engineering 12

4.5.2 Imbalanced data 13

4.5.3 Some common models 14

4.6 Clustering 14

4.6.1 K-Means Clustering 15

4.6.2 DBSCAN 15
4.6.3 Spectral Clustering 15

5 Methods **16**
5.1 Embeddings 16
5.2 Embedding quality 16
5.3 Embedding coherence 16

6 Results **17**

7 Discussion **18**

8 Conclusion **19**

Appendix A Code **20**

Appendix B Math **20**

Appendix C Dataset **20**

References **23**

Additional bibliography **25**

List of Figures

1 Json exemple 4
2 Xxd exemple 4

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* [5] 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“ [22] 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

A primary focus is on identifying the most suitable techniques for embedding byte sequences, particularly when the objective is to extract structures housing SSH keys for machine learning applications. As the research progresses, it becomes essential to discern if the designed embeddings manifest distinct variations depending on the diverse applications of OpenSSH. Given the extensive spectrum of SSH key sizes and the intricate workings of OpenSSH, ensuring the consistency and stability of these embeddings is paramount. Addressing these aspects will pave the way for a deeper comprehension of challenges and prospective solutions in the field of data embedding, aligning with the overarching goal of bridging digital forensics and machine learning.

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) [15] (Long Short-Term Memory (LSTM)[9] and Gated Recurrent Units (GRU)[3]) and sequence-to-sequence learning [24] offer unique perspectives on raw byte embedding. Furthermore, the efficacy of convolutional approaches (CNN), both standalone[16] 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 [7].Finally, we will dive into transformers[25] 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* [5].

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* [5].

4.1.3 Dataset

„We use SSHKex[22] 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* [5].

4.1.4 Entropy's Role in SSH Key Identification

Encryption keys[5] 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 [23], 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[27] 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. The Fisher's skewness[2] is :

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

Kurtosis Kurtosis[27] 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. The Fisher's kurtosis[2] is :

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

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[15]: 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)[9]: 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)[3]: 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)[16] 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)[6] 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.

Here are two Sequence-to-Sequence (Seq2Seq) models using CNN :

- **Autoencoders:** These neural network architectures[8] are designed for data compression and reconstruction. The encoder part compresses the input data into a compact representation, while the decoder reconstructs the original data from this representation. In the context of raw byte sequences, the encoder can be used to generate embeddings that capture the essential patterns and structures of the data.
- **Transformers :** Transformers[25] utilize self-attention mechanisms to weigh the significance of different parts of the input data. This allows them to capture long-range dependencies and relationships in the data. When applied to raw byte sequences, transformers can generate embeddings that consider both local and global patterns, making them particularly effective for tasks that require understanding the broader context of a sequence.

Yet, a significant challenge with traditional Sequence-to-Sequence (Seq2Seq) models using CNN is their constraint in managing inputs of varying sizes. Constructed with a set input size, they face difficulties when presented with sequences of diverse lengths, like raw byte sequences.

To address this limitation, various techniques have been employed to normalize the size of the input data. One of the most common methods is **padding**, where shorter sequences are filled with predefined placeholder values (often zeros) until they match the length of the longest sequence in the dataset. This ensures that all sequences fed into the model have a uniform length. Another approach is **bucketing**, where sequences of similar lengths are grouped together, minimizing the amount of padding required. Additionally, **truncation** can be used to shorten sequences that exceed a certain length, although this might result in the loss of some information. While these techniques enable CNN-based Sequence-to-Sequence (Seq2Seq) models to handle variable-sized inputs, it's crucial to ensure that the preprocessing steps do not introduce noise or distort the inherent patterns and relationships within the raw byte sequences.

4.4 Graph Embedding Methods

Wait Onyr

4.5 Machine learning

Machine learning, an integral part of artificial intelligence, revolves around designing algorithms and statistical models that allow computers to perform tasks without being directly programmed. Instead of relying on detailed instructions for every task, machine learning techniques empower systems to learn from data and make data-driven decisions. A key method in this field is supervised learning, in which models are trained using data that comes with predefined labels. Here, each piece of data in the training set has an associated known output. The primary goal of supervised learning is to establish

a relationship between inputs and outputs, enabling the model to predict or categorize new, unseen data based on this relationship.

A cornerstone in this realm is feature engineering, which involves the meticulous process of selecting and transforming variables to optimize model performance. Another challenge frequently encountered by practitioners is dealing with datasets where some classes are overrepresented, which can skew model predictions. Among the myriad of machine learning models available, certain ones have gained prominence due to their versatility and effectiveness. We will provide an overview of some of these notable models.

4.5.1 Features engineering

Feature engineering[11] is a cornerstone in the realm of machine learning. It involves the artful transformation of the given feature space to optimize the performance of predictive models. The significance of feature engineering cannot be overstated; it serves as a bridge between raw data and the predictive models, ensuring that the models are fed with the most relevant and informative features. Properly engineered features can drastically reduce modeling errors, leading to more accurate and reliable predictions. Here are some of the most common feature engineering techniques:

- **Normalization and Scaling** are preprocessing techniques used to standardize the range of independent features in the data. Many machine learning algorithms, especially those that rely on distance calculations like k-means clustering or support vector machines, are sensitive to the scale of the data. If features are on different scales, one feature might dominate others, leading to suboptimal model performance. Normalization typically scales features so that they have a unit norm, while other scaling methods, such as Min-Max scaling, transform features to lie in a given range, usually [0,1]. Z-score normalization or standard scaling is another method where features are scaled based on their mean and standard deviation. Properly scaled data ensures that each feature contributes equally to the model's decision, leading to more stable and accurate predictions.

- **Interaction Features[10]** refer to the creation of new features by combining two or more existing features, aiming to capture any synergistic effect between them. In many cases, the interaction between variables can provide more information than the individual variables themselves. For instance, while analyzing real estate prices, the individual features 'number of rooms' and 'location' might be informative, but their interaction, 'number of rooms in a specific location', might offer even more predictive power. Interaction features can be created by multiplying, adding, or even dividing original features, and they can help in capturing non-linear relationships in the data, enhancing the model's ability to make accurate predictions.

- **Feature Selection[10]** is a critical process in the machine learning pipeline that focuses on selecting the most relevant features from the original set, aiming to reduce the dimensionality and improve model performance. The primary goal is to eliminate redundant or irrelevant features that don't contribute significantly to the predictive power of the model. This not only helps in reducing the computational cost but also can lead to a more interpretable model. There are

various techniques for feature selection, including filter methods (based on statistical measures), wrapper methods (like recursive feature elimination), and embedded methods (where algorithms inherently perform feature selection, such as decision trees). By judiciously selecting features, one can build efficient models that are less prone to overfitting and have better generalization capabilities.

Following the aforementioned techniques, another essential facet in the feature engineering landscape is dimensionality reduction. As data grows in complexity, it often encompasses a vast number of features, leading to what is known as a high-dimensional space. While a plethora of features might seem advantageous, it introduces challenges, notably the *curse of dimensionality*[26, 12]. In such high-dimensional realms, data points tend to become increasingly sparse. This sparsity means that the relative distances between data points start to appear uniform, making it arduous for algorithms to discern meaningful patterns. This can lead to models that overfit the training data, capturing noise rather than the underlying data distribution. Additionally, the computational overhead increases, and deriving intuitive insights from the data becomes a daunting task.

Dimensionality reduction techniques come to the rescue by striving to trim down the number of features while preserving the crux of the information. Techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) are employed to transform the data from its original high-dimensional space to a more manageable, lower-dimensional one. This transformation aims to retain the significant patterns and structures inherent in the data. By judiciously reducing the dimensionality, not only can models be trained more efficiently, but they often yield better performance by focusing on the most pertinent features. This streamlined approach mitigates the challenges posed by the curse of dimensionality, ensuring models that are both robust and interpretable.

4.5.2 Imbalanced data

In machine learning, a frequent obstacle is the presence of datasets where one category vastly overshadows others[21]. This imbalance can skew models towards predicting the dominant class, often neglecting the less prevalent but potentially more critical class.

To counteract this, a variety of techniques have been devised:

- **Resampling:** This encompasses both increasing instances of the minority class (oversampling) and decreasing instances of the majority class (undersampling). A notable method for oversampling is the Synthetic Minority Over-sampling Technique (SMOTE), which generates artificial data points in the feature space.
- **Weighted Loss:** This strategy involves assigning greater weights to the minority class during the training phase, ensuring the model gives it due consideration.
- **Ensemble Methods:** Approaches such as bagging and boosting can be tailored to ensure a balanced class representation. For example, in bagging, each sample can be structured to

maintain a balanced class ratio.

- **Anomaly Detection:** This method reframes the task from classification to anomaly detection, viewing the minority class as an outlier or anomaly.

Selecting an appropriate strategy hinges on the specific problem and dataset characteristics. It's also crucial to evaluate the model's efficacy using suitable metrics, ensuring it genuinely addresses the imbalance.

4.5.3 Some common models

- **Logistic Regression[19] :** Logistic regression serves as a statistical technique tailored for binary classification tasks. While linear regression is designed to forecast continuous outcomes, logistic regression focuses on predicting the likelihood of a binary result. It leverages the logistic function to relate multiple independent variables to a binary outcome, ensuring the predicted values fall between 0 and 1. Typically, a 0.5 threshold is used to classify the final outcome. A key strength of logistic regression is its clarity and ease of interpretation, though it might face challenges with complex non-linear data unless further feature adjustments are made.

- **Decision Trees[13] :** Decision trees are machine learning models designed for both classification and regression tasks. They segment data into subsets based on feature values, making decisions at each node. While their hierarchical structure offers easy visualization and interpretation, they can be prone to overfitting. However, strategies like pruning can help in refining the tree and mitigating overfitting.

- **Random Forest[20] :** Random Forest is an ensemble method that creates a 'forest' of decision trees. Each tree is trained on a random subset of the data and makes its own predictions. The Random Forest algorithm then aggregates these predictions to produce a final result. This method is known for its high accuracy, ability to handle large datasets with higher dimensionality, and its capacity to manage missing values.

- **Support Vector Machines (SVM)[28] :** SVM are used for both regression and classification problems. They work by finding the hyperplane that best divides a dataset into classes. SVMs are effective in high-dimensional spaces and are versatile, as different Kernel functions can be specified for the decision function.

- **K-Nearest Neighbors (KNN)[14] :**KNN is a simple, instance-based learning algorithm. To make a prediction for a new data point, the algorithm finds the 'k' training examples that are closest to the point and returns the most common output value among them.

4.6 Clustering

Clustering is a key technique in unsupervised machine learning, aiming to group similar data points together. It's all about ensuring that items within a cluster are more alike than those in different clusters. This approach is great for uncovering hidden patterns in data. When it comes to checking

the quality of an embedding, clustering can be a handy tool. By forming clusters from embedded data, we can see how effectively similar structures come together. A top-notch embedding should make sure that data points from the same structure cluster closely. So, by looking at how well clustering works, we can gauge the strength of the embedding.

4.6.1 K-Means Clustering

K-Means [18] is a popular clustering method recognized for its straightforwardness and speed. It works by dividing a dataset into 'K' unique, separate groups (or clusters) based on how close each data point is to the cluster's center, termed the centroid. The method repeatedly places each data point with the closest centroid and then updates the centroid's position based on the points in its cluster. This cycle repeats until the centroids no longer move significantly. Though K-Means is great for clusters that are roughly spherical and similar in size, deciding on the best number of clusters (K) in advance can be tricky.

4.6.2 DBSCAN

DBSCAN [4] is a clustering method that identifies dense regions in data, considering sparse areas as outliers. Unlike K-Means, DBSCAN doesn't need a pre-defined number of clusters. It works on the principle that a cluster is a high-density area in data, surrounded by less dense regions. The algorithm is steered by two key parameters: the minimum points needed for a region to be dense and a distance measure determining how close points should be to create a cluster. DBSCAN shines in handling datasets where clusters have varying shapes but similar densities.

4.6.3 Spectral Clustering

Spectral clustering[17] is a method that emphasizes reducing the dimensionality of data using the eigenvalues of a similarity matrix. By constructing a similarity graph, where nodes represent data points and edges carry weights based on point similarities, the technique transforms the original space. Utilizing the eigenvectors of the graph's Laplacian, it creates a more compact and manageable representation. In this reduced space, traditional clustering methods like K-Means become more effective. Spectral clustering's strength lies in its ability to handle complex cluster structures, especially non-convex clusters, making it a valuable tool for diverse applications.

5 Methods

This research dives into the complexities of embedding byte sequences, focusing particularly on the extraction of structures containing SSH keys for machine learning purposes. The varied uses of OpenSSH introduce distinct challenges due to potential variations in the created embeddings. Given the wide array of SSH key dimensions and OpenSSH's intricate operations, maintaining the embeddings' stability and consistency is vital. In this methodological section, we will detail various embedding methods, present a framework for their assessment through a classifier model, and suggest another strategy to verify the embeddings' coherence between the different OpenSSH usage and key sizes.

5.1 Embeddings

5.2 Embedding quality

5.3 Embedding coherence

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

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] Meghan K. Cain, Zhiyong Zhang, and Ke-Hai Yuan. „Univariate and multivariate skewness and kurtosis for measuring nonnormality: Prevalence, influence and estimation". In: *Behavior Research Methods* 49.5 (Oct. 2017), pp. 1716–1735. ISSN: 1554-3528. DOI: 10.3758/s13428-016-0814-1. URL: http://link.springer.com/10.3758/s13428-016-0814-1 (visited on 08/30/2023).

[3] 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).

[4] Martin Ester et al. „A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *KDD-96* (Aug. 2, 1996).

[5] 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).

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

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

[8] G. E. Hinton and R. R. Salakhutdinov. „Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (July 28, 2006), pp. 504–507. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.1127647. URL: https://www.science.org/doi/10.1126/science.1127647 (visited on 08/30/2023).

[9] 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).

[10] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. „A survey of feature selection and feature extraction techniques in machine learning". In: *2014 Science and Information Conference*. 2014 Science and Information Conference. Aug. 2014, pp. 372–378. DOI: 10.1109/SAI.2014.6918213.

[11] Udayan Khurana, Horst Samulowitz, and Deepak Turaga. *Feature Engineering for Predictive Modeling using Reinforcement Learning*. Sept. 21, 2017. arXiv: 1709.07150[cs, stat]. URL: http://arxiv.org/abs/1709.07150 (visited on 08/30/2023).

[12] Mario Koppen. „The curse of dimensionality". In: 1 (2000), pp. 4–8.

[13] S. B. Kotsiantis. „Decision trees: a recent overview". In: *Artificial Intelligence Review* 39.4 (Apr. 2013), pp. 261–283. ISSN: 0269-2821, 1573-7462. DOI: 10.1007/s10462-011-9272-4. URL: http://link.springer.com/10.1007/s10462-011-9272-4 (visited on 08/30/2023).

[14] J. Laaksonen and E. Oja. „Classification with learning k-nearest neighbors“. In: *Proceedings of International Conference on Neural Networks (ICNN'96)*. International Conference on Neural Networks (ICNN'96). Vol. 3. Washington, DC, USA: IEEE, 1996, pp. 1480–1483. ISBN: 978-0-7803-3210-2. DOI: 10.1109/ICNN.1996.549118. URL: http://ieeexplore.ieee.org/document/549118/ (visited on 08/30/2023).

[15] 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).

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

[17] Ulrike von Luxburg. *A Tutorial on Spectral Clustering*. Nov. 1, 2007. arXiv: 0711.0189[cs]. URL: http://arxiv.org/abs/0711.0189 (visited on 09/05/2023).

[18] J Macqueen. „SOME METHODS FOR CLASSIFICATION AND ANALYSIS OF MULTIVARIATE OBSERVATIONS“. In: *MULTIVARIATE OBSERVATIONS* VOL. 5.1 (1967).

[19] Todd G Nick and Kathleen M Campbell. „Logistic regression“. In: *Topics in biostatistics* (2007). Publisher: Springer, pp. 273–301.

[20] Philipp Probst, Marvin Wright, and Anne-Laure Boulesteix. „Hyperparameters and Tuning Strategies for Random Forest“. In: *WIREs Data Mining and Knowledge Discovery* 9.3 (May 2019), e1301. ISSN: 1942-4787, 1942-4795. DOI: 10.1002/widm.1301. arXiv: 1804.03515[cs,stat]. URL: http://arxiv.org/abs/1804.03515 (visited on 08/30/2023).

[21] Dr D Ramyachitra and P Manikandan. „IMBALANCED DATASET CLASSIFICATION AND SOLUTIONS: A REVIEW“. In: *International Journal of Computing and Business Research* 5.4 (2014).

[22] 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).

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

[24] 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).

[25] 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).

[26] Michel Verleysen and Damien François. „The Curse of Dimensionality in Data Mining and Time Series Prediction“. In: *Computational Intelligence and Bioinspired Systems*. Ed. by Joan Cabestany, Alberto Prieto, and Francisco Sandoval. Red. by David Hutchison et al. Vol. 3512. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 758–770. ISBN: 978-3-540-26208-4 978-3-540-32106-4. DOI: 10.1007/11494669_93. URL: http://link.springer.com/10.1007/11494669_93 (visited on 08/30/2023).

[27] Donald J Wheeler. „Problems with Skewness and Kurtosis“. In: *Quality Digest Daily* (Aug. 1, 2011).

[28] Qiang Wu and Ding-Xuan Zhou. „Analysis of Support Vector Machine Classification“. In: *Journal of Computational Analysis & Applications* 8.2 (2006).

Additional bibliography

[29] Walter T. Ambrosius, ed. *Topics in biostatistics*. Methods in molecular biology 404. OCLC: ocn159977868. Totowa, N.J: Humana Press, 2007. 528 pp. ISBN: 978-1-58829-531-6.

[30] *CERT/CC Vulnerability Note VU#13877*. URL: `https://www.kb.cert.org` (visited on 08/30/2023).

[31] 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).

[32] 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).

[33] 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. 1, 2021), pp. 387–396. ISSN: 0167-739X. DOI: `10.1016/j.future.2020.09.004`. URL: `https://www.sciencedirect.com/science/article/pii/S0167739X20303265` (visited on 08/30/2023).

[34] 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.* 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`.

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 5, 2023

Lahoche, Clément Claude Martial