NeuroYara: Learning to Rank for Yara Rules Generation through Deep Language Modeling and Discriminative N-gram Encoding

Ziad Mansour*, Weihan Ou*, *Member, IEEE,* Steven H. H. Ding*, *Member, IEEE,* Mohammad Zulkernine*, *Senior Member, IEEE,* Philippe Charland†

Abstract—Signature-based malware detection methods are recognized for their simplicity, explainability, and efficiency. One of the most commonly used tools is Yara, which provides the syntax for crafting malware signatures. However, while developing high-quality Yara rules requires significant expertise in malware analysis, training such skilled analysts can be both resource-intensive and time-consuming. While a few works have been conducted to automate the generation of signatures, signatures generated by those works typically underperform the manually generated ones. In addition, these automated methods often depend on large static databases of hard-coded byte n-grams to minimize false positives. Instead of storing a large non-inclusive database to score byte n-grams, we propose a novel architecture utilizing two learning to rank neural networks to understand the underlying effectiveness and correlations among n-grams extracted for rule construction. This approach provides better flexibility and coverage of possible n-grams while reducing the required storage size from several GBs to only 10MBs. Combining these two models with a hierarchical density-based clustering method allows us to group multiple n-grams into logical conditions as Yara rules of higher quality. Experimental results show that our framework, NeuroYara, reduces the resources invested by analysts while generating rules with a low false-positive rate outperforming existing tools and manually-generated rules.

Index Terms—malware analysis, deep learning

1 Introduction

Malicious software, or malware, refers to programs intentionally designed to attack systems or steal information. According to literature, malware is the most common threat in the cyber world [27], [37]. In recent years, with the versatility and number of malware increasing dramatically, it is vital for cyber investigators to detect incoming malware and prevent it from being executed by potential victims. Machine learning-based models are popular approaches for malware detection and triage [33], since these models can better detect previously unseen samples. However, existing machine learning models have two main problems: first, they cannot easily accommodate some of the corner cases of malware samples, when the attackers employ techniques such as code obfuscation, encryption, and packing to evade detection; second, they lack explainability as to why a specific classification decision has been made and maintainability to patch the model when a wrong classification is observed.

Signature-based techniques are critical for malware detection, incidence response, and Indicator-of-Compromise (IoC) matching. A malware signature contains a group of elements, including strings, bytes, and section/header hashes, that are combined under certain conditions to uniquely identify a group of malware. Despite being one of the earliest methods in malware detection, robust and generalizable signatures can provide a high detection rate, while maintaining a low false positive rate [36]. It also provides better explainability and maintainability of the rules, such that when a new variant of malware is introduced to a malware family, analysts can easily modify the signature to incorporate the new malware sample, thanks to the technique's explainability. One of the unique approaches was developed by Newsome et al. [11]. Their approach, Polygraph, is an automated tool for different classes of signature generation based on distinct invariant substrings, that must be generally present in different variants of polymorphic worms. However, work [34] that was published one year after Polygraph shows that Polygraph can be easily misled, causing the resulting signatures to be useless.

This approach was proposed before the release of Yara [10], which is one of the most widely deployed tools to create malware signatures and rules. Yara rules can contain multiple strings, byte sequences, and conditions that are used to identify specific malware families (i.e., given a test binary, it checks whether it belongs to a specific malware family or not). It is time and resource-intensive to train junior analysts to develop high-quality rules and be on par with expert analysts [36]. It could take junior analysts several hours or weeks to fully analyze and construct a Yara rule for a malware sample [50].

So far, only a few works have been conducted to automate signatures and rules generation (e.g., [55], [36]). Current works state that they generate rules that are good enough to be used without significant changes to reduce the workload of malware analysts [36], [25]. However, the gen-

*Z. Mansour, S. H. H. Ding and M. Zulkernine are with the School of Computing, Queen's University, Ontario, Canada K7L 2N8. E-mails: (ziad.mansour, steven.ding, mz)@queensu.ca, †P. Charland is with Mission Critical Cyber Security Section, Defence R&D Canada - Valcartier, Quebec, QC, Canada. E-mail: philippe.charland@drdc-rddc.gc.ca
Manuscript received xxx; revised xxx; accepted xxx

erated rules by the current tools have much lower quality than the manually generated rules. In reality, the automated rules require significant modifications to become on par with the manual rules, which negates the purpose of the rules generation automation.

To the best of our knowledge, there exists no tool that can outperform or provide comparable performance to manually generated rules. One of the most common shortfalls of the existing tools is the need to store a huge database of hard-coded benignware strings as a reference set to compare with when scanning a binary file's strings to reduce the false positive (FP) rate. For example, YarGen [39] requires 4GBs-8GBs of memory while running and AutoYara [36] has to store a couple of hundreds of MBs of benign and malicious bytes, while the storage size required for NeuroYara is only 22 MB. Hence, our goal is to introduce an efficient automatic high-quality Yara rules generator that can provide performance on par with rules written by expert malware analysts, while maintaining a low FP rate and a flexible and inclusive compressed knowledge.

1.1 Challenges and Contributions

N-gram malicious modelling. The first challenge is to select n-grams for Yara rules construction. An important selection criterion is the maliciousness of the n-grams, as it helps to keep a low FP rate on the subsequent malware identification process using the Yara rules constructed with the selected n-grams. Existing works [22], [39], [36] determine the maliciousness of the n-grams by maintaining large manually-crafted databases of hard-coded reference strings for n-gram comparison. Those works do not consider the dependencies and relationships between n-grams, especially the ones that can be found in benignware and malware. Additionally, they do not consider out-of-vocabulary (OOV) n-grams, which means that their tools are not able to utilize some of the n-grams if they are not stored in their hard-coded database.

NeuroYara's novelty lies in modelling the maliciousness of n-grams without human interventions and a large storage space requirement. By training a bidirectional Long Short Term Memory (LSTM) [47] n-gram discriminative model on a large dataset of malware and benignware samples, it learns the semantic meanings of the n-grams, with considerations of the input context. It grasps the underlying maliciousness and benignness of byte sequences, and effectively scores and ranks n-grams by estimating the probability of maliciousness of n-grams. The n-gram discriminative model can generalize on novel data not observed in the training dataset.

Demanding on Storage space. Another challenge is to maintain low storage requirements while increasing the scale of the learned n-gram patterns. Prior works rely on large, manually crafted databases of reference strings for n-gram comparison, which can lead to exponential increases in storage needs when accommodating various lengths of n-grams [36]. In contrast, the bi-LSTM-based n-gram discriminative model used in NeuroYara distinguishes between benign and malicious n-grams with minimal model storage. Instead of several gigabytes of hard-coded benignware strings, NeuroYara requires much less storage space, even for long n-grams (around 10 MB).

```
rule ta505
{   strings:
      $x0 = {6D 65 20 ... 00 00 0A}
      ...
      $x35 = {65 2E 0D ... 00 12 2A}
     condition:
      (12 of ($x0,$x1,...,x10,$x11)) or ...
      or  (12 of ($x24,$x25,...,$x34,$x35)) }
```

Fig. 1. NeuroYara's generated rule for `ta505`. It contains Boolean expressions combining individual byte sequences.

Integrating expert knowledge. We acknowledge that utilizing the substantial knowledge gained from malware analysis can be advantageous for scoring and ranking n-grams. However, automated tools published in recent years for enhancing the Yara-rule generating process have limitations in incorporating expert knowledge on malware detection [12], [39], [8], [53]. With the new human language model, NeuroYara improves the effectiveness of the generated Yara rules by leveraging human expertise in rule generation. By training the human language model with Yara rules previously created by the experts, it learns to score the input n-grams based on their similarity to existing Yara rules. The proposed human language model, despite implicitly carrying massive knowledge, is only less than 10 MB.

Heterogeneous clusters and noise. Malware variants have a very different code structure. Consequently, malware samples are naturally grouped into heterogeneous clusters of different densities and shapes. These clusters require independent definitions of threshold and other parameters. Additionally, since the number of n-grams is far larger than the number of samples available for rule construction, there exist outliers as noise to the clusters. To mitigate heterogeneous malware variants and noisy outliers in the n-grams, NeuroYara combines the hierarchical clustering method (HDBSCAN [14]) with the previous learning to rank n-grams models in a unified novel architecture. HDBSCAN is an extended version of Density-based Spatial Clustering of Applications with Noise (DBSCAN) [19], which converts it into a hierarchical clustering algorithm. HDBSCAN does a great job in clustering complex-shaped clusters of data, with varying densities, noise, heterogeneous clusters, and outliers [14]. Therefore, utilizing the hierarchical approach of HDBSCAN in clustering n-grams and strings will allow grouping them in an optimum way, using Boolean expressions to provide high-quality Yara rules. With the unified architecture, there is no manual process needed for the complete rules-generating process.

Figure 1 depicts an example of a generated rule by NeuroYara for the malware family `ta505`. The set of n-grams (i.e., $x0 ... $x35) are the ones extracted and chosen to be used as signatures from the malware samples of the `ta505` family. To make the Yara rules able to capture malware variants, conditional rules are utilized to group n-grams and strings using logical and Boolean statements. Our **contributions** can be summarized as follows:

(1) Instead of maintaining a large, manually crafted string database, NeuroYara proposes the novel learning to rank discriminative model that automatically captures the maliciousness of n-grams. The proposed model incorporates the context information and relationships among the n-

grams and generalizes on data not observed in the past.

(2) NeuroYara incorporates expert knowledge on malware detection with its novel human language model that consists of the bidirectional LSTM. The proposed model learns the similarity among input n-grams with existing Yara rules generated by malware analysis experts.

(3) NeuroYara compresses the knowledge of benignware and malware bytes from several GBs, which was the case in previous works, to only around 10MBs, while outperforming these works and manually generated rules.

(4) NeuroYara combines the learning torank n-grams models with HDBSCAN in a unified novel architecture framework, which effectively mitigates heterogeneous clusters and noisy outliers in the Yara rule generation process.

(5) Compared to the state-of-the-art solutions YarGen [39], AutoYara [36], Yaraml [40] and manually generated rules, NeuroYara provides better performance in unknown malware prediction and false-positive testing across over 90 malware families, in different scenarios, and using only a few samples for rules construction as well as the evaluation in an adversarial environment.

The rest of the paper is organized as follows: Section 2 discusses our proposed methodology and how our framework's architecture operates; Section 3 explains the dataset and the experimental setup; Section 4 presents our methodology for evaluating the rules generated; Section 5 demonstrates our benchmark against the state-of-the-art tools AutoYara, YarGen, Yaraml and manually-generated rules; Section 6 discusses the related works; Section 7 concludes the paper.

2 METHODOLOGY

In this section, we present NeuroYara's overall architecture. We then discuss each component used in the framework in more detail. In order to present how NeuroYara constructs Yara rules for a certain binary file or a malware family containing more than one binary file, we discuss its architectural design. NeuroYara's architecture is displayed in Figure 2. The signature generation process in this work is completely automated. Specifically, our approach to developing the framework of NeuroYara can be summarized as follows:

Extraction. NeuroYara extracts byte sequences from the malware sample(s) and transforms them into an integer sequences format to be utilized by the neural network models. Then, using a sliding window approach, we extract n-grams for the two models to score and rank.

Probabilistic discrimination. Given the extracted n-grams, the bidirectional LSTM-based discriminative probabilistic model scores and ranks them based on their maliciousness. Since the model has been trained to understand the underlying maliciousness of n-gram sequences, it can score them by providing each n-gram with a probability value (i.e., a low probability n-gram indicates benignness, while a high probability indicates maliciousness). This step is typically done in previous works by saving a huge static database of benign n-grams and comparing the analyzed binary's n-grams with the database.

Human language scoring. Given the scored n-grams, the bidirectional LSTM-based human language model works on scoring n-grams based on how similar the pattern of this n-gram is compared to n-grams manually generated by malware analysts. This allows NeuroYara to employ our existing knowledge of Yara rules constructed manually in an efficient manner.

Clustering. We then select the top n-grams to be fed to HDBSCAN to cluster them in a way that allows to cover different variants from the same malware family, while maintaining a low FP rate. We take advantage of the semantics of Yara utilizing n-grams and strings to be used as signatures and bind the clusters using Boolean conditions and statements.

Algorithm 1: N-grams encoding and extraction

```
Input: Directory containing malware family binaries
Result: Malware family's n-grams
initialization;
Function readNgrams(directory):
    winSizeN := 16
    for binary in binaryFiles do
        rawBytes ← readBytes(binary)
        seqBytes ← bytes2seq(rawBytes) for i = 0 to
          len(bytes); step=1 do
            ngram ←slice[i:i+winSizeN]— bytes
            fileNgrams.append(ngram)
        end
        familyNgrams.append(fileNgrams)
    end
    return familyNgrams
```

2.1 N-grams Extraction

Given a malware sample or family, n-gram extraction is the first step in NeuroYara's process to generate Yara rules. In our context, we utilize byte n-grams, where the **n** character in n-grams stands for the number of bytes used. For example, a 7-byte n-gram can be written as {`6D 61 6C 77 61 72 65`} if it is in hexadecimal form (i.e., each 2 hexadecimal numbers represent a byte) or as `"malware"` if it is in ASCII form.

Most of the previous works utilized short n-grams (i.e., less than 8) (e.g., [21], [55]). However, short n-grams have downsides. They cause a higher FP rate since n-grams are not unique enough to the malware sample or family and therefore, they provide lower interpretability [12]. It should be noted that while longer n-grams can offer a unique signature that has a higher probability of matching a malware sample, which is good for a Yara signature, it will provide less flexibility and be more difficult for a malware analyst to alter the rule [12]. Autoyara [36] utilized long n-gram sequences which provide superior results than other shorter n-gram approaches. Given what was found empirically in previous works, we test our n-gram sizes performance from 8 grams to 16 grams since more than 16 grams would result in the downsides mentioned of the very long n-grams. Generally, 16-grams provide the best results for Yara rules signature creation. Therefore, to extract 16-gram sequences from the malware sample, we use the sliding window approach [38]. The process of n-grams extraction is described in Algorithm 1. For each input raw binary $X_b = [b_1, ..., b_n], b_i \in \{0, 1\}$, we first read the raw

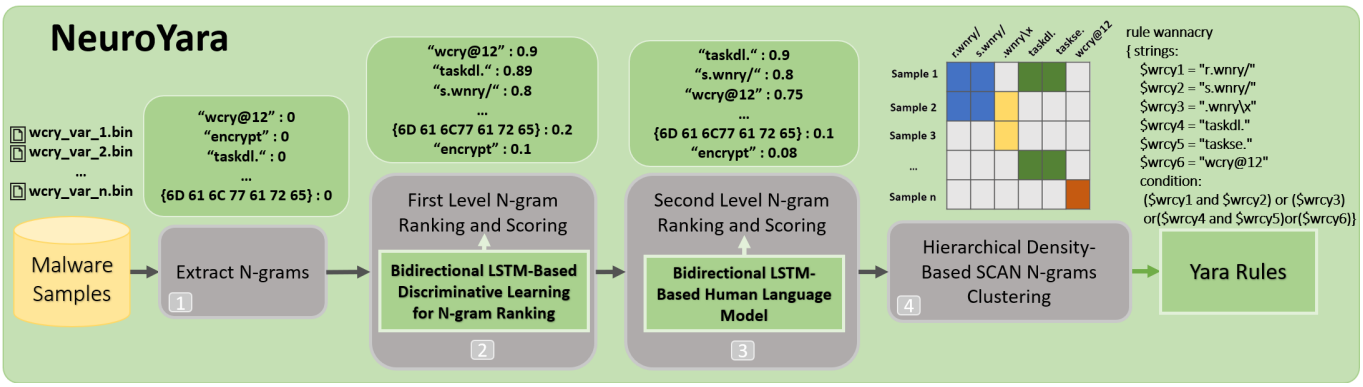


Fig. 2. Overall framework architecture of NeuroYara. After preparing the directory with binary samples of a malware family, in (1) we extract the n-grams from the given binaries using the sliding window approach. Then in (2) we do our first level of n-gram ranking and scoring using the discriminative probabilistic model. In (3) we utilize the human language model to impose our second level of n-gram ranking and scoring. Step (4) is grouping the top n-grams using HDSBSCAN clustering and transforming the clustered n-grams to signatures and Boolean conditions to finally develop the corresponding Yara rules.

bytes of the binary. Next, we transform the bytes to integer sequences format which the language model will require to perform its task. We denote the transformation process as $bytes2seq() : X_b \rightarrow X_h$. Then, we use the sliding window concept to take the 16-grams from the integer sequences. The 16-gram sequence for each input binary sample is denoted as X.

2.2 Discriminative Probabilistic Model

Model overview. The discriminative probabilistic model for n-grams scoring and ranking works on distinguishing between malicious and benign byte sequences by learning the meaning of the sequential data in the byte sequence. In other words, it determines the importance and usefulness of the extracted n-grams from a binary file in constructing signatures in a Yara rule. In the previous step, we obtained the 16-gram sequence for each input binary sample denoted as $X = \{x_0, ...x_n\}$, where each x represents a 16-gram sequence and n represents the max sequence length. In this step, our objective is to construct the discriminative probabilistic model F_d, which assigns a score ranging from 0 to 1 to each 16-gram sequence x, indicating the likelihood of its association with malicious operations. A score closer to 1 indicates a higher probability of a 16-gram being related to malicious operations. Current state-of-the-art models that perform well with sequential data are based on recurrent neural network (RNN) architectures [31], such as gated recurrent units (GRUs) [15] and LSTMs [17]. Generally, LSTMs, although more complex than other RNN-based models, provide superior performance for tasks dealing with sequential data [16]. An LSTM network works on learning past learned knowledge, while disregarding irrelevant data for the learning context task. This is accomplished by utilizing various gates (i.e., activation function layers). Hence, we utilize bidirectional LSTMs in the task of ranking n-grams based on their maliciousness and usefulness. Bidirectional LSTMs can be described as using two independent LSTMs together. This structure allows the networks to have both backward and forward information about the sequence (i.e., the network will preserve information from both past and future sequences). This network improves the performance compared to unidirectional LSTMs [43]. The architecture of LSTM-based models works by giving probabilities after each timestamp parsing each token, such as a character or a byte in a byte sequence. For each timestamp, one n-gram data in the input n-gram sequence flows into the LSTM network. After parsing the whole string or n-gram, the network concatenates the scores at each timestamp using a fully connected layer, in order to determine the probability of benignness/maliciousness of the n-gram. We will discuss in more detail how the bidirectional LSTM architecture provides such probabilities and learns the important parts of the sequential data.

2.2.1 *Discriminative Model's Methodology*

The architecture of the discriminative probabilistic model does two main tasks at the same time, where previous works and tools had to be done separately. It acts as the database for the benignware n-grams that previous works (e.g., [40], [36], [39], [8]) had to store to reduce the FP rate. However, given that we only store the model, it takes merely 10 MB of disk space to do the same. The second task performed by the model is giving higher scores to malicious n-grams, which will allow us to choose the top n-grams to be used as signatures. In order to know how this model performs those tasks, we have to discuss its architecture and how the model is trained. Figure 3 displays the training and architecture of the discriminative probabilistic model. The complete training process and architecture can be summarized as follows:

Preparation. In order to train the model, we prepared a dataset of 32 and 64-bit malware and benign binaries of different families and vendors. These families will not be included when testing the generated rules.

Extraction. We extract the byte sequences from the binaries and using the sliding window approach, we extract the n-grams required for the discriminative probabilistic model.

Embedding. Inside the model, we first get the n-grams as input and embed the n-grams by passing them to the embedding layer. It transforms the n-gram strings into a dense vector representation of a fixed length of positive integers. Each vector consists of d digits and the combination of the digits conveys the textual and semantic meaning of a n-gram string. This works on compressing the input feature

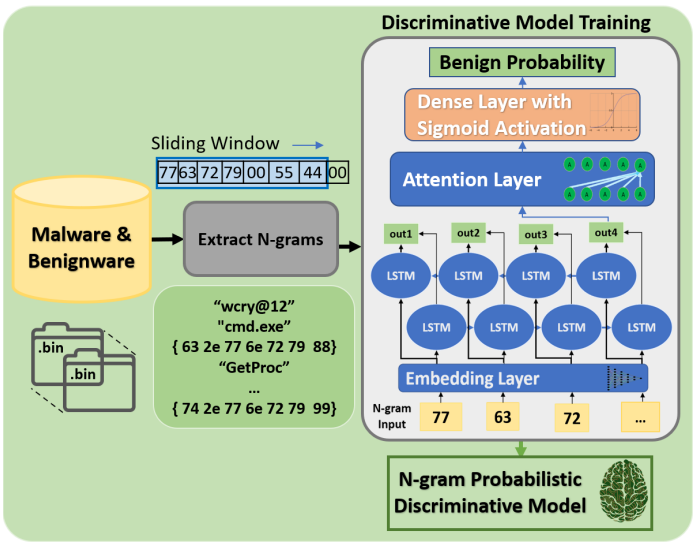


Fig. 3. Discriminative probabilistic model training, demonstrating the n-gram `wcry` in hexadecimal form obtained from the `wannacry` malware. It is analyzed by the model to provide a probability of maliciousness. After training, the model will be holding the discriminational capability between benign and malicious n-grams.

space. As an example of the process, in Figure 3, the n-gram `wcry` is extracted from the `wannacry` malware and passed to the embedding layer.

Encoding. The embedded n-grams are then passed to a bidirectional LSTM layer for semantic feature encoding. To generate the probability of maliciousness for the input n-grams, their semantic essence under given contexts needs to be captured, and the encoding layer is to learn their underlying functional patterns and semantic meanings of the n-grams. For each input binary sample $X_b = [b1, ..., bn], bi \in 0, 1$, n-grams are extracted as fixed-length integer sequences. Thus, an encoding layer that processes sequential inputs is necessary. We employ the bi-LSTM model to construct the encoding layer due to its ability to effectively handle sequential data. For each input n-gram, the model processes one byte per time step and produces high-dimensional feature representations of the complete n-gram. The output high-dimensional feature representations enable the following layers to determine the maliciousness of the n-grams within the given context.

Given the input code samples are structured in the sequential format, it is important to employ a model that can effectively handle long sequence data and catch the semantic meanings of the input. The Long Short-Term Memory (LSTM) network provides superior performance for tasks dealing with sequential data. With its gated structure that controls the data flow in and out of the network, the LSTM network excels in learning the context information from a long sequence while disregarding irrelevant data in the context. There are three kinds of gates in an LSTM model: i_t which is the input gate that controls how much of the input data should be taken into the calculation at timestamp t, f_t which is the forget gate that controls how much of the information from the previous timestamps should be remembered, and o_t controls how much information computed in this timestamp should be passed on to the next timestamp. Using LSTM in our work facilitates the process of capturing the semantic meanings of the n-grams in the input samples.

Attention. The next step is employing the attention mechanism technique [48]. This layer combines the outputs of the LSTM layer at different timestamps to better understand the context. The attention mechanism enhances the model's ability to determine the important parts of the input data in the current context of learning the maliciousness of n-grams [48]. Therefore, it provides better performance for this task compared to the single bidirectional LSTM layer. Attention can be described as utilizing a query and a set of key-value pair vectors to compute a weighted sum of the values which are dependent on the query and the corresponding keys. The query can be described as the values to focus on. To compute the query, key, and value, we should stack the h_t from the previous equation into matrix $H \in \mathbb{R}^{t \times v}$, where t implies the number of timestamps and v denotes the vector size. Then, utilizing linear transformation on the H matrix, we yield the keys K and values V matrices. The queries Q matrix is equal to h_{t-1}. Finally, to compute the attention, we use the following equation that allows the model to attend to the important parts of the n-gram, where d_k denotes the dimension of the queries and keys.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Prediction. Finally, our task determines how probable the input n-gram is benign or malicious (i.e., a lower score means benign while a higher one indicates maliciousness). The fully-connected dense layer will grasp the information from the attention layer and utilizing the Sigmoid function, we can determine the probability of maliciousness using the following equation, where x denotes the output from the dense layer:

$$h_(x) = \frac{1}{1 + e^{-x}}$$

After the complete training of the model, it is saved on disk (i.e., the model's weights, architecture, etc.) to be utilized in predicting n-grams maliciousness. It should be noted that n-grams fed to the model are from benign and malicious binaries, so there will be overlapping in some n-grams. Hence, the model can find the string `"cmd.exe"` in a benign binary as well as in a malicious one. One of the implications of the overlapping n-grams could be that it can reduce the performance of the detection process: as a malware detection system distinguishes malware and benignware based on the n-grams of the input samples, n-grams that appear both in the malicious and benign programs could bring to unclear information and make it more difficult to decide a clear boundary between the two classes. The above-mentioned problem can be relieved in this work: with the bi-directional LSTM model, we are able to capture the structural information of the input n-gram sequence across different timestamps, other than merely relying on the text-level information. In addition, with the attention mechanism applied to the discriminative model, NeuroYara can focus on the n-grams that provide most of the variants between malware-related and benignware-related patterns. Thus, the impact of the overlapping issue can be reduced,

and the proposed model is able to learn the distribution of n-grams that have a higher chance of being an indicator of a malicious binary.

2.3 Human N-grams Language Model

Here, we utilize a bidirectional LSTM language model to determine the importance and usefulness of the extracted n-grams based on learning the language of the n-grams utilized by malware analysts. This model works on giving probabilities to each n-gram based on the understating of the manually generated n-grams and how probable a specific n-gram belongs to this language. Therefore, given a sequence of tokens or bytes, the language model assigns a probability distribution to the next token, in order to understand the underlying meaning of the language in the given context. To further elaborate on the workflow of this step, we provide a running example with the symbols referencing the previous sections. In this step, we train the human n-gram language model F_h to generate the distribution over the vocabulary based on the input context. We train F_h to learn the most possible n-gram x_t for constructing effective Yara rules at each timestamp t with the human-extracted Yara rules x_h. Generally, language models have several applications such as the next word prediction, speech recognition, spelling correction, and machine translation. Similar to the discriminative probabilistic model, current state-of-the-art language models are based on RNNs architectures [31], such as GRUs [15], LSTMs, and transformers [18]. LSTM models have shown great potential in dealing with natural language processing (NLP) tasks [17], [35].

2.3.1 *Human Language Model's Methodology*

The human model after training takes only around 10MBs of disk space while holding the massive previous knowledge of manually generated Yara rules and signatures. The total number of manually generated n-grams the model is trained on is over 130 thousand n-grams. Figure 4 shows the training and architecture of the human language model. As in the discriminative probabilistic model, the process and model's architecture can be summarized as follows:

Preparation. The first step is the collection of high-quality and recently updated manual Yara rules. We pulled a huge amount of Yara rules from recently updated repositories. More details on the rules and their authors are available in Section 3.

Extraction. We extract byte sequences that are utilized as signatures in rules we pulled from the repositories. Using the sliding windows approach, we extract the n-grams.

Embedding. In the model, we utilized the input n-grams to embed them using the embedding layer, in the same way as the discriminative probabilistic model.

Encoding. The embedded n-grams are passed to a bidirectional LSTM layer. The insights for this encoding step are similar to the ones we discussed in the discriminative probabilistic model, where we would like to learn the semantic patterns from the input n-gram sequence. However, there are two main differences between this model and the discriminative probabilistic model. First, our task is to learn the language itself and the context, not just distinguish between being benign and malicious based on knowing the

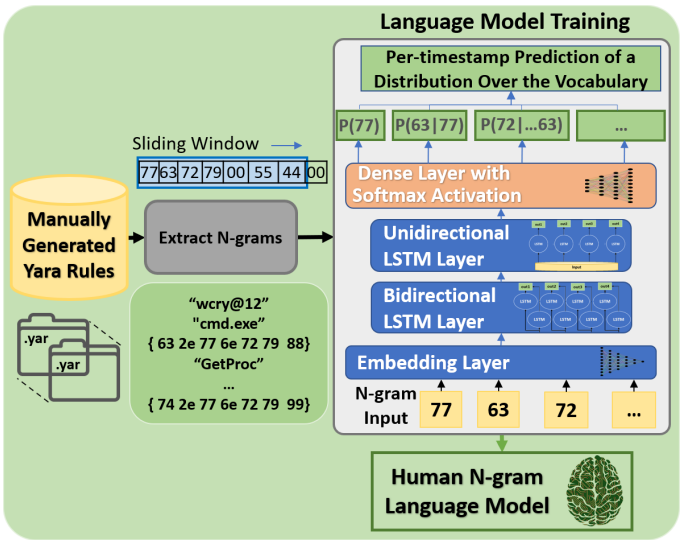


Fig. 4. Human language model demonstrating how it provides probabilities to n-grams, based on how likely their sequential dependencies are observed in manually-generated signatures. After training, the model will be holding manually generated signatures' language knowledge.

important parts of the sequence. Second, we cannot use an attention layer, since it will lead to information leakage while learning because it works on learning the language by the probability of the next token in the sequence. It will know the upcoming tokens beforehand, which is considered cheating, as it can give a false sense of high performance. Hence, we utilize the output of the bidirectional LSTM layer as input to a unidirectional LSTM layer, but not an attention layer. The model learns the context of the language by doing per-timestamp predictions of the distribution of possible tokens or bytes over the vocabulary of this language. To formalize, let $x_1, x_2, ...x_n$ be tokens from a n-gram. We want to compute the probability $P(x_1, x_2, \ldots, x_n)$, which is the probability of seeing these tokens in this specific order (i.e., belonging to this language's context). This can be calculated using the chain rule that can be formalized as:

$$P(x_1, \ldots, x_n) = P(x_1) \cdot P(x_2 \mid x_1) \cdots P(x_n \mid x_1, .., x_{n-1}) = \prod_{t=1}^{n} P(x_t \mid x < t)$$

Prediction. The fully connected dense layer will grasp the information fed by the unidirectional LSTM. This model works on understanding the language of the manually generated rules, by parsing the n-grams and trying to predict the next bytes in the n-gram. Therefore, the model can predict how probable a specific n-gram belongs to the language that it learned. That is why we utilize a Softmax layer for this task, rather than a Sigmoid. The Softmax calculates such probability using the following equation, where, $(X)_i$ is the input vector to the Softmax, e^{x_i} is the exponential of the output of the fully-connected dense layer, K is the number of the classification classes, and the summation formula is a normalization term to ensure that the output values sum to 1 and each is in the range (0, 1).

$$\text{softmax}(X)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

We can save the trained model on disk to be utilized to score and rank the n-grams scored by the language model. To combine the two models' scores, we attempted different methods, such as addition, multiplication, and averaging. However, it was found that simply adding the scores obtained from the two models provided the best ranking performance.

HDBSCAN. Density-based clustering is an algorithm that attempts to group data points that are closely packed together and mark data points that belong to a low-density region as outliers. One of the most commonly used density-based clustering algorithms is DBSCAN [19]. HDBSCAN extends DBSCAN by converting it into a hierarchical clustering algorithm. This allows to find clusters of variable densities, not just high-density data points. Therefore, it does a great job in clustering complex-shaped clusters, with uneven densities and with the existence of noise and outliers [14].

2.3.2 *HDBSCAN N-grams Clustering*

The NeuroYara pipeline models n-grams in three stages. In the first two stages, it models individual n-gram's potential maliciousness and usefulness separately. In the third stage, it models the correlations among n-grams on malware-level observations through clustering. Thus, before we finally construct the Yara rules based on the n-gram signatures we obtained from the first two steps, we run HDBSCAN [14] to cluster those n-grams. There are two main reasons to choose HDBSCAN for the clustering stage: first, the n-gram space of the malware samples is sparse and typically leads to varying densities across clusters, thus requiring independent thresholds for robust grouping. This introduces challenges for non-hierarchical clustering methods that use a single-density threshold or global hyperparameter such as K-Nearest Neighbours (KNN). Second, with a vast number of n-grams in the dataset, there exists a large number of outliers. since HDBSCAN is robust against outliers and can produce high-quality clusters in varying-density situations, it helps to relieve the above problem.

The purpose of the clustering task is to generate groups of n-grams and strings using Boolean `ANDing` and `ORing` conditions so that a Yara file can capture all variant samples of the same malware family. We provide the formal description of the input and output of the HDBSCAN n-grams clustering as follows: for each malware family, given its n-gram set $X = \{x_1, ..., x_n | x_i \in \mathbb{R}^d\}$ with d represents the dimension of each n-gram x_i, the goal of HDBSCAN in our work is to decide a function F_H that clusters the input into n-gram groups such that $F_H : X \rightarrow \{c_1, .., c_m\}$, where each $c_i = \{x_1, ..., x_k | x_j \in X\}$ represents an n-gram group, k represents the number of n-grams included in c_i, and m represents the total number of groups. In this work, the minimum number of n-grams included in each c_i is set to 1. Figure 5 shows an example of 5 binary samples of the known malware family `wannacry`, where 6 top n-grams were found. As can be seen, the string `"r.wnry/"` and `"s.wnry/"` together match the first two samples, `".wnry\x"` match samples 2 and 3, `"taskdl."` and `"taskse."` together matches samples 1 and 4, while `"wcry@12"` matches sample 5. Surely, this would be more complex in a real example, where we would have many more n-grams (byte sequences and strings) and maybe more samples as well. Additionally, some n-grams could just add a redundant condition that has already been covered by other n-grams. Hence, the task of HDBSCAN clustering is to cluster n-grams such that when formatted into Yara rule's syntax, it yields an output as in Figure 6.

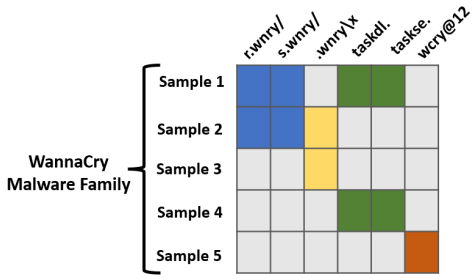


Fig. 5. Visualization of the clustering task of top n-grams.

One of the design choices that we had to perform empirically in order to determine the best value is the number of the top k n-grams to be chosen from each sample of a malware family. There seems to be no best practice for the number of top n-grams. We decided to try the ranges of **k** from 3 up to 100 n-grams per sample. After clustering the n-grams and generating the Yara rules, it was found that the best value for **k** is 12. Therefore, after scoring and ranking the n-grams, we pick the top 12 n-grams from each sample of a malware family. Then, we utilize HDBSCAN to group the useful n-grams by labelling the clustered n-grams together and employ logical statements to construct the Yara rules.

3 Experimental Setup

We prepared a collection of high-quality and recently updated manual Yara rules to be used as the state-of-the-art baseline to compare NeuroYara's generated rules. We pulled a huge amount of Yara rules from recently updated repositories such the CAPE malware sandbox [4], FireEye [5], as well as repositories containing a combination of rules [9], [6] combined with repositories of expert malware analysts from different companies, such as Microsoft [30], Fox-IT Security [20], Florian Roth, the National Security Agency (NSA), the Federal Bureau of Investigation (FBI) and FireEye. Other repositories, such as [2] maintained by the AIL Framework as well as [1], and [3] were also included. These rules are mainly utilized for the detection of malware families,

```
rule wannacry
{ strings:
   $wrcy1 = "r.wnry/"
   $wrcy2 = "s.wnry/"
   $wrcy3 = ".wnry\x"
   $wrcy4 = "taskdl."
   $wrcy5 = "taskse."
   $wrcy6 = "wcry@12"
   condition:
   ($wrcy1 and $wrcy2) or ($wrcy3) or ($wrcy4 and $wrcy5) or ($wrcy6)}
```

Fig. 6. Example of clustering output of `wannacry` malware family's n-grams, reformatted in Yara rule's syntax and incorporating Boolean expressions.

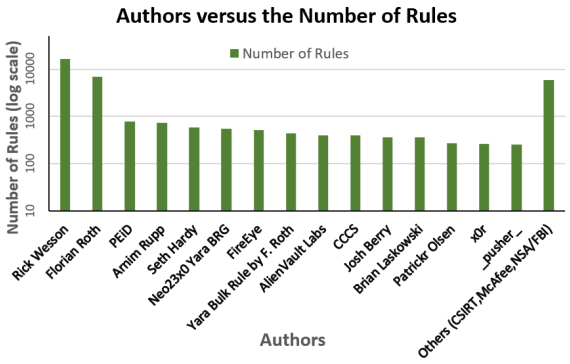


Fig. 7. Visualization of the top 15 recurring authors of the manually-generated rules versus the number of corresponding rules by each author in log scale.

but some of them may serve additional purposes, such as capturing packing techniques and indicators of compromise (IoC). Figure 7 summarizes the top recurring rules' authors and the number of rules in the log scale.

In real-life cases where a malware analyst is required to construct Yara rules for a specific family, he/she generally has very few binary samples or variants of the malware. At the same time, the malware analyst's task is to come up with high-quality Yara rules that should have a high detection rate for the binaries and all the variants of the same family, while maintaining a very low FP rate. The Yara rule should not match any of the benignware or misinterpret a malware sample to be from another malware family. Therefore, in order to test the performance of NeuroYara and other state-of-the-art tools that automatically generate Yara rules, we prepared different types of datasets. There is a huge dataset of various updated malware and benignware types to be used as the training and testing sets. The original dataset had over 100,000 32/64-bit packed and unpacked malware binaries, and over 7,000 benign executables from different software vendors. The dataset was labelled by VirusTotal [7] utilizing multiple Anti-Virus (AV) engines, and these labels were then aggregated using AVClass [44]. Our data corpus includes a collection of over 350 malware families, including notable examples like Dridex, DarkComet, and Azorult. The time frame covered in the malware samples is from 2011 to 2021, which encompasses the malware data from the last ten years when our experiment was conducted.

To simulate real-life tasks, we did 3 main experiments:

(1) Varying Malware Samples for Rules Construction. We created 4 datasets with varying numbers of malware for rules construction and did FP testing on the benign binaries. For each dataset, the families included in the training are not included in the testing. The input data is in the format of binary samples. The goal of the proposed model is to generate the Yara rules that can be used to correctly match the input malware binaries with their families. The datasets are labelled by VirusTotal, with the ground truth being the families of the input binaries. **(1) Varying Malware Samples for Rules Construction.** We created 4 datasets with varying numbers of malware for rules construction and did FP testing on the benign binaries. For each dataset, the families included in the training are not included in the testing. The input data is in the format of hexadecimal code. The goal of the proposed model is to generate the Yara rules that can be used to correctly match the input malware binaries with their families. The ground truth is the malware families, with benign binaries given a different family label from the malware families. The datasets are labelled by VirusTotal [7].

TABLE 1
Datasets description showing the number of benignware and malware used for rules construction and evaluation.

	Rules Construction		Evaluation	
	Malware Families	Malware Binaries	Malware Binaries	Benign Binaries
Scene 3	92	276	705	7828
Scene 4	85	340	680	7828
Scene 8	67	536	592	7828
Scene 16	55	880	474	7828

(2) Zero-day Malware Families. To Evaluate Neuro-Yara's ability to construct high-quality rules for malware families that had no manually generated rules in the reference set of the Human Language Model, we developed another dataset with malware families that had no manual rules. Moreover, we included more malware families and binaries.

(3) Adversarial Inputs Evaluation. In this experiment, we utilized a reinforcement-based model to manipulate binaries and generate variances that are meant to bypass detection based on the approach done by Anderson et al. [11]. We utilized the dataset used in experiment 2, which incorporated the zero-day malware with additional testing samples.

3.1 Varying Malware Samples for Rules Construction

To simulate real situations of the tasks of a malware analyst, we created 4 datasets from the malware and benignware dataset. In order to produce unbiased results, it should be noted that there is no overlap between the reference malware and benignware dataset used in the training of the discriminative probabilistic model, and the datasets we are using to construct and evaluate the resulting rules datasets are meant to evaluate NeuroYara's performance in all experiments.

The main difference between the 4 datasets is the number of samples per family used for rules construction. For example, the first dataset contains 92 malware families and for each family, we have 3 binary samples for signature construction, and each family's rules are constructed independently. Since the number of samples for rules construction is 3, we name this dataset setup scene 3. For the evaluation set, all the remaining binaries in the database of the families we have included in the rules construction set are added to the evaluation set. Additionally, we add the benignware binaries to the evaluation set for FP testing. Similarly, we created three other datasets with different numbers of binary samples for rule construction. Table 1 summarizes the four scenes, displaying the number of malware families and binaries for rules construction and the evaluation of malware and benignware's count.

In this experiment, we evaluate NeuroYara and the baselines using different techniques and metrics, which will be discussed in detail in the next section. Additionally,

TABLE 2
Zero-day malware dataset description showing the number of benignware and malware used for rules construction and evaluation.

	Rules Construction		Evaluation	
	Malware Families	Malware Binaries	Malware Binaries	Benign Binaries
Scene 3	208	624	1487	7828

we evaluate the tools based on their performance in FP predictions for benign binaries.

In order to construct the datasets provided in Table 1, we had to consider only malware that already has manually generated rules available to compare our performance with. This is the reason why the total number of malware samples is less than in the original dataset. Moreover, the number of binaries for rules construction and evaluation depends on another factor. For each scene, we select malware families that have a binary sample count at least greater than the scene number. For example, in scene 4, we have to select only malware families which have at least 5 binary samples (i.e., at least 4 for rules construction and 1 for evaluation). We chose the number of samples per malware family (i.e., scenes) based on what a real malware analyst would be facing.

3.2 Zero-day Malware Families

To evaluate NeuroYara's performance with malware families and samples that did not have rules in the manually generated rules used for training the human language model, we constructed a more challenging rules construction and evaluation dataset. We included only 3 binary samples per family to construct rules, for more than double the malware families, as well as added more testing variants and samples, compared to scene 3 in the previous experiment.

Table 2 summarizes the number of binaries and malware families, as well as the number of benign samples utilized in this experiment. Similar to the previous one, we chose malware families that have at least 4 binary samples, in order to create the rules construction and evaluation sets for scene 3 (i.e., at least 3 malware samples per family for rules construction and 1 for the evaluation set).

3.3 Adversarial Inputs Evaluation

This experiment is mostly overlooked in previous works. In an actual adversarial environment, the malware authors will try to break the detection methods utilized. They can deliberately manipulate malware binaries to confuse the automated detection to increase the FP and FN rates of the automated tool. Hence, we implemented a reinforcement-based model using a random agent to change the content of the binaries and generate variances that are meant to evade detection [11]. The model does several modifications to the binaries, such as renaming sections, modifying the machine type and timestamp, adding imports, padding, and benign strings, as well as break the optional header checksum.

We motivate our choice of using this method in the adversarial input evaluation experiment as follows: In real-world scenarios, adversaries continuously evolve and adapt their techniques to bypass malware detection systems, and the RL learning system simulates this process. By using a reinforcement learning model, we are able to mimic this adaptive behaviour by continuously producing adversarial malware that is evasive to existing antivirus products and solutions. Additionally, the system [11] explores various evasion strategies with its rich agent-action patterns. This reflects the diverse tactics that malicious actors might employ to bypass security mechanisms. The reinforcement adversarial agent conducts the modifications on the malware binaries by changing the content of the binaries. For example, it could rename the sections, modify the machine type and timestamp, append benign strings and break the optional header checksum.Our experiment shows that NeuroYara is able to generate satisfactory results under the adversarial scenario. One contributing factor to its robustness is the discriminative model of NeuroYara. It learns to score string tokens based on their maliciousness and avoid an on-purpose injection of benign strings. Additionally, the human language model of NeuroYara also contributes to this process to further filter out false positive strings.

We utilized scene 3 from experiment 2, which includes only zero-day malware and a more challenging testing set. We ran the reinforcement model on the evaluation set while maintaining the original rule-construction set. This would be similar to what an adversary would do to confuse the automated detection tool, by increasing the FP and FN rates.

3.4 Training Details

In this subsection, we would like to introduce the training parameters, loss function settings and necessary training details. Specifically, for the discriminative model, the dimension of embedding output is set to 8, the max sequence length is set to 16, and the number of LSTM units is set to 128. For the human n-grams language model, the dimension of embedding output is set to 100, the max sequence length is set to 256, and the number of LSTM units is set to 128. We use the dropout technique to avoid the over-fitting problem in the human n-grams language model, with the dropout rate being 0.3. The loss function for the discriminative model is Binary Crossentropy since the model is trained towards the benignware-malware classification problem; the loss function for the human n-grams language model is Sparse Categorical Crossentropy since the model is trained towards predicting the distribution over the token vocabulary. Adam (Adam) optimizer is used for the backpropagation process, with the learning rate of 1×10^{-5} for the discriminative model and 1×10^{-4} for the human n-grams language model. We utilize the validation set and the early-stopping technique to determine the training epochs and avoid the over-fitting problem. It should be noted that the training of different variances of the model, including the architecture we are using, took less than 6 hours to complete. This indicates that the training process for our model is relatively efficient.

4 Evaluation

To evaluate the performance of NeuroYara, we prepared the datasets setup mentioned in Section 3. To compare our performance with existing works and tools, we utilized three

TABLE 3
Demonstration of per-sample and per-match techniques.

Matches		Per-Sample		Per-Match	
True Family	Predicted Family	True Label	Predicted Label	True Label	Predicted Label
tat505	tat505 ramnit wannacry	1	0	tat505 tat505 tat505	tat505 ramnit wannacry
benign	benign	0	0	benign	benign

state-of-the-art tools, namely Autoyara [36], YarGen [39] and Yaraml [40], developed by Sophos, a security software and hardware company [46]. We tried all of them to see how they performed. Additionally, we utilized and compiled the existing updated high-quality Yara rules [4], [5], [9], [6], [2], [3], as mentioned in 2.3.

To evaluate the rules generated by NeuroYara and the other tools, we match the Yara rules generated on the evaluation set. As mentioned in Section 3, this set contains different variants of malware binaries from all the families in the rule-construction set, in addition to the benign binaries to test the FP rate. After having matched the generated Yara rules, we create a JSON file containing each family in the rule-construction set and in the benign as entries. Inside each entry are the names of the malware families that got matched with the Yara rules generated against the evaluation samples. With a JSON file for each tool and for each dataset, we can evaluate and present different metrics. We utilized 5 metrics in order to evaluate NeuroYara and the other baseline models, namely precision, recall, F1-Score, accuracy, and AUC. Each one of them expresses a different way to measure the performance of the tools.

Precision shows the percentage of malware families correctly predicted by the rules generated and how prone the model is to FP predictions. Recall demonstrates how sensitive the rules are in predicting the correct families and how they perform regarding FN predictions. F1-score is utilized to get the balance (i.e., harmonic mean) between precision and recall. F1-score is used since a Yara rule can have high precision but low recall. This means that it has a low FP rate (i.e., when the rule matches a binary file, its prediction is most probably correct), while it has a high FN rate (i.e., the rules cannot capture all variants of a family). Accuracy shows the true positive (TP) and true negative (TN) performance of the rules. AUC indicates the probability of predicting the correct malware families. We utilized the weighted metrics to account for class imbalance and provide more accurate interpretations of the results.

We developed two methods to evaluate the matched families. An example of the two methods, per-sample and per-match, is shown in Table 3. In this example, we see that we have two binary files. The first is from the `ta505` malware family, and the second is a benign binary file. Per-sample will check if the binary is malicious and only when the correct match is predicted, then we label this prediction as 1. Otherwise, it is 0. If the binary is benign then we set the predicted label to 0, if it was predicted as benign. Otherwise, we set it to 1. Per-match created a multi-class table for all Yara rules families that matched `ta505`. In this case, `ta505`'s rules got matched by 1 correct family (`ta505`) and two incorrect families (`ramnit` and `wannacry`). Hence, for every prediction for the same binary, there will be a separate entry in the table with the true label, as the actual family, and the prediction as the name of the predicted family. In the end, when the two methods are evaluated with metrics, the per-match table will be treated as if it has two out of four correct multi-class predictions. For per-sample, it will be seen that only one out of two was correctly predicted in a binary classification task.

These two methods are developed to evaluate how the rules are effective in predicting only the correct labels, without other malware families or benignware (i.e., in the per-sample technique). On the other hand, it evaluates how many incorrect predictions besides the correct ones are present (i.e., in the per-match technique). Therefore, the more incorrect malware families are predicted, the less score the rules will yield.

5 RESULTS

In this section, we will demonstrate the experimental results and findings based on comparing the manually generated rules, AutoYara, Yaraml, YarGen and NeuroYara on the four different datasets utilizing five different metrics: precision, recall, F1-score, accuracy, as well as AUC and using the two different evaluation techniques: per-sample and per-match.

5.1 Varying Malware Samples for Rules Construction

In a real-life situation, a malware analyst will have a small number of binary samples that behave similarly to learn from, in order to construct high-quality Yara rules for them. This can be depicted in the scene 4 setup, where the tools have four binary samples per family to generate Yara rules. Hence, we will start the discussion of our results by observing the performance of the tools in this setup. As can be seen in Table 4, we display NeuroYara's performance when utilizing only the discriminative model, and then when utilizing the two models to demonstrate the effectiveness of the two components. For the baselines, the manually generated rules perform the best in the per-sample and per-match evaluations. Previous works [55], [36] have indicated that the best-performing Yara rules are the ones generated by malware analysts. NeuroYara is the first automated tool to surpass the performance of manually generated rules. As we can see in Table 4, NeuroYara, with only the discriminative model, provided better performance than existing baselines. In per-sample, it correctly predicted only the actual family without much noise and performed better in multi-class prediction in per-match evaluation. The human language model enhanced the prediction performance across all malware families' overall metrics.

In order to check how the tools perform when having a very small number of binary samples to construct the rules from, we discuss the scene 3 setup shown in Table 5. In this experiment, NeuroYara is trained on both malicious binaries and benign binaries and then tested on malware samples from different families. Despite being counter-intuitive, all of the automated tools generally provided better performance in this scenario and NeuroYara provided a similar performance in scene 4. Since we should consider the F1-score as the balance between precision and recall, we can

TABLE 4
Scene 4 dataset's experimental results for the manually generated rules, AutoYara, Yaraml, YarGen, and NeuroYara showing the precision, recall, F1-score, accuracy, and AUC for per-sample and per-match evaluations. In NeuroYara, the letters **d** and **h** indicate the usage of the discriminative probabilistic model and the human language model respectively.

	Per-Sample					Per-Match				
	Precision	Recall	F1	Accuracy	AUC	Precision	Recall	F1	Accuracy	AUC
Manual	0.895	0.920	0.897	0.920	0.557	**0.898**	0.926	0.904	0.926	0.602
AutoYara	0.810	0.558	0.659	0.558	0.327	0.840	0.358	0.496	0.358	0.271
Yaraml	0.635	0.823	0.716	0.635	0.370	0.823	0.391	0.525	0.391	0.283
YarGen	0.840	0.793	0.815	0.793	0.447	0.806	0.359	0.485	0.359	0.349
NeuroYara $_{\mathbf{d}}$	0.932	0.932	0.910	0.932	0.582	0.888	0.932	0.904	0.932	0.631
NeuroYara $_{\mathbf{d,h}}$	**0.934**	**0.933**	**0.913**	**0.934**	**0.593**	0.897	**0.934**	**0.908**	**0.934**	**0.654**

TABLE 5
Scene 3 dataset's experimental results for the manually generated rules, AutoYara, Yaraml, YarGen and NeuroYara showing the precision, recall, F1-score, accuracy, and AUC for per-sample and per-match evaluations. In NeuroYara, the letters **d** and **h** indicate the usage of the discriminative probabilistic model and the human language model respectively.

	Per-Sample					Per-Match				
	Precision	Recall	F1	Accuracy	AUC	Precision	Recall	F1	Accuracy	AUC
Manual	0.892	0.918	0.894	0.918	0.556	0.891	0.922	0.898	0.922	**0.624**
AutoYara	0.825	0.663	0.733	0.663	0.400	0.876	0.503	0.632	0.503	0.346
Yaraml	0.835	0.822	0.828	0.822	0.452	0.837	0.620	0.711	0.620	0.382
YarGen	0.843	0.837	0.840	0.837	0.482	0.822	0.399	0.528	0.399	0.402
NeuroYara $_{\mathbf{d}}$	**0.933**	**0.932**	**0.910**	**0.932**	0.593	**0.932**	0.888	**0.904**	**0.932**	0.609
NeuroYara $_{\mathbf{d,h}}$	0.930	0.929	**0.910**	0.930	**0.596**	0.895	**0.930**	**0.904**	0.930	0.608

TABLE 6
Scene 8 dataset's experimental results for the manually generated rules, AutoYara, Yaraml, YarGen, and NeuroYara showing the precision, recall, F1-score, accuracy, and AUC for per-sample and per-match evaluations. In NeuroYara, the letters **d** and **h** indicate the usage of the discriminative probabilistic model and the human language model respectively.

	Per-Sample					Per-Match				
	Precision	Recall	F1	Accuracy	AUC	Precision	Recall	F1	Accuracy	AUC
Manual	0.898	0.927	0.905	0.927	0.542	0.899	0.931	0.910	0.931	**0.577**
AutoYara	0.000	0.001	0.000	0.001	0.007	0.049	0.015	0.016	0.015	0.034
Yaraml	0.161	0.649	0.258	0.161	0.091	0.824	0.051	0.086	0.051	0.049
YarGen	0.847	0.685	0.757	0.685	0.390	0.823	0.242	0.351	0.242	0.328
NeuroYara $_{\mathbf{d}}$	**0.939**	**0.939**	**0.918**	**0.939**	**0.574**	0.895	**0.938**	0.913	**0.938**	0.575
NeuroYara $_{\mathbf{d,h}}$	0.934	0.938	0.916	0.938	0.568	**0.902**	**0.938**	**0.914**	**0.938**	0.569

TABLE 7
Scene 16 dataset's experimental results for the manually generated rules, AutoYara, Yaraml, YarGen and NeuroYara showing the precision, recall, F1-score, accuracy, and AUC for per-sample and per-match evaluations. In NeuroYara, the letters **d** and **h** indicate the usage of the discriminative probabilistic model and the human language model respectively.

	Per-Sample					Per-Match				
	Precision	Recall	F1	Accuracy	AUC	Precision	Recall	F1	Accuracy	AUC
Manual	0.909	0.937	0.919	0.937	0.531	0.911	0.940	0.922	0.940	**0.632**
AutoYara	0.000	0.001	0.000	0.001	0.006	0.042	0.014	0.015	0.014	0.092
Yaraml	0.804	0.325	0.462	0.325	0.180	0.829	0.094	0.159	0.094	0.098
YarGen	0.878	0.725	0.794	0.725	0.410	0.838	0.224	0.331	0.224	0.190
NeuroYara $_{\mathbf{d}}$	**0.946**	**0.951**	**0.936**	**0.951**	**0.587**	0.916	**0.946**	0.928	**0.946**	0.593
NeuroYara $_{\mathbf{d,h}}$	0.935	0.947	0.932	0.947	0.578	**0.922**	**0.946**	**0.929**	**0.946**	0.590

see that in this case, NeuroYara with the discriminative model, only performed closer to NeuroYara using both models. Also, the addition of the human model improved the performance in the per-sample AUC, unlike the accuracy and AUC of the per-match.

To test the tools' ability to deal with a large number of malware samples, which is the case when malware has many variants, we will discuss scenes 8 and 16 as presented in Tables 6 and 7. Surprisingly, AutoYara and Yaraml performed much worse when their models were trying to learn on a bigger set of binary samples for rules construction. This is because the rules generated by these tools matched each binary file in the evaluation set with a lot of wrong predictions besides the correct one. This can be specifically demonstrated in the per-sample evaluation. This indicates signatures that are not robust and generalizable enough are used in the Yara rules generated by these tools. These signatures are not specific to this family, hence they match with other binary files. YarGen and the manual rules provided more consistent results, which were still lower than NeuroYara. However, YarGen generally tends to perform worse as the number of rules construction samples of a

malware family increases. Due to the high number of n-grams from the rule-construction set, they mislead the tools in choosing signatures that are not unique to the analyzed family.

NeuroYara provides a more consistent performance as the number of binaries per malware family used for rules construction increases. The addition of the human model to the discriminative probabilistic one in the scene 8 and 16 setups gives us a better intuition of its added value. We can see in Table 6 and 7 that the per-match performance of NeuroYara with the two models is better compared to when only the discriminative model is used, unlike for the per-sample evaluation. This indicates that the human models allow NeuroYara to correctly predict the family of the binary more accurately. However, it may add some incorrect predictions of other malware families. This is reflected in the per-sample evaluation since any other prediction besides the correct one will result in an incorrect prediction. However, we argue that in real-life situations, it is better to capture a malware sample indicating that it is a specific family besides other probable families, rather than not predicting that it is malware at all.

5.1.1 *False Positive Predictions Performance*

This test is one of the most overlooked ones in works that automate Yara rules generation. Here we test the generated rules performance when matched against benign binaries to examine the FP predictions rate. In other words, how frequently the Yara rules match one of the benignware, as the signatures in the Yara rules are not unique to this family. Authors of previous works tend to not report the performance of the rules generated on benignware [12], [36], [55]. This test is essential, especially if this tool will be deployed in an actual environment to assist malware analysts. Hence, we tested and compared the generated rules of NeuroYara, the manual rules, and YarGen, since it is the best-performing tool based on the results discussed in Section 5. The benign binaries utilized for evaluation were over 7,800 binaries, as indicated in Table 1.

Table 8 summarizes the results of the test. In the table, we report the five metrics against the four scenes. The numbers in bold are the highest ones for each metric per scene. As it can be concluded from the table, NeuroYara offers a much better performance and balance between FP and FN rates, as indicated by the F1 score and accuracy. NeuroYara also provides consistent and progressive performance across the scenes, whereas YarGen provided marginally higher AUC in the first scenes, but as we go up the scenes, NeuroYara performs better. However, it should be noted that the most important metric in this test is the F1 score since it is the one that reports how well the rules were able to not match the benign binaries and this is where NeuroYara significantly provides better performance. Therefore, despite being able to outperform the automated tools and manually generated rules, NeuroYara maintained lower FP predictions.

5.2 Zero-day Malware Families

As mentioned in Section 3, to evaluate NeuroYara's performance in constructing rules for malware families that had no manually generated rules to be trained on by the human language model, we developed another dataset with a rules construction set that contains only zero-day malware. Moreover, we added more binaries and variants in the evaluation dataset compared to the original Scene 3 dataset. We also assumed a very low number of binaries for rules construction. Hence, we included in the rule-construction set only 3 binary samples per family to construct rules for.

Table 9 summarizes the results of NeuroYara and the baselines, excluding the manually generated rules, since this dataset contains malware, for which there are no rules. This table demonstrates how NeuroYara's approach provides higher quality rules, despite being evaluated on zero-day malware and a more challenging dataset. NeuroYara provided a significantly better performance compared to the other baselines. Compared to the original scene 3, all tools provided inferior performance and compared to NeuroYara, the other baselines declined faster in performance. Additionally, the inclusion of the human language model enhanced the performance in the per-sample and per-match evaluations. This indicates that the human language model did not overfit on learning the pattern of n-grams and byte sequences of only known malware rule. It actually learned the language and the patterns used by expert malware analysts in developing their rules.

Comparing with Table 4 we could see that, even when the language model is excluded from NeuroYara, the Accuracy score still drops when facing the zero-day malware testing data. We consider the reason for this result as follows: When the language model is excluded from NeuroYara, NeuroYara is still a supervised learning model. The n-gram discriminative model is trained and optimized towards the task of categorizing benign n-grams and malicious n-grams. The training dataset consists of malware and benignware samples. When confronted with zero-day malware samples in testing, there could be new n-grams, n-gram combinations and unknown patterns of novel malware families. This can introduce generalizability challenges for the model when distinguishing n-grams related to benign or malicious zero-day samples. Thus, the filtered n-grams will be of degraded quality, resulting in a lower performance of the Yara rules generated from the n-grams in the zero-day testing case.

5.3 Adversarial Inputs Evaluation

This experiment is mostly overlooked in previous works that propose automated signatures and Yara rules generators. Since these automated approaches will be utilized in an adversarial environment, adversaries and malware developers will find methods to break and confuse these tools. For example, they will deliberately insert or remove bytes in the binaries to confuse the automated detection tools and the rules generated, in order to increase the FP and FN rates. Utilizing a reinforcement-based model using a random agent, we can simulate an adversary that manipulates binaries to generate variances and evade detection, without changing their functionality. The model does several adversarial manipulations to the binary, such as renaming sections, modifying machine type and timestamp, adding imports, padding, and benign strings, and breaking the optional header checksum.

TABLE 8
False positives performance comparison between YarGen, manually-generated rules, and NeuroYara based on testing the rules on benignware on the different 4 scenes, displaying precision (PRC), recall (RCL), F1 score (F1), accuracy (ACC), and AUC.

		Scene 3			Scene 4			Scene 8			Scene 16		
		YarGen	Manual	NeuroYara	YarGen	Manual	NeuroYara	YarGen	Manual	NeuroYara	YarGen	Manual	NeuroYara
Benignware	PRC	**0.971**	0.931	0.935	**0.977**	0.934	0.936	**0.983**	0.940	0.942	**0.987**	0.949	0.956
	RCL	0.463	0.990	**0.996**	0.412	0.990	**0.999**	0.244	0.990	**0.998**	0.229	0.990	**0.995**
	F1	0.627	0.959	**0.965**	0.580	0.961	**0.966**	0.390	0.964	**0.969**	0.372	0.969	**0.975**
	ACC	0.463	0.990	**0.996**	0.412	0.990	**0.999**	0.244	0.990	**0.998**	0.229	0.990	**0.995**
	AUC	**0.707**	0.630	0.620	**0.692**	0.639	0.607	0.614	**0.623**	0.600	0.608	0.589	**0.644**

TABLE 9
Zero-day malware dataset's performance of AutoYara, Yaraml, YarGen and NeuroYara showing the five metrics for per-sample and per-match evaluations. The letters **d** and **h** indicate the use of discriminative and language models respectively.

	Per-Sample					Per-Match				
	Precision	Recall	F1	Accuracy	AUC	Precision	Recall	F1	Accuracy	AUC
AutoYara	0.689	0.653	0.670	0.653	0.414	0.791	0.543	0.620	0.543	0.436
Yaraml	0.683	0.665	0.674	0.665	0.404	0.734	0.560	0.633	0.560	0.358
YarGen	0.690	0.711	0.700	0.711	0.428	0.682	0.330	0.419	0.330	0.348
NeuroYara $_{\mathbf{d}}$	0.870	0.846	0.781	0.846	0.517	0.762	0.846	0.779	0.846	0.558
NeuroYara $_{\mathbf{d,h}}$	**0.875**	**0.855**	**0.800**	**0.855**	**0.545**	**0.803**	**0.847**	**0.793**	**0.847**	**0.639**

TABLE 10
Adversarial dataset's performance of AutoYara, Yaraml, YarGen and NeuroYara showing the five metrics for per-sample and per-match evaluations. The letters **d** and **h** indicate the use of discriminative and language models respectively.

	Per-Sample					Per-Match				
	Precision	Recall	F1	Accuracy	AUC	Precision	Recall	F1	Accuracy	AUC
AutoYara	0.651	0.520	0.577	0.520	0.326	0.780	0.326	0.443	0.326	0.241
Yaraml	0.656	0.553	0.600	0.553	0.336	0.717	0.405	0.516	0.405	0.262
YarGen	0.685	0.676	0.680	0.676	0.411	0.688	0.304	0.403	0.304	0.255
NeuroYara $_{\mathbf{d}}$	**0.869**	0.844	0.777	0.844	0.513	0.758	**0.844**	0.776	**0.844**	0.549
NeuroYara $_{\mathbf{d,h}}$	0.851	**0.851**	**0.796**	**0.851**	**0.540**	**0.793**	**0.844**	**0.790**	**0.844**	**0.630**

We utilized the more challenging scene 3 from the previous experiment, which includes only zero-day malware and low number of malware binaries per family for rules construction. We employed a reinforcement learning-based adversarial malware generation tool [11] on the evaluation set while maintaining the original rule-construction set to simulate the adversarial environment and observe the degradation of matching performance. Table 10 shows the performance of NeuroYara and the baselines with the adversarial changes to the zero-day malware family dataset. NeuroYara provided the best performance by a margin compared to other baselines. Its performance only slightly declined in the adversarial environment compared to the other baselines, which declined more compared to the previous experiment. The addition of the language model provided a more resilient performance in the adversarial environment, despite the fact that the human language model was trained on a different set of malware families.

One possible attack on NeuroYara is to inject spurious Yara rules into the training data set of the human language model as a poisoning attack. As the human language model learns the Yara rule generation process from these input samples, introducing spurious Yara rules can introduce noise and confusion into the model's learning process. This may result in the generation of sequences that are not only ineffective as Yara rules but could also introduce false positives or other undesirable behaviour in the generated rules. To develop adversarial samples against the rules produced by NeuroYara, an attacker would need to have a good understanding of the architecture and components of the deep learning model, specifically the neurons, layers, the architecture of bi-LSTM networks and how a generative model works. The attacker would also need knowledge about the nature of the real-world malware data and the human-generated Yara rules, including the distribution of the training data, and how the data is pre-processed.

5.4 Limitations of the Method and Experiments

In this section, we would like to discuss the limitations of NeuroYara as well as the evaluation experiments. Since the human n-gram model is trained on human-defined Yara rules that are publicly accessible, there exists a potential threat that NeuroYara learns the lexical features of the input pre-existing Yara rule sequences rather than the inherited patterns of malicious and benign binaries. Such an issue could reduce the model's generalizability when encountering zero-day malware from families without human-defined Yara rules. Additionally, our work primarily focuses on the construction process from the perspective of sequential properties of the input binaries. However, malicious samples could differ from benign samples on multiple levels, such as graph structures. In the future, we could expect the model to be further refined by considering a broader array of factors in constructing Yara rule sequences.

Regarding the experiments, our work does not fully explore the impact of manually generated Yara rules. While our experiments with zero-day malware binaries suggest that our model can adapt to out-of-sample families while remaining feasible, a comprehensive investigation is needed to understand the mechanisms through how the manually

generated Yara rules can influence the proposed model in the construction of Yara rules for both in-sample and out-of-sample families. Case studies could be needed in this regard.

In Section 5.2, we present an experiment involving zero-day malware binaries for Scene 3. This experiment serves to evaluate NeuroYara's ability to generalize to out-of-sample binaries. The strong performance observed in this experiment demonstrates that NeuroYara does not merely memorize the training data but also effectively captures the underlying patterns of how to distinguish between different malware families and benign binaries as well as how to generate Yara rules. However, other experiments that do not include zero-day malware families might introduce a conceptual bias in the results. As in the training step, human-defined Yara rules are input into the human language model, and there is a chance that the human language model memorizes the input Yara rules without understanding the inherited properties. In the testing step, including only malware families with human-defined Yara rules may result in the combination of token sequences for distinguishing malware families all seen by the model in the training step. This makes it hard to evaluate whether the considered model generates the token sequence for Yara rule construction based on catching the inherited properties of malware families or merely memorizing the human-defined Yara rules that are input in the training step. Consequently, in the testing process, a high accuracy score could be attributed more to the presence of pre-existing rules than to our proposed model's ability to learn the underlying variant patterns between malware families and benignware.

In the initial design of our study, we made a deliberate choice to employ a conventional random split between the reference set and testing set. This decision was necessitated by the inherent limitations of our available data. Specifically, we encountered challenges in verifying the consistency between the timestamps associated with malware submissions to the data platform and the actual occurrence of these malware samples in the wild. Given these constraints, our primary goal was to focus on the robustness of matching variants for both retro-hunting and future threat-hunting, even when provided with as few as three samples for rule construction. To address this, we repeatedly conducted the evaluation process by randomly selecting only three samples per family for rule construction. This approach enabled us to encompass a wide range of variants for matching evaluation while mitigating potential bias introduced by timestamp constraints.

6 Related Works

Due to the exponential increase of sophisticated malware variants and families, this field has been growing massively. One of the oldest methods for malware detection is hash signature-based detection. It is extensively used by antivirus engines. However, this method, despite being indispensable, has its downsides. It requires keeping a huge database of signatures that has to be frequently updated with the hashes of the new samples [24]. Another approach uses machine learning models to classify malware samples [33]. It has been extensively researched [45]. Most works utilize Support Vector Machines (SVM), Naïve Bayes (NB), or Decision Trees (DT) [23], [26], [42]. More recent approaches utilize deep neural networks [41] and transformers to understand the semantics of malware [28], [54]. However, these complex models do not provide interpretations for their predictions and are prone to high FP rates.

Before the release of Yara rules [10], there were several approaches proposed for automatic malware signature generation. One of the unique approaches proposed by Newsome et al. is Polygraph [11]. It is an automated tool utilized to generate signatures based on distinct invariant substrings that must be generally present in different variants of polymorphic worms. It utilizes 4 different signature classes: Bayes signatures, conjunction signatures, longest common substrings, and token-subsequence signatures. This approach provides resilience against the variations in malware that are designed by malware authors. Li et al. introduced a similar approach to Polygraph called Hamsa [29]. The authors demonstrated this approach as being faster, more noise-tolerant, and attack-resilient.

Since the release of Yara signatures, a lot of organizations and tools have been proposed to supported the automatic Yara rule generation process. Despite being a very effective tool, the field of automatic Yara rules generation is still lacking many possible improvements. Some current tools that generate Yara rules automatically still require manual analysis or significant expert knowledge in malware analysis [12]. Current works indicate that their tools' objective is to expedite the process of Yara rules generation [36]. However, their generated rules have a much lower quality than the manually generated ones. One of the possible reasons is that existing methods have limitations on incorporating experts' knowledge into the malware signature generation process [22], [39], [36], which can lead to information insufficiency and thus decrease the quality of the generated rules. They require significant modifications, which negates the purpose of the rules generation automation. Another main common shortfall is the need to store a huge database of hard-coded benignware strings. This reference set is used to reduce the FP rate, as in the case of the work done by Griffin et al. [21], which stores several GBs worth of data.

Autoyara [36] is one of the most recent works in this area. The authors developed a new bi-clustering algorithm to cluster the n-grams extracted while utilizing large n-gram sizes. Their results indicated that the time spent on rules construction was over 44% faster while maintaining low FP and high detection rates. However, Autoyara has to store a couple of hundred MBs of benign and malicious bytes to function. YarGen [39] is a tool that utilizes Naive Bayes to score the usefulness of byte sequences. It gives scores based on hard-coded byte sequences. It uses a huge database of strings and opcodes of benignware to maintain a low FP rate while requiring 4GBs-8GBs of memory. In our experimental results, it was found that it provided better performance than AutoYara, despite being older. Yaraml [40] is another recent tool developed by Sophos [46]. This tool extracts strings as features from the binary samples and trains one of two machine learning models: Logistic Regression (LR) and Random Forest (RF). It then translates the output of the models into signatures to be written as Yara rules. Yabin [8] is developed by AlienVault's Open Threat Exchange (OTX). Using over 100GBs of stored goodware, it searches function

prologues and stack frames to capture rare functions based on the notion of code reuse in malware.

In the literature, there exists other approaches such as utilizing LCS algorithms to capture the common byte sequences in malware [13] and using the most frequently occurring byte sequences [52], [51]. However, very few works evaluated their automated tools in an adversarial environment [34], [32]. Any automated approach that will be utilized in rules generation or malware detection is vulnerable to adversarial attacks to confuse with the detection of malware. For example, Venkataraman et al. [49] discuss the limits of the automated tools that rely on pattern extraction for signature generation in an adversarial environment. Newsome et al. [32] proposed the Paragraph tool, in order to demonstrate the possible attacks on automated signatures generation tools, specifically on Polygraph. However, work [34] that published one year after Polygraph shows that Polygraph can be easily misled, and causing the resulting signatures to be useless. as the detection rate can drop to 0. Anderson et al. [11] developed a reinforcement-based framework that plays a series of games against an anti-malware engine to learn what changes can be done to the malware to evade detection, without changing its functionality.

Compared with the literatures, NeuroYara's novelty lies in modeling the maliciousness of n-grams without human interventions and a large storage space requirement by learning the semantic meanings of the n-grams, with considerations of the input malware and benignware contexts. The proposed n-gram discriminative model can generalize on novel data not observed in the past. In addition, NeuroYara improves the effectiveness of the generated Yara rules by leveraging human expertise in rule generation. NeuroYara mitigates heterogeneous malware variants and noisy outliers in the n-grams by combining the hierarchical clustering method (HDBSCAN [14]) with the previous learning to rank n-grams models in a unified novel architecture. From the literature review we conducted, YarGen, AutoYara, and Yaraml were found to be the most promising tools for constructing high-quality rules. Thus, we use these methods as baseline models in our experiment. Our experimental results show that NeuroYara outperforms the automated tools and the manual rules, while maintaining a lower FP rate and compressing the discriminational knowledge of benign n-grams, without using hard-coded signatures.

7 Conclusion

Signature-based malware detection solutions are essential in cyber threat intelligence and incidence response, due to their flexibility and maintainability. However, to create robust and generalizable signatures, one is required to have extensive experience in malware analysis. In this paper, we introduced our framework, NeuroYara, an automated neural network-based Yara rules generator for security analysts to expedite the rule construction process when facing a few fresh samples in an adversarial environment. It utilizes two neural network-based models that incorporate the knowledge of benignware n-grams and the manually generated Yara rules. The discriminative and language models work on ranking and filtering the analyzed binary's n-grams to create signatures capable of capturing the binaries being investigated, without being prone to a high FP rate. The framework works on clustering the top n-grams using HDBSCAN to construct Yara rules containing n-grams, strings, and Boolean statements. The experimental results suggest that NeuroYara outperforms state-of-the-art works for zero-day malware in adversarial environments. It is the first to surpass the performance of manual rules. Our promising results should lead to other works that attempt to compress and exploit the knowledge of benignware n-grams and existing Yara rules. One limitation that can be drawn from this work is using a subset of all existing malware families, which upon including some of them in the evaluation process, might affect the performance. However, our performance with the zero-day malware dataset suggests that NeuroYara can handle unseen and sophisticated malware.

References

[1] Ail framework - analysis information leak framework. https://github.com/ail-project/ail-framework. (Accessed on 10/05/2021).

[2] ail-yara-rules: A set of yara rules for the ail framework to detect leak or information disclosure. https://github.com/ail-project/ail-yara-rules. (Accessed on 09/16/2021).

[3] Burp-yara-rules: Yara rules to be used with the burp yara-scanner extension. https://github.com/codewatchorg/Burp-Yara-Rules. (Accessed on 09/16/2021).

[4] Capev2: Malware configuration and payload extraction. https://github.com/kevoreilly/CAPEv2. (Accessed on 09/16/2021).

[5] Fireeye. https://github.com/fireeye/red_team_tool_countermeasures. (Accessed on 09/16/2021).

[6] Neo23x0/signature-base. https://github.com/Neo23x0/signature-base/tree/master/yara. (Accessed on 09/16/2021).

[7] Virustotal. https://www.virustotal.com/gui/home/upload. (Accessed on 11/29/2021).

[8] Yabin: A yara rule generator for finding related samples and hunting. https://github.com/AlienVault-OTX/yabin. (Accessed on 09/16/2021).

[9] Yara-rules: Repository of yara rules. https://github.com/Yara-Rules/rules. (Accessed on 09/16/2021).

[10] Yara. the pattern matching swiss knife. https://github.com/VirusTotal/yara. (Accessed on 09/08/2021).

[11] H. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *ArXiv*, 2018.

[12] F. Bilstein. Automatic generation of code-based yara-signatures a practical approach for the generation of high-qality yara-rules. 2018.

[13] C. Blichmann. *Automatisierte Signaturgenerierung für MalwareStämme*. PhD thesis, Technical University of Dortmund, 2008.

[14] R. Campello, D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. pages 160–172, 2013.

[15] K. Cho, B. van Merriënboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 06 2014.

[16] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. 12 2014.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

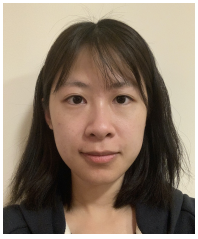
[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231, 1996.

[20] Fox-IT. Fox-it cybersecurity. https://www.fox-it.com/nl-en/. (Accessed on 10/05/2021).

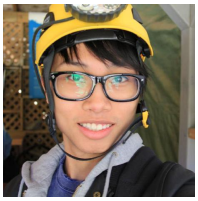
[21] K. Griffin, S. Schneider, X. Hu, and t.-c. Chiueh. Automatic generation of string signatures for malware detection. pages 101–120, 2009.

[22] B. K. HÁJEK. System for assisted creation of yara rules.
[23] O. Henchiri and N. Japkowicz. A feature selection and evaluation scheme for computer virus detection. In *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*, pages 891–895. IEEE Computer Society, 2006.
[24] F. Hsu, H. Chen, T. Ristenpart, J. Li, and Z. Su. Back to the future: A framework for automatic malware removal and system repair. In *22nd Annual Computer Security Applications Conference (ACSAC 2006), 11-15 December 2006, Miami Beach, Florida, USA*, pages 257–268, 2006.
[25] M. Khalid, M. Ismail, M. Hussain, and M. Hanif Durad. Automatic yara rule generation. In *2020 International Conference on Cyber Warfare and Security (ICCWS)*, pages 1–5, 2020.
[26] J. Z. Kolter and M. A. Maloof. Learning to detect malicious executables in the wild. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 470–478. ACM, 2004.
[27] A. .K.S. Impact of malware in modern society. *Journal of Scientific Research and Development*, 2:593–600, 06 2019.
[28] M. Q. Li, B. C. M. Fung, P. Charland, and S. H. H. Ding. I-MAD: A novel interpretable malware detector using hierarchical transformer. abs/1909.06865, 2019.
[29] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez. Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. In *2006 IEEE Symposium on Security and Privacy (S P'06)*, pages 15 pp.–47, 2006.
[30] Microsoft. Microsoft software. https://www.microsoft.com. (Accessed on 10/05/2021).
[31] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. Recurrent neural network based language model. volume 2, pages 1045–1048, 2010.
[32] J. Newsome, B. Karp, and D. Song. Paragraph: Thwarting signature learning by training maliciously. In D. Zamboni and C. Kruegel, editors, *Recent Advances in Intrusion Detection*, pages 81–105, 2006.
[33] O. Or-Meir, N. Nissim, Y. Elovici, and L. Rokach. Dynamic malware analysis in the modern era - A state of the art survey. *ACM Comput. Surv.*, 52(5):88:1–88:48, 2019.
[34] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *2006 IEEE Symposium on Security and Privacy (S P'06)*, pages 15 pp.–31, 2006.
[35] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.
[36] E. Raff, R. Zak, G. Lopez Munoz, W. Fleming, H. S. Anderson, B. Filar, C. Nicholas, and J. Holt. Automatic yara rule generation using biclustering. In *Proceedings of the 13th ACM Workshop on Artificial Intelligence and Security*, page 71–82, 2020.
[37] G. Ramesh and A. Menen. Automated dynamic approach for detecting ransomware using finite-state machine. *Decision Support Systems*, 138:113400, 11 2020.
[38] D. Reddy and A. K. Pujari. N-gram analysis for computer virus detection. *Journal in Computer Virology*, 2:231–239, 12 2006.
[39] F. Roth. yargen: a generator for yara rules. https://github.com/Neo23x0/yarGen, 2013.
[40] J. Saxe. Yaraml. https://github.com/sophos-ai/yaraml_rules/, 2020.
[41] J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features. 08 2015.
[42] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo. Data mining methods for detection of new malicious executables. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 38–49. IEEE Computer Society, 2001.
[43] M. Schuster and K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.
[44] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero. Avclass: A tool for massive malware labeling. In F. Monrose, M. Dacier, G. Blanc, and J. Garcia-Alfaro, editors, *Research in Attacks, Intrusions, and Defenses*, pages 230–253. Springer International Publishing, 2016.
[45] A. Shalaginov, S. Banin, A. Dehghantanha, and K. Franke. Machine learning aided static malware analysis: A survey and tutorial. 2018.
[46] Sophos. Sophos: a british security software and hardware company. https://www.sophos.com/en-us/index.aspx. (Accessed on 09/19/2021).
[47] M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. 09 2012.
[48] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. 2017.
[49] S. Venkataraman, A. Blum, and D. Song. Limits of learning-based signature generation with adversaries. In *NDSS*. The Internet Society, 2008.
[50] D. Votipka, S. Rabin, K. Micinski, J. Foster, and M. Mazurek. An observational investigation of reverse engineers' process and mental models. pages 1–6, 04 2019.
[51] H. Wang, S. Jha, and V. Ganapathy. Netspy: Automatic generation of spyware signatures for nids. *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 99–108, 2006.
[52] K. Wang, G. Ciocarlie, and S. Stolfo. Anomalous payload-based worm detection and signature generation. pages 227–246, 09 2005.
[53] L. Xu and M. Qiao. Yara rule enhancement using bert-based strings language model. In *2022 5th International Conference on Advanced Electronic Materials, Computers and Software Engineering (AEMCSE)*, pages 221–224, 2022.
[54] Z. Yu, R. Cao, Q. Tang, S. Nie, J. Huang, and S. Wu. Order matters: Semantic-aware neural networks for binary code similarity detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:1145–1152, 04 2020.
[55] A. Zhdanov. Generation of static yara-signatures using genetic algorithm. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pages 220–228, 2019.

Ziad Mansour Ziad Mansour received a B.E. degree from Cairo University in 2019. He is currently pursuing a master's degree at the School of Computing, Queen's University. His research focused on leveraging machine learning to solve complex cybersecurity challenges especially on malware analysis. In 2022, he joined Rapid7, Toronto, ON, Canada, and currently is responsible for developing and improving Rapid7's security platforms.



WEIHAN OU received her master's degree in machine learning from University College London (UCL), United Kingdom, in 2016. She is currently pursuing a Ph.D. degree in the School of Computing, at Queen's University, Canada, where she works in the L1NNA lab with Dr. Steven Ding. She worked with Dr. Nadia Berthouze and Dr. Youngjun Cho on stress level recognition at UCL, in 2017. Her current research focuses on solving cyber-security problems using machine learning models.



Steven H. H. Ding Dr. Steven Ding is an Assistant Professor in the School of Computing at Queen's University, where he leads the L1NNA Artificial Intelligence and Security Lab. His research bridges the domain of machine learning, data mining, and cybersecurity, aiming at addressing cybersecurity challenges using AI technologies and securing the future of AI systems. Dr. Ding obtained his Ph.D. from McGill University in 2019, and he was awarded the FRQNT Doctoral Research Scholarship of Quebec and the Dean's Graduate Award at McGill.

Mohammad Zulkernine Dr. Mohammad Zulkernine is a Professor and Tier 1 Canada Research Chair in Cyber-Physical System Security at the School of Computing of Queen's University, Canada. Dr. Zulkernine also serves as the Founding Director of the Queen's Centre for Security and Privacy. He is cross-appointed at the Department of Electrical and Computer Engineering of Queen's and the Royal Military College of Canada. Previously, Dr. Zulkernine served as the Graduate Chair of the School of Computing from 2019 to 2020 and held a Tier 2 Canada Research Chair in Software Dependability from 2011 to 2021. He joined Queen's in 2003 and spent his sabbatical at the University of Trento, Italy and Irdeto Canada. Dr. Zulkernine is a licensed professional engineer in Ontario, Canada.

Philippe Charland Philippe Charland is a Defence Scientist at Defence Research and Development Canada – Valcartier Research Centre, in the Mission Critical Cyber Security Section, where he leads the Systems Vulnerabilities and Lethality Group. His research focuses on software reverse engineering, more specifically on the development of binary analysis tools to accelerate the reverse engineering process involved in malware and embedded software analysis. Mr. Charland holds a bachelor's and a master's degree in Computer Science, both from Concordia University, Montreal, Canada.