Toward classified Instruction Set Architecture using Raw Byte Input to Convolutional Neural Network

Duc Tri Nguyen
George Mason University
Fairfax, VA
dnguye69@gmu.edu

***Abstract*—IDS/IPS is widely deployed in practice, it can be used to detect exploitation stage on network layer, according to our understanding, depend on header of malicious files, a certain mechanism is performed. In this paper, we tackle detecting shellcode or malformed instruction code send over networks without header or footer, our result shows that without expensive feature engineering, using direct raw byte data as input to our convolutional neural network, we are able to detect what instruction set architecture the shellcode belong to with the accuracy 93%. We propose a module help IDS/IPS reduce workload by only activate defend mechanism base on deep learning answers rather than applying on each incoming packet.**

***Index Terms*—Classify, Instruction Set Architecture, Deep Learning, Neural Network, Security**

I. Introduction

The detection of malicious code is an important topic in cybersecurity, especially as more of devices, architectures join an office network. Manual signature-based or heuristics-based and analysis procedures are still very prominent, however, with the increasing number of malware per day [1], the number of signatures applied in Intrusion Detection System and Intrusion Prevention System (IDS/IPS) also increases significantly per day, a good IDS/IPS can slow down network packet for a few seconds to check malicious traffic, it proved to work in the past, however, can IDS/IPS catch up with current malware rate on multiple platforms in the future is still an unknown question.

The most popular method is dynamic analysis. While intuitively appealing, there are many issues with dynamic analysis in practice, such as the instrumented environment or customized Virtual Machine which introduce high computational requirements, and usually not reflect the target environment of malware. Furthermore, in some case, malware can detect and alter its behaviors, allowing it to avoid discovery. In many cases, malware often obfuscates its code to hide its important keyword (urls, strings...) also increases its robustness from static analysis.

At network level, the only option one can have to filter malicious content is static analysis. Simply put, our goal is to use a fast and simple method to assist IDS/IPS in categorizing, classifying what architecture of the current packet belong to, E.g a downloading firmware by a printer in MIPS, malicious code in ARM webcam ...and apply certain filter mechanism on certain platforms. Furthermore, it does not only avoid repeating useless filters across platforms but also improves the performance of IDS/IPS.

From the point of view at IDS/IPS, which operate at network level, we decide to take a static analysis approach, our goal is to make our approach practical, instead of spending time on preprocessing data and feature extracting like traditional machine learning approach, we feed raw data into Neural Network (perhaps very small portion of time for decapsulate packets) and return what architecture the packets belong to, then let IDS/IPS does its job.

This leads us to our **Research Question**: Given raw bytes of data, is it random data or executing code, if it is executing code, can we identify which Instruction Set Architecture to by using Neural Network?

Contribution of this paper:

- First in class in applying Neural Network in classifying Instruction Set Architecture from raw data,
- Convolutional Neural Network model that can classify Instruction Set Architecture base on byte sequence with high accuracy,
- The module that can be easily applied, extend to existing IDS/IPSs.

II. Background

A. Instruction Set Architecture

In simple words, Instruction Set Architecture (ISA) is assembly code run on a certain platform, E.g: x86, nios, powerpc, arm....

An assembly instruction can contain:

- Opcode
- Registers
- Memory address
- Operands

In the scope of classifying ISA, the same sequence of raw bytes can be interpreted differently by different platforms, due to different platforms have different raw bytes opcodes for the same operation, it's also applied to registers, operands, etc ...

Despite many differences between many ISAs, we can categorize them into 2 sets:

1) CISC: Complex Instruction Set Computer, instructions have dynamic length.

TABLE I
ASSEMBLY INSTRUCTIONS OF X86_64 VERSUS ARMV8-RP3

CISC		RISC	
4155	push r13	014040e3	movt r4, 1
4531e4	xor r12d, r12d	0950a0e1	mov r5, sb
55	push rbp	0960a0e1	mov r6, sb
4889742448	mov qword [var_8h], rsi	24008de5	str r0, [sp + var_8h]

TABLE II
FEATURE LIST OF ISAS

Arch	?-bit	32-bit	64-bit	Little Endian	Big Endian
alpha		✓			
arc		✓		✓	
arm		✓	✓	✓	✓
avr	8				
m68k		✓			✓
mips		✓	✓		✓
mipsel		✓	✓	✓	
msp430	16				
nios2		✓		✓	
powerpc		✓	✓	✓	✓
riscv		✓	✓	✓	
s390		✓	✓		✓
sh		✓		✓	
sparc		✓	✓		✓
x86_64		✓	✓	✓	
xtensa		✓		✓	

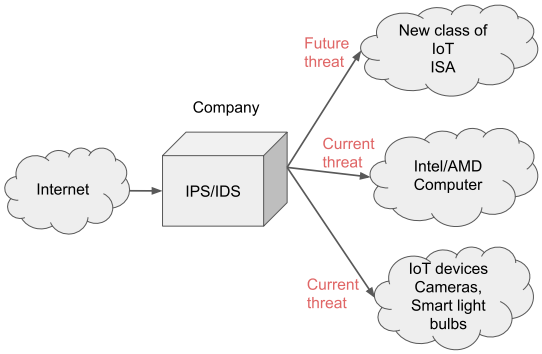

Fig. 1. Threat model with current IDS/IPS

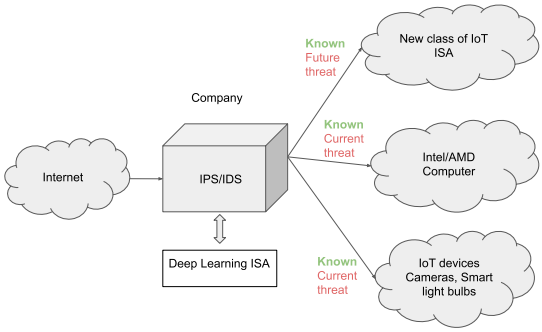

Fig. 2. Improved our threat model with Neural Network

2) RISC: Reduce Instruct Set Computer, instructions have fix length.

As shown in Table. II-A2, the raw bytes of instructions in CISC and RISC have different length, it is useful features can be used in classifying ISA. Thus, in training, with RISC, raw bytes input to Neural Network doesn't require padding in contrast to CISC.

Table. II-A2, show the number of features that can affect to raw bytes instruction sequence, such as the order of bytes like Little Endian, Big Endian, also the architecture bitwidth can affect raw bytes instruction as well.

B. Threat model

Excluding network topology and only take into account ISA classes, our threat model is displayed in Fig. II-B. We take a random company office as an example, inside the company infrastructure has multiple devices, most of them are office computers using *x86_64* ISA, some of them are smart devices using *MIPS*, *ARM* ISA, etc ..., and future IoT devices with unknown ISA.

One could argue that IDS/IPS can stop almost all the threat such as malware, virus, worm, etc... in x86_64 ISA computer due to its long history fighting malware on such platform, then placing an IDS/IPS at the network level, combine with antivirus software installed on each computer on each laptop, computer in company network then the network is probably secure.

Although we agree with the argument and see nothing wrong with that (what could be better?), however, we see that it is not enough to prevent all threats coming to network company, one can see that we cannot have protection mechanism on Application Layer (OSI model) on IoT devices, e.g it is not possible to install antivirus for MIPS, ARM devices are not possible due to its lack of OS. That lead to an insecure situation, the current IoT devices threat and future threat are depended on IDS/IPS.

On the surface, we know that IDS/IPS works by detecting exploitation stage, hence we can simplify how it detect exploit:

- By signatures
- By heuristics

As state in [1], each day there are hundred thousand of new malware, if there is no known signature for new malware, the IDS/IPS cannot detect new threat. Additionally, if there is an IoT malware that goes through IDS/IPS, it is obviously that applying x86_64 malware signature to e.g a MIPS device is a waste of time and computing power.

To overcome the challenge, our goal is to build a module that helps IDS/IPS to identify the ISAs, thus, it can apply its mechanism to filter out malicious content as shown in Fig. II-B, Deep Learning ISA is the module that IDS/IPS can simply send to and receive responses from and act according to the response. For example, if the return answer from Deep Learning ISA module is known ISA, then IDS/IPS can activate its signature and heuristic filter, otherwise, let the packets go through.

III. METHODOLOGY

In this section, we describe the methodology we use to perform the task of ISA classification. We first provide in-

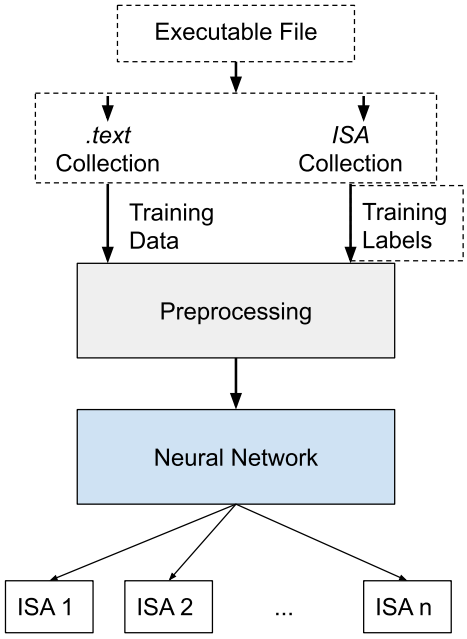

Fig. 3. Overview of ISA training system

formation regarding our set up environment and then proceed with our deployed techniques.

A. System Description

Our ISA classification process is shown in Figure. III-A. It begins with executable files, we use radare2 [2] script to extract raw bytes instruction from *.text* section and its ISA to our collection, the output of raw bytes instructions and ISA are treated as training data and labels.

The result of these executions are preprocessed to get numerical feature vectors, these vectors are then forwarded as inputs to neural networks for training.

The classification system is similar to our training system, but without the collection step and training labels, the raw byte input is considered as Training Data and then forward into preprocessing step and then Neural Network, the expected output is predefined ISA families.

B. Dataset

For neural network to work properly, it requires a huge amount of training data to perform higher accuracy than traditional machine learning algorithms. Within the scope of the class project, the dataset is only limited to *coreutils* source code [3].

It took approximately 48 hours to compile *crosstool-ng* for 94 build features, among them, there are 32 unique ISA features, since we focus on the raw byte instruction set, any feature affect the raw bytes instruction presentation is selected as build options as well, for example: 64-bit vs. 32-bits, Little Endian vs. Big Endian, and ISA version such as ARMv6, ARMv7, ARMv8..

For each ISA feature, it is used to cross-compile *coreutils* source code, the executable files are forwarded as input to our training system. Some build features do not work in Ubuntu18.04 Linux-gnu platform, such as Linux-musl, unknown-elf (for bare metal) build option, in some cases, we drop a build feature if it takes more than 5 hours to fix compilation bugs.

Finally, we have dataset of 19 ISA features, in total 817Mb from 2040 Linux executable files, all executable files are dynamically linked, we actively avoid repeated function, which in turn reduces bias in training data.

In [4], in Section. 3, it stated that the public training dataset is in severe overfitting, so they use the private dataset from an anti-virus industry partner, where samples were taken from real machines. It raises a question that for raw bytes malware analysis, what is the metric to determine either training data is good or bad. As a result, when we preprocessing data, we believe data taken from *.text* section is developed by community therefore the coding style and logic of the raw bytes instruction are not biased.

C. Feature Preprocessing

In our preprocessing stage, training data and labels are converted to numerical vectors.

Training data input is raw bytes sequence, we split them into 64 bytes chunk and pad a chunk with NULL bytes if its length is less than chunk 64 bytes. Next, we use Tokenizer function in Keras [5] to convert the input into numerical vectors. Training labels are encoded by one-hot encoding into numerical vectors.

D. Deep Neural Network

One of the benefits of deep learning over other machine learning techniques is its ability to be applied over raw data without the need for manual and domain-specific feature engineering.

To maximize the utilization of the possibilities of neural network methodologies, in [6] they combine convolutional and recurrent layers in one neural network. Due to our limited understanding of neural network, we were only able to construct a convolutional neural network (CNN).

As shown in Fig. III-D, we use 8 layers, among them, we use Dropout to prevent over-fitting, a Softmax layer to output label probabilities, Dense layers act as input or hidden layers, Activation layer act as combine function.

IV. Evaluation

A. ISA detection challenges

From the best of our knowledge, there are 2 methods can help identify ISA by looking at a sequence of raw byte:

- Heuristic: Need expensive analysis
- Neural Network

Heuristic analysis is used in many popular disassemblers, such as IDA Pro, radare2, thus, this is trial and error process with high false-positive in disassembler, one raw byte sequence can belong to one ISA but can be disassembled to mnemonic instruction in other ISAs, also it is very easy to fail into junk code trap [7], instruction reordering, which in turn increase the analysis time significantly. Besides, this approach

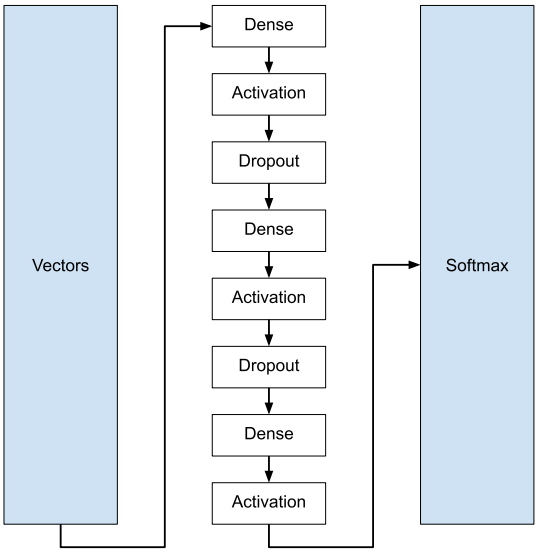

Fig. 4. Deep Neural Network Architecture

TABLE III
NO. OF ATTEMPTS WITH ACCURACY AS TRAINING METRIC

Neural Network	Encoder	Layers	Accuracy
CNN 3rd	**Tokenizer**	**8**	**93%**
CNN 2nd	One-hot	4	15%
CNN 1st	Tokenizer	4	09%

does not fit to our threat model at network level because of its expensive analysis cause delay in network, our goal is to identify ISA as fast as possible manner.

Seeing that Neural Network is excellent at classifying, categorizing data, since there is no previous work in this topic, we think this is the right time to apply Neural Network in ISA detection.

B. Neural Network in ISA detection

In this section, we describe the outcomes of our experiments executed using the dataset in Section. III-B. We separate the dataset into 2 sets with ratio 80/20 for training and testing.

As shown in Table. IV-B, we applied various settings and configuration, the *CNN 1st* is to use Tokenizer as encoder with only 4 layers, however, the accuracy is only 9%.

On the 2nd attempt, we use One-hot as our encoder with the same setting, the accuracy result slightly increases, we believe that the encoder contributes to the outcome. Thinking that the input plays a significant role in neural network, we improve the Preprocessing step, consider the preprocessing step as formal, we now can proceed to tweak our neural network without further experimenting the use cases of Encoder.

As a result, in the last attempt, we add 2 Dropout, 1 Dense and 1 Activation layers, the number of layers now increase to 8, we see the result increase significantly to 93% with the number of *epoch* as 20. After a few tweaks with *epoch* numbers, we stop after we see that the accuracy of our system is not improved further than 93%.

V. DISCUSSION

Here are the limitations of our work:

- Instruction reordering can fool heuristic search, at this time
- Not taken into account the alignment, we believe the sliding window method will solve this.
- No support for raw bytes SIMD instruction, we think that SIMD instructions are mainly used to accelerate computation, if the attacker uses SIMD instruction for software exploitation, eventually they have to switch to normal instruction to make system calls, where it could be detected by our classification system. We are not aware of pure SIMD instructions in software exploitation.
- Our dataset is only limited to Linux-gnu, musl ABI is considered as future work, Window binaries are considered as well.
- We acknowledge that the output of raw bytes instruction sequence heavily depends on compilers, such as Clang, GCC, Visual Studio C/C++, etc ... and compiler version, to overcome this difficulty, we must expand our training data although we think that the hidden structure of an ISA will not change much.

VI. RELATED WORK

In 2015, [8] discuss an approach whereby they apply a convolutional neural network for binary classification to disassembled malware byte code. The raw disassembled code is further processed to generate a more regularized set of features. For example, they extract the individual x86 processor instructions, which are variable length, and then apply padding or truncation to create fixed length features. They also parse the disassembled code to extract code imports, which they use to generate a further fixed-length feature vector for each example.

[4] propose the deep learning malware classification model by feeding entire raw bytes malware PE files, almost no preprocessing and require no domain knowledge, in which, close to our work as we training on raw bytes. The proposed model split the input data into chunks, then forward them to 2D convolution net, 13 layers, use RNN, and embedding layers to support variable input length.

There was comparison [9] between three deep learning-based approaches for IoT malware detection (but their scope only limit at x86), at their experiment result show, by converting raw byte malware into image and use convolutional network CNN_IMG the accuracy tune-up to 100%, the second approach CNN_SEQ is close to our work, they take raw data sequence in executable and writable section from executable files (inspired from [4]), which render low effort in preprocessing step but only achieve 90.58% accuracy, their last approach use CNN_ASM, require a disassembler tool to generalize assembly instructions, then forward them to train, the result is approximately 99%-100%.

In [10], it shows that without complex feature engineering, their classifier achieves a high accuracy 98.2% and only take

0.02s to process one binary file on regular desktop workstations, make it practical in real-world. They combine CNN with LSTM architecture, improve the performance compared to other CNN models.

In 2016, [6] combines CNN and RNN, input data is one-hot encoding system-call API in dynamic analysis, in comparison, deep learning is outperforming best traditional machine learning approach, require no specialist for feature engineering, in all experiments deep learning is way better than machine learning using the same input.

Recently, [11] take raw bytes sequence approach

All of the related works attempt to classify malware to which family its belongs to. Many approaches use raw byte sequence demonstrate high accuracy without expensive feature engineering in the preprocessing step. To some extent, our methodology and their have similarities, but none has attempted to classify ISAs.

VII. Future Work

Our current architecture only uses CNN architecture and support fixed-size input, adopting RNN, LSTM to our architecture is considered as future work. Adopting techniques such as Max Pooling, Embedding to improve our architecture accuracy. Lastly, deploy our deep learning module to real-world IDS/IPS and perform benchmarking.

References

[1] AV-Test, "Malware statistic per day," 2020.

[2] "radare2 UNIX-like reverse engineering framework and command-line toolset ," https://github.com/radareorg/radare2, 2020.

[3] "GNU coreutils," https://github.com/coreutils/coreutils, 2020.

[4] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. Nicholas, "Malware detection by eating a whole exe," 2017.

[5] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[6] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2016, pp. 137–149.

[7] "Recognizing and Avoiding Disassembled Junk," https://www.fireeye.com/blog/threat-research/2017/12/recognizing-and-avoiding-disassembled-junk.html, 2017.

[8] A. Davis and M. Wolff, "Deep learning on disassembly data," *BlackHat USA*, 2015.

[9] K. D. T. Nguyen, T. M. Tuan, S. H. Le, A. P. Viet, M. Ogawa, and N. L. Minh, "Comparison of three deep learning-based approaches for iot malware detection," in *2018 10th International Conference on Knowledge and Systems Engineering (KSE)*, 2018, pp. 382–388.

[10] Q. Le, O. Boydell, B. Mac Namee, and M. Scanlon, "Deep learning at the shallow end: Malware classification for non-domain experts," *Digital Investigation*, vol. 26, p. S118–S126, Jul 2018. [Online]. Available: http://dx.doi.org/10.1016/j.diin.2018.04.024

[11] S. E. Coull and C. Gardner, "Activation analysis of a byte-based deep neural network for malware classification," *2019 IEEE Security and Privacy Workshops (SPW)*, May 2019. [Online]. Available: http://dx.doi.org/10.1109/SPW.2019.00017