CFDVex: A Novel Feature Extraction Method for Detecting Cross-Architecture IoT Malware

Tran Nghi Phu
VNU University of Engineering and Technology
People's Security Academy (PSA)
Hanoi, Vietnam
tnphvan@gmail.com

Le Huy Hoang
People's Security Academy (PSA)
Hanoi, Vietnam
hoangle.hvan@gmail.com

Nguyen Ngoc Toan
People's Security Academy (PSA)
Hanoi, Vietnam
ngoctoan.hvan@gmail.com

Nguyen Dai Tho
VNU University of Engineering and Technology
Hanoi, Vietnam
UMI UMMISCO 209 (IRD/UPMC)
nguyendaitho@vnu.edu.vn

Nguyen Ngoc Binh
The Kyoto College of Graduate Studies for Informatics (KCGI)
Kyoto, Japan
nn_binh@kcg.edu

ABSTRACT

The widespread adoption of Internet of Things (IoT) devices built on different architectures gave rise to the creation and development of multi-architecture malware for mass compromise. Cross-architecture malware detection plays an important role in detecting malware early on devices using new or strange architectures. Prior knowledge of malware detection on traditional architectures can be inherited for the same task on new and uncommon ones. Basing on CFD and Vex intermediate representation, we propose a feature selection method to detect cross-architecture malware, called CFDVex. Experimental evaluation of the proposed approach on our large IoT dataset achieved good results for cross-architecture malware detection. We only trained a SVM model by Intel 80386 architecture samples, our method could detect the IoT malware for the MIPS architecture samples with 95.72% of accuracy and 2.81% false positive rate.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; • **Security and Privacy** → *Systems Security*; Intrusion/anomaly detection and malware mitigation.

KEYWORDS

IoT, Malware detection, CFDVex, Cross-architecture

ACM Reference Format:
Tran Nghi Phu, Le Huy Hoang, Nguyen Ngoc Toan, Nguyen Dai Tho, and Nguyen Ngoc Binh. 2019. CFDVex: A Novel Feature Extraction Method for Detecting Cross-Architecture IoT Malware. In *SoICT' 19: The Tenth International Symposium on Information and Communication Technology, December 4–6, 2019, Hanoi - Ha Long Bay, Vietnam.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3368926.3369702

1 INTRODUCTION

In the last few years, the Internet of Things (IoT) becomes a more and more important trend in the world. It has an emergent range of application domains such as healthcare, energy management, intelligent transport systems, smart building, smart city, military. This rapidly increasing popularity has attracted the attention of malware developers, therefore malware is a potential security challenge. In the first half of 2018, there were more than 120,000 IoT malware instances detected by Kaspersky IoT Lab, while there were only 193 malware samples in 2014 and 7,242 ones reported in 2017 [1]. Only some emerging IoT malware such as Tsunami, Mirai, Brickerbot are really significant and worldwide. Embedded Linux is known as the most popular operating system (OS) for IoT devices [2]. Therefore, detecting novel malware on Embedded Linux OS of IoT devices is a big challenge, due to the diversity of types, a broad range of applications, increasing computing and processing capabilities of IoT devices.

One of the most major challenges in designing IoT malware detection systems is the generation of a lightweight cross-architecture signature generation scheme for detecting and classifying IoT malware [3]. The signature-based malware detection methods [4] attempt to model the malicious behavior of malware and use the model in the detection of malware. These methods are widely used by security vendors, but they are ineffective against IoT malware, especially those that exploit zero-day vulnerabilities, or unknown malware

Different from traditional malware, one unique characteristic of IoT malware is cross-platform capability. In heterogeneous IoT infrastructures, different processor architectures and OSs are supported [5]. There is a sharp increase in the number of malware samples that can run on different OSs such as Windows, Linux, Android [6]. Besides, a malware source could be compiled on many CPU architectures, therefore its compiled instances could run on IoT devices using different CPU architectures such as Intel 80386, ARM, MIPS, PowerPC, etc, these kinds of malware are called cross-architecture malware. There are much research that mention how to detect multi-platform malware such as Alhanahnah *et al.* [3], Alam *et al.* [7].

IoT devices used to be hardware components with small to medium size software drivers and applications to enable a limited interface to those components [8]. Static analysis is a method of analyzing and examining malware based on their characteristics without execution, which is one of two important research directions in the malware analysis and detection. Hence, static analysis can become an efficient method on IoT malware analysis and detection because encryption and obfuscation techniques are not commonly used by the malware. In the past, there had been many researches on static analysis achieved good results in malware detection such as API call [9], PSI (Printable String Information) [10], opcode (Operation Code) [11], CFG (Control Flow Graph) [12], etc.

Some feature types at high abstract level do not depend on architectures such as PSI, API call, etc. They can be used to detect cross-architecture malware. In the recent years, cross-architecture malware detection focuses on intermediate representation (IR). Kim *et al.* [13] proposed an intermediate representation for binary analysis, but they can only find semantic bugs in binary lifters. Sepp *et al.* [14] proposed an extension of REIL with relational information by translating the flags (an instruction's side effects) calculations into arithmetic instructions. However, REIL cannot handle self-modifying code because REIL instructions can not be overwritten or modified during the interpretation of REIL code. It does not support all the architectures that we wanted. Zhoa *et al.* [7] proposed a new intermediate language major in malware analysis, named MAIL, which can analyze and detect metamorphic malware. MAIL provides an abstract representation of an assembly program and hence the ability for a tool to automate malware analysis and detection. The experiments of MAIL referred to testing with ARM and Intel 80386, but its dataset is a little samples (250 malware and 1,137 benign), unbalance in size of CFG and do not mention cross-architecture detection clearly. Therefore, it's hard to fairly evaluate its accuracy. In summary, in our knowledge, there is no research to solve cross-architecture malware detection with full scripts and a large size dataset enough.

Furthermore, Vex is an intermediate representation used in Valgrind [15], a famous dynamic binary instrumentation tool. It has an architecture-agnostic, side-effects-free representation of a number of target machine languages. The uplifting of binary code into Vex is quite well supported. It abstracts machine code into a representation designed to make program analysis easier [16]. Additionally, the VINE intermediate language (VINE-IL) proposed by Song *et al.* [17] is an intermediate language of the static analysis framework VINE used in the BitBlaze project, which is used in tool Panorama [18] for malware analysis and detection. VINE first translates a binary to Vex, and then to VINE-IL.

Internally, Vex is used by Angr [19, 36], a famous open source malware analysis. Angr chosen Vex as its intermediate representation because reliable translation methods from many architectures already existed in Valgrind [20]. Vex was the only choice that offered an open library and supported for many architectures. As a bonus, it is very well documented and designed specifically for program analysis, making it very easy to use in Angr.

Control flow-based features, which combine both CFG and opcode, achieve high malware detection accuracy. Using opcode to detect malware, was firstly proposed by Bilar [21]. Afterward, many researches based on opcode like [11, 22, 23] have been done. Santos *et al.* [23] have suggested the Idea method to detect variants of known malware families based on frequency of appearance of opcode sequences. From opcode sequences, they built vector representation of the executable binaries.

The Control flow-based feature extraction method proposed by Ding *et al.* has the ability to detect malicious code with higher accuracy than traditional Text-based methods. However, The Ding *et al.*'s method encountered NP-hard problem in a graph, therefore, it is not feasible with the large-size and high-complexity programs. In our previous work [24], we proposed the CFD (called as C500-CFG) algorithm for extraction of Control flow-based features based on the idea of dynamic programming with polynomial complexity O(N^2), where N is the number of basic blocks in decompiled executable codes. Thus, it is more efficient and more effective in detecting malware than the old one: processing faster, extracting feature of large files, using less memory and detecting IoT malware with high accuracy.

Thus, we propose, in this paper, a feature selection method for cross-architecture malware detection, named CFDVex. Our method is based on Vex intermediate representation and the idea of CFD method. The CFD method extracted Control flow-based features based on opcode, therefore we calculated n-gram of opcode stream concatenated from all execution paths of CFG. The CFDVex extracts Control flow-based features based on Vex instead of opcode, by calculating n-gram of Vex stream concatenated from all execution path of CFG. By translating to Vex, Angr could extract CFG from executable program with high accuracy. Hence, we can achieve good results by combining synchronously Vex of basic blocks with CFG extracted based on Vex compare with other IRs. The Vex intermediate representation is extracted at two levels of information including Vex command type information and specific Vex command information, respectively. CFDVex is trained by Support Vector Machine (SVM)[25] in three scenarios including Vex-based malware detection, mixed playback capabilities, and the ability to detect cross-architecture malware. Vex-based malware detection is a malware detection method based on evaluation and training on the same architecture dataset. Mixed detection means training and testing data are multi-architectures dataset. Malware cross-architecture detection is a malware detection method that evaluates the model with a different architecture dataset from the training model.

Costin *et al.* [26] analyzed 32,000 firmware images, reported that Linux was the most frequently encountered embedded OS in their dataset – being present in more than three quarters (86%) of all analyzed firmware images. Pa *et al.* [27] proposed IoTPOT, a honeypot has been collected about 4,000 IoT malware samples such as Tsunami, Mirai, Bashlite etc. Another IoT malware database that we can mention is Detux [28] with more than 9,000 samples. Beside IoT malware samples, it is crucial to collect benign files to be able to implement detection algorithms. Brash [29] has collected 1,078 benign and 128 malware samples for ARM-based IoT applications. In their experiments, Alhanahnah *et al* only collected 130 benign IoT samples, which is also small compared with the number of malware samples. Alhanahnah *et al.* [3] said that IoT malware dataset provided by IoTPOT was the largest IoT malware dataset currently available. But with these above datasets, it is not true. A sufficiently large and dataset with full architecture types are needed to ensure fairly and accurately in algorithm evaluation. Therefore, we collected the IoT dataset which is the largest IoT dataset currently available.

In this paper, our experimental dataset focuses on two main architectures of Intel 80386 and MIPS. There are 8 types of architecture in the IoT dataset, but MIPS and Intel 80386 are larger, and they represent two popular platforms: PCs and embedded devices. Experimental evaluation of the proposed approach using our dataset yields malware detection achieved good results. The main contributions of the paper include:

- We propose an novel CFDVex feature selection method based on combining between Vex IR and CFD, as a novel feature extraction method for cross-architecture malware detection. The CFDVex achieved a high malware detection accuracy and a low FPR on cross-architecture IoT dataset.
- We make an assessment and evaluate the relationships of malware on different architectures based on experimentation using our dataset yields malware detection.
- We build IoT dataset which is the largest IoT dataset currently available for multi-architecture.

The remainder of this paper is organized as follows: Section II shows knowledge of Vex intermediate representation; In section III, we present the main idea and introduce our proposal; We experiment and evaluate the performance of the methods in section IV; Finally, the conclusions are given in section V.

2 VEX INTERMEDIATE REPRESENTATION

The Vex IR [16] abstracts away several architecture differences when dealing with different architectures, allowing a single analysis to run on all of them: Register names, Memory access, Memory segmentation, Instruction side-effects. This representation has four main classes of objects:

- *IR Expressions* represent a calculated or constant value;
- *IR Operations* describe a modification of IR Expressions;
- *IR Temporary variables* are used as internal registers, *IR Expressions* are stored in temporary variables;
- *IR Statements* model changes in the state of the target machine, such as the effect of memory stores and register writes, *IR Statements* use *IR Expressions* for values they may need;
- *IR Blocks* are a collection of IR Statements, representing an extended basic block in the target architecture.

Vex IR is actually well documented in the libvex_ir.h file [30]. An example of IR translation from opcodes on MIPS architecture is presented in Table 1. In the example, the (*push ebp*) opcode is translated into 5 IR Statements; the (*mov ebp, esp*) opcode is translated into 2 IR Statements, each of which contains at least one IR Expression.

There are 11 types of IR Statements [30] shown in Statement type Column of Table 2. A statement has many templates for how to cooperate IR Expressions, Operations and IR Temporary variables.

3 OUR METHOD

The CFDVex feature extraction method extracts Control flow-based Vex IR behaviors using CFD algorithm idea [24]. Each vertex is a basic block of a Vex representation sequence instead of an opcode sequence. An opcode statement contains opcode name and parameters, CFD only gets opcode name, which is first word in an opcode statement, to generate the opcode sequence. Vex statements have many forms than opcode statements. They have a lot of types, each type includes many templates, therefore we must find out how to select their representation.

We propose two ways to select a Vex statement's representation called CFDVex level 1 and level 2. At CFDVex level 1, we only get an IR Statement type as a Vex statement's representation. It means a Vex's representation of a Vex statement is determined as a Statement type in Statement type Column of Table 2. And at the CFDVex level 2, a main expression of each statement is selected as the **proposed representations** column in Table 2. There are many IR operations such as Add8, Add32, Sub32, Mul32, Shl32, CmpEQ32 etc, therefore the *Opnames* in the **proposed representations** column in Table 2 get the value corresponding to that statement. There are 2 examples presented in Table 1 that show how to get Vex level 1 and Vex level 2 from an IR Statement.

Table 1: Opcode translation to Vex IR, Vex level 1 and Vex level 2

Opcode	Vex IR	Level 1	Level 2
push ebp	- IMark(0x6570, 1, 0) -		
	t0 = GET:I32(offset=28)	WrTmp	Get
	t10 = GET:I32(offset=24)	WrTmp	Get
	t9 = Sub32(t10,0x0004)	WrTmp	Sub32
	PUT(offset=24) = t9	Put	Put
	STle(t9) = t0	Store	STle
mov ebp, esp	- IMark(0x6571, 2, 0) -		
	PUT(offset=28) = t9	Put	Put
	PUT(offset=68) = 0x6573	Put	Put.cons

According to CFD Algorithm [24], there are 3 phases to extract Control flow-based features from a decompiled executable program. Firstly, a CFG is extracted from the decompiled executable program. Secondly, a Execution graph (E-Graph, called C500-Graph) is constructed from the CFG based on the E-Graph Algorithm [24]. Finally, from the E-Graph, Control flow-based features with n-gram based on Vex is computed by Algorithm 1.

In Algorithm 1, the *getNgramVexConnect(u,v)* function computes an n-gram Vex frequency vector of the Vex sequence that includes *(n-1)* Vex's representations at the end of vertex *u* and *(n-1)* Vex's representations at the begin of vertex *v*, where *n* is the length of n-gram. The function *getNgramVexOfVertex(u)* calculates an n-gram Vex's representation frequency vector of the Vex's representation sequence of vertex *u*.

Example for a Vex Level 1 stream of a MIPS ELF block:

Put Put Put WrTmp WrTmp WrTmp Put Store Put Put StoreG CAS WrTmp WrTmp WrTmp WrTmp WrTmp WrTmp

Example for a Vex Level 2 stream of a MIPS ELF block:

```
Input: E-Graph GC = (V, A, r, L, C, D)
n: the length of n-gram
Output: A Control flow-based Vex features vector with n-gram feature
1: feature = 0
2: for u in V do
3:     sumU = 0
4:     for v in getChildNodesOf(u) do
5:         sumU = sumU + D[u,v]
6:         feature = feature + D[u,v] * getNgramVexConnect(u,v)
7:     end for
8:     feature = feature + getNgramVexOfVertex(u) * sumU
9: end for
```

Algorithm 1: Extracting Control flow-based Vex features with n-gram from E-Graph

Table 2: Vex IR statement types, statement templates list and its proposed representation

Statement type	Templates	Proposed representation
Put	Put(add) = tmp Put(add) = constant	Put Put.cons
PutI	PutI(add) = tmp PutI(add) = constant	PutI PutI.cons

WrTmp	tmp = GET (add)
	Tmp = Constant
	Tmp = tmp
	Tmp = op
	Tmp = LDle(tmp)
	Tmp = LDle(constant)
		Get
		Constant
		Copy
		Opnames
		LDle
		LDle

Store	STle (add) = tmp STle (add) = constant STle(tmp) = tmp STle(tmp) = constant	STle STle.cons STle STle.cons
LoadG	LoadG	LoadG
StoreG	StoreG	StoreG
CAS	CAS	CAS
LLSC	LLSC	LLSC

Dirty	Dirty(constant)
	Dirty(RdTmp, constant)
		Dirty
		Dirty.cons

MBE	MBE	MBE
Exit	Exit	Exit

Put Put.cons Put.cons Get Get Sub32 Put STle Put Put.cons StoreG CAS Add32 Sub32 Mul32 Mul32 Shl32 CmpEQ32

After a Control flow-based features with n-gram of an executable program was extracted by Algorithm 1, a machine learning method will be used to train and detect malware.

The complexity of E-Graph building algorithm is O(N^2), where N is the number of basic blocks in a decompiled executable program [24]. The complexity of Control flow-based features from E-Graph extracting algorithm with n-gram is also O(N^2), because it is determined by the number of *for* loops at line number 2, 4 in the Algorithm 1. In summary, the complexity of CFDVex algorithm is O(N^2).

4 THE EXPERIMENTS

4.1 IoT Dataset

We collected IoT malware from many sources such as IoTPoT [27], Detux [28], and VirrusShare [33]. After collecting, we filtered only executable ELF files and checked in VirusTotal [34]. Benign samples are extracted from more than 23,000 firmware image of routers [35] such as Asus, Belkin, Tenvis, Dlink, TP Link, Linksys, Trendnet, Centurylink, Zyxel, Openwrt, etc. Intel 80386 and X86-64 benign samples were collected from new installation Ubuntu OSs with some common applications on PCs. Malware and benign samples spread almost common IoT architectures such as MIPS, ARM, PowerPC, Motorola, SPARC, RenesasSH and PC architectures like Intel X86-64. The number of samples distributing follow architecture is presented in Table 3. The ***Shared*** column, shows the number of samples existed in the three sources (Virus-Share, IoTPoT, Detux), is below 5% of the total malware samples. It means that the collected malware dataset from the three sources are almost dependent on each other. Our IoT dataset is a useful dataset for cross-architecture malware detection. The number of benign samples is one of the most limited previous researches, but it is large enough in this dataset. Although there are still absent benign samples of some architectures, but to our knowledge, our IoT dataset is the largest IoT dataset currently available, the size of the IoT dataset is 9,380 MB and can be get from *http://firmware.vn*.

Table 3: Our IoT dataset statistic information

	Virus -Share	IoTPot	Detux	Shared	Total Mal	Total Benign
MIPS	1,603	935	3,282	798	5,022	1,899
ARM	2,117	912	35	26	3,038	530
Intel 80386	5,492	570	29	5	6,086	1,438
X86-64	586	320	11	3	914	180
SPARC	584	299	8	4	887	
Motorola	1,455	294	4	5	1,748	
PowerPC	699	353	12	4	1,060	60
Renesas	646	310	2	3	955	
	13,182	**3,993**	**3,383**	**848**	**19,710**	**4,107**

4.2 Environment Setup

Our experiments were run on the 64-bits Ubuntu 16.04.3 operating system, with 2x12-core CPUs, Intel Xeon E5-1600 family, 64GB RAM. A CFG of ELF file was extracted by Angr's CFGEmulated method because it reached high accuracy [24]. We got a Vex sequence from a basic block by the Angr framework Angr because of supporting Vex IR at both level 1 and level 2. Feature extraction and machine learning method [25] were installed on the Scikit-learn Python library 0.19.2.

4.3 Measurements

The performance of the classifiers is evaluated by four criteria: Accuracy, F1-Score, False Negative Rate (FNR), and False Positive Rate (FPR). We define the following measures:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4}$$

$$FPR = \frac{FP}{TN + FP} \tag{5}$$

$$FNR = \frac{FN}{TP + FN} \tag{6}$$

where:

- TP: The number of malware samples truly predicted to be malware
- FP: The number of benign samples truly predicted to be malware.
- FN: The number of malware samples truly predicted to be benign.
- TN: The number of benign samples truly predicted to be benign.

4.4 Feature reduction and Machine learning method

Chi-Square [31] feature reduction method is used to check the relevance between two events, which are the appearance of features and class labels. It is one of most efficient feature reduction methods in the text classification and get high accuracy when cooperate with CFD [24]. In Chi-Square, K is an important paramater, define as the number of dimentions of feature vector after reducing. If K is too small, it lacks of information to classify. The larger it is, the lower speed of classification reduce.

Chi-Square applied in [24] reached a high accuracy at $K = 300$. The experimental results in CFD [24] clarified that the 2-gram feature extraction is better than the 3-gram feature extraction for the same MIPS dataset. Hence, we selected $K = 300$ and 2-gram for our experiments.

For Vex IR level 1, there are only 11 elements of the Vex IR statement types as mentioning in Table 2. If we apply 2-gram feature selection to CFDVex, there are 121 dimensions for the feature vector. It's small, therefore we do not need to reduce dimensions of the feature vector. For the Vex IR level 2, there are more than 300 elements of the Vex IR statement representation as mentioning in Table 2. The number of 2-gram vector's dimensions are large, therefore we will use a feature reduction method.

The SVM classification is a highly efficient method of binary classification, also reached high accuracy with CFD [24]. We used SVM with the sigmoid function kernel and grid search method, which can find the best parameter set. In the experiments, we use a 5-folk cross-validation method with the best parameter set. Data are divided into five different parts with four training parts and one testing part for each experiment. Measures such as accuracy, FPR, FNR, F1-Score are calculated as the average of five times in these experiments.

4.5 Performing single comparison on MIPS, Intel 80386

Ding *et al.*'s method was too slow to extract features of all MIPS samples in our IoT dataset, therefore we only use T1 set was a small part of the IoT dataset, which was presented in our previous research [24] with 844 MIPS samples of 300 benign and 544 malware samples. Figure 1 shows comparison between Ding *et al.*'s method [12], CFD [24] and CFDVex level 2 on MIPS architecture samples of T1. In the figure, the dotted bar shows accuracy of the three methods, the solid bar shows F1-Score of the three methods. We noted that, CFDVex level 2 got a higher accuracy and a higher F1-Score than Ding *et al.*'s method, but a little lower than CFD based on opcode. It means that the Vex IR has a good ability to detect malware, of course, there is still a loss of information in the translation process compared to using opcode.

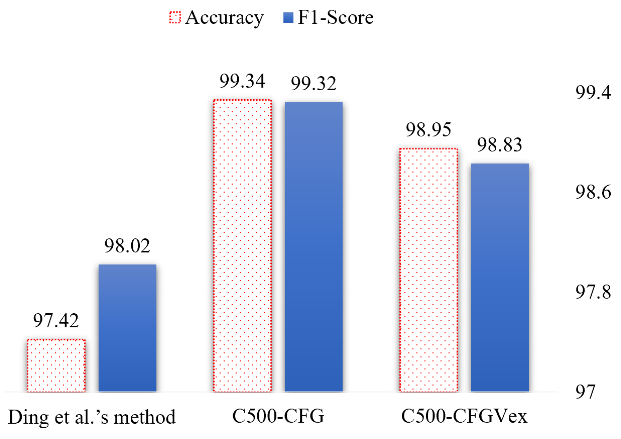

Figure 1: Comparison between CFD, Ding *et al.*'s method and CFDVex level 2 on MIPS architecture dataset

The experimental results for the CFDVex method's malware detection capacity are shown in Table 4. The average accuracy for Intel 80386 architecture is 98.6% at the CFDVex level 1 and 98.96% at the CFDVex level 2, the average FPR is approximately 0.62-0.71%. With MIPS architecture, the average accuracy is 97.98% at the CFDVex level 1 and 98.30% at the CFDVex level 2. The average FPR is approximately 1.27-1.28%. It proves that the CFDVex has a high capacity to detect malware running on Intel 80386 and MIPS architecture with affordable FPR. The CFDVex level 2 got better outcome than the CFDVex level 1.

4.6 Evaluation of crossed architecture samples

Table 5 shows experimental results of crossed-architecture malware detection. When we trained by the Intel 80386 dataset and evaluated by the MIPS dataset, it achieved a high accuracy (95.72%) and a affordable FPR (3.2%). But if we trained by the MIPS dataset and evaluated by the Intel 80386 dataset, it got bad results. Its reason is

Table 4: Malware detection on single architecture

		CFDVex Level 1	CFDVex Level 2
Intel 80386	FPR	0.71	0.62
	FNR	2.06	1.6
	ACC	98.60	98.96
MIPS	FPR	1.28	1.27
	FNR	2.5	1.9
	ACC	97.98	98.30

the Intel 80386 dataset contains more types of malware than the MIPS dataset, only some malware instants of Intel 80386 appeared on MIPS. Thus, it's predicted that in the near future there will be a huge amount of malware that can appear in the direction of moving from Intel 80386 to MIPS. Although there is no high accuracy detection as with single architecture, but the experiments also proved that we could detect malware instance on a new architecture by learning malware samples from existing and common architectures like Intel 80386.

Table 5: Evaluation of cross architecture malware detection

		CFDVex Level 1	CFDVex Level 2
Intel 80386 for training Testing by MIPS	FPR	3.1	**2.81**
	FNR	4.77	**2.56**
	ACC	94.20	**95.72**
MIPS for training Testing by Intel 80386	FPR	0.62	1.12
	FNR	92.02	82.3
	ACC	52.7	58.2

4.7 Detection on mixed architecture samples

We generated a mixed dataset from Intel 80386 and MIPS samples at training and evaluating steps. The experimental results are shown in Table 6 with ACC is 97.02%, FPR is 1.05% and FNR is 2.51% with CFDVex level 2. It is still a higher accuracy compare with accuracy reported in [3] is 85.2% even with larger data samples.

Table 6: Detection on mixed architecture samples

	CFDVex Level 1	CFDVex Level 2
FPR	1.27	**1.05**
FNR	4.04	**2.51**
ACC	95.98	**97.02**

5 CONCLUSIONS AND FUTURE WORK

In this paper we proposed the new method CFDVex to detect cross-architecture malware by reusing our previous developed tools and other methods. The CFD has gotten a high detection accuracy with each architecture of IoT malware through opcode. The Vex intermediate representation is used efficiently in many cross-architecture tools like Valgrind and Angr. Thus, our CFDVex is a feature selection method for cross-architecture ELF file that is based on Vex intermediate representation and our CFD method. We generated the IoT dataset which is the largest IoT dataset currently available for multi-architecture and conducted systematic experiments of detection of malware on each architecture, mixed architectures to improve accuracy, and cross-architecture to detect malware on new architecture.

Experimental evaluation of the proposed approach using our IoT dataset achieved good results with rate of the ability to detect Vex-based malware reaching 98.96%, mixed detection reached 97.02% and across from Intel 80386 to MIPS architecture detection reached 95.72%. Two proposed feature extraction methods have good capacity of malware detection and CFDVex level 2 get higher accuracy and lower FPR than level 1 in all experiments.

As our future work, we will (1) verify the CFDVex algorithm with all datasets to evaluate the performance and effectiveness; (2) find out a relation of malware instances between different architectures; and (3) improve the CFDVex at level 2 by choosing suitable representation for each template.

6 ACKNOWLEDGMENT

This research is funded by Vietnam Ministry of Public Security under Grant no. BX.2017.T31.01.

REFERENCES

[1] Kaspersky IoT Lab Report. *New IoT malware grew three fold in H1 2018.* [Online]. Available: https://www.kaspersky.com/about/press-releases/2018_new-iot-malware-grew-three-fold-in-h1-2018. [Accessed: 02-Sep-2019].

[2] Yin Minn Pa Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. *IoTPOT: Analysing the Rise of IoT Compromises.* In Proceedings of the 9th USENIX Conference on Offensive Technologies, 9–19. WOOT'15. Berkeley, CA, USA: USENIX Association, 2015.

[3] Alhanahnah, Mohannad, Qicheng Lin, Qiben Yan, Ninh Zhang, and Zhenxiang Chen. *Efficient Signature Generation for Classifying Cross-Architecture IoT Malware.* 2018 IEEE Conference on Communications and Network Security (CNS), 1–9, 2018.

[4] N. Idika, A.P. Mathur. *A Survey of Malware Detection Techniques.* Technical Report, Purdue University, 2007

[5] Evanson Mwangi karanja, Shedden Masupe, Jeffrey Mandu. *Internet of Things Malware: A Survey.* IJCSES, vol. 8, No.3, 2017.

[6] Xuxian Jiang, Xinyuan Wang, Dongyan Xu. *Stealthy malware detection and monitoring through VMM-based out-of-the-box semantic view reconstruction.* ACM Transactions on Information and System Security (TISSEC), Volume 13 Issue 2, February 2010.

[7] Shahid Alam, R. Nigel Horspool, and Issa Traore. *MAIL: Malware Analysis Intermediate Language: A Step Towards Automating and Optimizing Malware Detection.* In Proceedings of the 6th International Conference on Security of Information and Networks, 233–240. SIN '13. New York, NY, USA: ACM, 2013.

[8] Ralf Huuck. *Iot: The internet of threats and static program analysis defense.* Embedded World 2015, Exibition & Conferences, pp. 493–495.

[9] Rafiqul Islam, Ronghua Tian and Lynn Batten. *Classification of Malware Based on String and Function Feature Selection.* Second Cybercrime and Trustworthy Computing Workshop, 2010.

[10] Huy Trung Nguyen, Quoc Dung Ngo, and Van Hoang Le. *IoT Botnet Detection Approach Based on PSI Graph and DGCNN Classifier.* In 2018 IEEE International Conference on Information Communication and Signal Processing (ICICSP), 118–122, 2018.

[11] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero and Pablo Garcia Bringas. *Opcode Sequences as Representation of Executables for Data-Mining-Based Unknown Malware Detection.* Information Sciences, Data Mining for Information Security, 231 (May 10, 2013): 64–82.

[12] Yuxin Ding, Wei Dai, Shengli Yan and Yumei Zhang. *Control Flow-Based Opcode Behavior Analysis for Malware Detection.* Computers & Security 44 (July 1, 2014): 65–74.

[13] Soomin Kim, Markus Faerevaag, Minkyu Jung, SeungIl Jung, DongYeop Oh, JongHyup Lee, and Sang Kil Cha. *Testing Intermediate Representations for Binary Analysis.* In Proceedings of the 32Nd IEEE/ACM International Conference on

Automated Software Engineering, 353–364. ASE 2017. Piscataway, NJ, USA: IEEE Press, 2017.

[14] Alexander Sepp, Bogdan Mihaila, and Axel Simon. *Precise Static Analysis of Binaries by Extracting Relational Information.* In 18th Working Conference on Reverse Engineering, 357–366. Limerick, Ireland: IEEE, 2011.

[15] N. Nethercote and J. Seward. *Valgrind: a framework for heavyweight dynamic binary instrumentation.* SIGPLAN Not, 42(6):89 -100, June 2007.

[16] Intermediate Representation in Angr. Available *https://docs.angr.io/advanced-topics/ir*

[17] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. Bitblaze: A new approach to computer security via binary analysis. In ICISS '08, pages 1-25, Berlin, Heidelberg, 2008. Springer-Verlag.

[18] H. Yin and D. Song. Privacy-Breaching Behavior Analysis. *In Automatic Malware Analysis*. Springer Briefs in Computer Science, pages 27-42. Springer New York, 2013

[19] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel and Giovanni Vigna. *State of The Art of War: Offensive Techniques in Binary Analysis*, IEEE Symposium on Security and Privacy (SP), 2016.

[20] *Frequently Asked Questions.* [Online]. Available: https://docs.angr.io/introductory-errata/faq. [Accessed: 16-Jun-2019].

[21] Daniel Bilar. *Opcodes as Predictor for Malware.* International Journal of Electronic Security and Digital Forensics 1, no. 2 (2007): 156.

[22] Robert Moskovitch, Clint Feher, Nir Tzachar, Eugene Berger, Marina Gitelman, Shlomi Dolev and Yuval Elovici. *Unknown Malcode Detection Using OPCODE Representation.* In Intelligence and Security Informatics, 204–215. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008.

[23] Igor Santos, Felix Brezo, Javier Nieves, Yoseba K. Penya, Borja Sanz, Carlos Laorden, and Pablo Garcia Bringas. *Idea: Opcode-Sequence-Based Malware Detection.* In Engineering Secure Software and Systems, Second International Symposium, ESSoS 2010, Pisa, Italy, (pp.35-43), 2010.

[24] Tran Nghi Phu, Nguyen Ngoc Toan, Le Hoang, Nguyen Dai Tho, Nguyen Ngoc Binh. *C500-CFG: A Novel Algorithm to Extract Control Flow-based Features for IoT Malware Detection.*19th International Symposium on Communications and Information Technologies (ISCIT), 2019, Hochiminh, Vietnam.

[25] Shunichi Amari, Si Wu. *Improving support vector machine classifiers by modifying kernel functions.* Neural Netw 1999;12:783-789.

[26] Andrei Costin, Jonas Zaddach, Aurélien Francillon and Davide Balzarotti, *A large-scale analysis of the security of embedded firmwares*, in Proceedings of the 23rd USENIX Security Symposium, 2014, pp.95-110.

[27] Pa Yin Minn Pa, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, Takahiro Kasama, and Christian Rossow. *IoTPOT: A Novel Honeypot for Revealing Current IoT Threats.* Journal of Information Processing 24, no. 3 (2016): 522–533.

[28] Detux [Online]. Available *https://github.com/detuxsandbox/detux*

[29] David Brash. *Recent Additions to the ARMv7-A Architecture.* In 2010 IEEE International Conference on Computer Design, 2010.

[30] Vex IR Document. *https://github.com/angr/vex/blob/master/pub/libvex_ir.h*

[31] Hiroshi Ogura, Hiromi Amano and Masato Kondo. *Feature Selection with a Measure of Deviations from Poisson in Text Categorization.* Expert Systems with Applications 36, no. 3, Part 2 (April 1, 2009): 6826–6832.

[32] Y. Yang and J. O. Pedersen, *A comparative study on feature selection in text categorization.* Proceedings of the 14th International Conference on Machine Learning (ICML '97), p. 412-420, 1997.

[33] Virusshare [Online]. Available *https://virusshare.com/*

[34] Virus Total [Online]. Available *https://virustotal.com/*

[35] Tran Nghi Phu, Nguyen Ngoc Binh, Ngo Quoc Dung, and Le Van Hoang. *Towards Malware Detection in Routers with C500-Toolkit.* In 2017 5th International Conference on Information and Communication Technology (ICoIC7), 1–5, 2017.

[36] Christopher Kruegel and Yan Shoshitaishvili. *Using static binary analysis to find vulnerabilities and backdoors in firmware.* in: Black Hat USA, 2015.