EAI Endorsed Transactions
on Security and Safety Research Article EAI.EU

Malware Detection Based on Opcode Dynamic Analysis

Jing Zhang[*] and Yu Wen

Abstract

Malware detection is an important problem in the field of information security. Opcodes are the most direct information reflecting the execution behavior of malware, The malware based on the dynamic analysis of opcodes also faces some challenges: the acquisition of the operating code information in the execution process of the malware; the high false alarm rate in the detection process and the large system overhead caused by the malware detection in the application layer. In order to deal with the above problems, this paper proposes a new scheme for dynamic opcode acquisition, the opcode information obtained from the software runtime is used for offline analysis. The detection accuracy of off-line malware can reach 99.85%, which is superior to the traditional technology. Moreover, this paper proposes an online detection scheme: CPU built-in malware monitoring model (CBMM), which can solve the problem that it is difficult to accurately identify the execution trajectory of malware in the current malware detection process, at the same time, this model can monitor malware in real time. Finally, we implement our model by VerilogHDL, functional simulation was carried out in modelsim simulation software and its implementation cost was analyzed.

Keywords: Malware Detection, Opcodes, Dynamic Detection.

Received on 16 July 2020, accepted on 15 October 2020, published on 20 October 2020

doi: 10.4108/eai.22-6-2021.170239

[*] Corresponding author. Email: zjj7ucas@163.com

1 Introduction

With the development of the Internet technology in recent years, it is also used by criminals to carry out malicious activities due to the openness of the Internet. So information security is becoming more and more important. An important problem in the field of information security is malware detection. Most antivirus software use a combination of signatures and heuristics method to detect malware. The problem with this approach is that it is susceptible to malware obfuscation mechanisms, making it difficult to accurately identify the trajectory of malware execution.

Because the malware will eventually execute the code with malicious behavior in the execution process, the malware can be detected by analyzing the behavior information of the malware. Since opcodes are the most direct information reflecting the execution behavior of malware, this paper analyzes the information of opcodes in the execution process of malware for malware detection. However, the malware based on the dynamic analysis of the operating code also faces some challenges: the acquisition of the operating code information in the execution process of the malware; the high false alarm rate in the detection process and the large system overhead caused by the malware detection in the application layer. In order to deal with the above problems, this paper firstly analyzes the progress and existing problems of existing malware detection technology based on dynamic opcode analysis, then proposes a new scheme for dynamic opcode acquisition, the opcode information obtained from the software runtime is used for offline analysis. In the off-line analysis, this paper uses a variety of feature selection algorithms to extract features of the operating code information when the software is running, we use the extracted feature subset combined with a variety of machine learning algorithms to conduct cross-comparison experiments. Finally, the detection accuracy of off-line malware can reach 99.85% and the false alarm rate can reach 0.5%. Based on the above research results, this paper proposes an online detection scheme: CPU built-in malware monitoring model (CBMM), which can solve the problem that it is difficult to accurately identify the execution trajectory of malware in the current malware detection process, at the same time, this model can monitor malware in real time. Finally, we implement our model by VerilogHDL, functional simulation was carried out in modelsim simulation software and its implementation cost was analyzed.

2 Related Work

Malware detection can be divided into dynamic malware detection and static malware detection according to the way of obtaining malware information. Dynamic malware detection is a way to expand malware detection by running malware and obtaining behavior information (including opcode, register, API call, etc.) during its operation.

In order to deal with the problem that the signature-based malware detection method is susceptible to the confusion mechanism, the current malware detection research begins to focus on how to detect malware by analyzing the behavior information of software runtime. The opcode information at the time of software execution reflects the operation performed by the processor. By analyzing the opcode information at the time of software running, the existence of malware can be detected, it is not affected by the confusion mechanism [2].

Ozsoy et al. [3] used Intel Pin tools to insert instructions into malware, they collected instructions during program operation and analyzed the information of instruction type, frequency of access operation, and they used neural network and logical regression to construct classifier. The sensitivity of classifier to malware reached 100% and the false alarm rate was 9%. [4] [5] also proposed to use opcode for malware detection Testing. In [6], the author proposed to use the instruction sequence with variable length as the feature. After obtaining the feature, the malware detection was carried out using bagging integrated learning algorithm, finally a better detection effect was obtained. There are also scholars who directly use hexadecimal opcode data for malware detection. The advantage of this approach is that the complexity of opcode types can be reduced because one hexadecimal data may correspond to multiple opcode [8].

It can be seen that opcode can be used as a good feature in malware detection, due to the use of opcode information in dynamic analysis, we can solve the problem of malware detection caused by code confusion. But most of the previously proposed methods have a high false alarm rate, high performance overhead and resource requirements. On-line detection of malware and accuracy and false alarm rate need to be further improved.

In terms of dynamic opcode acquisition, there are two main ways: using application layer tools to obtain (e.g. Intel Pin tool [52], valgrind [53]); using system level tools to obtain (mainly sandbox) [55]. During the use of application level tools, there is a problem that kernel-space information cannot be obtained, when using sandbox tools, there is the problem of low monitoring efficiency. We propose a new opcode dynamic acquisition scheme in Section 3.

3 Proposed Methodology

The malware detection model determines malware as malware or benign software according to the input characteristic data. The malware detection model M can be understood as a function whose domain is a set of all programs P, and the range is {Y, N}:

$$\forall p \in P,\ M(p) = \begin{cases} Y, & if\ p\ is\ infected \\ N, & otherwise \end{cases} \tag{1}$$

The detection model M scans the program p, determine whether the program contains malicious behavior. The desired result is: if the M returns Y, the program is detected to contain malicious behavior; otherwise, malicious behavior is not detected. In this paper, the detector is the malware detection technology of opcode dynamic analysis, and the domain of definition is the set of programs.

The traditional dynamic analysis method is difficult to obtain the complete sequence of malware opcode. In this paper, a method of obtaining the operation-time opcode information is proposed, and this method uses the complete opcode information. We use different feature extraction algorithms and classification algorithms to extract and classify them, then we analyze the effects of different feature selection algorithms, N-gram length and classification algorithms on malware detection. Experiments show that the detection accuracy of malware is 99.85%, which is superior to traditional technology.

We divides the current malware detection based on opcode dynamic analysis into three parts and two stages (see Fig. 1). Three parts refer to dynamic opcode acquisition part, feature extraction part and decision classification part. Two stages refer to offline data analysis stage and online detection stage. The relationship between the three parts and the two stages is as follows: in the off-line data analysis stage, the required dynamic opcode information is obtained by the dynamic opcode acquisition part, a feature subset is obtained by the feature extraction part, the decision and classification of the third part is expanded by the acquired feature subset. At the end of the off-line data analysis stage, we can get a malware detection classifier which is input into the dynamic opcode information feature. The online detection stage mainly uses the results obtained after offline analysis to detect malicious code online. The next step is to introduce the offline opcode dynamic acquisition scheme in Section 3.1. Section 3.2 introduces the feature algorithm and decision algorithm used in the off-line analysis stage, and section 3.3 presents the online malware detection scheme in this paper.

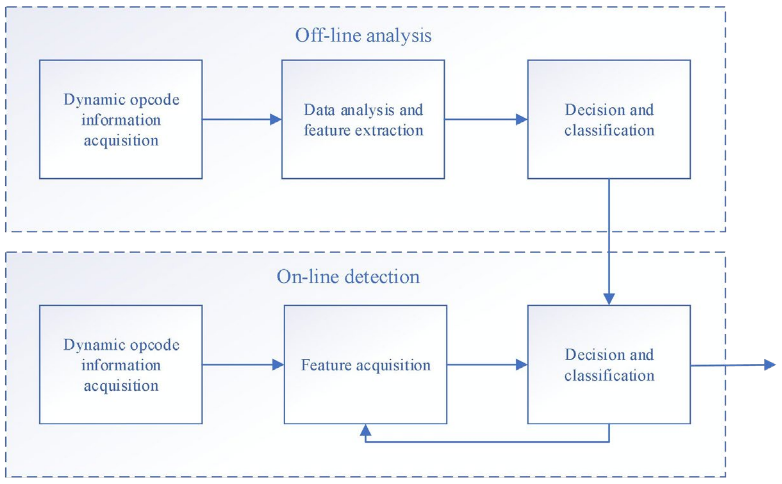
Off-line analysis
Dynamic opcode information acquisition
Data analysis and feature extraction
Decision and classification
On-line detection
Dynamic opcode information acquisition
Feature acquisition
Decision and classification

Fig. 1. Dynamic analysis of malware detection based on opcodes

3.1 Opcode Acquisition Scheme

The current dynamic opcode acquisition method still has some problems, such as incomplete information and low monitoring efficiency, so we propose a dynamic opcode acquisition scheme. This scheme is mainly based on the QEMU [58] binary translation mechanism, as shown in Fig. 2, in the process of translating guest opcode into host operation, the data saving module is inserted to save the required information. A sandbox system based on QEMU is designed through the above ideas. This sandbox system can provide instruction level monitoring granularity. A QEMU sandbox system will be implemented in section 4.

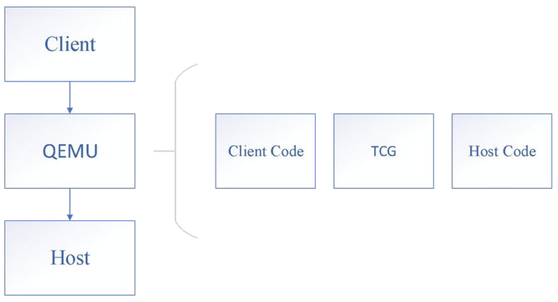

Fig. 2. QEMU binary translation technology

Fig. 3 shows the dynamic opcode acquisition scheme of this paper based on QEMU sandbox system, which includes three parts: data source, data filtering, and QEMU sandbox system. The sections are described below.

- Data source: software to be analyzed. Malicious software is downloaded from Virus share website in this paper. It is a website dedicated to providing malware analysis samples for researchers. The benign software obtains from the Linux system software, mainly from the system software under the /usr/bin and /bin directory.
- Data filtering: To ensure the reliability of malicious samples, filter them before analyzing them. This paper uses the virustotal [59] for sample filtering, by writing an interface program, we use virustotal to analyze malware. Virusshare provide nearly 10 mainstream antivirus engines including McAfee, Symantec, 360 and so on, which can ensure the reliability of malware.
- QEMU based sandbox system: this part is the core part of this scheme, the main work is to run the software to be analyzed and obtain its dynamic opcode data. This system will be described in detail in section 4.

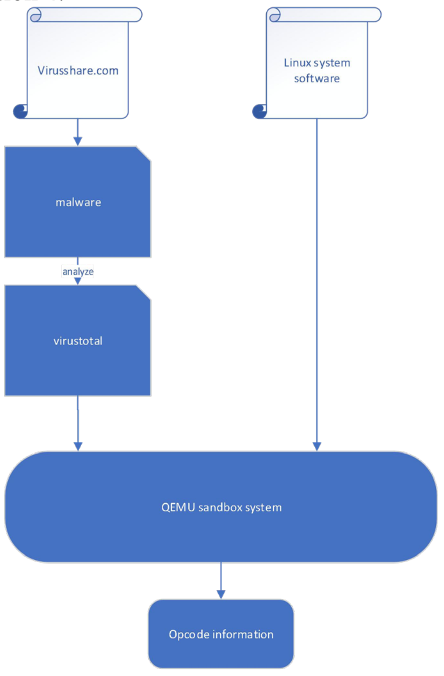

Fig. 3. Dynamic opcode acquisition scheme

3.2 Feature Selection And Decision Classification

Feature Selection:

The feature selection algorithms adopted in this paper are as follows.

- Correlation-based Feature Selection (CFS) Algorithm.
- Chi party Test Feature Selection Algorithm.
- Information Gain and Information Gain Rate.
- Symmetric Uncertainty Feature Selection Algorithm.
- N-gram Algorithms.

In section 4, we will use these feature selection algorithms for feature extraction, and analyze the effect of feature selection algorithms on the final detection effect.

Table 1. Examples of n-gram

Operand sequence	Mov cmp push jmp add
1-gram	Mov,cmp,push,jmp,add
2-gram	Mov cmp ,cmp push, push jmp, jmp add
3-gram	Mov cmp push, cmp push jmp, push jmp add

Decision Classification:

The decision classification algorithms used in offline analysis are decision tree, SVM, Bayesian network, ensemble learning algorithm and the neural network.

3.3 Online Malware Detection Scheme

We propose an online malware detection scheme. As shown in the following figure, this scheme mainly includes three parts: data separation module, feature extraction module and decision module.

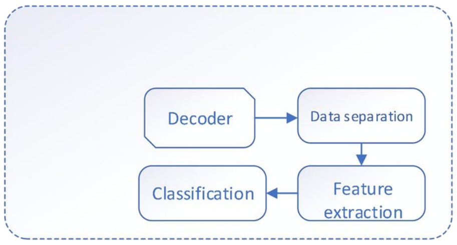

Fig. 4. Schematic diagram of malware detection scheme

Data separation module works to split the opcode according to the CR3 value. The data source of the data separation module is the decoder of the processor. The feature extraction module and the decision module will be designed by using the detection module obtained after offline analysis. So this part will be explained in detail after section 4 offline data analysis.

4 System Implementation And Experiment

Analytical experiments will be carried out in this section. The acquisition mode will use the scheme proposed in section 3 to obtain dynamic opcode. After the dynamic opcodes information is obtained, the opcodes will be analyzed by empirical and computational methods. The computationally based analysis method will be applied to the feature selection algorithm and classification algorithm introduced in section 3. In the process of analyzing the experimental results, the analysis method of control variables is mainly used to analyze the factors that affect the detection accuracy of malware. We further design online malware detection scheme: CPU built-in malware monitoring module (CBMM), the model can detect malware online during CPU operation. We realize the CBMM through Verilog HDL, use modelsim simulation software for functional simulation and analyze the actual modern price.

4.1 Data Collection Environment

The environment in which the data collection system operates is shown in Table 2.

Table 2. Data collection environment

	operating system	architecture	processor	memory
host	Windows10	64bit	Intel i7-8750H	16G DDR4
client	Ubuntu16.4	32bit	Qemu core	4G

As shown in Fig. 5, the QEMU based sandbox system consists of three modules: microarchitecture information preservation module, software scheduling module and system operation anomaly monitoring module.

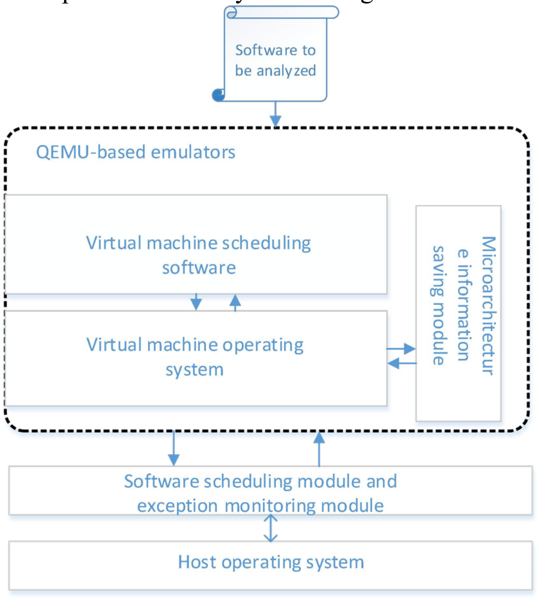
Software to be analyzed
QEMU-based emulators
Virtual machine scheduling software
Virtual machine operating system
Microarchitecture information saving module
Software scheduling module and exception monitoring module
Host operating system

Fig. 5. QEMU based sandbox system

The three modules described above are described below:

- Opcode information saving module. According to the principle of QEMU binary translation, we modify the source code in the process of QEMU intermediate translation, add the data saving program and design the trigger point where the software needs to save the data at run time, so that the information such as opcode can be saved under the specified scenario. Through analyzing the x86 instruction set, the instructions 0 xf1 and 0 xd6 which do not exist in the x86 instruction set are selected as the trigger points for the preservation of microarchitecture information.

- Use two long integer variables ϕ and φ to represent the number of times 0 xf1 and 0 xd6 two instructions are executed.

$$\begin{cases} if\ \emptyset == \varphi\ and\ \emptyset\ \%\ 2 == 0, start\ flag = True \\ if\ \emptyset == \varphi\ and\ \emptyset\ \%\ 2 == 1, start\ flag = True \\ else\ start\ flag = False\ and\ stop\ flag = False \end{cases} \quad (2)$$

  Start flag： start recording opcode information. Stop flag： stop recording opc in formation.

- Software scheduling module. The microarchitecture information saving trigger point and information saving program designed according to step 1 enables the user to trigger the microarchitecture saving program of the simulator under the condition that the microarchitecture information needs to be saved. At the same time, the scheduling software also needs to keep the initial state of each simulator running platform consistent, so it needs to be restored to the initial state before each system running, and the function flow of the scheduling module is shown in Fig. 6.

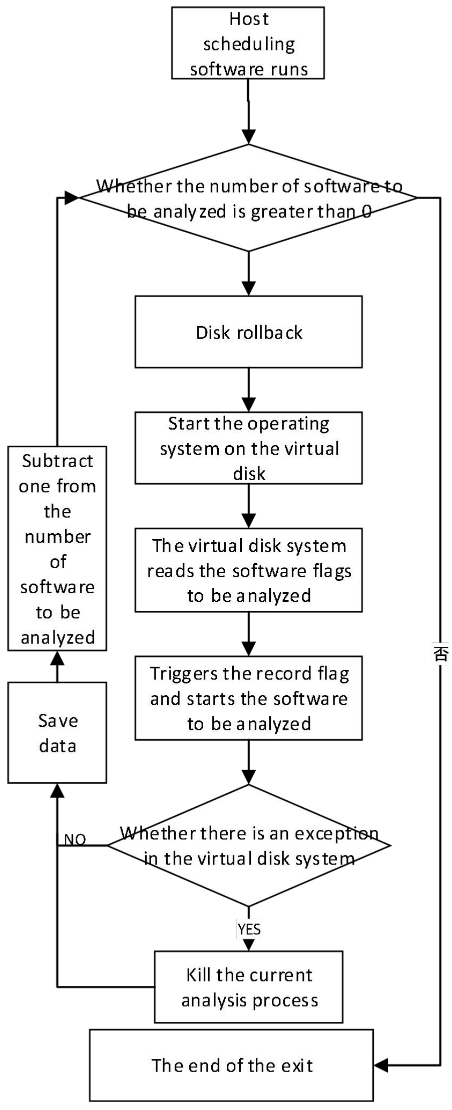
Host scheduling software runs
Whether the number of software to be analyzed is greater than 0
Disk rollback
Start the operating system on the virtual disk
The virtual disk system reads the software flags to be analyzed
Triggers the record flag and starts the software to be analyzed
Whether there is an exception in the virtual disk system
YES
NO
Kill the current analysis process
Save data
Subtract one from the number of software to be analyzed
The end of the exit
否

Fig. 6. Software scheduling module

- Abnormal monitoring module. Scheduling software automatically triggers and stops the system's ability to hold processor microarchitecture information under certain conditions. But in some situations, such as running, some malware can cause unrecoverable damage to the system. For improving the robustness of the whole system, the abnormal monitoring module is designed, and the analysis is carried out according to the existence time PID the second dispatching subprocess of the module. If the abnormal time is set, the abnormal monitoring software will deal with the current system scheduling software call subprocess.

Through the above opcode collection scheme, we can get the opcode information that needs to be analyzed in the offline analysis stage in this paper.

4.2 Datasets

Datasets: Malware gets system software from virusshare [69], benign software gets from 32-bit Ubuntu 16.04. Table 3 shows the number of malware and benign software in the text.

Table 3. Datasets

	Number
benign Software	1166
malicious software	2189

Evaluation criterion: accuracy, recall, ROC curve.

Malware is defined as a positive class sample and benign software as a negative class sample.

Accuracy:

$$Acc = \frac{TP+TN}{TP+TN+FP+FN} \times 100\% \quad (3)$$

Recall:

$$recall = \frac{TP}{TP+FN} \times 100\% \quad (4)$$

ROC curve: ROC curve is a curve with false positive rate and true rate as axes. The area under the ROC curve is called AUC. Briefly, the greater the AUC, the better the classification effect.

4.3 Data Processing

Experimental result:

A total of 168 different opcodes appeared in this paper, and the top18 number of opcodes appeared in malware accounted for 93.6% of all opcodes (Fig. 7). We also counted opcodes that only appear in malware (Fig. 8).The main functions of the most frequent opcodes are listed in Appendices Table 7, Data of N-gram N=2, 3 were also extracted in this experiment (Appendices Fig. 14~Fig. 19). There are 3064 different sequences in the 2- gram and 20,387 in the 3- gram.

We use feature selection algorithm to process opcode data, first construct arff format file to meet weka data format, which contains data feature name and feature data. The frequency characteristic information of different opcodes is mainly considered in this paper. Feature selection can reduce the dimension of feature data and reduce the computational complexity of classifier. Then we will choose different classification algorithms to experiment, and compare the experimental results under the combination of different feature algorithms and different classification algorithms. Feature selection algorithms as well as classifier algorithms will act in the case of Ngram N=1, 2, 3 (Fig. 7).

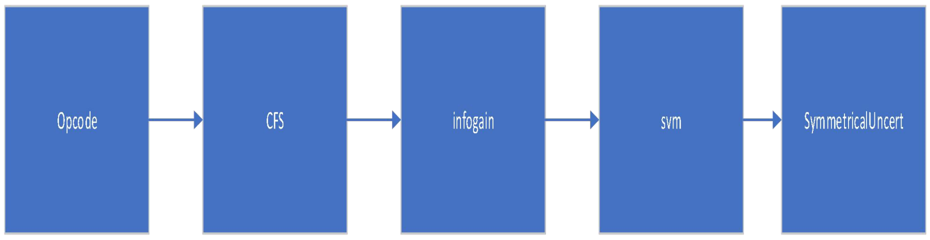

Fig. 7. 1-gram opcode feature selection

Experiment of Classification Algorithm.

When the feature selection algorithm is CFS, the random forest classifier has the best performance for a single opcode feature, and its accuracy is 99.85%, followed by Bayes Net, accuracy of 99.79% (Table 4).

Table 4. Results of classifier test (accuracy)

Feature Selection Algorithm	classifier	Acc 1-gram	Acc 2-gram	Acc 3-gram
CFS	SVM	97.91%	99.01%	99.58%
	Bayes net	99.79%	99.34%	99.79%
	ANN	97.43%	98.65%	99.88%
	IBk(k=1)	98.78%	98.65%	99.73%
	Random Forest	99.85%	99.61%	99.85%
	J48(dt)	99.73%	99.49%	99.73%
	bagging	99.76%	99.43%	99.76%
infogain	SVM	74.87%	89.77%	99.43%
	Bayes net	89.50%	89.77%	97.04%
	ANN	97.31%	97.52%	99.61%
	IBk(k=1)	99.1%	98.80%	99.73%
	Random Forest	99.61%	99.61%	99.76%

	J48	99.43%	99.43%	99.73%
	bagging	99.2%	99.43%	97.23%
chisquare	SVM	77.28%	93.97%	92.22%
	Bayes net	89.23%	89.71%	90.61%
	ANN	97.43%	98.06%	99.76%
	IBk(k=1)	98.83%	98.74%	99.80%
	Random Forest	99.70%	99.58%	99.67%
	J48	99.31%	99.58%	99.73%
	Bagging	99.31%	99.58%	99.73%
Gainratio	SVM	74.69%	99.34%	97.38%
	Bayes net	89.68%	92.93%	99.22%
	ANN	97.25%	99.01%	99.73%
	IBk(k=1)	98.71%	98.65%	99.64%
	Random Forest	99.22%	99.55%	99.64%
	J48	98.92%	99.55%	97.70%
	Bagging	98.98%	99.49%	99.73%
symmetrical	SVM	74.78%	98.00%	98.87%
	Bayes net	89.12%	92.85%	99.31%
	ANN	97.44%	98.90%	99.85%
	IBk(k=1)	98.89%	99.10%	99.76%
	Random Forest	99.73%	99.64%	99.79%
	J48	99.43%	99.55%	99.70%
	Bagging	99.28%	99.52%	99.76%

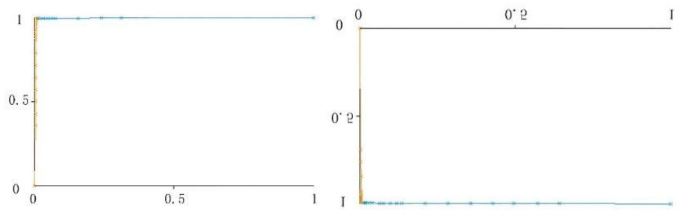

Fig. 8. Bayes Net and random forest ROC curve

Table 5. Bayes Net and random Forest recall rate

	Bayes Net	Random forest
False alarm rate	0.9%	0.5%
Recall rate	0.993	0.997

According to the figure above, the CFS feature selection algorithm is the best one. Remove the information gain rate feature selection algorithm at N=1. In other cases, the accuracy rate is above 99%. Except for the CFS feature selection algorithm, the accuracy of classification algorithm increases with the N of N-gram under other feature selection algorithms.

4.4 Design and Implementation of On-line Malware Detection Scheme

This section will design the online malware detection module in this article: CPU built-in malware monitoring module (CBMM).

The data separation module is responsible for diverting the received opcode data stream according to the CR3 value, obtaining the opcode data after the processing data preprocessing module is shunted by multiple channel; feature extraction modules associated with the CR3 value to generate the feature vector; the decision module detects the indicated by the current CR3 according to the feature vector generated by the feature extraction module. The advantages of the CPU built-in malware monitoring model are hardware implementation, the response speed is fast, it can be monitored in real time without the influence of malware confusion mechanism. As shown in Figure 9. The diagram shows where the security module is built in Schematic architecture. The input of the CBMM in this architecture is the opcode section in the instruction, as well as the value of the cr3register (Fig. 10). The model determine whether the current running program or software has malicious behavior through the detection of the input data stream. During on-line monitoring, the module generates feature vectors and judges malware when the CR3 changes and the input opcode reaches the preset threshold. Before the decision condition is reached, the whole module will only use the register to record the number of times that the specific opcode is executed, the decision module will only be used in the decision stage, because the data separation module the multiple channel obtained by the block (set to 64 channel in this paper) is running in parallel, the expected value of the data received by each channel is 1/64 of the processor processing data, which can greatly alleviate the problem of high processor processing speed.

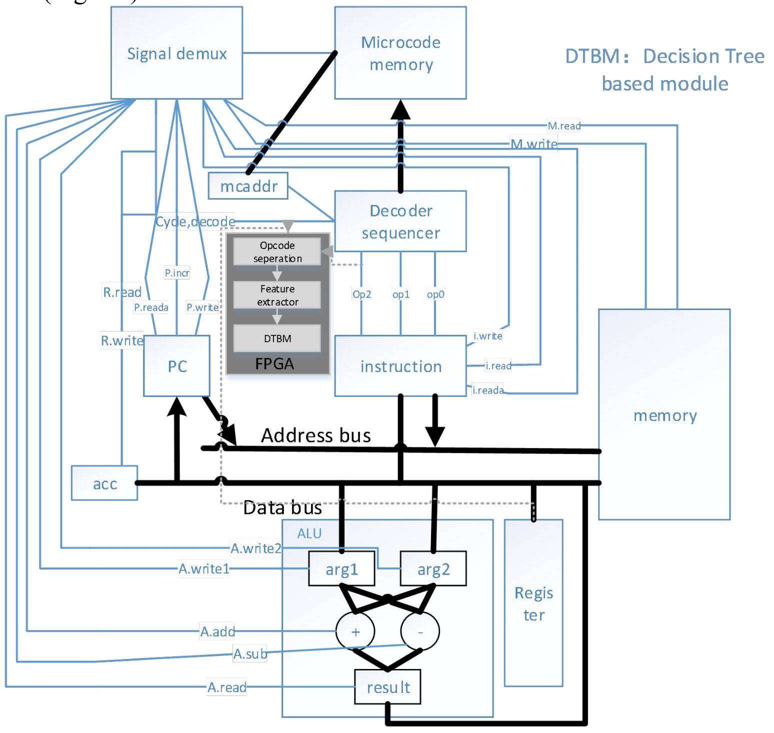

Fig. 9. Schematic diagram of secure processor structure embedded in CBMM module

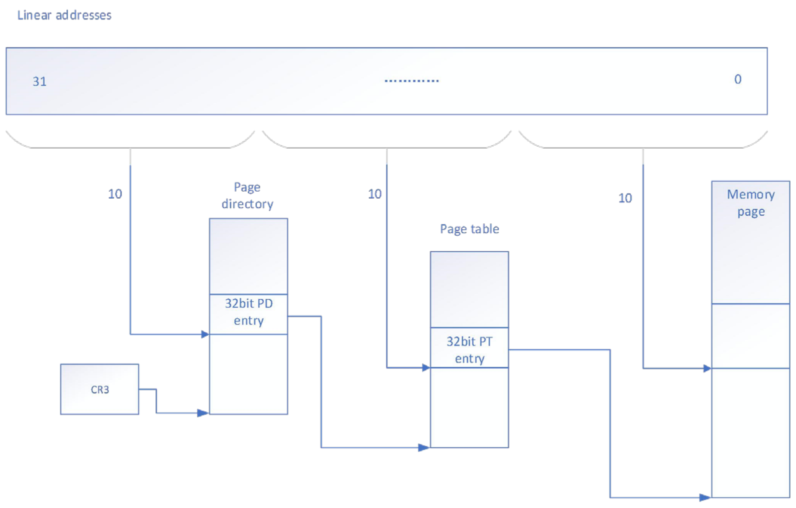

Fig. 10. Typical use of CR3 in address translation

Hardware Implementation Cost Analysis.

Implementation and simulation environment: Quartus Prime Standard Edition 16Modelsim. The functional modules are simulated at RTL level using modelsim and power consumption prediction using PowerPlay power Analyzertool.

Feature extraction module will produce feature vectors when the current CR3 value changes or the recorded opcode reaches a certain threshold. Decision module uses the decision tree algorithm. This algorithm is a C4.5 algorithm. The information gain rate is used to construct the decision tree, pruning is carried out in the process of constructing the tree.

CBMM module is a complete functional module after integrating data separation module, feature extraction module and decision module. This paper uses modelsim to verify it. The required input data will be constructed in the simulation experiment. By writing the testbench to satisfy all the excitation of the module, we analyze its accuracy according to its simulation output. In the experiment, all opcode data are from offline collected data, CR3 data are added for themselves. In the experiment, the CR3 data range is 0-99. In the online simulation experiment, 50 benign samples and 50 evil samples were used Italian software to test.

4.5 Experimental Results Of Algorithm Simulation

As shown below, the input is CR3 and the opcode and the output is the decision result of the decision module, 01 means the decision is malware and 10 means the decision is benign software. During the experiment of CBMM module, the module can detect the malicious software accurately, and accord with the expected effect of offline.

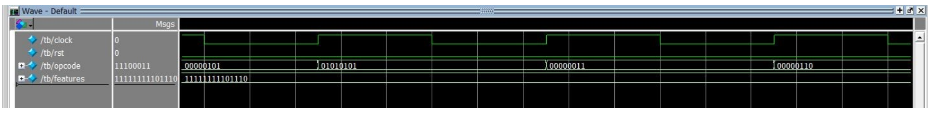

Fig. 11. Simulation results of feature extraction module

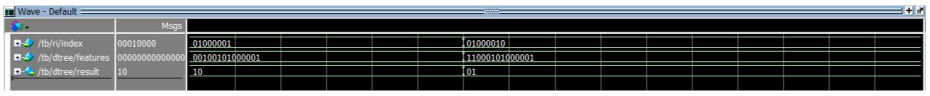

Fig. 12. Decision Tree Algorithm Simulation

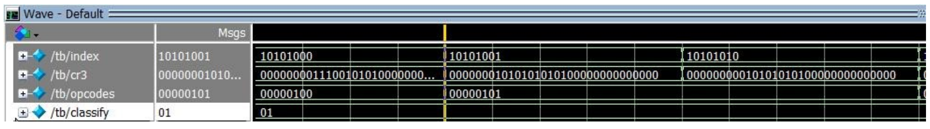

Fig. 13. CBMM simulation waveform

Thermal power consumption analysis is carried out after functional simulation. The following table shows the estimated thermal power consumption and resource usage of each module. LE logic unit in the FPGA development board. According to Table 9, the whole on-line detection scheme can be completed with few logical units.

Table 9. Analysis of resources and estimated power consumption

	resource	Feature extraction module	Decision module	Complete module
LU	LE	163	39	172
Power	heat dissipation	138.14mW	129.86mW	141.86mW

5 Conclusion

A CPU built-in malware monitoring model is proposed in this paper to complete the design and simulation of the module, but this module does not take into account the performance factors in hardware implementation. In the future work, the implementation of this module will be optimized and built into the real processor to achieve the effect of its practical application.

References

[1] Mcafee, “McAfee Labs Threat Report,” no. December, p. 50, 2016.

[2] A. H. Sung, J. Xu, P. Chavez, and S. Mukkamala, “Static Analyzer of Vicious Executables (SAVE),” Proc. - Annu. Comput. Secur. Appl. Conf. ACSAC, no. January, pp. 326–334, 2004.

[3] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, “Data mining methods for detection of new malicious executables,” no. February 2001, pp. 38–49, 2002.

[4] J. Kolter, “Learning to detect and classify malicious executables in the wild,” J. Mach. Learn. Res., vol. 7, pp. 2721–2744, 2006.

[5] A. Pfeffer et al., “Malware analysis and attribution using genetic information,” Proc. 2012 7th Int. Conf. Malicious Unwanted Software, Malware 2012, pp. 39–45, 2012.

[6] G. Bonfante, “Control flow graphs as malware signatures,” Int. Work. Theory Comput. Viruses, no. May 2007, pp. 1–6, 2007.

[7] J. Kinder, S. Katzenbeisser, C. Schallhart, and H. Veith, “Detecting Malicious Code by Model Checking,” pp. 174–187, 2010.

[8] M. D. Preda, M. Christodorescu, S. Jha, and S. Debray, “A semantics-based approach to malware detection,” ACM Trans. Program. Lang. Syst., vol. 30, no. 5, pp. 1–54, 2008.

[9] S. Wehner, “Analyzing worms and network traffic using compression,” J. Comput. Secure. vol. 15, no. 3, pp. 303–320, 2007.

[10] C. Liangboonprakong and O. Sornil, “Classification of malware families based on Ngrams sequential pattern features,” Proc. 2013 IEEE 8th Conf. Ind. Electron. Appl. ICIEA 2013, pp. 777–782, 2013.

[11] P. Venu, D. Vaman, and R. Prasad, “N-Gram Analysis in SVM Training Phase Reduction Using Dataset Feature Filtering for Malware Detection,” Int. J. Sci. Res., vol. 3, no. 9, pp. 550–554, 2014.

[12] K.-H.-T. Dam and T. Touili, “Malware Detection based on Graph Classification,” no. Icissp, pp. 455–463, 2017.

[13] C. K. Patanaik, F. A. Barbhuiya, and S. Nandi, “Obfuscated malware detection using API call dependency,” no. June 2014, pp. 185–193, 2013.

[14] A. A. E. Elhadi, M. A. Maarof, and B. I. A. Barry, “Improving the detection of malware behaviour using simplified data dependent API call graph,” Int. J. Secur. It’s Appl., vol. 7, no. 5, pp. 29–42, 2013.

[15] M. Ozsoy, K. N. Khasawneh, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, “Hardware-Based Malware Detection Using Low-Level Architectural Features,” IEEE Trans. Comput., vol. 65, no. 11, pp. 3332–3344, 2016.

[16] S. Dolev, Y. Elovici, A. Shabtai, R. Moskovitch, and C. Feher, “Detecting unknown malicious code by applying classification techniques on OpCode patterns,” Secur. Inform. vol. 1, no. 1, 2012.

[17] I. Santos, B. Sanz, C. Laorden, F. Brezo, and P. G. Bringas, “Opcode-sequence-based semi-supervSantos, I., Sanz, B., Laorden, C., Brezo, F., & Bringas, P. G. (2011). Opcodesequence-based semi-supervised unknown malware detection. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial ,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2011, vol. 6694 LNCS, pp. 50–57.

[18] S. R. Bragen, “Malware detection through opcode sequence analysis using machine learning,” 2015.

[19] P. Agarwal and K. Bansal, “Malware Classification Challenge.”

[20] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, “Ensemble learning for Low-Level Hardware-Supported malware detection,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 9404, pp. 3–25, 2015.

[21] I. T. Supplement, N. Crime, and V. Survey, “Types of Malware and Malware Distribution Strategies,” no. November 2013, pp. 33–47, 2015.

[22] V. Bontchev, “Current status of the caro malware naming scheme,” Proc. 15th Int. Virus Bull. …, 2005.

[23] F. Cohen, “Computer Viruses Theory and Experiments Fred,” vol. 21A, pp. 22–35, 1978.

[24] R. Fox and W. Hao, “An Introduction to Networks,” Internet Infrastruct., no. November, pp. 1–41, 2018.

[25] W. Stallings et al., Computer SeCurity PrinciPles and Practice Third Edition. 2015.

[26] C. L. and A. Lakhotia, Malware and Machine Learning, vol. 563. 2015.

[27] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, “A taxonomy of computer worms,” no. January 2003, p. 11, 2005.

[28] Spencer Smith, “Rootkits what are Rootkits ?” Symantec Secur., no. Rootkits, pp. 1–9, 2012.

[29] S. Rizvi, G. Labrador, M. Guyan, and J. Savan, “Advocating for Hybrid Intrusion Detection Prevention System and Framework Improvement,” Procedia Comput. Sci., vol. 95, pp. 369–374, 2016.

[30] L. McLaughlin, “Bot software spreads, causes new worries,” IEEE Distrib. Syst. Online, vol. 5, no. 6, p. 1, 2007.

[31] J. Goebel and T. Holz, “Rishi: identify bot contaminated hosts by IRC nickname evaluation,” HotBots, 7, no. June, 2007.

[32] Wiley, Pratical Reverse enginering. 2014.

[33] I. You and K. Yim, “Malware obfuscation techniques: A brief survey,” Proc. - 2010 Int. Conf. Broadband, Wirel. Comput. Commun. Appl. BWCCA 2010, no. October, pp. 297– 300, 2010.

[34] K. M. PhiliP O’Kane, SaKir Sezer, “Obfuscation: The Hidden Malware,” pp. 41–47, 2009.

[35] T. Schmidt, “Evaluating Techniques for Full System Memory Tracing,” 2017.

[36] M. S. Artem Dinaburg*†, Paul Royal†* and Wenke Lee†*, “Ether: Malware Analysis via Hardware Virtualization Extensions,” J. Neuroradiol., vol. 30, no. 5, pp. 283–285, 2003.

[37] F. Bellard, “QEMU, a Fast and Portable Dynamic Translator,” Eur. J. Pharmacol., vol. 394, no. 1, pp. 85–90, 2000.

[38] Https://www.virustotal.com/#/home/search, “virustotal.”.

[39] M. Hall, “Correlation-based feature selection for machine learning ,” Diss. Univ. Waikato, vol. 21i195-i20, no. April, pp. 1–5, 1999.

[40] M. A. Hall, “Feature Selection for Discrete and Numeric Class Machine Learning,” pp. 1– 135, 2013.

[41] E. H. Jr., “Information Gain versus Gain Ratio: A Study of Split Method Biases,” pp. 1– 20, 2001.

[42] L. Yu and H. Liu, “Feature selection for high-dimensional data: a fast correlation-based filter solution. Proceedings of the twentieth international conference on machine learning,” 2003.

[43] W. T. V. B. P. Flannery, S. A. Teukolsky, Numerical recipes: the art of scientific computing. 1385.

[44] B. HSSINA, A. MERBOUHA, H. EZZIKOURI, and M. ERRITALI, “A comparative study of decision tree ID3 and C4.5,” Int. J. Adv. Comput. Sci. Appl., vol. 4, no. 2, pp. 13– 19, 2014.

[45] M. Ettaouil, E. Abdelatifi, F. Belhabib, and K. El Moutaouakil, “Learning algorithm of kohonen network with selection phase,” WSEAS Trans. Comput., vol. 11, no. 11, pp. 387– 396, 2012.

[46] J. V. Tu, “Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes,” J. Clin. Epidemiol. vol. 49, no. 11, pp. 1225– 1231, 1996.

Appendices

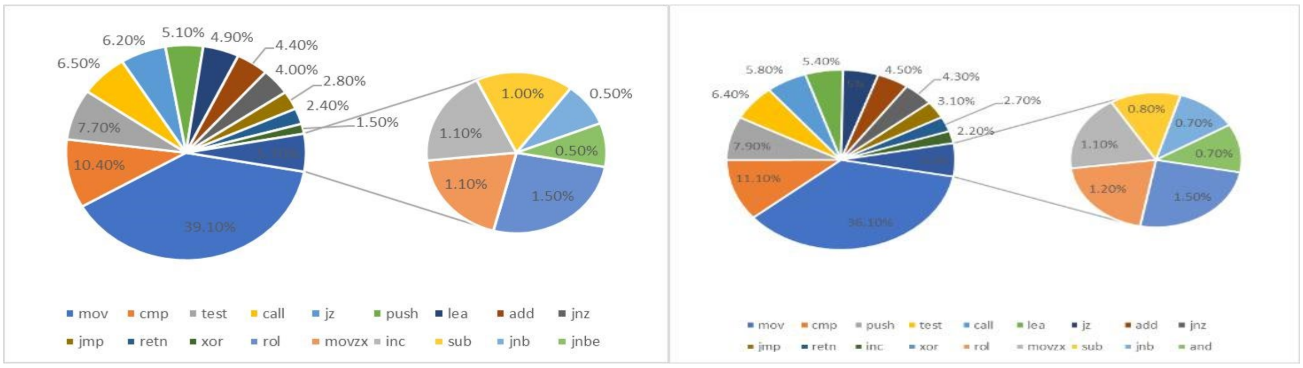

Fig. 14. Benign and malware top18 opcode (left: benign software, same below)

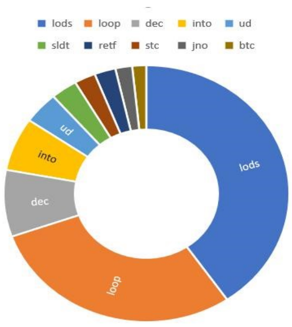

Fig. 15. Opcodes that only appear in malware

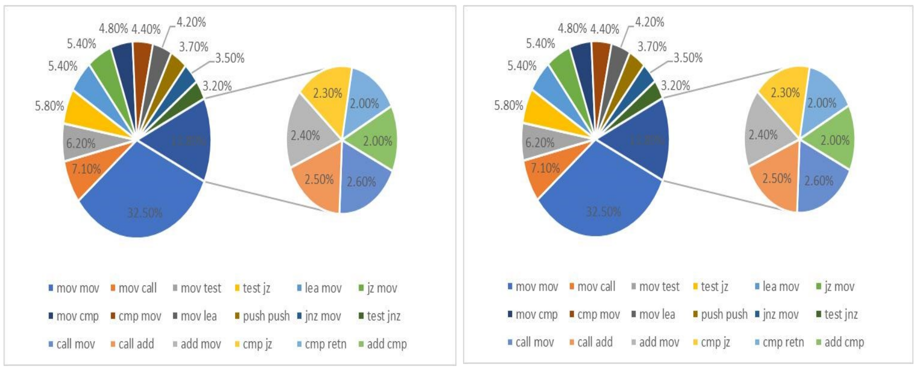

Fig. 16. Benign Software and malware 2-gram top18

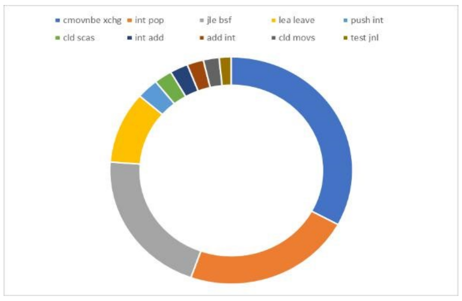

Fig. 17. Malware-specific sequence

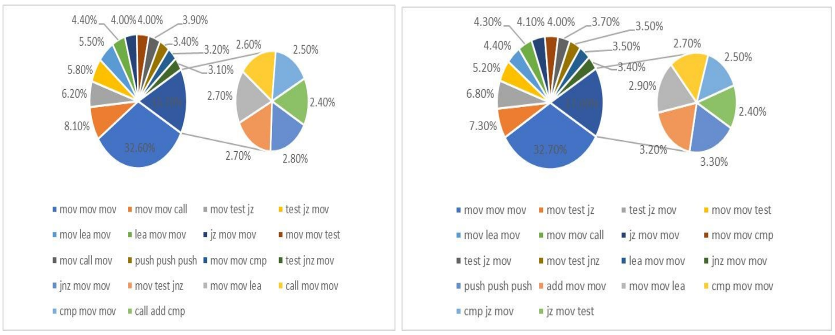

Fig. 18. 3-gram benign software and malware sequence top18

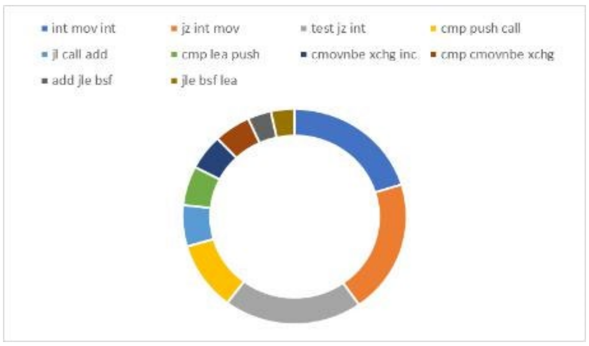

Fig. 19. 3-gram sequence top10 only in malware

Table 6. Operation code description

Operation code	description
MOV	Move
CMP	Compare Two operands
TEST	Logical compare
CALL	Call Procedure
JZ	Jump near if zero/equal (ZF=1)
PUSH	Push Word, Doubleword or Quadword Onto the Stack
LEA	Load Effective Address
ADD	add
JNZ	Jump near if not zero/not equal (ZF=0)
JMP	Jump
RETN	Return from procedure

Table 7. Contribution of single opcodes to classification top10

opcodes	description

JECXZ	Jump short if rCX register is 0
JP	Jump short if parity/parity even (PF=1)
PUNPCKLBW	Unpack Low Data
REX.WB	REX.W and REX.B combination
PMOVMSKB	Move Byte Mask
BSR	Bit Scan Reverse
REX.WRX	REX.W, REX.R and REX.X combination
REX.WX	REX.W and REX.X combination
SUB	Subtract
TEST	Logical Compare

Table 8. Under different feature selection algorithms, N-gram N=1, 2, 3 feature top10

Method top10	CFS	infogain	chisquare	GainRatio	symmetrical
1-gram	MOV	JECXZ	JECXZ	JP	JECXZ
2-gram	Test jz	Movzx jecxz	Movzx jecxz	Sub rol	SUB ROL
3-gram	mov mov mov	Retn lea sub	Ret lea sub	retn lea sub	Retn lea sub
1-gram	CMP	REX.RXB	REX.RXB	CPUID	JP
2-gram	Jz mov	Rol jnz	Rol jnz	Jecxz rol	Movzx jecxz
3-gram	Mov test jz	Sub rol test	Sub rol test	Sub rol test	Sub rol test

1-gram	TEST	JPS	JP	JECXZ	PUNPCKLBW
2-gram	Lea mov	Sub rol	Sub rol	movzxjecxz	Rol jnz
3-gram	Test jz mov	Jbe lea rol	Jbe lea rol	Jbe lea rol	Jbe lea rol
1-gram	CALL	SUB	JNBE	SETBE	CMOVL
2-gram	Jnz mov	Lea rol	Jmp rol	Nop inc	Jecxz rol
3-gram	Mov mov cmp	Lea rol cmp	Neg mov jmp	jbe sub cmp	Jbe sub cmp
1-gram	ADD	ROL	MOVZX	STOS	REX.WB
Method top10	CFS	infogain	chisquare	GainRatio	symmetrical
2-gram	Push push	Jbe lea	Lea rol	Sub or	Jbe lea
3-gram	Mov mov lea	Jz sub mov	Jz sub mov	Rol jnz retn	Rol jnz retn
1-gram	INC	MOV	JBE	BSR	PMOVMSKB
2-gram	Test jnz	Jmp rol	Jbe lea	Jz jnbe	Nop inc
3-gram	Mov cmp mov	Neg mov jmp	Jbe sub cmp	Jmp sub rol	Jmp sub rol
1-gram	ROL	ADD	ROL	CMPXCHG8B	BSR
2-gram	Add mov	Lea sub	Movzx xor	Mov jl	Div cmp
3-gram	call cmp mov	Jbe sub cmp	Lea rol cmp	Jz jmp sub	Neg mov jmp
1-gram	SUB	CMP	MOV	LOOP	REX.WRX
2-gram	Cmp jnz	Rol add	Lea sub	Jz js	Jz jnbe

3-gram	Push call add	Add rol jnz	Jmp rol mov	negmovj mp jmp	jz jmp sub
1-gram	JNB	CALL	SUB	CMOVL	JBE
2-gram	Mov add	Movzx xor	Cmp rol	Add cmovb	Rol imul
3-gram	Call add cmp	Jmp rol mov	Add rol jnz	jz movzxlsl lsl	Add rol jnz
1-gram	NEG	TEST	LEA	REX.WX B	CMOVB
2-gram	Mov jmp	Add cmp	Add cmp	Rol jnz	Jmp rol
3-gram	Push mov push	Jnbe test neg	Jnbe test neg	Call add jnz	Jz movzx lsl