Journal of Computers Vol. 31 No. 1, 2020, pp. 184-191
doi:10.3966/199115992020023101015

Research on Cache Timing Template Attack Based on KNN Algorithm

Cai-Sen Chen[1], Zhi-Wei Cheng[2], Yu-Bo Wang[3*], An Wang[4], Xiang-liang Ma[5]

[1] Military Exercise and Training Center, Army Academy of Armored Forces, Beijing 100072, China
caisenchen@163.com

[2] The 32167 troops of the Chinese People's Liberation Army, LaSha 850000, China
cheng.zw@mail.scut.edu.cn

[3] Teaching and Research Support Center, Army Academy of Armored Forces, Beijing 100072, China
355094556@qq.com

[4] School of Computer Science, Beijing Institute of Technology, Beijing 100081, China
wanganl@bit.edu.cn

[5] Institute of Software, Chinese Academy of Sciences, Beijing 100049, China
maxiangliang@163.com

Received 13 January 2019; Revised 6 March 2019; Accepted 12 March 2019

Abstract. For the Cache timing template attack, a large amount of timing information needs to be collected during attacking other processes. For the processing of high-noise data, the attack efficiency is low and the success rate is not high. It is proposed to establish a template matrix with the Cache hit rate feature and use the KNN algorithm. Analyze and judge. First, normalize the collected Cache timing data, and then use the hit rate to create a template for attack. Experiment results show that the accuracy of attack using KNN algorithm is higher than that of traditional mean square error method, and can reach about 90%.

Keywords: Cache timing attack, Cache timing information, KNN algorithm, template attack

1 Introduction

Template attack is proposed by Chari et al. [1]. It is a side channel analysis method combined with probability statistics. The attack process is divided into two steps: template establishment and analysis. By analyzing the bypass information of a device running leak when the template is established, the operation of the device is in one-to-one correspondence with the characteristics of the leaked bypass information. The analysis and utilization is to obtain the bypass information leaked by other similar devices to deduce the operation of the device. Cache timing template attack is to establish a template to attack by analyzing the Cache timing information leaked during the device operation. During the template establishment process of Chari et al. [2]. When the acquired energy trajectory feature points are relatively large, the attack is particularly time consuming and requires a lot of computing resources, because the attack requires a large amount of bypass information.

The premise of Cache timing template attack is that the computer operating system has the characteristics of shared memory. When the system is running, the process will load the shared byte library into the Cache, and query the byte library to obtain the required value. In Linux, the GDK framework is used as the default user interface framework. When the user inputs the keyboard, the GDK library is queried, and the signal input by the keyboard is converted into Unicode characters through the query operation. The Cache timing template attack is to obtain the timing attack template by querying the

* Corresponding Author

Cache timing information leaked by the GDK library, and finally use the template to attack the system of the same version of the GDK library, thereby obtaining the value input by the user through the keyboard.

In the Cache timing template attack proposed by Brumley and Hakala [3], the Prime+Probe attack model is used to attack the L1 Cache group. This attack method requires the attack process and the victim process to run in the same core, which is difficult to implement. Brumley and Hakala use the Prime+Probe attack model. The efficiency and accuracy of attacking the Cache group are not high. This study uses the Flush+Probe attack model to attack the L3 Cache line. It does not require the attack process and the victim process to run in the same way. In the core, and the efficiency and accuracy of attack are improved. For the Cache timing template attack, when the data is collected, it is vulnerable to other processes, the collected data is noisy, and the accuracy is not high when judging. A method of using the Cache hit rate of the access address to establish the timing template is proposed. The KNN algorithm is used for predictive judgment.

The Cache timing template attack firstly uses the Flush+Reload attack model to get the keyboard input of the computer, and obtain the cache hit rate of each address in the GDK library multiple times. The KNN algorithm is used to analyze the Cache timing template, and the hit rate of the specified address is used to establish a template. Through the KNN algorithm, the input key values are sampled and classified, and during the attack phase, the collected hit rate trajectory is calculated to the classification. The distance is used to determine the key value entered. This attack method consumes a large amount of resources during the attack phase, but the accuracy of the judgment is better.

2 Related theory research

2.1 Flush+Reload Attack Model

In 2011, Gullasch et al. [4] proposed using the *clflush* command to implement a more efficient access-driven Cache timing attack. Using the *clflush* command to evict the data in the Cache into memory, firstly, executing a small instruction of the victim process, and then determine the Cache of the eviction. Whether the row data is reloaded into the Cache. In 2014, Yarom and Falkner discovered that the *clflush* command would drive all levels of Cache row data into memory, including the shared LLC (Last-Level-Cache). According to the characteristics of the *clflush* directive, they confirmed that this attack can achieve cross-core attacks, that is, the victim process and the spy process can run on different cores of the same CPU.

The algorithm of the Flush+Reload attack method is described as follows:

Flush+Reload attack method
1. Map binary (shared library) into the shared address space (into the Cache).
2. Use the *clflush* command from the Cache to refresh a shared Cache line.
3. Call the password process to execute the process.
4. It is detected whether the cached line that is refreshed in step 2 is loaded into the Cache by the password process.

Compared with the Prime+Probe attack method, the Flush+Reload attack method has a higher precision. The former can determine the Cache group accessed by the victim process, but the Cache group generally has multiple Cache lines. The Prime+Probe attack method cannot be accurate to the Cache line. Underneath, the Flush+Reload attack method can accurately determine the Cache line accessed by the victim process. This attack method is recognized by the cryptanalysis community because it can determine the Cache line accessed by the victim process. Irazoqui et al. [5] used this attack method to recover all the keys of the AES cryptographic algorithm in the VMWare virtual machine. In 2015, Gulmezoglu et al. [6] successfully exploited the AES algorithm implemented on OpenSSL by using the shared resource optimization technology of memory deduplication. In the same device, Irazoqui et al. [7] use this technology to detect and distinguish different cipher libraries running in different virtual machines. In addition, Irazoqui et al. [5]. also establish a precise hidden channel between virtual machines working together on the cloud platform. In 2015, Gruss et al. [2]. used Flush+Reload to collect the timing information input by the user's keyboard to create a template, and the template can automatically recognize the key information input by the user. This attack method has caused a huge

threat to keyboard input, cloud platform security, and software-implemented cryptographic algorithms.

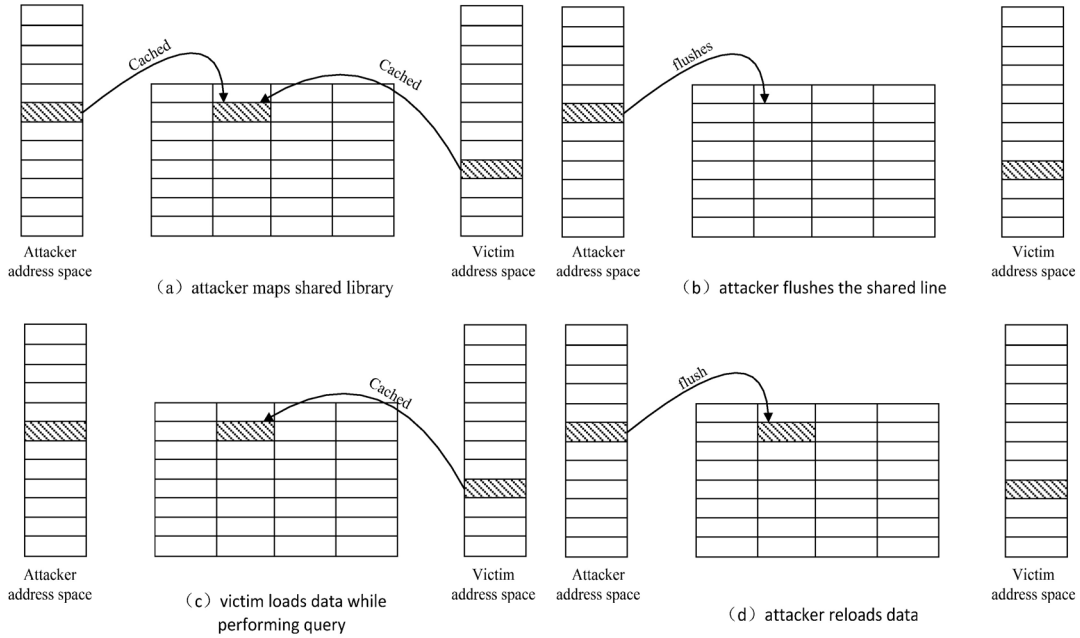

Fig. 1. Flush+Reload attack steps

2.2 KNN Algorithm Model

KNN (*K* Nearest Neighbor) is a simple classification and prediction algorithm, first proposed by Cover and Hart [8]. It is a classification algorithm for supervised learning in machine learning algorithms. The algorithm is easy to implement and does not require parameter estimation. The classification problem of multiple categories is particularly excellent, and the accuracy of prediction is higher than the SVM (Support Vector Machine) algorithm [9]. The KNN algorithm is a lazy algorithm. It is computationally intensive and consumes a large amount of memory, and the prediction result takes a long time [10].

The essence of KNN algorithm is a data analysis method based on data statistics. If all instances have *N* attributes, corresponding to *N*-dimensional space, all instances are divided into n classes. When judging a new instance, pass Calculate the distance between this instance and all other instances, select the *K* instance points closest to the instance, and finally count the *K* instance categories. The *K* categories in the *K* instances are the most judged. There are three main factors in the algorithm: the instance set, the distance and the size of *K*. The instance set is the data with the category label, the distance is the criterion for calculating the similarity between the two instances, and *K* is the nearest *K* from the newly added instance. For example, the size of *K* indicates the number of votes in the final judgment, and directly affects the accuracy of the algorithm. If *K*=1, the instance is directly assigned to his nearest class. Now suppose there are 3 classes, each with two properties, as shown in Fig. 2.

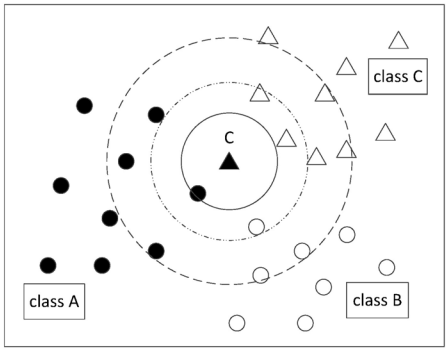

Fig. 2 KNN algorithm example

As shown in Fig. 2, the black solid circle represents class *A*, the open circle represents class *B*, the open triangle represents class *C*, the solid triangle represents a new instance, and each instance has two properties, so each face can be represented by a polygon Example. If *K*=1, the closest to the new instance *C* is a point of class A, the algorithm classifies the new instance as class *A*; if *K*=4, the nearest 4 points from the new instance, there are two classes *C*, There is one class *A* and one class *B* respectively. According to the principle of the algorithm, among the four instances, the instance belonging to class *C* is the most, and the new instance is classified as class C. If *K*=12, in the last 12 instances. The most common class *C*, there are five, the new instance is classified as class *C*.

3 Template Information Collection and Data Processing

The attack mainly targets the program query table operation, maps the shared binary to the memory, and monitors the access situation of the specified range address space. In the Linux operating system, the GDK framework is generally used as the default user interface framework. We can determine the value of the keyboard input by monitoring a part of the address in the GDK library, because when the keyboard is input, the keyboard signal needs to be converted into Unicode code and then displayed on the display. In the process of conversion, the *gdk_keysym_to_unicode_tab* array will be queried. Once the query needs to load the relevant data into the Cache, the information leaked by the Cache can be utilized. This array is included in the *libgdk-3.so.0.2200.24* part of the GDK library, so we can determine the value of the keyboard input by monitoring certain address segments of the GDK. Under non-privileged conditions, we use the map function to map *libgdk-3.so.0.2200.24* into memory and monitor this segment of address. To quickly capture a large amount of key information, we use the *libxdotool* tool for automatic analog input.

The algorithm is described as follows:

```
Input:  Set of events E, target program binary B, duration d
Output: Cache Template matrix T
Map binary B into memory
foreach event e in E do
   foreach address a in binary B do
      while duration d not passed do
         simultaneously
         Trigger event e and Flush+Reload attack on address a
         Save the number of Cache hits k and the number of times the event e is triggered.
       end
         Calculate the hit rate k / n and save it to T
    end
end
```

In the process of collecting Cache timing information, when the keyboard input is triggered, the specified target address is attacked. Each target address needs to collect the number of cache hits in time t and record the number of times of event e triggered during time *t*. In order to ensure accurate data collection, it is guaranteed to trigger multiple events *e* for *a* duration *t*. The duration *t* determines the time required to collect timing information. If the time *t* is too short, the number of hit information collected is small and not accurate enough; if the time t is too long, the acquisition takes a lot of time. When a cache failure occurs, the CPU always fetches the entire cache line when it acquires data, so we cannot distinguish the address offset bits in the same row, and we will get the same information in the address in the same cache line. On X86 processors, a Cache line typically has 64 bytes. When collecting data, a Cache line only supervises an address without losing accuracy.

We use the *libxdotool* tool to continuously simulate inputting a certain amount of the same character. At the same time, we use the Flush+Reload attack method to monitor the address. Firstly, refresh an address, then reload to determine whether the address is in life, record the number of hits and the number of times the button is triggered. And record the hit rate, get the hit rate matrix at each address for each key input time.

The traditional Cache timing template attack needs to collect the cache hit rate of the entire address line, then manually analyze the hit rate, remove the invalid address line, and then select the threshold, set the value greater than the threshold to 1, and the hit ratio is set to 0, to establish a Cache timing attack template for attack. Since the threshold value is required to perform secondary processing on the data, the effective information of too much original data is lost, and a certain error is brought to the judgment result. In this section, the original data is retained as much as possible, and the original data is directly used for judgment.

The template data collection is divided into two steps. First, the cache hit rate data is collected for all the addresses of the *gdk_keysym_to_unicode_tab* array. Through analysis, the valid and distinguishable button addresses are used as the monitoring address, and then the selected addresses are directly monitored, through libxdotool. The tool simulates the input button value, and collects the Cache timing information of the specified address multiple times to establish a Cache timing attack template. When collecting data, you first need to trigger an event, record the Cache hit rate trajectory of this event during this time, and calculate the Cache hit rate of the event in an address, and then save it in the Cache hit rate template.

The time taken to collect the Cache timing information is proportional to the duration of the analog button. In the first data acquisition, in order to ensure that the selected address is valid, the button duration of the first data acquisition is set to 2.5×108 CPU cycles. Different buttons should have a certain difference in the selected address. The experimental platform is running the kali Linux 2017.3 version of the operating system, the attack address library is libgdk-3.so.0.2200.24, in order to simplify the experiment, the experiment only collects the Cache timing information of the digital button part and judges and predicts the digital input. After completing the first experiment, 9 addresses are selected, which are *0x29140, 0x29180, 0x291C0, 2A380, 0x2A5C0, 0x2BA40, 0x2BE40, 0x2FC00*, and *0x30FC0*. The address is relative to the attacked address library *libgdk-3.so.0*. The offset of the first address 2200.24. The second time of the data is collected, the duration *t* can be reduced appropriately, because it is necessary to collect the Cache timing information of the button multiple times, and the appropriate amount of time t can shorten the data collection time. In the second experiment, we respectively have the duration. t is 1.5×108, 2.0×108, 2.5×108 and 3.0×108 CPU cycles to collect 100 Cache timing for each key input, and then use 25, 50, 75 and 100 key information to create a template for judgment.

4 Template Establishment and Experimental Analysis

4.1 Template Establishment and Utilization

A simple analysis is performed on the Cache timing information collected by the digital key input. When the input key is related to a certain address, a cache hit occurs, and the collected cache hit ratio is greater than one, and there is also a case less than one; Cache failure occurs regardless of an address. However, the collected Cache timing information has a cache hit ratio greater than 1. These situations are caused by noise. When collecting timing information, it is impossible to remove unrelated processes. The effect is that it is difficult to establish an effective Cache Timing Template according to a hit rate trajectory. We can collect a hit rate trajectory of a specified address multiple times to establish a valid template from multiple Cache hit rate trajectories.

In order to prevent the collected extreme data from predicting the algorithm, firstly, each Cache hit rate trajectory is Z-score normalized, so that the value of each hit rate trajectory conforms to the normal distribution, and the influence of extreme data is minimized. For the data that has been standardized, the Euclidean distance is chosen as a measure of the similarity of the two hit rate trajectories. Since the collected data has noise, when selecting the K value, it can contain the noise and remove the influence of a small amount of noise.

After establishing the template from the collected data, it is necessary to determine the appropriate K value through experiments, randomly select 20% from the data standardized by Z-score for cross-validation experiments, and verify the accuracy of K value by cross-validation. After selecting appropriate K value, the experimental analysis of the amount of data needed to establish the template is carried out, and the template is established using 25, 50, 75 and 100 times of each button, and the accuracy of the algorithm prediction is compared.

4.2 Experiment and Result Analysis

The experimental environment is the operating system of kali Linux 2017.3 version, the CPU is Intel i7-4720HQ, the memory is 16G, and the attack object is *libgdk-3.so.0.2200.24* library. In order to obtain a reasonable K value, the experiment uses 1000 data collected at a duration of t=1.5×108, each of which has 100 data, and randomly selects 20% of the data for cross-validation to obtain the accuracy of the algorithm. The result is shown in Fig. 3.

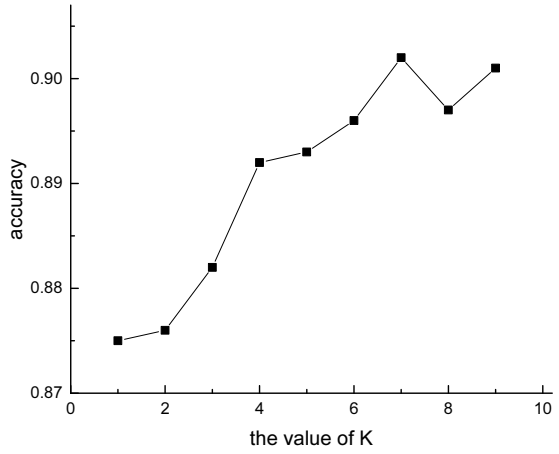

Fig. 3. Relationship between K value and accuracy

From Fig. 3, when *K*=1, the accuracy rate is 87.5%. With the increase of *K* value, the correctness rate of the algorithm generally increases. When *K*=7, the accuracy rate is up to 90.2%. When *K*=8, the accuracy rate decreases, and it is proved that K is not as large as possible in this case, so *K*=7 is selected as a suitable value.

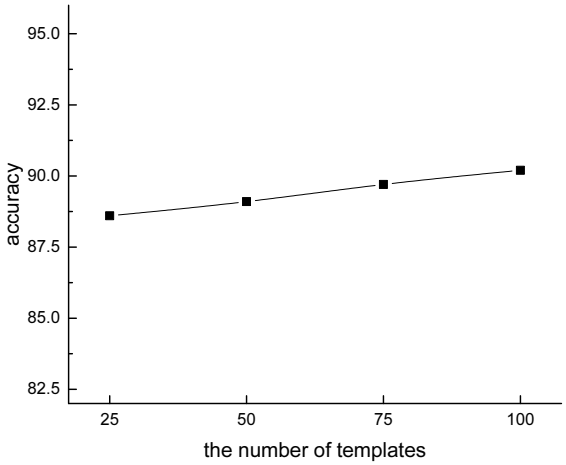

Fig. 4. Relationship between the number of templates and the accuracy

The experiment selects *K*=7, and uses the data collected at the duration *t*=1.5×108, and uses each of the 25, 50, 75 and 100 data of each button to establish a template to judge the result as shown in Fig. 4.

As shown in Fig. 4, the more the number of templates is established, the higher the accuracy is when the KNN algorithm is used for judgment. When only 25 data is used to create the template, the accuracy is 88.6%. When 100 data is used to establish the template. The accuracy rate has increased to 90.2%.

If the duration *t* is large enough, the more accurate the template is established, the better the accuracy can be achieved by using the mean square error method, but the prediction using the mean square error method is that the accuracy convergence is slow, and the convergence is fast using the KNN algorithm for the duration t. In the shorter case, good accuracy can be maintained. In order to compare the advantages of using the KNN algorithm and the mean square error method, data is collected when the

duration t is 0.5×108, 1.0×108, 1.5×108, 2.0×108, 2.5×108, and 3.0×108 CPU cycles. Each button value collects 100 Cache timing information for experiment, and the experimental results are shown in Fig. 5.

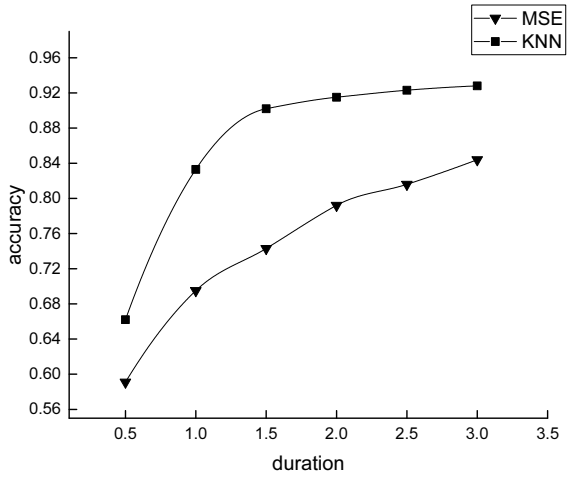

Fig. 5. Comparison of MSE and KNN methods

From Fig. 5, when the duration $t = 1.5 \times 108$, the accuracy of using the KNN algorithm is 90.2%, the accuracy of using the mean square error method is 74.3%, and when the duration is $t = 2.0 \times 108$; using KNN The accuracy of the algorithm is 91.5%, and the accuracy gradually converges. Collecting data takes a lot of time, which increases the efficiency of the attack. It can be clearly seen from the figure that as the duration t increases, the accuracy of the KNN algorithm gradually increases, and the growth is slower, but still superior to the method using the mean square error.

4 Conclusion

Firstly, the principle of Flush+Reload attack model and KNN algorithm model are analyzed. Then the Cache timing information collection technology is deeply studied. Finally, the attack template is built by using Flush+Reload model, and the attack is obtained in Linux system environment. Input, using KNN algorithm to process data, compared with the mean square error method, the accuracy rate has been significantly improved.

Acknowledgments

The authors would like to thank anonymous reviewers for their valuable comments. This research was supported by the National Natural Science Foundation of China under Grant No. U1836101, No. 61402528, National Cryptography Development Fund (No. MMJJ20170201).

References

[1] S. Chari, J.R Rao, P. Rohatgi, Template Attacks, in: Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2002), 2002.

[2] D. Gruss, R. Spreitzer, S. Mangard, Cache template attacks: automating attacks on inclusive last-level caches, in: Proc. Usenix Conference on Security Symposium, 2015.

[3] B.B. Brumley, R.M. Hakala, Cache-timing template Attacks, in: Proc. Advances in Cryptology- ASIACRYPT 2009- 15th International Conference on the Theory and Application of Cryptology and Information Security, 2009.

[4] D. Gullasch, E. Bangerter, S. Krenn, Cache games--bringing access-based cache attacks on AES to practice, in: Proc. IEEE Symposium on Security and Privacy, 2011.

[5] G. Irazoqui, M.S. Inci, T. Eisenbarth, B. Sunar, Wait a Minute! A fast, Cross-VM Attack on AES, in: Proc. International Workshop on Recent Advances in Intrusion Detection, 2014.

[6] B. Gülmezoğlu, M.S. İnci, G. Irazoqui, T. Eisenbarth, B. Sunar, A Faster and More Realistic Flush+Reload Attack on AES, in: Proc. Constructive Side-Channel Analysis and Secure Design (COSADE 2015), 2015.

[7] G. Irazoqui, M.S. İnci, T. Eisenbarth, B. Sunar, Know thy neighbor: crypto library detection in cloud, in: Proc. Privacy Enhancing Technologie, 2015.

[8] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory 13(1)(1967) 21-27.

[9] R.A. Nugrahaeni, K. Mutijarsa, Comparative analysis of machine learning KNN, SVM, and random forests algorithm for facial expression classification, in: Proc. Technology of Information and Communication, 2017.

[10] Y. Zhao, Y. Qian, C. Li, Improved KNN text classification algorithm with MapReduce implementation, in: Proc. International Conference on Systems and Informatics, 2018.