Secure Programming
Assignment 3

Puzhi Yao a1205593

October 26, 2016

1 Implementation

In this assignment, I have implemented the cache template attack on libgdk-3 through gedit based on [1]. The detail information of attacked target are listed below:

Attacked Machines:

- Virtual Machine: Elementary OS
- Ubuntu Kylin

Attacked process: gedit 3.10.4

Attacked library: $libgdk-3.so.0.1000.8$

First, we have to estimate the threshold for minimum cache miss cycles which can be determined by calibration tool. The minimum cycle is then replaced in the profiling.c and exploitation.c code. This minimum cache miss cycle is used to determine whether the attack is valid or not.

Secondly, the main tools we used to perform cache template attack are profiling.c and exploitation.c. These two files are adopted from spy.c[1]. Due to the performance difference between virtual machine and actual operating system, the example spy.c can not perform full run or take too much time to scan large libraries on the virtual machine. Therefore, the following changes I made to the original source code to make it work on virtual machine.

- Increase the block size from 64 to 1024, which reduces the time cost and cache hit rate.
- The attack script only targets libgdk-3 without scanning other shared libraries.
- The results of profiling.c shows the simple post-process results rather than cache template matrix.
- Using the simple post-process result as the input of exploitation to validate the obtain address.
- The time, cache hit and CPU cycles are recorded in spreadsheet and based on plotted figures to determine whether the post-process result is correct or not.

2 Result and Validation

The profiling.c will generate all possible events per address once we have finished profiling phase. Then, I run the exploitation.c on each single event memory address to confirm that the library is indeed leaking data at certain memory addresses. The exploitation.c will constantly perform flush and reload on the offset specified and record the cache hit that higher than minimum cache miss cycle. Due to the total number of addresses and attack complexity, we only validate single event address (e.g. 0x85d00:q) which means that if the validation is true, then we can predict certain key press while monitoring this memory address with high accuracy.

The Figure 1 illustrate the example results of profiling for $libgdk-3.so.0.1000.8$. The profiling.c scanned 10241 memory addresses and found 94 single event addresses. As shown in Figure 1, The first column is

2	Address	Keys	Exploitation Check	/usr/lib/x86_64-linux-gnu/libgdk-3.so.0.1000.8
1779	0x1bc00	d	Triggered when Press any key/Move mouse	
1780	0x1bc40	v	Triggered when Press any key/Move mouse	
1811	0x1c400	l	Triggered when Press any key/Move mouse	
1948	0x1e640	a	Triggered when Press any key/Move mouse	
1953	0x1e780	y	FALSE	
2049	0x1ff80	t	TRUE	
2332	0x24640	v	FALSE	
2429	0x25e80	c	TRUE	
2442	0x261c0	e	TRUE	
2457	0x26580	t	FALSE	
2458	0x265c0	r	FALSE	
2498	0x26fc0	t	FALSE	
2531	0x27800	x	FALSE	

Figure 1: Example results of profiling

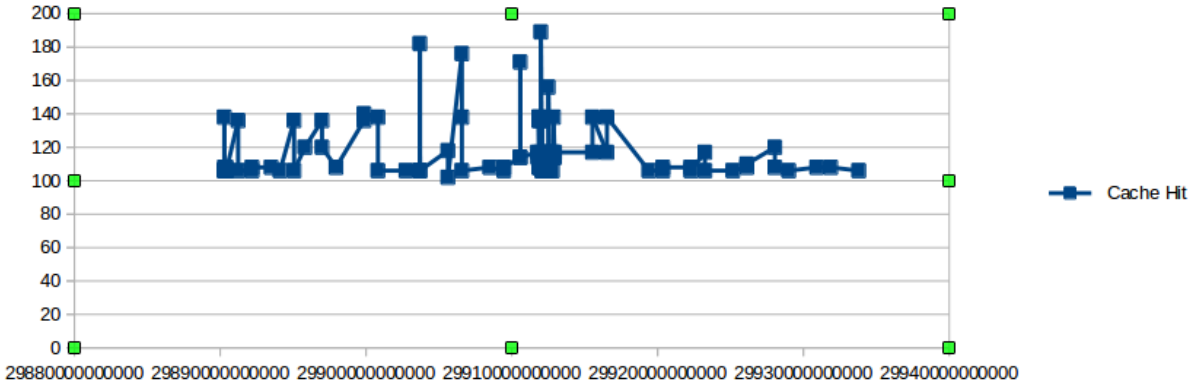

Figure 2: Exploitation results for 0x2c340 with key press d

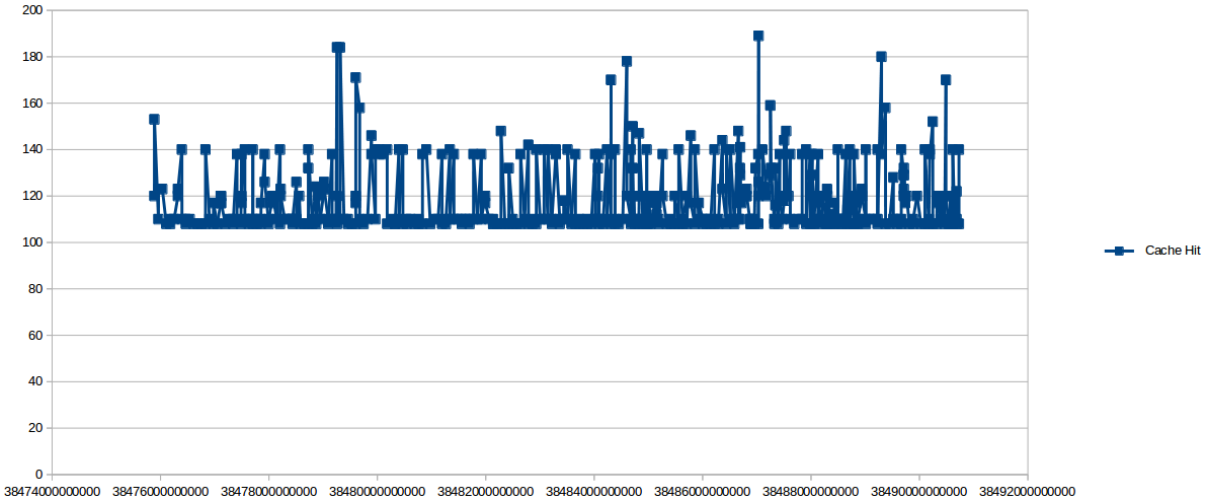

Figure 3: Exploitation results for 0x3bfc0 with key press w

the scanned memory address. The second column is the event key and third column is the exploitation checking result. The exploitation checking result is obtained by validating the cache hit plot while key pressed. The Figure 2 and Figure 3 illustrate the exploitation results for different addresses. In these two plots, the cache hit rate is the vertical axis and horizontal axis is the time cost. The detail results can be found in $Assignment3/result/Geditgdk3$ directory.

3 Analysis

This section will analyze cache template attack based on different phases.

3.1 Calibration

The minimum cache miss cycles may vary for different CPU architecture. Calibration tool should be run several times and the minimum cycle threshold should be the lowest one as it captures more data. However, if the minimum cycle is too low, it may increase variants or noise during profiling and exploitation phases.

3.2 Profiling

The profiling results can be categorized into three groups:

- No event addresses: Profiling.c does not capture any cache hit on these addresses.
- Multiple event addresses: Profiling.c captures multiple events when scanning theses addresses. Theses addresses may have leaks, but they are hard to be identified. Therefore, we will ignore this type of addresses.
- Single event addresses: Profiling.c captures single event on these addresses, which means that these addresses have high possibilities that leaks key information and can be exploited.

After the exploitation phase, some single event addresses are identified as false results and some addresses is triggered by other types of event. As shown in Figure 1, address 0x1bc00,0x1bc40, 0x1c400, 0x1e640 may leak the mouse movement and click information which can be used to perform mouse track in other attacks or exploitation.

3.3 Exploitation

From Figure 2 and Figure 3, the exploitation results can be categorized into two groups:

- Low noise
- High noise

In Figure 2, it shows the low noise results. Most cache hits are below 140 which is the average cache hits. By observing the peaks that above 140, we can quickly infer whether the victim is currently pressing the certain key or not. In Figure 3, it depicts the high noise results. During the exploitation of 0x3bfe0, we only pressed 4 times 'w'. Although most cache hits are between 100 and 140, there are still several peaks that over 160 which are not triggered by our key pressing.

These two types of results illustrate that even we found the memory address that leaks information, it is still hard to set the peak threshold and predict key pressing accurately. By comparing my results with Gruss et al [1], I found that the leaking address may vary based on different version of target machines, processes and libraries, which increases the complexity to reuse previous found result addresses.

4 Extension

For the extension, I have implemented the cache template attack on Firefox shared library libthread. The detail information of attacked target are listed below:

- Attacked Machines: Ubuntu Kylin

- Attacked process: FireFox 49.0
- Attacked library: $libpthread - 2.19.so$

The results I found are listed in directory: Assignment3/results/FireFox

References

[1] Gruss, D., Spreitzer, R., & Mangard, S. (2015). Cache template attacks: Automating attacks on inclusive last-level caches. In 24th USENIX Security Symposium (USENIX Security 15) (pp. 897-912). `https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-gruss.pdf`