Lab 5 - Flush+Reload Attack against AES

Due on Mar. 19th (T) 2019

1 Description

In this lab, you will learn a widely used cache side-channel attack: Flush+Reload attack. It relies on a special x86 instruction, *clflush*, and an OS optimization, *memory deduplication*. *clflush* instruction evicts cache lines corresponding to a specified memory address. *memory deduplication* merges memory pages with the same data that belong to different processes, meaning when two processes load the same file into the main memory, only one copy of the file exists in memory and both processes share it.

You will use the *clflush* instruction to monitor the usage of AES T-tables in the OpenSSL library. Based on the usage pattern, you will recover the last round encryption key.

2 System Profiling

To understand the cache hierarchy of your system, you will first need to determine the timing characteristic of the main memory and caches on your testing platform. In this step, you need to create a micro-benchmark to measure the time distributions for last-level cache hits and misses, respectively. Show the timing distributions in the same graph with two different colors. Based on the graph, determine an appropriate threshold that can be used to separate cache hit and cache miss from the measured time. You also need to figure out the cache line size of the last level cache.

3 Using OpenSSL Library

We have used the OpenSSL library for Lab 1. Download it again from https://github.com/openssl/openssl.git and refresh your memory. You will need to checkout the repository and compile it on your machine.
To checkout the repository:

```
$ git clone https://github.com/openssl/openssl.git
```

To compile it on your machine:

```
$ cd openssl
$ ./config no-hw no-asm
$ make
```

Note that it is important to have *no-hw* and *no-asm* flags during the Makefile configuration. These two flags are enforcing to use the T-table implementations and also avoid using the AES-NI instruction extension. Once it is compiled, you will see *libcrypto.so* in the openssl directory. This is the shared library which will be loaded with the victim and attack binaries.

4 Preparing Flush + Reload Attack

Finding T-table Address: For Flush + Reload, the spy uses the *clflush* instruction to monitor one selected cache line of a specific T table. For example, you can choose the first cache line of the first T-table *Te0*. To do that, you will need to determine the offset of your target within the shared library. You will first disassemble the shared library and locate your target. You can use the following command to do so:

```
$ objdump -D libcrypto.so > disassembled-crypto.txt
```

Now, open *disassembled-crypto.txt* file in any text editor of your choice and find the *Te0*. You will see something similar to the following:

```
00000000001a3d60 <Te0>:
  1a3d60:   a5                      movsl  %ds:(%rsi),%es:(%rdi)
  1a3d61:   63 63 c6                movslq -0x3a(%rbx),%esp
  1a3d64:   84 7c 7c f8             test   %bh,-0x8(%rsp,%rdi,2)
  1a3d68:   99                      cltd
```

The left hand side is the offset of your target within the shared library. In this case, *1a3d60* is the offset of the target for the first cache line of *Te0*.

You can also use another way to find the Te0 address (the first cache line):

```
$ readelf -a libcrypto.so > ~/aeslib.txt
```

Setting Up the Attack: The experiment has two processes running at the same time: attacker and victim. You are given the two c files. The victim source code file is complete and no need to change. You should still study it and understand what it is doing before moving onto attacking. Hint: victim and attacker are set up in a server-client socket model. You will need to complete attacker.c file. Then when compiling these two files, you need to compile them against the shared library (libcrypto.so) you created in the previous step.

Most of the code for attacker.c is already given except for a target setting line and a function called doTrace(). The target setting line is in function of init(), which is to figure out the memory address for the given offset in the shared library (e.g., the first cache line for Te0). Check out the source code of attacker.c and find the function to use for setting the target memory address.

```
target = ; // setup the target for monitoring
```

Function doTrace() implements the major steps for one round of Flush+Reload operations, and is called in an iterative loop (e.g., 1M times). For collecting one time sample, the operations are: 1. generate a random plaintext; 2. flush the corresponding memory address (using clflush); 3. send the plaintext to the server (victim) for encryption; 4. on receiving the ciphertext, reload the memory address and time it; 5. save the trace (the ciphertext and timing sample). Hints for steps are given in the function definition place.

```
void doTrace()
```

Note if you are using our ecehss server for this lab, you may want to change the port number in both *attacker.c* and *victim.c* as your classmates may be using the server at the same default port number.

```
server.sin_port = htons([Put your port number here]);
```

Once you have completed the attacker source code, you can proceed to compile both attacker and victim sources. Note that you need to link the shared library you just created. The compilation example given below assumes you have put the attacker.c and victim.c in one directory above the root directory of openssl (you have downloaded and compiled).

```
$ gcc attacker.c -Iopenssl/includes -Lopenssl/ -o attacker -lcrypto
$ gcc victim.c -Iopenssl/includes -Lopenssl/ -o victim -lcrypto
```

When you run an executable (e.g., victim or attacker) compiled with a shared library (also called dynamic library), you will need to tell the loader where to find that library. You can set *LD_LIBRARY_PATH* environment variable in this way: export LD_LIBRARY_PATH=[openssl root directory]. You can refer to https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html if you don't know how to run the binary.

5 Attack - Online Phase: Collect Time Samples

You need two terminals: on one terminal you run the victim, and on the other terminal, you run the attack. Once you have collected 1 million samples, compile and run analysis program to analyze those samples. You are given analysis.c. The analysis program calculates the average reload time for each cipher value in each cipher byte. Using *plot.py* to plot the result from the analysis program. If you have successfully monitored one cache line of a lookup table, you should observe 16 outliers in each of 4 figures.

The dependency packages for running *plot.py* are noted in *requirements.txt*. You can use *virtualenv* command when you are working on a Python related script/project. Follow the following commands to initialize the Python environment and run *plot.py* script. Refer to https://docs.python-guide.org/dev/virtualenvs/#lower-level-virtualenv and https://docs.python-guide.org/dev/virtualenvs/#other-notes for more detail on how to use *virtualenv* command.

```
$ pip install virtualenv
$ virtualenv -p python2.7 env
$ source env/bin/activate
$ pip install -r requirements.txt
$ python plot.py result.txt
```

6 Attack - Offline Analysis: Recover Last Round Key Bytes

To recover key bytes, you will need the threshold you found in Section 2 and samples you collected in Section 5. For each sample, first you use the threshold to determine whether to keep this sample or not - keep if the time is lower than the threshold. You want to keep samples that used the monitored cache line and discard the rest. Use the following algorithm to recover each key byte value individually. The k value with the highest counter is the correct key.

```
For every ciphertext in the kept samples
  For k = 0...255
    1. Calculate Inverse_Sbox[ k xor ciphertext[j] ] => s
    2. Check s in {0,1,2,3,...,15}?
    3. If so, increment the counter for k
```

By monitoring one cache line, you should be able to recover four key bytes. Pick cache lines in other T tables, and recover other 12 key bytes as well.

You can refer to https://eprint.iacr.org/2014/435.pdf and https://ieeexplore.ieee.org/document/8203771 for AES key recovery methods.

7 What You Need to Turn In

To receive credit on this lab, you will need to turn in all your code, and a brief report with following items:

1. Figures showing the timing distributions for L3 cache hit and miss.
2. Cache line size of your platform.
3. Your threshold.
4. The figure generated using *plot.py*.
5. Report 16 last round key bytes you have recovered.
6. Discuss the advantages and disadvantages for using a shared library.
7. Suggest two countermeasures to prevent this type of attacks and describe how.