Lab 03 Experimenting with Dynamic Taint Analysis in DECAF

Preparation

Download DECAF from github (https://github.com/sycurelab/DECAF), and go to the "decaf" subfolder.

1. You will need to install some dependencies:

```
sudo apt-get install qemu
sudo apt-get build-dep qemu
```

2. Start to configure and make using the following commands:

```
./configure –target-list=i386-softmmu –enable-tcg-taint –enable-vmi
make
```

The "configure" indicates that we like to build DECAF for the guest architecture of 32-bit x86, and enable TCG-level tainting support, and enable Virtual Machine Introspection (VMI). After "make" is done, an executable will be generated in the "i386-softmmu" folder.

3. Prepare a virtual machine image.

For your convenience, I can share my Windows XP image just for the purpose of this course. Due to the license restriction, please do not use it for other purposes. You can download it from http://www.cs.ucr.edu/~heng/teaching/cs260-winter2017/xpsp3.zip. It is protected by a password, which will be disseminated via iLearn announcement.

4. How to use DECAF

Read tutorials https://code.google.com/archive/p/decaf-platform/wikis/DECAF.wiki

Use the following command line to launch DECAF:

```
/path/to/qemu-system-i386 –m 1024 –monitor stdio -netdev user,id=mynet -device
rtl8139,netdev=mynet /path/to/xpsp3.img
```

In this command, you specify to allocate 1024MB RAM to the virtual machine, and redirect the monitor window to the standard IO, and launch Windows XP from the provided image.

You can type "help" to review what commands are available for you to use. Several useful ones are below:

ps: list the running processes
lsmod: list the kernel modules
guest_modules: list the modules for a given process
load_plugin: load an analysis plugin
unload_plugin: unload the analysis plugin
tainted_bytes: show the number of tainted bytes in the virtual machine

Task 1: Implement a simple taint analysis plugin

Objective: Implement a simple plugin that performs dynamic taint analysis and detects control-flow hijacking attacks. We will still use the same example01.c as the test program. More specifically, when we give a normal input to this program, we would like to see dynamic taint analysis gives no alarms, whereas when giving a malicious input, we would expect to see dynamic taint analysis detects a malicious control flow transfer (more precisely, the program counter (EIP) becomes tainted).

Hints:

1. Since we are running Windows inside the virtual machine, we need to compile example01.c into a Windows executable. You can use a compiler for windows such as MSVC, but here I suggest a more convenient alternative, a cross compiler MinGW. To install it on Ubuntu, run:

   ```
   sudo apt-get install mingw32
   ```

   Then, you can compile it like below:

   ```
   i586-mingw32msvc-gcc –m32 –fno-stack-protector –o ex01.exe example01.c
   ```

   It will generate an executable called ex01.exe. Then you need to transport this file into the virtual machine. There are various ways to do so. You can google it. One way I suggest is to just put this file into an HTTP server, and point your browser in Windows to get it.

2. To understand how to write a plugin that uses dynamic taint analysis, please read the keylogger plugin under plugins/keylogger/ directory carefully. Much of the code can be reused to this assignment.

   To taint the program input, you can just use taint_sendkey command to enter tainted keystrokes.

   To detect a tainted EIP, you need to register a DECAF_EIP_CHECK_CB callback (defined in shared/DECAF_callback_common.h), by using the following DECAF API (defined in shared/DECAF_callback.h).

   ```
   DECAF_Handle DECAF_register_callback(
               DECAF_callback_type_t cb_type,
               DECAF_callback_func_t cb_func,
               int *cb_cond
               );
   ```

   Here is an example how to do it:

   ```
   eip_check_handler = DECAF_register_callback(DECAF_EIP_CHECK_CB,
   my_eip_check, NULL);

   void my_eip_check(DECAF_Callback_Param * p) {
      DECAF_EIP_Check_Params *param = (DECAF_EIP_Check_Params *)p;
   ```

```
    //param->target_eip_taint will hold the taint status for EIP.
}
```

3. More complexities when performing whole-system taint analysis:
    a. Shall we enable pointer tainting?
       Answer: You can first try tainting with pointer tainting disabled (by default). If you could not observe tainted EIP even when you supply a malicious input, then it probably means you need to turn it on.
    b. What if we observe tainted EIP in the kernel or other processes than the one of interest?
       Answer: With the VMI functionality provided by DECAF, you can tell which module and process it is from, and thus filter them out.