18-739L – Research Proposal

Rajiv Kulkarni (rajivk)

1) Introduction

Many problems in CTFs involve hijacking the control flow during the execution of a program. This includes the likes of stack based buffer overflows, heap overflows and format string vulnerabilities. The exploit involves providing user input such that the control-flow of the program is manipulated to perform unintended actions. All the stack based buffer overflow problems where one has to overwrite the return address and execute a shell code fall under this category.

I will be writing a tool to automate the process of identifying which of the elements of a program can be controlled by the attacker. The tool helps in identifying what input is required in order to control the eip register (which means that the execution flow can be hijacked). In addition to that, the tool also identifies which registers can be controlled as well as what memory regions in the stack, heap and global space. An additional feature of the tool is to scan the usual format string functions used in the input program to identify if they pose a threat.

2) Approach

One type of vulnerabilities this tool aims to exploit is a stack-based buffer overflow. A simple and straight forward example would be the following code:

```
#include <stdio.h>
  int main(int argc, char **argv)
  {
        char buf[8]; // buffer for eight characters
        gets(buf); // read from stdio (sensitive function!)
        printf("%s\n", buf); // print out data stored in buf
        return 0; // 0 as return value
  }
```

In the above example, since there is no length-check on the *buf* variable, an attacker can input an arbitrary length input eventually overwriting the return address of *main* function. When the program returns from main, it now tries to return to the new address. If the new return address is a valid address, the program starts executing instructions at the new address which forms the basis of buffer overflow attacks. If the return address is an invalid one, the program usually crashes with a SIGSEGV or segmentation fault. The tool utilizes this fact to determine the change in the control flow. It uses the technique which is used by many other tools, and generates a random input.

The program is most likely to crash if the input overwrites the return address. At this crash point, the tool analyses the core dump of the program. It first scans the EIP for the random input it provided. Thus the exact offset that caused the crash can be established. Also, the tool scans the other registers to

determine what other registers are controlled by the input. This is followed by a scan of stack and heap to check which areas of memory is controllable by the input. Also, if the tool has access to debugging symbols, it matches the memory location with variable names to precisely indicate which variable is controlled by the input.

The tool, in parallel looks out for format string functions. When a format string function is encountered, and its first argument is in writable memory, the tool indicates a potential format string vulnerability.

Once the analysis is complete, the tool also provides potential exploit strings. These exploits can directly be used as inputs to the vulnerable programs. Hence from a user's perspective, this tool can be used to perform a complete analysis of which elements in the program are controllable and if the program is simple enough, use the generated exploit string to directly exploit the program.

3) Implementation and Evaluation

This tool would be written in python and would also be leveraging the python scripting ability of gdb. The user would start this tool by giving the path of the program to be exploited. The tool then scans the assembly of this program to determine which functions take user-inputs and the length of the input expected, and calculates the distance between the return address and the input buffer. The tool then generates appropriate random inputs and stores them. The program is then started and these random values are provided as input.

The program is then allowed to continue its course till it generates a segmentation fault and dumps a core file. This core file is then opened by the tool using gdb. Once the core file is opened in gdb, a python script is started which performs all the analysis mentioned above. Another script searches scans for format functions and checks if their first argument in pointing to writable memory. This analysis is later presented to the user. Also, the tool leverages the 'checksec' functionality like the one used in gdb-peda. Based on the security options enabled for the binary, the tool generates potential exploit strings. The exploit strings include a normal stack based exploit string, an environment variable based exploit, a ret2reg type exploit, etc. The tool also utilizes another open-source tool "ROPGadget" to generate a ROP exploit.

This tool would first be tested on the PicoCTF problem "Execute Me". This program just executes whatever is given as input. When the tool first generates random input, it would segfault at the first instruction, thus indicating that the execution begins from the beginning of the input itself. Then, a shell code is generated which can be passed as input to the program.

This tool would then be evaluated on ROP1, another simple PicoCTF problem with ASLR enabled. The tool would generate the random input, and calculate the offset that is giving us control. However, since it is ASLR enabled, a reliable exploit string can't be generated using traditional methods. This tool should detect that the value of eax register is controllable by the user input and hence should search for a *jmp eax/call eax* and generate a ret2reg type exploit.

On passing these tests, the tool can be evaluated on more complex problems of PicoCTF and other CTFs.

4) Related Work

1) *https://github.com/longld/peda*

A lot of the features of the tool can be found in gdb-peda. However, this tool aims at reducing the number of steps and manual intervention, while determining all the elements of a vulnerable program a malicious input can control.

2) *http://shell-storm.org/project/ROPgadget/*

ROPgadget is used by this tool to generate a ROP exploit

5) Updates