Writeup

Kevin Turkington

I. General Exploitation

In Brad Antoiewiez's lectures he describes the basics to manipulating software as using vulnerabilities in a program unintended by the programmer. This can be through the use of finding bugs within software. As well as taking advantage of poor programming practices by the developer for example Mac OS bug of gaining root access into the "User Groups" in system preferences using 'root' as a user name as seen in Robert Hackett's article Apple Macs Have Yet Another Password-Bypassing Bug. When looking for these vulrabilities intent can not always be nefarious, but can be turned into to specific companies 'Bug Bounty programs' for monetary rewards. For most of the labs within this weeks lectures we used the tool WinDbg to analyze and determine weather our programs exploits were completed correctly.

II. Stack overflow exploit

The stack overflow exploit is simply a case of stack smashing. Where a hacker loads the stack with a large set of repeated jibberish ending with an address in memory. Specifically the hacker is trying to overwrite any return address instructions with an address of their own pointing towards a separate payload with in stack memory. For example in Dhaval Kapils 'Buffer Overflow Exploit' blog post he provides the following compilable code for stack overflow exploitation:

```
#include <stdio.h>

void secretFunction()
{
    printf("Congratulations!\n");
    printf("You have entered in the secret function!\n");
}

void echo()
{
    char buffer[20];
    printf("Enter some text:\n");
    scanf("%s", buffer);
    printf("You entered: %s\n", buffer);
}
```

```
int main()
{
    echo();
    return 0;
}
```

When the user is prompted to Enter some text the program can be exploited by providing a repeated set of hexadecimal values leading to the `secretFunction()` how ever this will differ machine to machine. For most modern computer this exploit is protected by utilizing a stack canary or stack cookie. Which is normally a randomly generated non-unicode or ascii character placed right before the return address pointer.

The general step to using this exploit as shown in the slides are:

1) Crash triage
2) Determine return address offset
3) position shell code
4) Find the address of our shell code

A. Free after use exploit

In a similar way to the stack overflow exploit where we overwrite the return address instruction, we are finding in heap memory where a class or data was recently freed and replacing the address of that freed data to a payload or shell code. So when the next time that page is reallocated and used it will contain a payload and will be promptly executed. A possible deterrent to this exploit could be immediately overwriting newly allocated heap memory or clearing it before use.

1) Free the object
2) Replace the object with your object
    a) Figure out the address
    b) make allocations of the same size
3) position shell code
4) Use object again