John Rocco, Albert Yeung, David Dvorkin 12/10/18

CS458: Introduction to Computer Security Assignment 4

Heap Spray

Heap spraying is a popular code injection technique used in exploiting a program. It streamlines the method to execute code in areas that should otherwise have been protected from outside injections. Heap sprays, as the name implies, spray bytes into a location of memory on a process' heap. It fills a heap's blocks with specific values and flags them so it can be returned to later in order to exploit a target system.

Heap sprays do not exploit security issues. They are instead used to make a currently present vulnerability easier to exploit. Therefore, a heap spray is generally accompanied with some security exploit to break security boundaries. Memory is often aligned randomly from the hacker's point of view due to address space layout randomization. This, along with timing, makes it difficult to pinpoint where one should exploit security flaws. Heap sprays account for the randomness and increase chances of success. Most operating systems have large heap allocations in the same general locations and sequential allocation. Therefore, whenever a heap is sprayed, it will be in approximately the same place each time.

The attacker most commonly sprays the heap with a string consisting of a NOP slide and shellcode. A NOP is a "no operation" (0x90 in hex) assembly instruction. When it is executed, the program will simply continue on to the next line. A NOP slide is therefore a long chain of NOP instructions. It is extremely easy to physically spot a NOP slide when looking at the heap. The shellcode afterwards is the code that will actually perform the malicious action. This is the binary code visible after executing objdump on a file. For this to execute properly once it is

sprayed onto the heap, the shellcode must be executable on the CPU's instruction set architecture, so it will not need to be compiled.

These two together, the NOP slide and the shellcode, form the basic string that is used in a heap spray attack. The purpose of the NOP slide is to more easily reach the shellcode. If the hacker only knows the general location of the injected shellcode, it will be difficult to execute it since they would need the exact line on the heap for it to run. The NOP slide makes it so that if the hacker uses any line from that slide, the shellcode will execute anyway, since the NOPs will just continue to the next line until they run out. The program will then run the shellcode which is stored immediately after the NOP slide.

```
void f3(int * p){
    *p = 12;
}

00000000004012ff <_Z2f3Pi>:
  4012ff:   55                      push   %rbp
  401300:   48 89 e5                mov    %rsp, %rbp
  401303:   48 89 7d f8             mov    %rdi, -0x8(%rbp)
  401307:   48 8b 45 f8             mov    -0x8(%rbp),%rax
  40130b:   c7 00 0c 00 00 00       movl   $0xc,(%rax)
  401311:   90                      nop
  401312:   5d                      pop    %rbp
  401313:   c3                      retq
```

Figure 1. A simple function and its shellcode equivalent

Shellcode is a very powerful tool that can be used by a hacker. Shellcode is the binary in the executables that the machine can understand. When a function in a language is written and compiled, binary is created. As such, a hacker can simply write a function in any language, compile it, and then object dump the executable to get working shellcode. Figure 1 shows an example of an extremely simple function, and its shellcode equivalent when compiled with g++. With the string of binaries starting from 55, 48, 89, e5, …, etc. all the way to c3, if just written to

the heap and ran, would operate as if the function f3 was called. Being able to write any function without having to require access to the source code pre-compilation is what makes heap spraying so effective.

```
#include <stdio.h>

unsigned char shellcode[] =
            "\x50\x53\x51\x52\x56\x57\x55\x89\xe5\x83\xec\x18\x31\xf6\x56\x6a"
            "\x63\x66\x68\x78\x65\x68\x57\x69\x6e\x45\x89\x65\xfc\x31\xf6\x64"
            "\x8b\x5e\x30\x8b\x5b\x0c\x8b\x5b\x14\x8b\x1b\x8b\x1b\x8b\x5b\x10"
            "\x89\x5d\xf8\x31\xc0\x8b\x43\x3c\x01\xd8\x8b\x40\x78\x01\xd8\x8b"
            "\x48\x24\x01\xd9\x89\x4d\xf4\x8b\x78\x20\x01\xdf\x89\x7d\xf0\x8b"
            "\x50\x1c\x01\xda\x89\x55\xec\x8b\x58\x14\x31\xc0\x8b\x55\xf8\x8b"
            "\x7d\xf0\x8b\x75\xfc\x31\xc9\xfc\x8b\x3c\x87\x01\xd7\x66\x83\xc1"
            "\x08\xf3\xa6\x74\x0a\x40\x39\xd8\x72\xe5\x83\xc4\x26\xeb\x41\x8b"
            "\x4d\xf4\x89\xd3\x8b\x55\xec\x66\x8b\x04\x41\x8b\x04\x82\x01\xd8"
            "\x31\xd2\x52\x68\x2e\x65\x78\x65\x68\x63\x61\x6c\x63\x68\x6d\x33"
            "\x32\x5c\x68\x79\x73\x74\x65\x68\x77\x73\x5c\x53\x68\x69\x6e\x64"
            "\x6f\x68\x43\x3a\x5c\x57\x89\xe6\x6a\x0a\x56\xff\xd0\x83\xc4\x46"
            "\x5d\x5f\x5e\x5a\x59\x5b\x58\xc3";

int main() {
    ((void(*)())shellcode)();
    return 0;
}
```

Figure 2. Shellcode to run calc.exe on Windows 10 machine

The name shellcode comes from the fact that it traditionally starts a command shell on the system being targeted, but any piece of injected malicious code can be considered shellcode. Once inside the command shell, a hacker can gain near complete control of a targeted system. They can gain information on the system and network, run new processes, download malware, launch more attacks on other machines, or even crash the system permanently. Figure 2 gives an example of how a hacker can translate these malicious activities into shellcode. The C program introduced by Dafchev creates an unsigned character array as the shellcode. This string was derived from x86 machine code instructions that opens a command shell and enters the command "calc" to run calc.exe. The function then creates a function pointer to the array, and uses the

notation “()” to run it, which will read in the first instruction in the shell code into the instruction pointer register.

Heap sprays are most commonly carried out by a script that allocates the NOP slide and shellcode string onto a heap. C library function malloc or the ‘new’ keyword in Python or Javascript are used to this end, which will add the injection dynamically to the heap as the script iterates. The hacker must then find an exploit in the browser or program to cause the heap to be read. If the heap spray is successful, a read will have a high probability of hitting somewhere on the NOP slide, which will result in the shellcode to execute. Heap locations are generally predetermined. Thus, it is easy to take advantage of writes being stored consecutively. For instance, a typed array in Javascript is guaranteed to be stored on the heap consecutively.

Heap spraying has a history dating back to the early 2000s. It was most commonly seen with exploits on Internet Explorer 6. There were common drive-by download attacks, where a malicious web page would put code into a machine’s browser’s memory. While the web page itself would be unable to run the user’s heap, it would be pretty common for for the now injected malicious heap code to be accidentally run. The concept was so simple that different viruses only needed very slight modifications to the code.

Since heap spraying is generally used to make use of a security exploit, it is not always easy to notice if a heap spray was used on a hacked device. Therefore, it is not often reported that a heap spray was used in an attack. However, in 2010 Google shocked the cyber security world by coming forward and admitting to being hacked. A presumably China-based hack was done targeting Chinese human rights activists. They successfully compromised two accounts, however they did not steal any personal information. They managed to get relatively useless information,

such as the date of the accounts' creation. Google later remarked in their blog post on the matter that the hack was "highly sophisticated and targeted". Google even went on to say that they know at least 20 other companies that did not come forward were also attacked. While only speculated, heap spraying is very likely to have been used in these attacks. (Mills, 2010)

Since these attacks, most exploits involving heap sprays have been patched out. Two such protective measures are The Nozzle Project and BuBBle. Nozzle was introduced in a paper by Microsoft in 2008. It provides runtime monitoring for attempts at heap sprays by comparing the heap to a concept they introduced known as "global heap health". If the heap does not look healthy, they will remove the heap spray. BuBBle takes a slightly different approach. When BuBBle detects that there is an attempt to spray the heap, it will just prevent the source code allocations of the NOP sled and shellcode altogether. They are currently implemented in various browsers, such as BuBBle in FireFox. (Prorootect, 2012)

Heap spraying is an effective code injection technique that is reasonably easy to implement. It demonstrates how powerful a NOP slide can be when used for arbitrary code execution. While today exploits attainable through a heap spray attack have been mostly patched out, it has seen a good deal of use up to about 2010. Software engineers, and most notably web developers, should understand the techniques involved in a heap spray attack in order to ensure the security of new softwares and systems.

Bibliography

Dafchev, I. (2017, September 26). Basics of Windows shellcode writing. Retrieved from https://idafchev.github.io/exploit/2017/09/26/writing_windows_shellcode.html

Exploit writing tutorial part 11 : Heap Spraying Demystified. (2017, November 06). Retrieved from https://www.corelan.be/index.php/2011/12/31/exploit-writing-tutorial-part-11-heap-spraying-demystified/

Mills, E. (2010, January 14). Behind the China attacks on Google (FAQ). Retrieved from https://www.cnet.com/news/behind-the-china-attacks-on-google-faq/

Prorootect. (2012, April 21). Heap Spray Protections. Retrieved from https://malwaretips.com/threads/heap-spray-protections.6580/