Dimitrios Kalemis

I am exactly like Jesus Christ: an atheist and anarchist against society and bad people with influence and power (judges, social workers, politicians, priests, and teachers).

The need for a POP POP RET instruction sequence

Posted on October 27, 2010

The purpose of this post is to explain the need for a POP POP RET instruction sequence. You often read or hear that exploit writers search for this instruction sequence because it is an essential part of their exploit. But why? What is so useful about this instruction sequence and how is it used? These are the questions I will try to answer.

For the following analysis I am going to assume that we have a 32-bit little-endian architecture. I am also going to leave out a lot of details, in order to focus on only some of the concepts. Also remember that ESP is the Extended Stack Pointer and EIP is the Extended Instruction Pointer.

OK, let us begin.

POP POP RET is a sequence of instructions needed in order to create SEH (Structured Exception Handler) exploits. The registers to which the popped values go are not important for the exploits to succeed, only the fact that ESP is moved towards higher addresses twice and then a RET is executed. Thus, either POP EAX, POP EBX, RET, or POP ECX, POP ECX, RET or POP EDX, POP EAX, RET (and so on) will do.

Each time a POP <register> occurs, ESP is moved towards higher addresses by one position (1 position = 4 bytes for a 32-bit architecture). Each time a RET occurs, the contents of the address ESP points at are put in EIP and executed (also ESP is moved, but this is not important here).

In order to create successful SEH exploits, the address of a POP POP RET sequence has to be found. This will enable the attacker to move ESP towards higher addresses twice and then transfer execution at the address where ESP points at.

All that remains now for me is to explain why we need to move ESP towards higher addresses twice and continue execution from the address that resides there in the stack.

It is beyond the scope of this post to explain in detail how Structured Exception Handling works. SEH is comprised for the most part by a linked list of records with each record corresponding to an exception handler. The first field of each record is the pointer to the next record and the second field is the address of the exception handler.

SEH exploits are based on the fact that the attacker can alter a portion of the stack (by a buffer overflow) and put values there that can misdirect the execution of the SEH handler after an exception is raised. After an exception, execution begins at the address that is pointed by the first SEH handler residing in the stack. The attacker can find this address and change it.

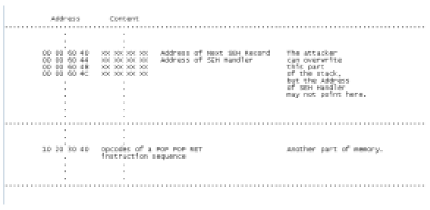
So the attacker could alter the stack by placing shellcode in the stack in order to alter the address that the SEH handler will begin execution from, to point to the s
successfully exploited the system? Well, it could be
safeguards in place that will keep such an exploit f
scope of this post to explain how SafeSEH protects

out more about SafeSEH with an Internet search.

Something more involved than the previous technique will succeed, though. The attacker has to overcome SafeSEH, and to do so, she has to find a module that has the following characteristics:
1) It is loaded in memory.
2) It has a POP POP RET instruction sequence somewhere.
3) It has not been built with SafeSEH safeguards.
If such a module exists, the attacker can use a buffer overflow to point the Address of the first SEH handler there. Then, the exploit will succeed.

So, let us see step-by-step how such a SEH exploit works. I provide an example below. The addresses are completely fictional. (Any similarity to actual addresses, cases, facts or people is purely coincidental!)

Before the buffer overflow, the memory looks like this:



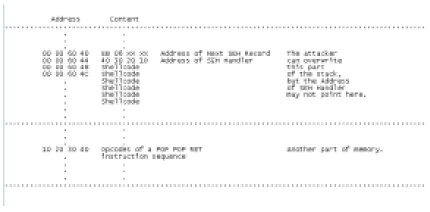
```
   Address      Content
......................................................................
      :            :
00 00 60 40   xx xx xx xx   Address of Next SEH Record   The attacker
00 00 60 44   xx xx xx xx   Address of SEH Handler       can overwrite
00 00 60 48   xx xx xx xx                                this part
00 00 60 4C   xx xx xx xx                                of the stack,
      :            :                                     but the Address
      :            :                                     of SEH Handler
      :            :                                     may not point here.
......................................................................
      :            :
10 20 30 40   opcodes of a POP POP RET                   Another part of memory.
      :       instruction sequence
      :            :
......................................................................
```

Figure 1: Before the buffer overflow

Figure 1: Before the buffer overflow.

The attacker will overflow a buffer (not shown here). The buffer resides in an address lower than 00 00 60 40. The attacker will overflow the buffer in order to appropriately set the stack. Also, please note that the module with the POP POP RET instruction sequence resides in "Another part of memory".

After the buffer overflow, the memory looks like this:



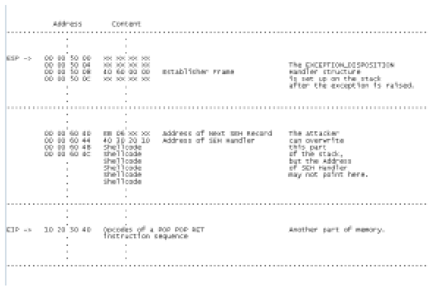
```
   Address      Content
......................................................................
      :            :
00 00 60 40   EB 06 xx xx   Address of Next SEH Record   The attacker
00 00 60 44   40 30 20 10   Address of SEH Handler       can overwrite
00 00 60 48   Shellcode                                  this part
00 00 60 4C   Shellcode                                  of the stack,
      :       Shellcode                                  but the Address
      :       Shellcode                                  of SEH Handler
      :       Shellcode                                  may not point here.
      :       Shellcode
      :            :
......................................................................
      :            :
10 20 30 40   opcodes of a POP POP RET                   Another part of memory.
      :       instruction sequence
      :            :
......................................................................
```

Figure 2: After the buffer overflow

Figure 2: After the buffer overflow.

The attacker has put shellcode beginning at 00 00 60 48. She has also altered the Address of Next SEH Record to contain the opcodes for a 6-byte jump to shellcode (EB 06 are the opcodes for the assembly instruction "JMP 06"). And she has also altered the Address of SEH Handler to point to the POP POP RET instruction sequence.

Now, an exception will be raised and after that, the memory looks like this:



```
          Address       Content
....................................................................
                 :         :
ESP ->    00 00 50 00   xx xx xx xx
          00 00 50 04   xx xx xx xx                          The EXCEPTION_DISPOSITION
          00 00 50 08   40 60 00 00   Establisher Frame      Handler structure
          00 00 50 0C   xx xx xx xx                          is set up on the stack
                 :         :                                 after the exception is raised.
                 :         :
....................................................................
                 :         :
          00 00 60 40   EB 06 xx xx   Address of Next SEH Record   The attacker
          00 00 60 44   40 30 20 10   Address of SEH Handler       can overwrite
          00 00 60 48   Shellcode                                  this part
          00 00 60 4C   Shellcode                                  of the stack,
                 :      Shellcode                                  but the Address
                 :      Shellcode                                  of SEH Handler
                 :      Shellcode                                  may not point here.
                 :      Shellcode
                 :         :
....................................................................
                 :         :
EIP ->    10 20 30 40   Opcodes of a POP POP RET             Another part of memory.
                 :      instruction sequence
                 :         :
....................................................................
```

Figure 3: After the exception is raised

Figure 3: After the exception is raised.

Because the system tries to handle the exception that is raised, it sets up the EXCEPTION_DISPOSITION Handler structure on the stack. This structure's Establisher Frame points to the first handler record. The first handler record contains the Address of Next SEH Record and the Address of SEH Handler and begins at 00 00 60 40. The system sets the stack in such a way that ESP points at the beginning of the the EXCEPTION_DISPOSITION Handler structure, here at 00 00 50 00.

From this figure, it is easy to understand the need for the POP POP RET instruction sequence. We move ESP two places towards higher addresses (with the two POPs) and we continue execution (with the RET) at the contents of the address ESP points to now.

Let us proceed step-by-step from the beginning:

After the buffer overflow, an exception is raised, so execution begins at the Address of SEH Handler. (This address belongs to the first handler record that is pointed by the establisher frame.) The buffer overflow set the Address of SEH Handler to be the address of the POP POP RET sequence (which is 10 20 30 40 in our example).

So, execution begins at address 10 20 30 40.
Before the first POP is executed, ESP points at 00 00 50 00.
After the first POP is executed, ESP points at 00 00 50 04.
After the second POP is executed, ESP points at 00 00 50 08.
After the RET, EIP points at 00 00 60 40,
which are the contents of the address 00 00 50 08 ESP pointed at.
So, execution will continue at 00 00 60 40.
So, the instruction EB 06 is executed, which is a 6-byte jump to 00 00 60 48,
the beginning of the shellcode. Thus we have a successful exploit.

It is obvious that if the attacker could overwrite the SEH Handler with whatever values she wished, she could overwrite it with the address of the shellcode, but since this is not possible due to SafeSEH safeguards, the SEH handler is made to point to the POP POP RET sequence.

It is also obvious that for the purposes of the exploit, Address of Next SEH Record is not used as an address, but contains code, a 6-byte jump over the SEH handler. The two bytes that follow EB 06 have no importance.

Some people believe that the way SEH exploits work is by faking and rethrowing the exception. I respectfully disagree. In my opinion, the original exception is neither "faked" nor "rethrown". I think I can prove that. It is easy. Look: After execution of the POP POP RET in
address of the Address of Next SEH Record, so the
rethrown, the Address of Next SEH Record would
transferred to, and not an executable jump code as

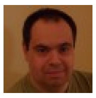
moment EIP starts off from the the Address of SEH Handler 10 20 30 40, until the shellcode is reached and executed.

Related

Create and test buffer overflow exploits
In "Security"

Stack-based buffer overflow proof of concept
In "Security"

How to lock a text file for reading and writing using C#
In "Development"



About Dimitrios Kalemis
I am a systems engineer specializing in Microsoft products and technologies. I am also an author. Please visit my blog to see the blog posts I have written, the books I have written and the applications I have created. I definitely recommend my blog posts under the category "Management", all my books and all my applications. I believe that you will find them interesting and useful. I am in the process of writing more blog posts and books, so please visit my blog from time to time to see what I come up with next. I am also active on other sites; links to those you can find in the "About me" page of my blog.
View all posts by Dimitrios Kalemis →

This entry was posted in Security. Bookmark the permalink.

Dimitrios Kalemis
Blog at WordPress.com. *Do Not Sell My Personal Information*