Table of contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTRODUCTION</td>
<td>2</td>
</tr>
<tr>
<td>REQUISITES</td>
<td>2</td>
</tr>
<tr>
<td>THE EXAMPLE BUG</td>
<td>3</td>
</tr>
<tr>
<td>DEBUGGING THE BUG</td>
<td>4</td>
</tr>
<tr>
<td>CONCLUSION</td>
<td>12</td>
</tr>
<tr>
<td>GREETINGS</td>
<td>13</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>13</td>
</tr>
</tbody>
</table>
Introduction

The intention to write this paper was when I’ve started working on an exploit of a daemon, which is described here later. After I was able to trigger the bug, I’ve noticed that a normal overflow would fail, because I had to fulfill too many requisites to get the exploit working. So I came to the conclusion that I have to use another technique. As I’ve heard a lot of SEH (Structured Exception Handler) hacking, I’ve started my browser and googled for a SEH paper, but failed. It seemed that nobody ever described it in a paper. I just found some examples of exploits using this technique, like the well known worm Code Red. As I was too lazy to debug that shit, I’ve started reversing the exception by myself and solved the trick very fast. The following paper will describe my lessons I’ve learned when I tried to get the exploit working.

Requisites

To understand all the shit I’ll try to explain you, you should fulfill the following requisites:

- X86 assembly [4]
- debugging with softice [3]
- basic knowledge of exploitation
- basics in structured exception handling [1] [2]

For all requisites I’ve listed some references at the bottom of this paper. And of course you should have installed a Serv-FTP Server 4.x and one of the Windows targets which I offer in the sample exploit [5], to get our stuff running. ;)

The example bug

The bug I’ve exploited is the latest SERV-U FTP-Server bug that was found by kkqq of the Superman Anti Security Team some weeks ago. To trigger the bug you’ll need a valid normal user-account on a serv-u server and a writeable directory. If the homedirectory isn’t writable, but another, we have to change to that directory. After you’ve logged onto the box, you just have to type:

    site chmod 666 <overlong-nonexisting-file>

and the daemon stops it’s activities. For further information on this bug, consult the 0x557 site [6] which can be found in the references. A sample exploit can be downloaded from the THC website [5]. Although it would be better if you first read the two SEH papers in the references, if you haven’t skills in that area, i just wanna give you a taste, why it is better to use the SEH technique, than the normal stack esp overwriting stuff here.

In the following you can see a memory map in the servudaemon.exe after I’ve send the bad data:

```
0013ac50c  :  550 /c:/<400 bytes of crap we filled in>
0013ac6a4  :  <SEH address in little endian format>
0013ac6fc  :  <ESP>
```

between SEH address and ESP are different pointers which are needed by the program flow. If the pointers are overwritten with data to unmapped memory we earn a page fault, the server dies and we have no control. :(

You see clearly that we had to fulfill to many requisites if we would plan to overwrite the ESP to get control over the daemon.

The easier and, you can imagine, more stable trick is to get control by overwriting the pointer of the Structured Exception Handler, which was installed by the servudaemon.exe itself and points to a routine, that is able to handle an unexpected error (exception). That’s exactly what we wanna do. We overwrite the SEH pointer with an address in a stable area, which then jumps into our shellcode and owns the box. The SEH gets triggered, because we’ve also overwritten some pointers which the program flow needs. As they point to unmapped memory, we get lucky. ;)

So before I’ve started coding the exploit I’ve just created a text-file to trigger the bug and piped the data to the daemon.

Sample servu.txt:

```
user lamer
pass test123
site chmod 666 <400 bytes crap><0xeb+0x06><2 bytes crap><SEH address in little endian format><shellcode>
```

If we have to change the directory, because our homedir isn’t writable, we have to substract the length of the pathname - <drive:\> from the 400 bytes of crap.
Debugging the bug

After creation of the servu.txt first join softice by pressing the hotkey STRG-D and enter the following commands:

addr servudaemon
px kiuserexceptiondispatcher

which enables us to find out very fast where the bug occurs exactly. After that, leave softice and fire up the named netcat command.

netcat command: nc <ftp-server host> 21 < servu.txt

If the user/password was ok for the servu-server and he munched the site chmod command, softice should popup now and showing us the following:
We can see that the breakpoint we’ve set, worked very well, because we triggered the bug successfully. The daemon wants to handle the exception now and jumps to API:

    ntdll!KiUserCallbackDispatcher

when you now trace (command is ‘t’) the first 4 commands you should stand right before:

    call 778ac57e

Now just type:

    d ebx

and cool… look at the first softice snapshot where the arrow points to. In little endian format we can see the address: 0x00419589

This is exactly where the bug occurred. Now just leave the debugger with the command ‘x’ and let the daemon crash. Start it again and when he’s in running mode, join softice and kill the old bpix kiuserexeptiondispatcher with the command ‘bc 0’ (should be the only breakpoint ;)

Now add some new commands to softice:

    addr servudaemon
    bpx 00419589

leave softice and fire up the netcat command again and you see the following now:
As you can see, we are in the ServuDaemon now at address 0x00419589 ;)
Now just trace once using ‘t’ and we’ll join the KiUserCallbackDispatcher again.
Trace again till call 778ac57a and enter this subcall now.
Now we are in the RTLTraceDatabaseEnumerate API which we pass best buy using the F10 key in softice till the call 778b98b0 and step into this one buy using the ‘t’ command.
As you can see, we are now in the RtlConvertUlongToLargeInteger+004F API. Just trace till that call ecx command but don’t enter the call yet.
When you are directly on the call ecx command just have a look at ecx!
It’s our overwritten SEH pointer which we placed at offset 404-407 in little endian format. In our case it’s at 0x74f92ac4, which is a very stable offset for all windows 2000 editions (SP0-SP4) and lays in c:\winnt\system32\ws2help.dll
The handler gets installed in the servudaemon.exe at 0x419124 (mov fs:[00000000],eax)
In this snapshot I’ve used a german w2k professional edition.
For english Servers the offset is at 0x75022ac4. Now enter the call ecx with ‘t’.
After we entered our overwritten SEH address we pop 2 registers and return.

Note to unix exploit writers:

On Windows it’s not a good idea to jump directly to the offset where the shellcode lies, because we are in a multi-threaded environment where offsets can change, which makes our exploit unstable. For that reason we’re jumping to a stable address first, ie. long time not changed dll, which is loaded when the exploited service runs.

So, what do you guess where we are now?
Yes, you are right! We are back in our self-created buffer where we placed the shellcode. Just leave the debugger now and you should be able to netcat to port 31337 where you get a shell. The sample exploit [5] on the THC website tries directly to connect to the port 31337.

**Conclusion**

As I’ve showed, exploitation of SEH is quite simple and a very elegant way to own a service. I hope everyone who wasn’t aware with this technique, has learned something useful.
Greetings

This time the greetings fly to: THC, Halvar, FX, Gera, Scut, Hendy, Random, Stealth, FtR, Dvorak, MaXX and especially g0dzilla who loaned me his digicam for the softice snapshots.

References


-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: Weitere Infos: siehe http://www.gnupg.org

mQGiBDzw5yMRBACGJIo25Bbb6mBkP2+qwde0CtvCm5uJGdXWOW8BqQwDHkoO4h
sdouA+0JtdFtFqriCzhZWbspNsWePPOA8WvG3f6qIuQid6A2J1h+bNw0WEqx9t
8TkooEVS3L3wIDCig3cEtmvAlj0C9g4jj5B/QwHJYrWNFoAxc2S1IxxwcG8Wk9
Lawhv+Xnc6c6/w5O8Ipn6d/S24/fqFqFtvN22Jd6nCQ75A64fB7mH7zUSVPY9
BetYXm4GhcHx7zf0hAbJQW0nNymYGiFv99UvO9G5nG+y9j9l6l6oQ777dIZUEpL
F5FeVeFyrtDGYmBhG9hwz83C9n9gqPzz1xYThDxkvlTMleNRC3GIJCRpF6b
a7u4a/9nCqRx2HbRkaj39zurC6Yz289fSimGzu7PTVw3bxDBo8ig4CyHcjmHe+j
DResUKGgdyE+/d07ofbFEODQjgdaX1mmsw34pCILK0yrdQMtdbgDyDPjw5KGHX
G0hrHv/Uhigk3W6nC43ZvPzbh3HVF01u8jDTQRgwaRyldGc45dthQkam90ob551GN5
YmVycHVuayA5A9obm55yNbrQGdteC5uZXQ+IFcEExECAcEFajZW5YMFCwcKAwQD
FQMCAxYCAQIQxGAKCRD3cEGutqj/MW7AJ90SnrB+V0MgPfVOT4edv7C++RNhwCf
byT/qkEswaxsF8g4HeX33fPe25Ag0EPPDnrRAIALdcN8E2Z8ZUa4p8fjwXNO
ip6GOANUN5XlpmvC9e5ErFk+NM2ArB707QJFlkmKV8voPnj4IPUyUyGlGeOhzj
r865l6p88Rsrv05JKTW+rizAmd8lJ1B4YqlLzmt9OuzoOeACjq3CuQtPMYrNuOkPL9
nX51EgnLnYaUYAkysAhYLRhur3/maNdjtn2T63MoJauA4B4PvKveg3Fs1pAp5mj
9yyfu6Gz0Wn8Xp5vSd2W3PUBJ+Q7J3On35byebIKiuGsti6Y5L0Z5DIW2rvePz9g
cR5Q06j+mxAooTUMBBJwMmXjhm5nTGr50X8mbP+73MGrhtsRR+WJ+EWSLQ8A
AvcH/iqRCMmPB/yIlMheFEPUMNBsZOJ+yV3KpunUNLbAPht7E2ZmEpgdL3tjHTC
vZ06k4O11BkodmdfKChEwzlhWeF8P3a+wgW2lCm6c7F6Pef9kPXD43uIWTLL4RF
hcpMyrs45B2uht7aE3Pc08gbsnWwaj87Stwab+ezOmngmrRvZKnYREVW1RRHRSI3I6
C4rex3DuUHFjFdniEXiw97h1GyOPVDx6sICk2SunfzQAEZC2n7/lQwPdw6ZAbAf
Ay9WmTqPBFd2fApUtz1h8epS6Mb6a5R2BDQjilNh2npFQnZlih8ehjvdOc67dKyay
R/g0EpU0tIVvAeacolCJJE8h30IRqGqYEQIAbgUCPPDrQAKCRD3cEGutqj/NuP
AJ979Idhs926vxxhRA5Y8G0hLyDAwCgo8eWQW1Y7+oQfWbhG8XCzei4oAIo=
=2B7h

-----END PGP PUBLIC KEY BLOCK-----