

Macro-Reliability in Win32 Exploits

“A la conquete du monde...”

Kostya Kortchinsky

<http://www.immunityinc.com/>



Agenda

- Problems with large scale exploitation
- Immunity's Solutions
 - Common Addresses
 - Remote Language Fingerprinting
- The Future

Problems in Large Scale Remote Exploitation

- Targets are not homogeneous
- Targets have host protection layers
- Targets have network protection layers
- Targets vary over time

Windows Machine Types

- Targeting a remote exploit requires:
 - Major/Minor versions
 - Service Packs
 - Patches
 - Configurations
 - Language Packs
 - Software version and configuration
 - Networking conditions between attacker and target
 - Host protections on target

Exploits and Magic Numbers

- Most exploits contain a list of “magic numbers” that help them target remote machines
 - shellcode offsets
 - return addresses
 - writable addresses
 - etc
- Each magic number decreases the reliability of the exploit in the wild

Minimizing Magic Numbers

- Two obvious approaches
 - Find common addresses that are the same across all your target types
 - Find a way to do fine-grained fingerprinting on your targets to accurately determine their magic numbers
- Hardest and best way
 - Rewrite the exploit to not need magic numbers at all

Common Addresses

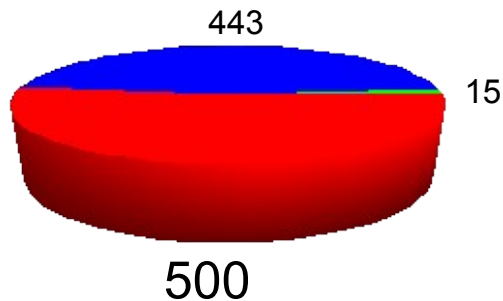
- Avoid fingerprinting as much as possible
 - Fingerprinting is usually noisy
 - SP fingerprinting is not that reliable
 - Usually using MSRPC interfaces
 - AFAIK, localization fingerprinting is pretty nonexistent
- Major Windows version fingerprinting is quite reliable
 - Some work was already done on SP independent return addresses
- “Universal address” often means English only

Naïve Approach


- Try and find addresses as independent as possible of the targets
 - In DLLs: image base address usually changes with language pack
 - In EXEs: image base doesn't change much
 - In EXEs and DLLs: different versions usually means different offsets relatively to image base
- DLLs with same version and same image base might provide common return addresses...
 - Small C program: `dllvers.c`

Some Results

Windows 2000¹ \system32 DLLs



¹English, Japanese, Italian, Dutch, German, Spanish, Chinese, Russian, French SP0 to SP4 up to date

 Common accross Language
 Common accross SP
 Others

- Common DLLs

admparse.dll	5.0.2920.0	0x80000000
bootvid.dll	5.0.2172.1	0x80010000
dbmsadsn.dll	1999.10.20.0	0x42bd0000
dbmssocn.dll	1999.10.20.0	0x73330000
dbmsspxn.dll	1999.10.20.0	0x42be0000
gpkcsp.dll	5.0.2134.1	0x80000000
mcdsrv32.dll	5.0.2160.1	0x80010000
msvcirt.dll	6.1.8637.0	0x780a0000
msvcp50.dll	5.0.0.7051	0x780c0000
rtipxmib.dll	5.0.2168.1	0xd0000000
slbcsp.dll	5.0.2134.1	0x80000000
slbkygen.dll	5.0.2144.1	0x80000000
sqlwid.dll	1999.10.20.0	0x412f0000
vcdex.dll	5.0.2134.1	0x0ffb0000
vdmredir.dll	5.0.2134.1	0x0ffa0000

Pretty useless!

IMMUNITY 

In Memory

- Not only DLLs and EXEs and memory
 - Stacks
 - Heaps
 - File mappings
 - PEB, TEBs
 - Various different kinds of sections...
- Do not only stick to EXEs or DLLs to search for opcodes, look into the whole memory space
 - Small C program: `dumpop.c`

NLS File Mappings

- Several NLS files are mapped by default by Windows before the process even starts
 - unicode.nls
 - locale.nls
 - sortkey.nls
 - sorttbls.nls
- Others can be loaded at runtime depending on the locale used
 - ctype.nls for example
- Mapping base address is (almost) fixed for a given binary on the same major version of Windows

NLS File Mappings (cont.)

- Mapping base address will depend on previously allocated pages:
 - Stack of main thread
 - Based on `SizeOfStackReserve` parameter in PE header
 - Imported DLLs
 - Based on their image base address
- Include a lot of `jmp reg`, `call reg`, `push reg` & `ret`
- Haven't changed since Windows NT 4.0
- Contain 1 NULL byte, not executable
 - Still can be used quite efficiently

Memory Mapping Example

Address	Size	Owner	Section	Contains	Type	Access	Initial	Mapped as
00010000	00001000				Priv	RW	RW	
00020000	00001000				Priv	RW	RW	
00120000	00001000				Priv	RW	Gua: RW	
0012E000	00002000			stack of na	Priv	RW	Gua: RW	
00130000	00003000				Map	R	R	
00140000	00003000				Priv	RW	RW	
00240000	00006000				Priv	RW	RW	
00250000	00003000				Map	RW	RW	
00260000	00016000				Map	R	R	\\Device\HarddiskVolume1\WINDOWS\system32\unicode.nls
00280000	00030000				Map	R	R	\\Device\HarddiskVolume1\WINDOWS\system32\locale.nls
002C0000	00041000				Map	R	R	\\Device\HarddiskVolume1\WINDOWS\system32\sortkey.nls
00310000	00006000				Map	R	R	\\Device\HarddiskVolume1\WINDOWS\system32\sorttbls.nls
00400000	00001000	test		PE header	Imag	R	RWE	
00401000	00007000	test	.text	code	Imag	R	RWE	
00408000	00002000	test	.rdata	imports	Imag	R	RWE	
0040A000	00002000	test	.data	data	Imag	R	RWE	
7C800000	00001000	kernel32		PE header	Imag	R	RWE	
7C801000	00002000	kernel32	.text	code, import:	Imag	R	RWE	
7C803000	00005000	kernel32	.data	data	Imag	R	RWE	
7C808000	00006000	kernel32	.rsrc	resources	Imag	R	RWE	
7C8EE000	00006000	kernel32	.reloc	relocations	Imag	R	RWE	
7C900000	00001000	ntdll		PE header	Imag	R	RWE	
7C901000	00007000	ntdll	.text	code, export:	Imag	R	RWE	
7C97C000	00005000	ntdll	.data	data	Imag	R	RWE	
7C981000	00002000	ntdll	.rsrc	resources	Imag	R	RWE	
7C9AD000	00003000	ntdll	.reloc	relocations	Imag	R	RWE	
7F6F0000	00007000				Map	R E	R E	
7FFB0000	000024000				Map	R	R	
7FFD6000	00001000				Priv	RW	RW	
7FFDF000	00001000			data block	Priv	RW	RW	
7FFE0000	00001000				Priv	R	R	

Remote options

- **Passive**
 - SIGINT can tell you a lot of things about a machine, including language strings
 - This is mostly useful for client-side attacks
- **Active**
 - Scanning may correlate your SIGINT data with a particular machine after it moves IP addresses
 - Various services on the remote machine may offer “localized” strings which can be used for language detection

Determining Language Pack Remotely

- Microsoft Windows does not offer a remote and anonymous way to correctly determine the language pack of a Windows install
- The applied language pack changes offsets and base addresses within DLLs which affect our exploits
- Some vulnerabilities and/or exploits are only effective on certain languages
 - [MS06-009](#): Korean Input Method Editor
 - [MS07-001](#): Brazilian Portuguese Grammar Checker

Why care so much about language pack?

- Most research on exploit reliability assumes English Windows
- But any large company has branches in places where the native language is not English
- Consultants come from all countries and place their non-English Windows laptops onto corporate networks

The Same Path Principle

- When exploiting a vulnerability we want to reduce the number of services and ports used
 - All services might not be running
 - All ports might not be opened
- Try and find as many ways as possible to remotely fingerprint a Windows language
 - MSRPC
 - SNMP
 - Web browsers
 - ...

MSRPC Localization using Shares

- Works by matching “remark” unicode field of a SHARE_INFO_1 structure returned by the NetShareEnum() API
 - Interface 4b324fc8-1670-01d3-1278-5a47bf6ee188 v3.0, opnum 15 in services.exe (2000)
 - Endpoints on ncacn_np, ncadg_ip_udp (old SP)
- Needs IPC\$ and/or C\$ share to exist
 - Usually better be if exploiting a RPC bug
- Will work anonymously against NT 4.0, 2000, XP < SP2 and 2003 SP0

Shares Results

- Uniquely matched

- French
- Spanish
- Russian
- German
- Dutch
- Polish
- Simplified Chinese
- Traditional Chinese
- Turkish
- Hungarian
- Czech
- Norwegian
- Swedish
- Greek
- Danish
- Finnish

- “Collisions”

- Common (no translation)

- English
- Arabic
- Hebrew
- Japanese
- Korean

- On IPC\$ share

- Italian
- Portuguese
- Brazilian

- On C\$ share (or any disk)

- Portuguese
- Brazilian

MSRPC Localization using Users

- List users on a system using LsaLookupSids() API by bruteforcing SIDs, match the default ones that are localization dependent
 - Interface 12345778-1234-abcd-ef00-0123456789ab v0.0, opnum 57
 - Endpoints on ncacn_np
- Will work anonymously against NT 4.0 and 2000
 - Useful in some case to refine previous technique results
- Works against XP SP1a with fake credentials if a Share has been setup

MSRPC Localization using Print Providers

- Best of the RPC methods, unique to CANVAS
- Works by matching the “comment” unicode field of a `PRINTER_INFO_1` structure returned by the `EnumPrinters()` API
 - API itself doesn't support remote listing of Print Providers
- Needs access to the `spoolsv.exe` service
 - Interface 12345678-1234-abcd-ef00-0123456789ab v1.0, opnum 0
 - Usually through `ncacn_np:\PIPE\spoolss`
- Works anonymously against up to and including XP SP2!
 - No access on 2003 unless configured as a Printer Server

Print Providers

- Windows based clients and servers have 3 print providers by default
 - `win32spl.dll` comment string is localized
- 3rd party software can install their own print provider
- Side note: multiple vulnerabilities in the recent past, PP enumeration is interesting for that too
 - [MS05-043](#): Heap overflow in `win32spl.dll`
 - [Novell TID #3125538](#): Stack overflow in `nwspool.dll`
 - [CTX111686](#): Stack overflow in `cpprov.dll`
 - And more...

Print Providers Results

- Uniquely matched

- French
- Spanish
- Russian
- German
- Dutch
- Polish
- Simplified Chinese
- Traditional Chinese
- Turkish
- Hungarian
- Czech
- Norwegian
- Swedish
- Greek
- Danish
- Finnish
- Japanese
- Korean
- Portuguese
- Italian
- Brazilian

- “Collisions”

- English
- Arabic
- Hebrew
- Probably due to lazy translators

SNMP Localization

- No such thing as a Windows Language OID :-(
 - Well at least I haven't found one
 - SNMPv2-MIB::sysLocation.0 is pretty useless
- Hopefully, Windows provides a list of installed software accessible from the public community
 - HOST-RESOURCES-MIB::hrSWInstalledName.*
 - Hopefully the term “Hotfix” is localized
 - “Correctif” in French, “Revisión” in Spanish
- Needs at least some hotfixes installed
 - No hotfix usually means no trouble for us though :>

IIS & IE Localization

- IIS is not very talkative about its localization
- 40x errors are localized
 - 404 error string
 - 404 pages
 - If customized, several other 40x pages to try
- Localization through IE might be useful for client-side exploits
 - Accept-Language header can give an hint
 - Nowadays heap-spray provides a mean to disregard this

Configuration Options

- Of we can't get the localization of the remote target:
 - Assume it is English or another particular localization
 - Don't run the exploit
 - Assume the target has the same localization of the nearest neighbor

CANVAS Example

The screenshot shows the Immunity CANVAS interface. The top bar displays the current callback IP as 10.10.11.1. A table lists various exploits, with 'ms04_011' (Microsoft Windows LsaSs RPC Overflow) selected. The Node Tree on the right shows the target host 10.10.11.129. The CANVAS Log shows the following output:

```
Gussed languages: ['German']  
Get Remote Language found: ['German']  
Found os of 10.10.11.129 as Windows 2000 German
```

The bottom status bar shows a successful exploit: 'Microsoft Windows LsaSs RPC Overflow attacking 10.10.11.129:445 (succeeded!)'.

The dialog box for the Microsoft Windows LsaSs RPC Overflow exploit is shown. The host is set to 10.10.11.129. The 'Autoversioning' option is selected. The 'Versions' list includes:

- Windows 2000 SP0-SP4 English
- Windows 2000 SP0-SP4 French, Simplified Chinese
- Windows 2000 SP0-SP4 Japanese
- Windows 2000 SP0-SP4 German
- Windows 2000 SP0-SP4 Dutch, Italian, Spanish
- Windows XP SP0-SP1a

The 'Covertness Bar' is set to 'As Covert As Possible'.

Some CANVAS Exploits

Exploit	Vulnerability	Method	Target
ms01_023	IPP ISAPI Overflow	NLS mapping	2000 SP0-SP1
ms01_033	Index Server ISAPI Overflow	NLS mapping	2000 SP0-SP1
ms03_001	RPC Locator Overflow	NLS mapping	NT 4.0 SP6a, 2000 SP0-SP3
ms03_022	Media Services ISAPI Overflow	NLS mapping	2000 SP0-SP4
ms03_026	RPC Interface Overflow	NLS mapping	NT 4.0 SP6a, 2000 SP0-SP4, XP SP0-SP1a, 2003 SP0
ms03_049	WksSvc Overflow	ws2help.dll address based on localization	2000 SP0-SP4, XP SP0-SP1a
ms04_011	LsaSs Overflow	ws2help.dll address based on localization	2000 SP0-SP4, XP SP0-SP1a
ms04_031	NetDDE RPC Overflow	NLS mapping	2000 SP0-SP4, XP SP0-SP1a
ms05_039	UPNP RPC Overflow	NLS mapping	NT 4.0 SP6a, 2000 SP0-SP4, XP SP0-SP1a
ms06_066	Netware Service Overflow	NLS mapping	2000 SP0-SP4, XP SP0-SP1a
ms06_070	WksSvc Overflow	NLS mapping	2000 SP0-SP4

Heap Overflows

- Usually needs a function pointer overwritten
 - UEF should be considered last resort since depending on SP and language
 - PEB lock functions are at a fixed location but might not be triggered when we want
 - To avoid an exception, we might want to find a writable location
 - Might be in .data section of a binary
- Memory leaks will help a lot

MSRPC Pointer Leak

- MIDL [unique] attribute leaks a pointer in the target process memory space on the wire if combined with [out]
 - <http://msdn2.microsoft.com/en-us/library/aa367294.2>

- Example

```
- long _RpcEnumPrinters (  
    [in] long arg_1,  
    [in][unique][string] wchar_t * arg_2,  
    [in] long arg_3,  
    [in, out][unique][size_is(arg_5)] char * arg_4,  
    [in] long arg_5,  
    [out] long * arg_6,  
    [out] long * arg_7  
);
```

Wireshark Capture

(Untitled) - Wireshark

File Edit View Go Capture Analyze Statistics Help

Filter: Expression... Clear Apply

No.	Time	Source	Destination	Protocol	Info
26	0.129575	10.10.11.1	10.10.11.134	SMB	Read AndX Request, FID: 0x4000, 16 bytes at offset 0
27	0.129655	10.10.11.134	10.10.11.1	SMB	Read AndX Response, FID: 0x4000, 16 bytes
28	0.131780	10.10.11.1	10.10.11.134	SMB	Read AndX Request, FID: 0x4000, 588 bytes at offset 0
29	0.131851	10.10.11.134	10.10.11.1	SP00LSS	EnumPrinters response
30	0.168794	10.10.11.1	10.10.11.134	TCP	37389 > microsoft-ds [ACK] Seq=1672 Ack=1601 Win=9312 Len=0 TSV=2438450 TSER=57617
31	0.182789	10.10.11.1	10.10.11.134	TCP	37389 > microsoft-ds [FIN, ACK] Seq=1672 Ack=1601 Win=9312 Len=0 TSV=2438451 TSER=57617
32	0.183532	10.10.11.134	10.10.11.1	TCP	microsoft-ds > 37389 [FIN, ACK] Seq=1601 Ack=1673 Win=17394 Len=0 TSV=57618 TSER=2438451

Ethernet II, Src: vmware_e1.e1.e1.co (00:0c:29:e1:e1.co), Dst: vmware_c0.00.00 (00:50:56:c0:00:00)

Internet Protocol, Src: 10.10.11.134 (10.10.11.134), Dst: 10.10.11.1 (10.10.11.1)

Transmission Control Protocol, Src Port: microsoft-ds (445), Dst Port: 37389 (37389), Seq: 949, Ack: 1672, Len: 652

NetBIOS Session Service

SMB (Server Message Block Protocol)

DCE RPC Response, Fragment: Single, FragLen: 604, Call: 2 Ctx: 0, [Req: #24]

Microsoft Spool Subsystem, EnumPrinters

Operation: EnumPrinters (0)

[Request in frame: 24]

buffer

Referent ID: 0x00097e34

Buffer size: 558

Buffer data: 00800100FA010000BA0100008401000000C0010046010000...

Needed: 558

```
0000 05 00 02 03 10 00 00 00 3c 02 00 00 02 00 00 00 ..... \.....
0010 44 02 00 00 00 00 00 00 34 7e 09 00 0e 02 00 00 D..... 4~.....
0020 00 80 01 00 fa 01 00 00 0a 01 00 00 04 01 00 00 ....F.....
0030 00 c0 01 00 46 01 00 00 f8 00 00 00 c0 00 00 00 ...F.....
0040 00 80 01 01 76 00 00 00 3c 00 00 00 10 00 00 00 ..... <.....
0050 49 00 6e 00 74 00 65 00 72 00 6e 00 65 00 74 00 I.n.t.e.r.n.e.t.
0060 20 00 55 00 52 00 4c 00 20 00 50 00 72 00 69 00 .U.R.L. .P.r.i.
0070 6e 00 74 00 65 00 72 00 73 00 00 00 57 00 69 00 n.t.e.r. s..W.i.
0080 6e 00 64 00 6f 00 77 00 73 00 20 00 4e 00 54 00 n.d.o.w. s..N.T.
0090 20 00 49 00 6e 00 74 00 65 00 72 00 6e 00 65 00 .I.n.t.e.r.n.e.
00a0 74 00 20 00 50 00 72 00 6f 00 76 00 69 00 64 00 t..P.r.o.v.i.d.
00b0 65 00 72 00 00 00 57 00 69 00 6e 00 64 00 6f 00 e.r..W. i.n.d.o.
00c0 77 00 73 00 20 00 4e 00 54 00 20 00 49 00 6e 00 w.s. .N. T. .I.n.
00d0 74 00 65 00 72 00 6e 00 65 00 74 00 70 00 50 00 t e r n e t P
```

Frame (718 bytes) DCERPC over SMB (604 bytes)

Referent ID for this NDR encoded pointer (dcerpc.referent_id), 4 bytes

P: 62 D: 62 M: 0 Drops: 0



MSRPC Pointer Leak (cont.)

- Ideal use:
 - Populate target memory with an entry of your own using a 1st RPC function
 - Retrieve the entry using a 2nd RPC function with the MSRPC Pointer Leak
 - You have the pointer to your entry!
- Doesn't happen that often:
 - [MS05-010](#): License Logging Service overflow
- Will give a good idea of the base address of the heap anyway

HEROES: MS06-070

- Description of Vulnerability

- Pseudo-code

```
array=(unsigned int *)malloc(n*sizeof(unsigned int *))
//initialization and various operations on array
...
for (i=0;condition==true;i++) {
    free(array[i]);
    //process some more, update condition
    ...
}
```

- We can influence condition based on the content of the SNMP request, thus freeing pointers outside of array

HEROES: MS06-070 (cont.)

- Several issues arise when attempting to exploit this vulnerability:
 - How can we control the pointer that will be freed?
 - Given pointer control, what do we actually want to free?
 - Once we get our Write4 primitive, what will we overwrite?
 - How do we leverage our Write4 primitive into full blown code execution?

HEROES: MS06-070 (cont.)

- Exploitation stages
 - Crash
 - Find information leak
 - Get working on a language dependent way
 - Only writable function pointers are in .data section of snmpapi.dll:
 - Image base depends on language
 - Offset relative to image base depends on version
 - Get working with special OID for global lock function pointer
 - Using the PEB lock routines

Other similar vulnerabilities

- VERDE
 - Arbitrary Free in DHCP MSRPC Service on Windows 2000 SP2/SP3
- DTLOGIN
 - Arbitrary Free in XDMCP service of dtlogin on Solaris (or other commercial Unixes)

Networking Issues

- Attacking an entire class-B you will find many networking setups
 - Port forwarding
 - Load balancing
 - NAT (perhaps both the attacker and target are behind different NATs)
 - Firewalls with ex-filtration filters
 - Poorly configured routers
- Each of these setups forces complications on your exploit efforts

Defeating network speed-bumps

- Accurate network reconnaissance is hugely expensive in memory, network traffic, time, and technology
- Ideally the solution is to re-use the socket we came in on
- Alternately, we could use a shellcode that did not require socket connections at all, such as an HTTP downloader shellcode
 - But this does require SOME network connectivity, and our target may be in a strict DMZ

Socket Stealing on Windows

- Windows socket stealing is difficult
 - Common technique is to call `getpeername()` on all handles and check to see which ones come from our host and/or source port
 - This fails to handle NAT and other networking setups properly
 - `Getpeername` will freeze when called on named pipes and other handles, causing the shellcode to sometimes fail
 - Immunity's 3rd generation Windows socket stealing shellcode launches one thread per handle and sends a GOOO to the client
 - This handshake ensures proper operation over all network types

Socket Stealing on Windows (cont)

- Sometimes stealing a socket is not possible
 - MSRPC calls typically go through the SMB stack and no socket is available
 - In this case a “bind-to-an-MSRPC function” shellcode is useful
 - Overflows are often in a different process than the socket, for example, ISAPIs

ISAPI stealing

- Immunity's ISAPI-GO-Code will search the stack for the currently used ISAPI structure
- This contains a Read and Write function, which can be used to send and receive data from the Inetinfo.exe process
- Using this code allows exploits to steal SSL sockets, even though the process being exploited is not the Inetinfo process!

The Future

- Windows XP SP2
 - Remote language fingerprinting is – I think – absolutely necessary to work out DEP issues
 - Most addresses are language-dependent
 - Microsoft Netware Service stack overflow
 - Novell Netware Client for Windows PP stack overflow
- Vista
 - Even more languages supported!
- OS X/Linux
 - Getting more important all the time!

Conclusion

- Attacking large scale global networks can be done effectively by spending a fairly reasonable amount of time doing effective fingerprinting
- Questions?