Nathan Rittenhouse – nathan_@mit.edu

MS08-067 – NETPRPATHCANONICALIZE

Importance

- Used for the Downadup worm
- Affects almost every version of NT based Windows systems

Background on DCERPC

- Two main binary RPC protocols – SunRPC/ONCRPC and DCERPC
  - Both are open specs
- DCERPC got adopted by Microsoft
- SunRPC got adopted on Unix based platforms

DCERPC / COM / DCOM

- Reason for COM's existence
  - Decouple interfaces from implementations
- Example
  - Take two different DLLs written in C++ with different compilers
  - Have one attempt to import/create an instance of a class in the other
  - Attempt to call a member function of that class
  - Look for the Dr. Watson screen

What COM attempts to fix

- There isn't a standard C++ calling convention
- There aren't standard calling conventions between multiple languages
- COM intends to fix that
  - If a language supports COM, any COM object created can be used by any other language that supports COM

DCERPC's role

- DCERPC is the transport mechanism for DCOM
- DCOM is Distributed COM
- The interface and implementation are *so* decoupled, function calls over a network are possible
  - Without the programmer having to do anything different
- DCERPC is the network transport protocol

DCERPC internals

- Microsoft uses SMB as a transport mechanism for DCERPC
  - SMB can run on top of UDP/TCP
  - Originally chosen because SMB was versatile and could run on top of many different protocols
    - Including Microsoft's NetBEUI
  - Also provides authentication, allows remote user impersonation

Comparison of SunRPC and DCERPC

- Closed source vs. open source difference
- Both have languages for specifying definitions
- SunRPC apps have interfaces generated via 'rpcgen,' which simply spits out some C files
  - Does marshalling INSIDE of the target app
  - XDR encoding method

DCERPC

- DCERPC based apps are easier to RE if there's only binary
  - Format strings which describe the interface AND its member functions, arguments, data types are embedded IN THE BINARY
  - RPCRT4.DLL parses these and marshals data to interfaces/functions accordingly
- Uses NDR marshalling method

Why is DCERPC awesome?

- A LOT of the work of figuring out an application's interface is taken out
  - Enables us to write fuzzers that can extract an IDL from another app and talk to it correctly
- Many assume since they don't give out the IDL for their app, no one can talk to it
  - Wrong ☺

Steps to RE a Microsoft Patch

- Look at the security bulletin: http://www.microsoft.com/technet/security/Bulletin/MS08-067.mspx
- Look at the KB article: http://support.microsoft.com/?kbid=958644
- Note which files are patched

Notes

- "Server service"
  - srvsvc.dll
- "RPC request"
  - Means there's a vulnerable RPC function

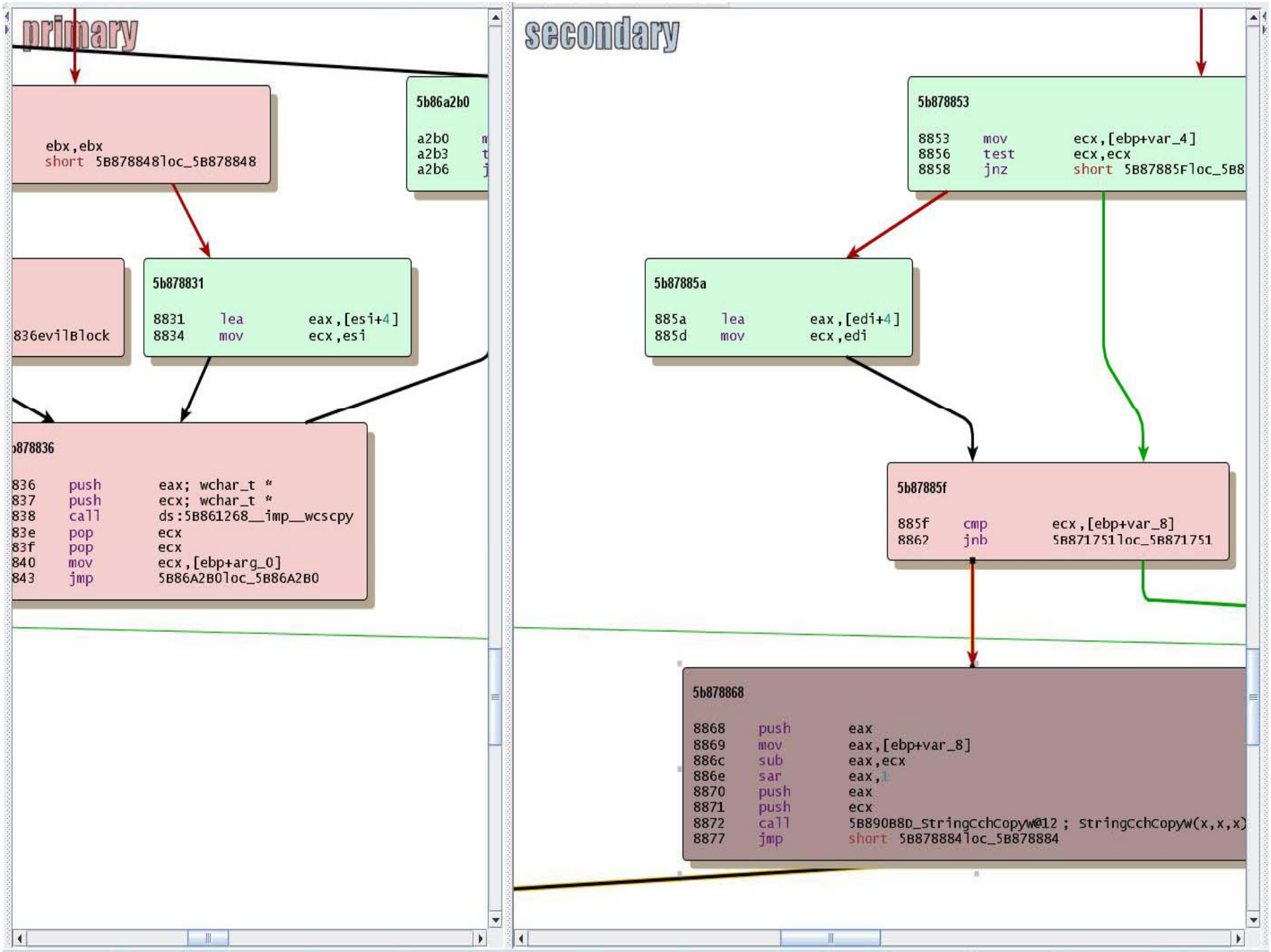
Bindiffing

- Use Bindiff or Binary Diffing Studio
  - Bindiff is MUCH nicer
- We will cover Bindiff, since BDS has caused me lots of pain in the past

Bindiffing

- Load both patched and unpatched versions into IDA
- Tell Bindiff to run its algorithm against the two
  - This may take some time


primary
secondary


primary
ebx,ebx
short 5B878848loc_5B878848
5b86a2b0
a2b0
a2b3
a2b6
836evilBlock
5b878831
8831 lea eax,[esi+4]
8834 mov ecx,esi
878836
836 push eax; wchar_t *
837 push ecx; wchar_t *
838 call ds:5B861268__imp__wcscpy
83e pop ecx
83f pop ecx
840 mov ecx,[ebp+arg_0]
843 jmp 5B86A2B0loc_5B86A2B0
secondary
5b878853
8853 mov ecx,[ebp+var_4]
8856 test ecx,ecx
8858 jnz short 5B87885Floc_5B8
5b87885a
885a lea eax,[edi+4]
885d mov ecx,edi
5b87885f
885f cmp ecx,[ebp+var_8]
8862 jnb 5B871751loc_5B871751
5b878868
8868 push eax
8869 mov eax,[ebp+var_8]
886c sub eax,ecx
886e sar eax,1
8870 push eax
8871 push ecx
8872 call 5B890B8D_StringCchCopyW@12 ; StringCchCopyW(x,x,x)
8877 jmp short 5B878884loc_5B878884

Vulnerable control flow

- Notice the addition of a ‘jnb’ instruction before a string copy
- Notice wcscpy -> StringCchCopy

Finding the interface

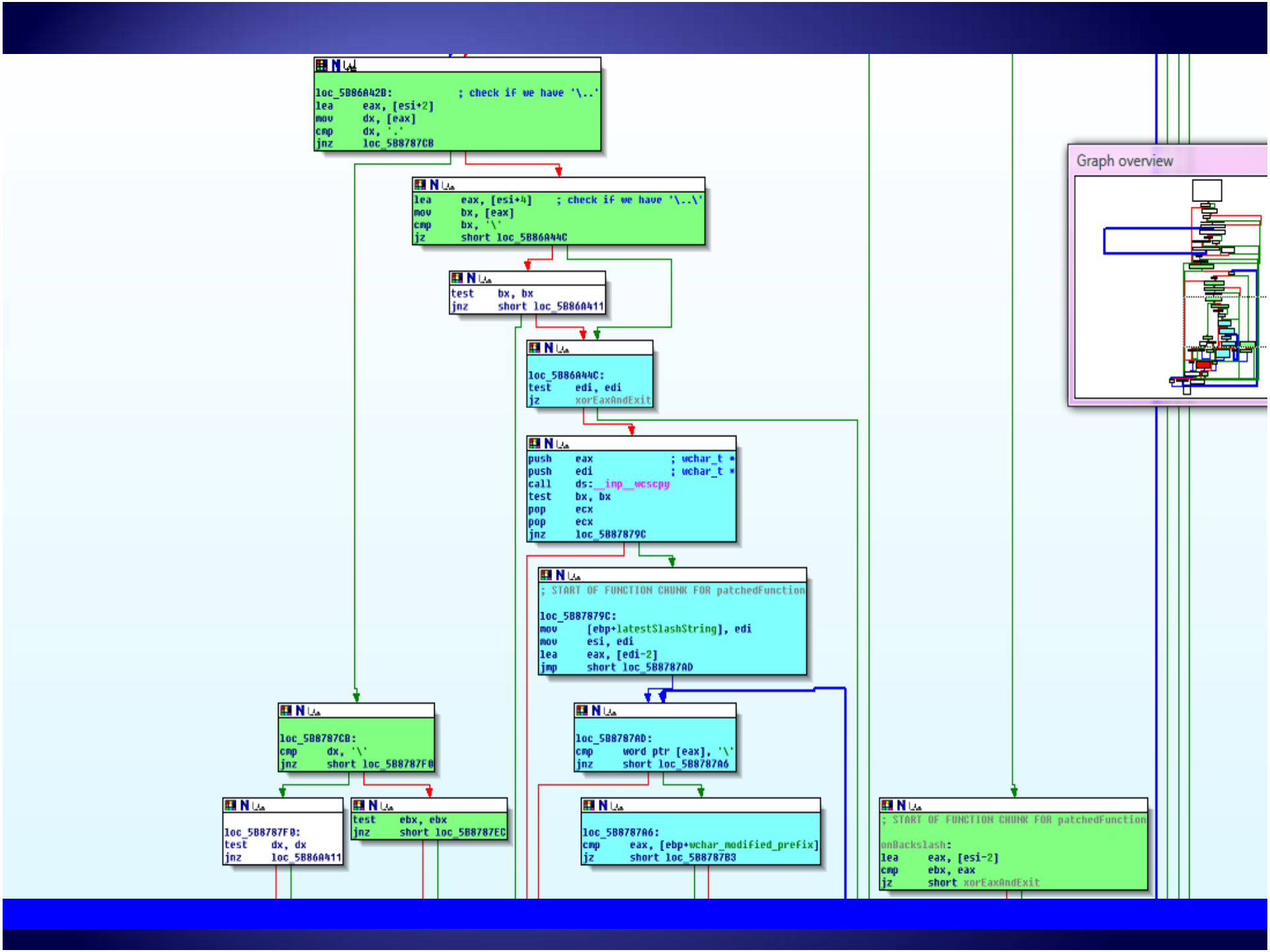
- Load both srvsvc.dll and netapi32.dll into IDA
- Run mIDA
- Notice no RPC interfaces found in netapi32.dll
  - All are contained in srvsvc.dll

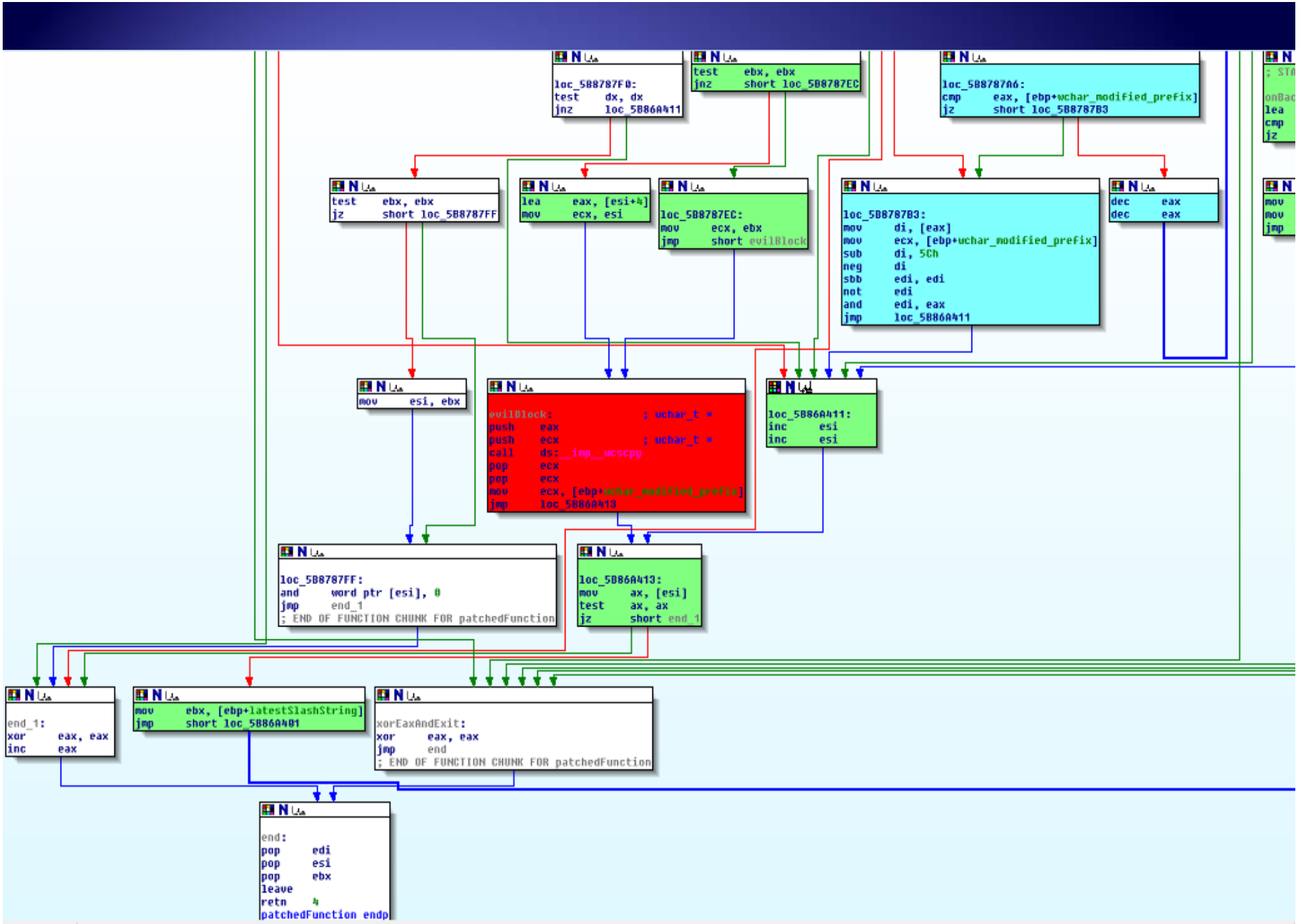
Finding the RPC function

- Use xref feature in IDA
  - Track back to CanonicalizePathName
  - Then _NetpwPathCanonicalize
- Examine imports of srvsvc, search for this function
  - Match to NetprPathCanonicalize

Hitting the endpoint

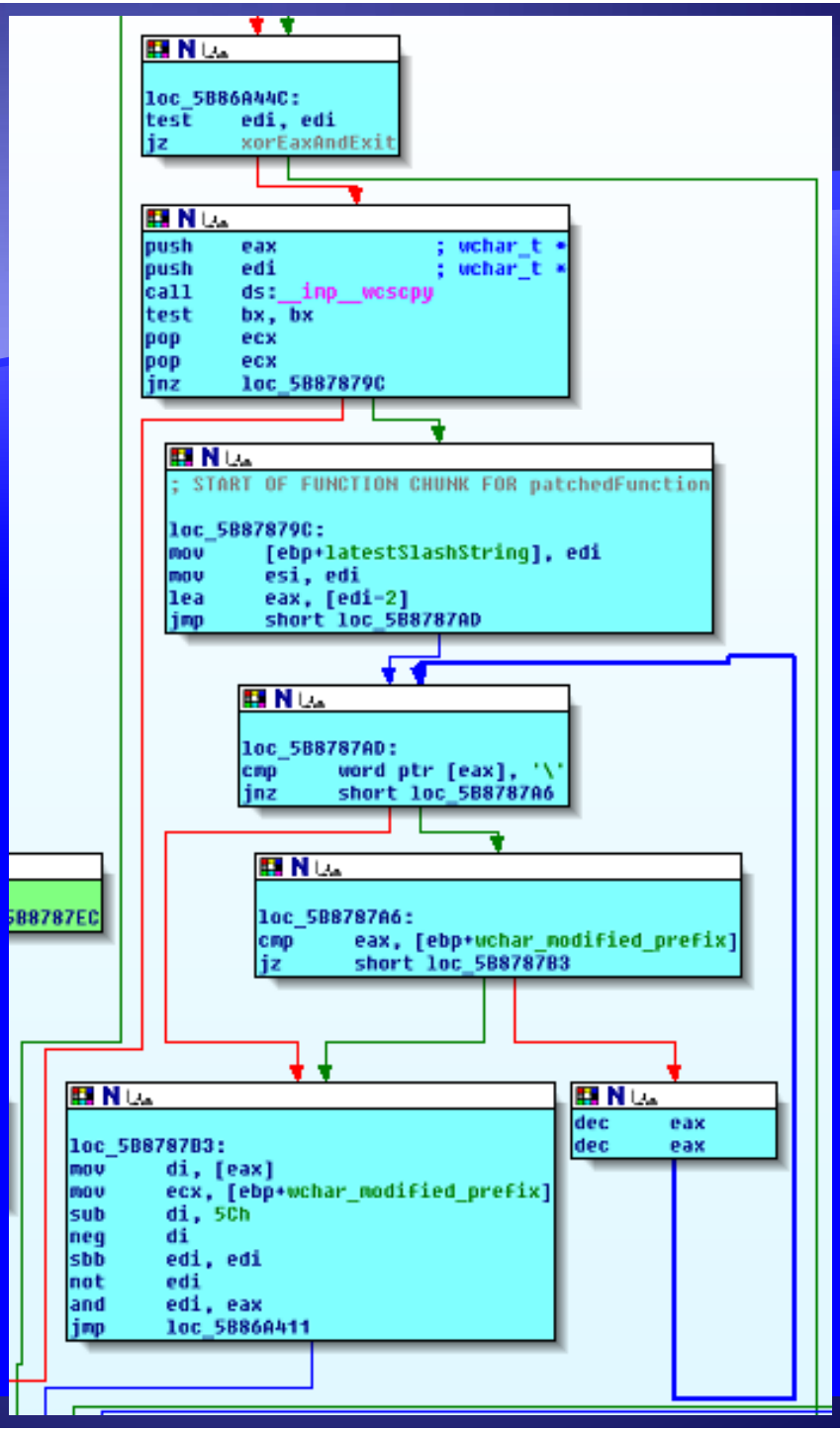
- Use similar method to samba vulnerability
- Notice the nacn_np, note other UUID info
- Modify an existing Metasploit exploit to hit this function
- Use NDR encoding rules found at TippingPoint website
  - Could 'reverse' them from other Metasploit exploits
  - Or reverse RPCRT4.dll


loc_5B86A42B: ; check if we have '\..'
lea eax, [esi+2]
mov dx, [eax]
cmp dx, '.'
jnz loc_5B8787CB
lea eax, [esi+4] ; check if we have '\..\'
mov bx, [eax]
cmp bx, '\'
jz short loc_5B86A44C
test bx, bx
jnz short loc_5B86A411
loc_5B86A44C:
test edi, edi
jz xorEaxAndExit
push eax ; wchar_t *
push edi ; wchar_t *
call ds:__imp__wcscpy
test bx, bx
pop ecx
pop ecx
jnz loc_5B87879C
; START OF FUNCTION CHUNK FOR patchedFunction
loc_5B87879C:
mov [ebp+latestSlashString], edi
mov esi, edi
lea eax, [edi-2]
jmp short loc_5B8787AD
loc_5B8787CB:
cmp dx, '\'
jnz short loc_5B8787F0
loc_5B8787AD:
cmp word ptr [eax], '\'
jnz short loc_5B8787A6
loc_5B8787F0:
test dx, dx
jnz loc_5B86A411
test ebx, ebx
jnz short loc_5B8787EC
loc_5B8787A6:
cmp eax, [ebp+wchar_modified_prefix]
jz short loc_5B8787B3
; START OF FUNCTION CHUNK FOR patchedFunction
onBackslash:
lea eax, [esi-2]
cmp ebx, eax
jz short xorEaxAndExit
Graph overview


loc_5B8787F0:
test dx, dx
jnz loc_5B86A411
test ebx, ebx
jnz short loc_5B8787EC
loc_5B8787A6:
cmp eax, [ebp+wchar_modified_prefix]
jz short loc_5B8787B3
test ebx, ebx
jz short loc_5B8787FF
lea eax, [esi+4]
mov ecx, esi
loc_5B8787EC:
mov ecx, ebx
jmp short evilBlock
loc_5B8787B3:
mov di, [eax]
mov ecx, [ebp+uchar_modified_prefix]
sub di, 5Ch
neg di
sbb edi, edi
not edi
and edi, eax
jmp loc_5B86A411
dec eax
dec eax
mov esi, ebx
evilBlock: ; uchar_t *
push eax
push ecx ; uchar_t *
call ds:__imp__wcscpy
pop ecx
pop ecx
mov ecx, [ebp+wchar_modified_prefix]
jmp loc_5B86A413
loc_5B86A411:
inc esi
inc esi
loc_5B8787FF:
and word ptr [esi], 0
jmp end_1
; END OF FUNCTION CHUNK FOR patchedFunction
loc_5B86A413:
mov ax, [esi]
test ax, ax
jz short end_1
end_1:
xor eax, eax
inc eax
mov ebx, [ebp+latestSlashString]
jmp short loc_5B86A401
xorEaxAndExit:
xor eax, eax
jmp end
; END OF FUNCTION CHUNK FOR patchedFunction
end:
pop edi
pop esi
pop ebx
leave
retn 4
patchedFunction endp

Vulnerability

- Not a straight stack overflow
- Due to an unbound searching loop for a '\' character
  - Then concatenating the result with something else
  - A string can become much longer than intended
- Done on stack, so stack smash
  - If no other '\' exists before the loop, stack overflow


loc_5B86A44C:
test edi, edi
jz xorEaxAndExit
push eax ; wchar_t *
push edi ; wchar_t *
call ds:__imp__wcscpy
test bx, bx
pop ecx
pop ecx
jnz loc_5B87879C
; START OF FUNCTION CHUNK FOR patchedFunction
loc_5B87879C:
mov [ebp+latestSlashString], edi
mov esi, edi
lea eax, [edi-2]
jmp short loc_5B8787AD
loc_5B8787AD:
cmp word ptr [eax], '\'
jnz short loc_5B8787A6
loc_5B8787A6:
cmp eax, [ebp+wchar_modified_prefix]
jz short loc_5B8787B3
loc_5B8787B3:
mov di, [eax]
mov ecx, [ebp+wchar_modified_prefix]
sub di, 5Ch
neg di
sbb edi, edi
not edi
and edi, eax
jmp loc_5B86A411
dec eax
dec eax

Links

- Full exploit (done by Metasploit, not me)
  - http://metasploit.com/svn/framework3/trunk/modules/exploits/windows/smb/ms08_067_netapi.rb
- Using pyMSRPC to trigger this
  - http://dvlabs.tippingpoint.com/blog/2008/11/06/using-pymsrpc-to-trigger-ms08-067
- TippingPoint NDR encoding examples
  - http://dvlabs.tippingpoint.com/blog/2007/11/24/msrpc-ndr-types
- Technical analysis
  - http://www.dontstuffbeansupyournose.com/?p=35