Exploit Targets a11y.text Exploit Targets Coding Exploit Targets in your Metasploit Module a11y.text Coding Exploit Targets in your Metasploit Module Exploits define a list of targets that includes a name, number, and options. Targets are specified by number when launched. Sample Target Code for an Exploit Module: 'Targets' =>

```
[
    # Windows 2000 â€“ TARGET = 0
    [
        'Windows 2000 English',
        {
            'Rets' => [ 0x773242e0 ],
        },
    ],
    # Windows XP - TARGET = 1
    [
        'Windows XP English',
        {
            'Rets' => [ 0x7449bf1a ],
        },
    ],
],
```

'DefaultTarget' => 0)) Target Options Block a11y.text Target Options Block The options block within the target section is nearly free-form although there are some special option names. â€˜Retâ€™ is short-cutted as target.ret() â€˜Payloadâ€™ overloads the exploits info block Options are where you store target data. For example: The return address for a Windows 2000 target 500 bytes of padding need to be added for Windows XP targets Windows Vista NX bypass address Accessing Target Information a11y.text Accessing Target Information The â€˜targetâ€™ object inside the exploit is the

users selected target and is accessed in the exploit as a hash. target[â€˜padcountâ€™] target[â€˜Retsâ€™][0] target[â€˜Payloadâ€™][â€˜BadCharsâ€™] target[â€˜opnumâ€™] Adding and Fixing Exploit Targets a11y.text Adding and Fixing Exploit Targets Sometimes you need new targets because a particular language pack changes addresses, a different version of the software is available, or the addresses are shifted due to hooks. Adding a new target only requires 3 steps. Determine the type of return address you require. This could be a simple â€˜jmp espâ€™, a jump to a specific register, or a â€˜pop/pop/retâ€™. Comments in the exploit code can help you determine what is required. Obtain a copy of the target binaries Use msfpescan to locate a suitable return address msfpescan help file â€“ Metasploit Unleashed Getting a Return Address withÂ msfpescan a11y.text Getting a Return Address withÂ msfpescan If the exploit code doesnâ€™t explicitly tell you what type of return address is required but is good enough to tell you the dll name for the existing exploit, you can find out what type of return address you are looking for. Consider the following example that provides a return address for a Windows 2000 SP0-SP4 target. 'Windows 2000 SP0-SP4',

{

'Ret' => 0x767a38f6, # umpnpmgr.dll

} To find out what type of return address the exploit currently uses, we just need to find a copy of umpnpmgr.dll from a Windows 2000 machine machine and run msfpescan with the provided address to determine the return type. In the example below, we can see that this exploit requires a pop/pop/ret. root@kali : ~ # msfpescan -D -a 0x767a38f6 umpnpmgr.dll [umpnpmgr.dll] 0x767a38f6 5f5ec3558bec6aff68003c7a7668e427

00000000 5F pop edi

00000001 5E pop esi

00000002 C3 ret

00000003 55 push ebp

00000004 8BEC mov ebp,esp

00000006 6AFF push byte -0x1

00000008 68003C7A76 push 0x767a3c00

0000000D 68 db 0x68

0000000E E427 in al,0x27 Now, we just need to grab a copy of the target dll and use msfpescan to find a usable pop/pop/ret address for us. root@kali : ~ # msfpescan -p umpnpmgr.dll [targetos.umpnpmgr.dll]

0x79001567 pop eax; pop esi; ret

0x79011e0b pop eax; pop esi; retn 0x0008

0x79012749 pop esi; pop ebp; retn 0x0010

0x7901285c pop edi; pop esi; retn 0x0004 Now that weâ€™ve found a suitable return address, we add our new target to the exploit. 'Windows 2000 SP0-SP4 Russian Language',

{

'Ret' => 0x7901285c, # umpnpmgr.dll

} Next Exploit Payloads Prev Exploit Mixins