

PowerPwning: Post-Exploiting By Overpowering PowerShell

Joe Bialek

About Me

- Joe Bialek
- Security Engineer
- Twitter: @JosephBialek
- Blog: <http://clymb3r.wordpress.com>
- GitHub: <https://github.com/clymb3r>

PowerShell is Awesome

- Provides access to the Win32 API
- Doesn't write to disk when scripts are run on remote computers
- Script runs inside PowerShell.exe or WsmProvHost.exe (when run remotely)
 - Don't have to execute suspicious or unsigned processes

What I Want To Do With PowerShell

- Run existing tools in PowerShell without rewriting them in PowerShell
 - Use existing tools but leverage PowerShell's forensic benefits
- Solution: Write a PowerShell script to reflectively load and execute PE's (EXE/DLL) in the PowerShell process

How To Load A PE

1. Allocate memory for PE
2. Copy PE headers to memory
3. Copy sections to memory (.text, .data, etc.)
4. Perform “base relocations” on the sections loaded
5. Load DLL’s the PE requires
6. Adjust memory permissions
7. Call the entry function
 - For DLL: Calls DllMain which lets the DLL know it is loaded
 - For EXE: Function which sets up the process, gets command line arguments and calls int main()

DLL Specific Stuff

- After loading DLL, call exported DLL functions
- Remote PowerShell can't capture "stdout", you won't see anything your program outputs
 - printf
 - cout
- To capture output: Make the DLL function return a "char*" or "wchar_t*"
- PowerShell can Marshal this pointer to a managed string and print the output

Problems Reflectively Loading An EXE: Prevent PowerShell From Exiting

- When EXE exits, it calls `ExitProcess`
 - PowerShell is the running process so it is killed by `ExitProcess`
 - I want the EXE to exit, not PowerShell
- Solution:
 - Call the EXE entry function in its own thread
 - Overwrite `ExitProcess` function with a call to `ExitThread`

Assembly To Overwrite ExitProcess

; Set a var to 1, let PS know exe is exiting

```
mov    rbx, 0x4141414141414141
```

```
mov    [rbx], byte 0x01
```

; Call exitthread instead of exitprocess

```
sub    rsp, 0xc0
```

```
and    sp, 0xFFF0 ; Needed for stack alignment
```

```
mov    rbx, 0x4141414141414141
```

```
call  rbx
```


Problems Reflectively Loading An EXE: Pass Command Line Arguments

- EXE entry function retrieves command line arguments and passes them to “int main(argc, argv)”
 - Functions which are used to get command line:
 - GetCommandLine()
 - __getcmdln()
 - Function called appears to depend on how the EXE was compiled (/MD vs /MDd in Visual Studio)
 - Built solutions to deal with both cases

Patch GetCommandLine()

For EXE compiled with Visual Studio as “Multi-Threaded”:

- Overwrite GetCommandLineA() and GetCommandLineW() with shell code to return a string I allocate

Assembly to overwrite GetCommandLine

```
; X64 code  
mov     rax, 0x4141414141414141  
ret
```

```
; X86 code  
mov     eax, 0x41414141  
ret
```

Patch `__getcmdln()`

For EXE compiled with Visual Studio “Multi-Threaded DLL”:

- The runtime DLL `msvcrXXX.dll` or `msvcrXXXd.dll` exports the variables `__acmdln` and `__wcmdln`, which are `char*` and `wchar_t*`
- Replace these with our own strings we allocate using PowerShell
- When the DLL function `__getcmdln` is called, it will parse the strings we set into `argc` and `argv` and return them

Remote Reflective DLL Injection

- Stephen Fewer method:
 - Write DLL bytes AND his reflective DLL loader to remote process memory
 - CreateRemoteThread for his reflective DLL loader, which then reflectively loads the actual DLL
- I can't write PowerShell code in to a remote process, so this method doesn't work for me

Remote Reflective DLL Injection

- My method:
 - Allocate memory in remote process
 - Load needed libraries in remote process
 - Have to write assembly for remote LoadLibrary and remote GetProcAddress functionality
 - Stage DLL in the PowerShell process
 - Perform relocations and whatnot on the DLL bytes while it is in the PowerShell process
 - Base relocation calculations are done based on the address of memory allocated in the remote process
 - Write the bytes to the remote process
 - Create a thread to begin DLL execution

Remote LoadLibrary (x64)

```
; Save rsp and setup stack for function call
```

```
push rbx
```

```
mov rbx, rsp
```

```
sub rsp, 0x20
```

```
and sp, 0xffc0
```

```
; Call LoadLibraryA
```

```
mov rcx, 0x4141414141414141
```

```
; Ptr to string of library, set by PS
```

```
mov rdx, 0x4141414141414141
```

```
; Address of LoadLibrary, set by PS
```

```
call rdx
```

```
mov rdx, 0x4141414141414141
```

```
; Ptr to save result, set by PS
```

```
mov [rdx], rax
```

```
; Fix stack
```

```
mov rsp, rbx
```

```
pop rbx
```

```
ret
```

Remote GetProcAddress (x64)

```
; Save state of rbx and stack
```

```
push rbx
```

```
mov rbx, rsp
```

```
; Set up stack for function call to GetProcAddress
```

```
sub rsp, 0x20
```

```
and sp, 0xffc0
```

```
; Call getprocaddress
```

```
mov rcx, 0x4141414141414141
```

```
; DllHandle, set by PS
```

```
mov rdx, 0x4141414141414141
```

```
; Ptr to FuncName string, set by PS
```

```
mov rax, 0x4141414141414141
```

```
; GetProcAddress address, set by PS
```

```
call rax
```

```
; Store the result
```

```
mov rcx, 0x4141414141414141
```

```
; Ptr to buffer to save result, set by PS
```

```
mov [rcx], rax
```

```
; Restore stack
```

```
mov rsp, rbx
```

```
pop rbx
```

```
ret
```


Demos

Detection & Prevention

- PowerShell remoting requires administrator access
- PowerShell pipeline logging MAY help detection
- Constrained run spaces help limit the power of PowerShell
- Standard stuff like firewalls, limiting powerful accounts, etc.. will help prevent the remote aspect
- Machine wide profile to log actions to a transcript

Closing Thoughts

- **This is NOT a vulnerability!**
 - PowerShell is a Turing complete programming language, it can do all this by design
 - Basically any programming language can be used to create similar functionality
- PowerShell is a great way to manage Windows systems and has good security
- Don't let this talk scare you away from PowerShell

Links

- Invoke-ReflectivePEInjection:
<https://github.com/clymb3r/powershell>
 - Also part of PowerSploit
- Blog: <http://clymb3r.wordpress.com>

References

MSDN documentation on PE's and DLL loading:

- <http://msdn.microsoft.com/en-us/magazine/bb985992.aspx>
- <http://msdn.microsoft.com/en-us/magazine/cc301808.aspx>
- <http://msdn.microsoft.com/library/windows/hardware/gg463125>

Other reflective loaders:

- <https://github.com/stephenfewer/ReflectiveDLLInjection>
- <http://www.joachim-bauch.de/tutorials/loading-a-dll-from-memory/>

Good PowerShell related blogs:

- <http://www.exploit-monday.com/>
- <http://www.leeholmes.com/blog/>