Weaponizing the Windows API

With Metasploit's Railgun

Who is this guy?

- Twitter & IRC: thelightcosine
- Core Developer for Metasploit Pro
- Community Contributor
- Penetration Tester

Meterpre-what?

Windows Meterpreter

- Goto Payload for Windows
- DLL, compiled C
- Usually injected into process memory
- Enhanced CMD shell
- Provides basic post-exploitation API

Windows Meterpreter

- Often run with SYSTEM Privs
- Can be migrated into a user's process

So what is Railgun?

Railgun

- Railgun is an extension to the Meterpreter STDAPI
- Allows Arbitrary Loading of DLLs
- As long as you know the path of the DLL, you can access it's functions

Railgun

> Since Windows API DLLs are always at known paths, we can always load them

The Windows API

- Dynamic access to the entirety of the Windows API on the system
- By calling APIs from user processes, we can impersonate users
- Anything becomes possible

Let's talk about Railgun

A brief History of Railgun

- June 2010 – Railgun submitted to Metasploit by Patrick HVE
- Sept 2010 – 64bit support added by Stephen Fewer
- Feb 2011 – Chao-mu takes over Railgun support, resumes new feature work
- Fall 2011 – Chao-mu disappears
- Aug 2012 – YOU start contributing to Railgun
- Dec 2012 – Mayans predict Railgun-related Apocalypse?

How it works

On the C side

- LoadLibrary function opens a Handle to the DLL
- GetProcAddress maps a function pointer to the specified function
- Memread and Memwrite functions for manipulating memory space

How it works

On the Ruby side

- Ruby code lives in lib/rex/post/meterpreter/extensions/stdapi/railgun
- User/module writer defines the DLL and the needed functions
- Functions are then avilable as methods
- Can define at runtime or use definition files

```
def self.create_dll(dll_path = 'advapi32')
  dll = DLL.new(dll_path, ApiConstants.manager)


  dll.add_function('CredEnumerateA', 'BOOL', [
    ['PCHAR', 'Filter', 'in'],
    ['DWORD', 'Flags', 'in'],
    ['PDWORD', 'Count', 'out'],
    ['PBLOB', 'Credentials', 'out']])
```

A look at Railgun Definitions

Anatomy of a Function

1. Function Name
2. Function Return Type
3. Array of Parameters
    1. Param type
    2. Param Name
    3. IN/OUT/INOUT Parameter

A word about constants

- Railgun knows about Windows constants
- They are defined in api_constants.rb in the railgun folder
- Easy to add new constants as needed there

Supported Data Types

DWORD

- If it quacks like a duck…
- Pass as a Fixnum or Bignum
- String representation of constants can also be passed in

PDWORD

- Pointer to a DWORD
- Pass a Fixnum
- Pass the Content of the DWORD not the pointer
- If it is an OUT only paramter, pass a 4 (size of a DWORD)
- Pass nil for a NULL Pointer

PCHAR and PWCHAR

- Pass as Ruby strings. Will be converted seamlessly
- If OUT only, pass fixnum of the size of the buffer (including null byte)

Definition

```
dll.add_function(
    'CryptAcquireContextW',
    'BOOL',[
['PDWORD', 'phProv', 'out'],
['PWCHAR', 'pszContainer',
    'in'],
['PWCHAR', 'pszProvider', 'in'],
['DWORD', 'dwProvType', 'in'],
['DWORD', 'dwflags', 'in']])
```

Usage

```
ms_enhanced_prov = "Microsoft
    Enhanced Cryptographic
    Provider v1.0"
prov_rsa_full = 1
crypt_verify_context =
    0xF0000000
alg_md5 = 32771
alg_rc4 = 26625
advapi32 =
    client.railgun.advapi32
acquirecontext =
    advapi32.CryptAcquireConte
    xtW(4, nil,
    ms_enhanced_prov,
    prov_rsa_full,
    crypt_verify_context)
```

Used in the SmartFTP password Recovery Module

Bool

> Pass in Ruby True/False values exactly as expected

Definition:

dll.add_function('IsDebuggerPresent', 'BOOL',[])

Usage:

>> client.railgun.kernel32.IsDebuggerPresent()

=> {"GetLastError"=>0, "return"=>**false**}

Bytes and Words

"Handled the same as DWORDs but Fixnums passed in will be truncated to the appropriate length

PBLOB

- Anything that's not a string or a DWORD
- Treated as a ruby string
- Railgun will not help you parse structures

Definition

```
dll.add_function(
    'WlanGetProfile', 'DWORD',[
['DWORD', 'hClientHandle', 'in'],
['PBLOB', 'pInterfaceGuid', 'in'],
['PBLOB', 'strProfileName', 'in'],
['LPVOID', 'pReserved', 'in'],
['PDWORD', 'pstrProfileXML',
    'out'],
['PDWORD', 'pdwFlags', 'inout'],
['PDWORD', 'pdwGrantedAccess',
    'out']])
```

Usage

```
profile['name'] =
    @host_process.memory.re
    ad(ppointer,512)
ppointer = (ppointer + 516)


rprofile =
    @wlanapi.WlanGetProfile(
    wlan_handle,guid,profile['n
    ame'],nil,4,4,4)
```

Used in the wlan_profile post module

Faking unsupported Data Types

- Pointers and Handles of any kind are really just numbers, so treat them as DWORDs
- If it can be treated as a number it's a DWORD
- Otherwise it's a PBLOB
- If neither works, add support for it yourself =)

Dealing with Return Values

- The function will return a hash
- Hash will always contian at least GetLastError
- Hash will return any OUT values

GetLastError

- Will return 0 if there was no error
- Otherwise will contain the windows system Error code encountered
- Errors codes can be looked up at http://msdn.microsoft.com/en-us/library/windows/desktop/ms681381(v=vs.85).aspx

Returned OUT Parameters

```
acquirecontext =
    advapi32.CryptAcquireC
    ontextW(4, nil,
    ms_enhanced_prov,
    prov_rsa_full,
    crypt_verify_context)


createhash =
    advapi32.CryptCreateHas
    h(acquirecontext['phPro
    v'], alg_md5, 0, 0, 4)
```

Tricky Situations

- Complex structure types that you will have to parse yourself
- Strings you don't know the length of
- Large number of string reads (SLOWWWW)

So What?

Why do we care about all this stuff?

What it means

- Anything you can do with the windows API is available
- Without increasing the size of the payload

Example Mayhem

- Get the OS to Decrypt stored SmartFTP Passwords
- Enumerate and decrypt stored RDP passwords
- Scan for Wireless Aps
- Enumerates Domain controllers on the victim's network

Demo time

- “ Enough of these ugly slides
- “ Let’s see it in action