ADOBE SANDBOX
WHEN THE BROKER IS BROKEN

Peter Vreugdenhil

Exodus Intelligence

Intro

- Peter Vreugdenhil
- VP of Operations at Exodus Intelligence
- @WTFuzz
- peter@exodusintel.com

Goal of this talk

- Explaining the code responsible for the interesting parts of the Sandbox
- Making it easier for other researchers to find sandbox escapes
- Show some potential sandbox escapes

Content

- Sandbox basics
- The Adobe Sandbox
- Attack surface
- Finding all Broker endpoints
- Finding intercepted API functions
- (Ab)using the broker to escape

Previous work on Adobe Sandbox

- Zhenhua Liu - Breeding Sandworms: How To Fuzz Your Way Out of Adobe Reader's Sandbox
- Paul Sabanal & Mark Vincent Yason : PLAYING IN THE READER X SANDBOX

What is a sandbox?

- Wikipedia:

  A **sandbox** is a security mechanism for separating running programs. It is often used to execute untested code, or untrusted programs from unverified third-parties, suppliers, untrusted users and untrusted websites.

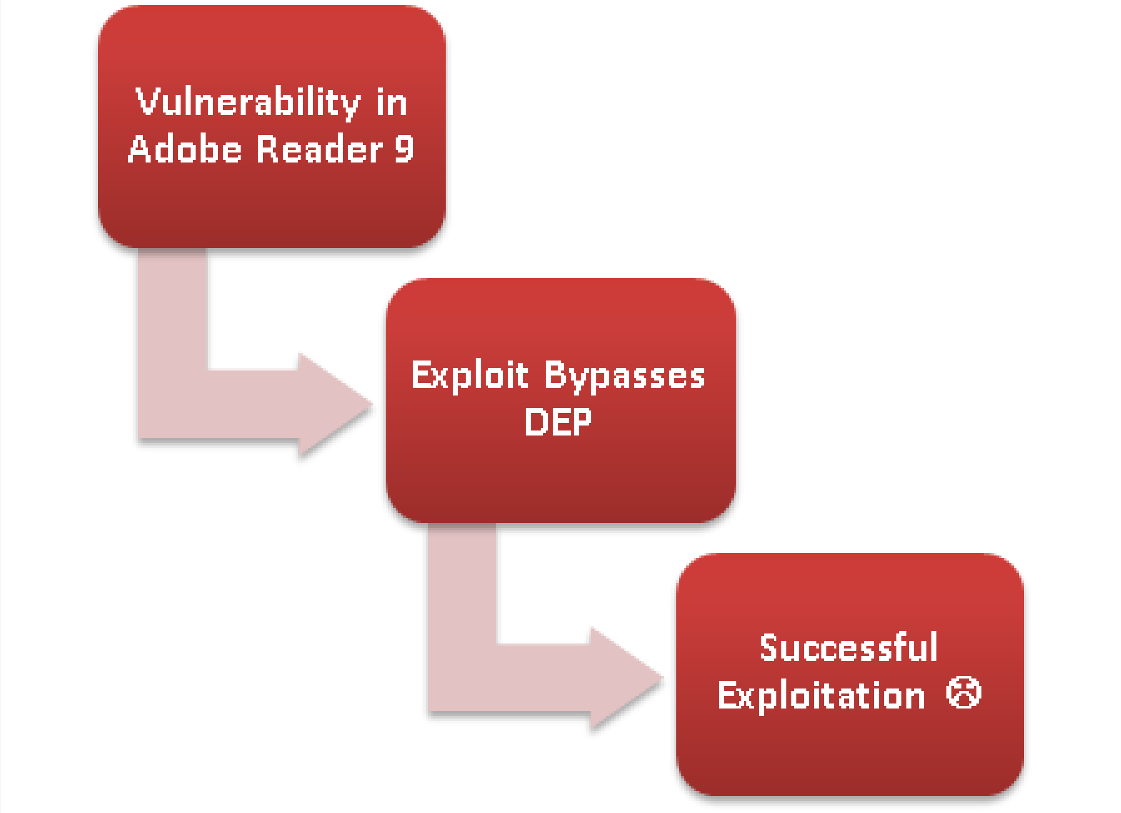
Sandbox workings

- Untrusted code is running with low/limited privileges
- Anything requiring elevated privileges goes through a broker
- Usually certain windows API calls are intercepted for transparency

Terminology

- **Broker**: Medium integrity process
- **Client**: LOW integrity process
- **Cross Call**: request from Client to Broker
- **Endpoint**: Code running in the broker responsible for handling the Cross Call
- **Escape**: Executing arbitrary code with Medium Integrity

Adobe on Sandboxing

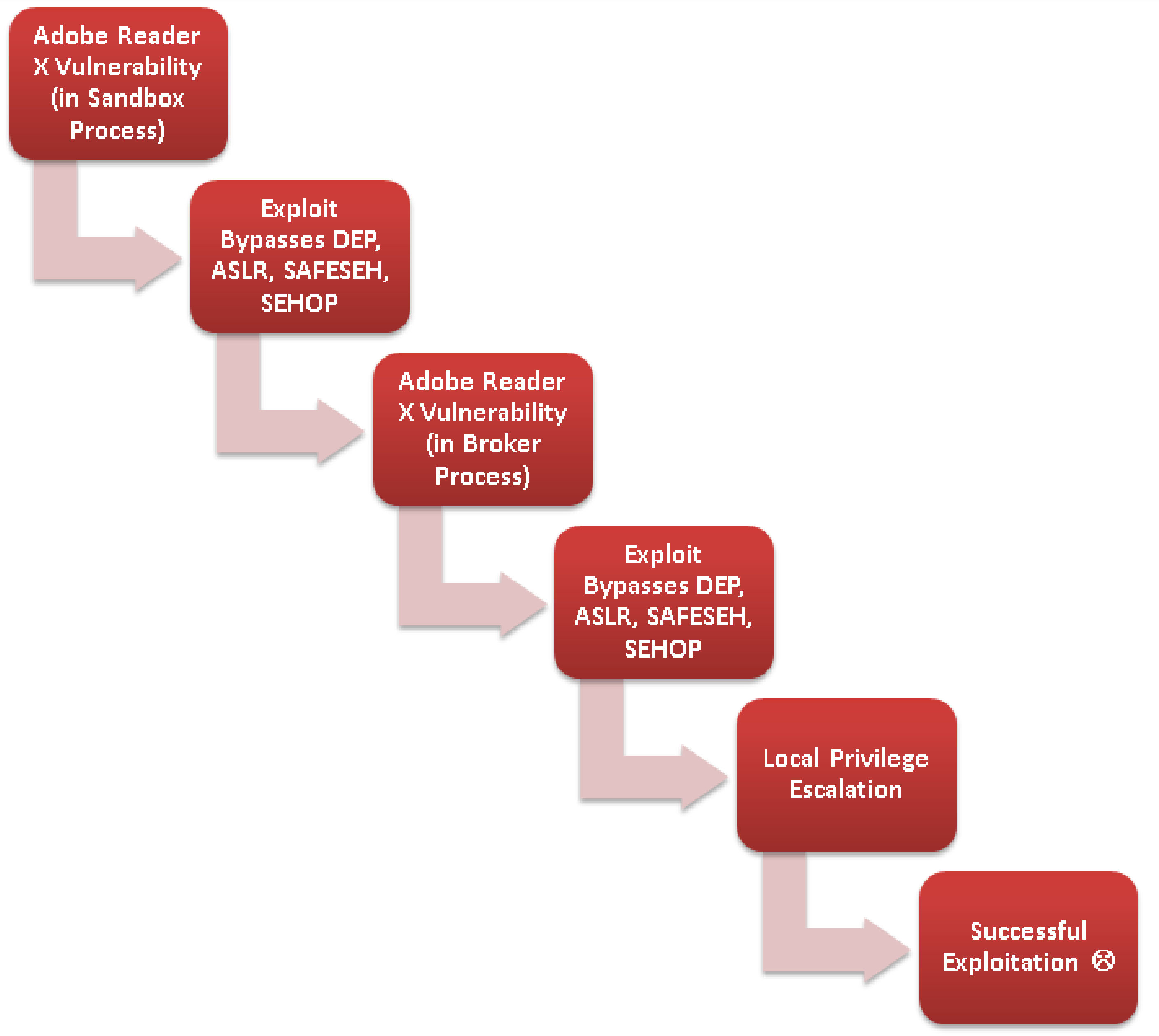


Vulnerability in Adobe Reader 9

Exploit Bypasses DEP

Successful Exploitation ☹

http://blogs.adobe.com/asset/files/2010/11/WinXP-A9-Exploit-Steps1.png



Adobe Reader X Vulnerability (in Sandbox Process)

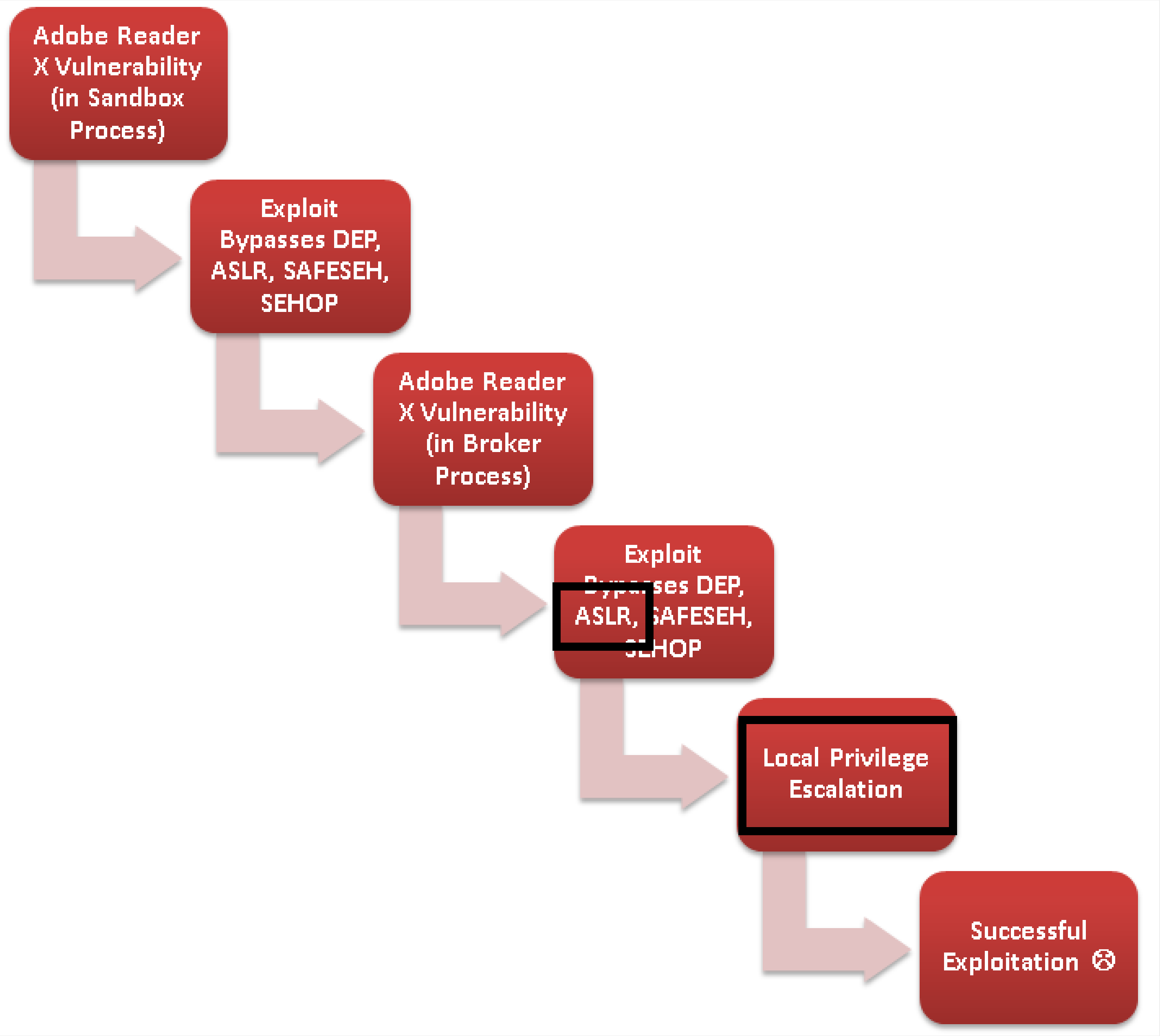
Exploit Bypasses DEP, ASLR, SAFESEH, SEHOP

Adobe Reader X Vulnerability (in Broker Process)

Exploit Bypasses DEP, ASLR, SAFESEH, SEHOP

Local Privilege Escalation

Successful Exploitation ☹



Adobe Reader X Vulnerability (in Sandbox Process)

Exploit Bypasses DEP, ASLR, SAFESEH, SEHOP

Adobe Reader X Vulnerability (in Broker Process)

Exploit Bypasses DEP, ASLR, SAFESEH, SEHOP

Local Privilege Escalation

Successful Exploitation ☹

Adobe Sandbox Basics

- Available since Adobe Reader X
- Improved in Adobe Reader XI
- Based on the Chromium sandbox
  - Less restricted
  - Much more communication between client and broker
- 1 confirmed Adobe Sandbox escape in the wild (so far)
- 1 unconfirmed escape for sale in Russia

Adobe Sandbox on Windows

- Restricted Token
- Windows Integrity levels
- Separate Desktops
- Separate Jobs

Adobe Sandbox Restricted Token

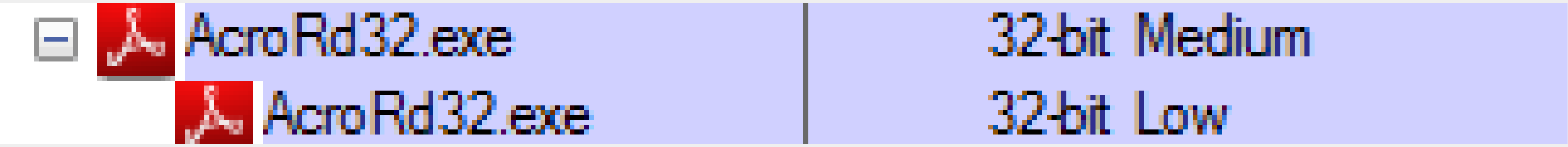
- Everything is denied.
- Privileges: SeChangeNotifyPrivilege enabled

Adobe Sandbox Integrity Levels

- Windows has 5 predefined Integrity levels
  - Untrusted
  - **Low**
  - **Medium**
  - High
  - System

Adobe Sandbox Integrity levels

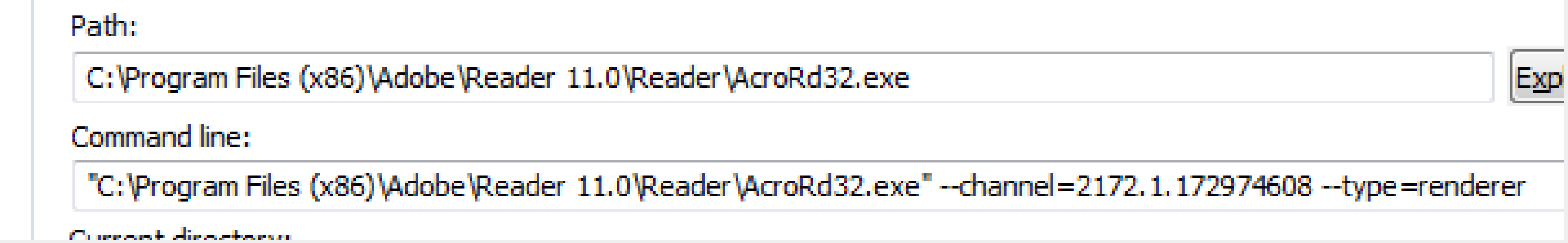
- Adobe starts as a MEDium Integrity process
- Spawns a child process as LOW Integrity
- Child process is responsible for parsing and rendering pdf files



```
⊟ AcroRd32.exe        32-bit  Medium
    AcroRd32.exe      32-bit  Low
```

Adobe Sandbox

- Child process command line arguments specify communication channel details and process type



Path:

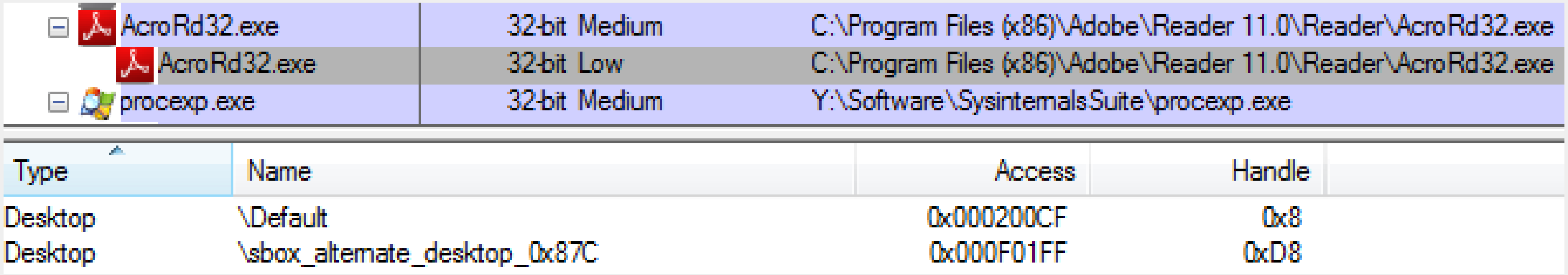
C:\Program Files (x86)\Adobe\Reader 11.0\Reader\AcroRd32.exe

Command line:

"C:\Program Files (x86)\Adobe\Reader 11.0\Reader\AcroRd32.exe" --channel=2172.1.172974608 --type=renderer

Adobe Sandbox Desktop

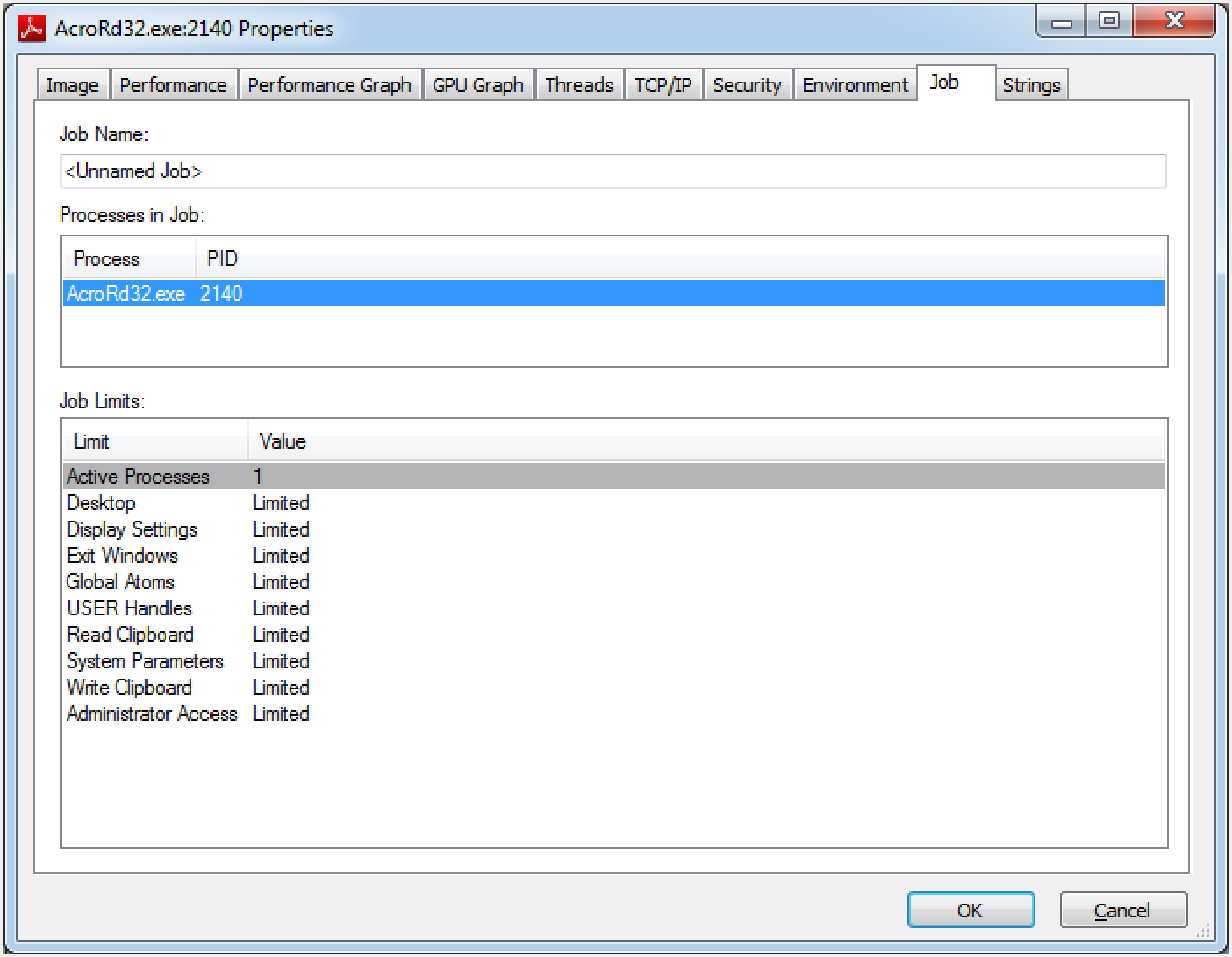
- LOW Integrity child process has its own desktop (since Reader XI).
- sbox_alternate_desktop_0x<ParentPID>
- Limited access to the Default desktop
- Protects against (among other) shatter attacks



```
⊟ AcroRd32.exe        32-bit Medium    C:\Program Files (x86)\Adobe\Reader 11.0\Reader\AcroRd32.exe
    AcroRd32.exe      32-bit Low       C:\Program Files (x86)\Adobe\Reader 11.0\Reader\AcroRd32.exe
⊟ procexp.exe         32-bit Medium    Y:\Software\SysinternalsSuite\procexp.exe
```

Type	Name	Access	Handle
Desktop	\Default	0x000200CF	0x8
Desktop	\sbox_alternate_desktop_0x87C	0x000F01FF	0xD8

Adobe Sandbox Job



Job Name:

<Unnamed Job>

Processes in Job:

Process	PID
AcroRd32.exe	2140

Job Limits:

Limit	Value
Active Processes	1
Desktop	Limited
Display Settings	Limited
Exit Windows	Limited
Global Atoms	Limited
USER Handles	Limited
Read Clipboard	Limited
System Parameters	Limited
Write Clipboard	Limited
Administrator Access	Limited

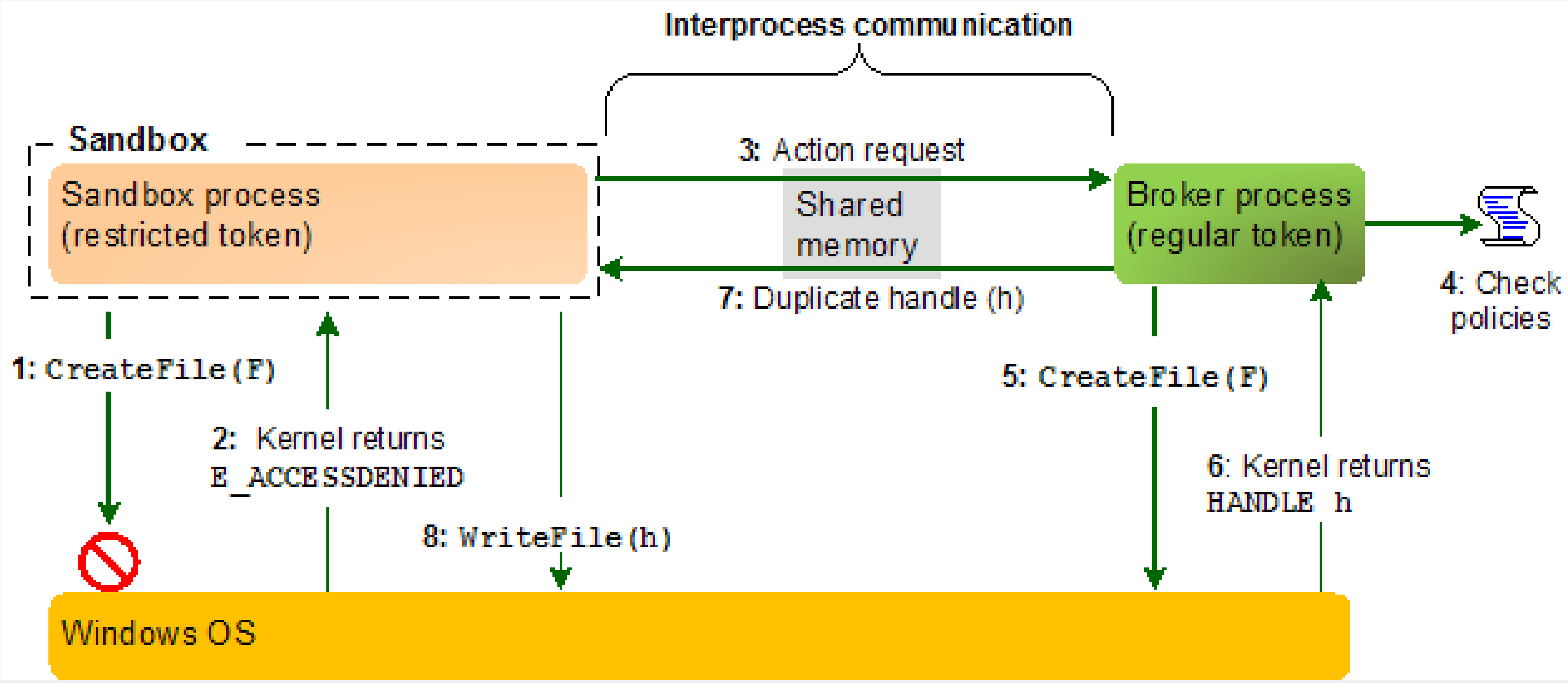
Adobe Sandbox Attack Surface

- Windows Kernel vulnerabilities
- IPC Communications errors
- Incorrect default permissions
- Logical flaws in Cross Calls
- Memory corruption in Cross Calls

Adobe Sandbox Attack Surface

- ~~Windows Kernel vulnerabilities~~
- ~~IPC Communications errors~~
- Incorrect default permissions
- **Logical flaws in Cross Calls**
- ~~Memory corruption in Cross Calls~~

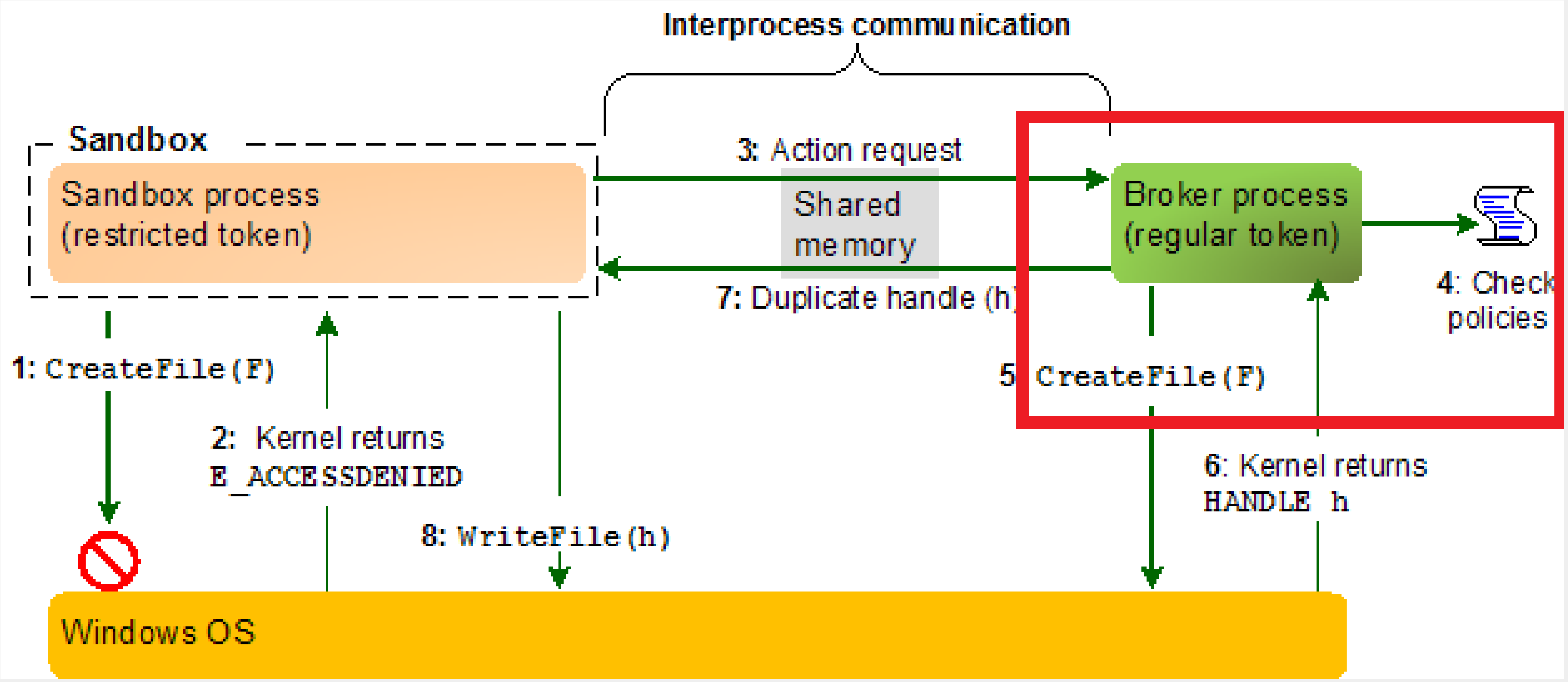
Broker Client communication



http://blogs.adobe.com/asset/files/2010/11/Sandbox-and-Broker-Process-IPC.png

Broker Client communication

We will focus on the Broker endpoints



http://blogs.adobe.com/asset/files/2010/11/Sandbox-and-Broker-Process-IPC.png

Client Broker communication

- AcroRD32.exe responsible for Cross Calls
- Changes with updates
- Finding all Cross Calls through different versions is possible
- Easy even

Client Broker communication

- Uses Shared Memory for communication
- Structures and Parameters for Cross Calls are written to memory by the Client process
- Broker reads them back and acts on them
- Some Parameters can be used to receive results
- Vulnerabilities can exists in this part of the process

Cross Call Parameters

```
// [ tag                4 bytes]
// [ IsOnOut            4 bytes]
// [ call return       52 bytes]
// [ params count       4 bytes]
// [ parameter 0 type   4 bytes]
// [ parameter 0 offset 4 bytes] ---delta to ---\
// [ parameter 0 size   4 bytes]                |
// [ parameter 1 type   4 bytes]                |
// [ parameter 1 offset 4 bytes] ---------------|--\
// [ parameter 1 size   4 bytes]                |  |
// [ parameter 2 type   4 bytes]                |  |
// [ parameter 2 offset 4 bytes] ----------------------\
// [ parameter 2 size   4 bytes]                |  |   |
// |---------------------------|                |  |   |
// | value 0     (x bytes)     | <--------------/  |   |
// | value 1     (y bytes)     | <-----------------/   |
// |                           |                       |
// | end of buffer             | <---------------------/
// |---------------------------|
```

Cross Call Parameters

- Cross Call tag/ID
- Number of Parameters
- Types of Parameters

Cross Call IDs

- Chromium has 19 Cross Calls predefined
- 16 are actually used
- ID 0 is unused
- ID 1 and 2 are test only
- Adobe Reader has 260 Cross Calls defined

Cross Call Parameters Types

Chromium code defines 6 valid Parameter types

```
enum ArgType {
  INVALID_TYPE = 0,
  WCHAR_TYPE,
  ULONG_TYPE,
  UNISTR_TYPE,
  VOIDPTR_TYPE,
  INPTR_TYPE,
  INOUTPTR_TYPE,
  LAST_TYPE
};
```

Adobe sandbox implementation adds two more

Broker Endpoints

- Every Cross Call is linked to a Broker function
- Finding all the end points would allow us to RE the broker code
- Finding all the parameters for the functions would make it easier

Broker Endpoints

- One function is responsible for defining Cross Calls

```
static const IPCCall set_info = {
       {IPC_NTSETINFO_RENAME_TAG,
        VOIDPTR_TYPE,
        INOUTPTR_TYPE,
        INOUTPTR_TYPE,
        ULONG_TYPE,
        ULONG_TYPE},
        reinterpret_cast<CallbackGeneric>(
           &FilesystemDispatcher::NtSetInformationFile
        )
       };
ipc_calls_.push_back(set_info);
```

Broker Endpoints

If we can find that function we might be able to find:

- Cross Call ID
- Parameter info
- Broker endpoint function

Finding Broker Endpoints

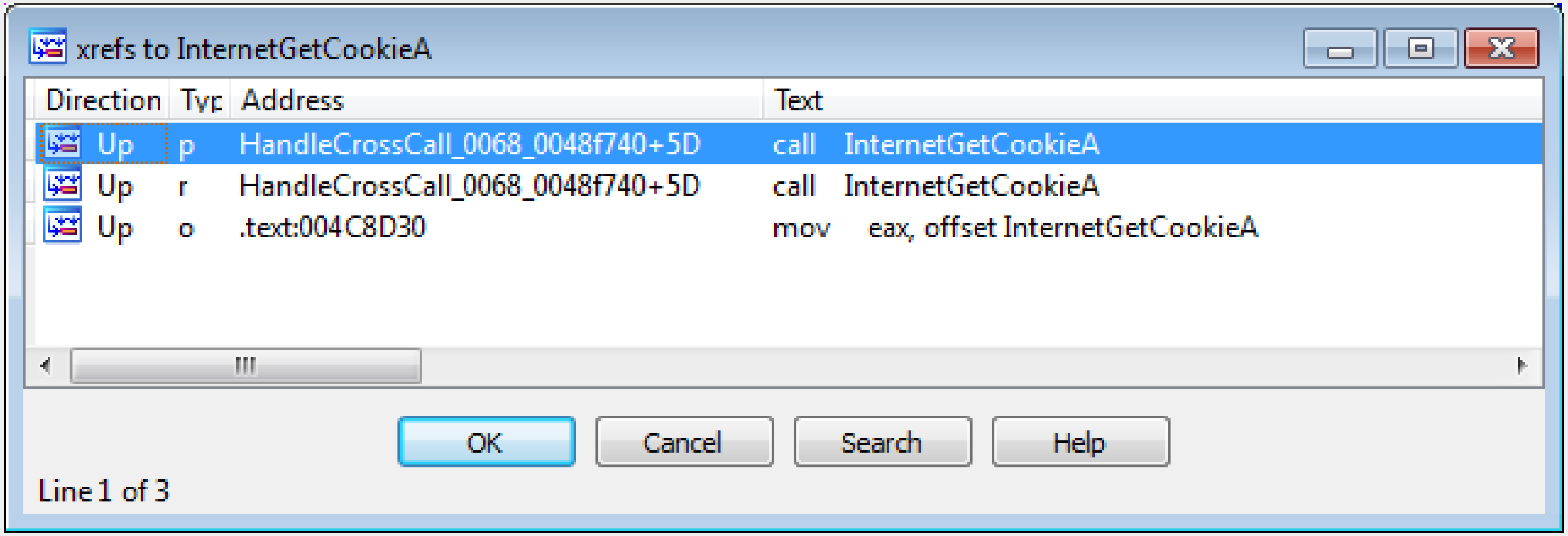
1. Finding one broker endpoint function
2. Find structure containing pointer to endpoint function
3. Find function responsible for adding this Cross Call
4. Find all Cross Call structures
5. Find all Cross Call endpoints and parameters

Step 1: Finding one Endpoint

- There are 107 imported functions that are only called directly from a Cross Call endpoint
- Examples:
  - InternetGetCookieA
  - DeleteSecurityContext
  - FreeCredentialsHandle
  - DeviceCapabilitiesW
  - DeviceCapabilitiesA

Step 1: Finding one Endpoint

- Find all Xrefs for InternetGetCookieA



Direction	Typ	Address	Text
Up	p	HandleCrossCall_0068_0048f740+5D	call InternetGetCookieA
Up	r	HandleCrossCall_0068_0048f740+5D	call InternetGetCookieA
Up	o	.text:004C8D30	mov eax, offset InternetGetCookieA

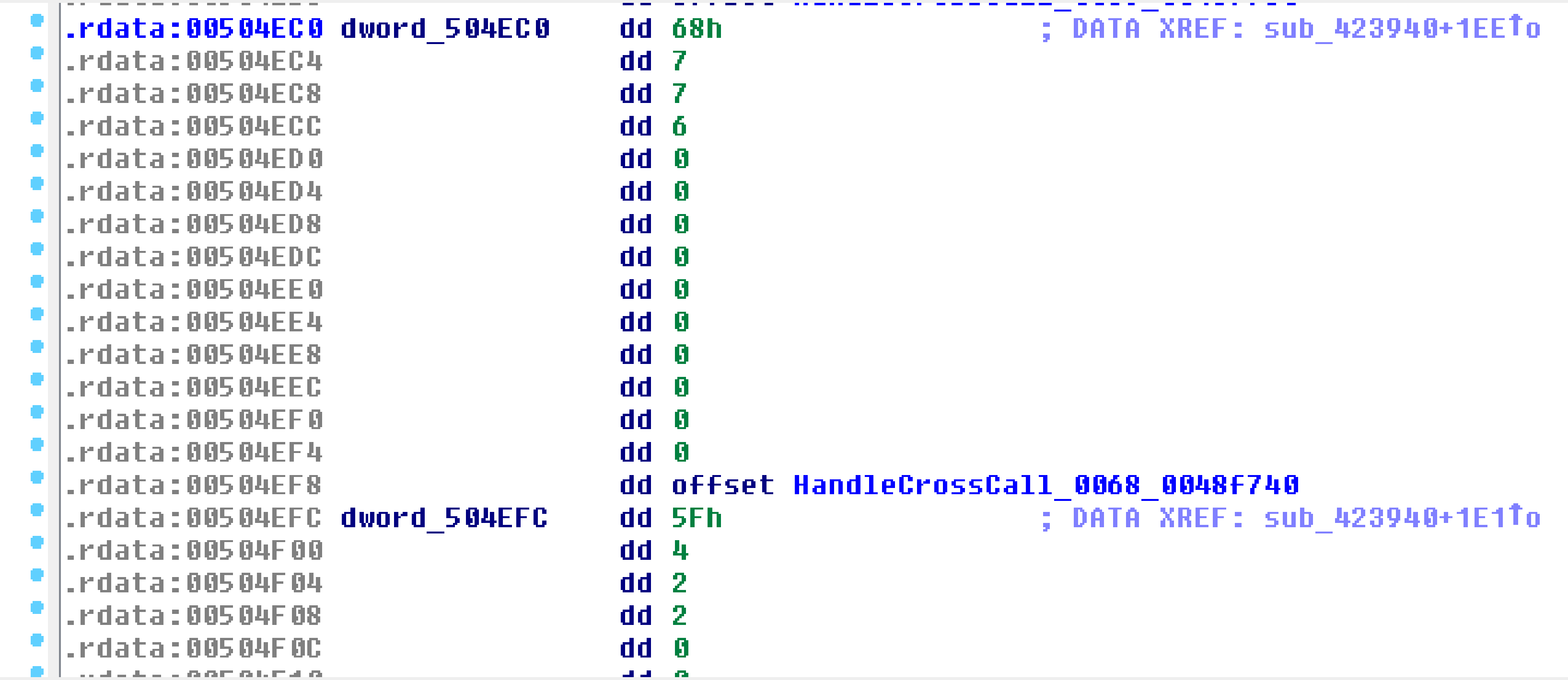
Line 1 of 3

Finding Broker Endpoints

1. Finding one broker endpoint function
2. **Find structure containing pointer to endpoint function**
3. Find function responsible for adding this Cross Call
4. Find all Cross Call structures
5. Find all Cross Call endpoints and parameters

Step 2: Find Cross Call Structure

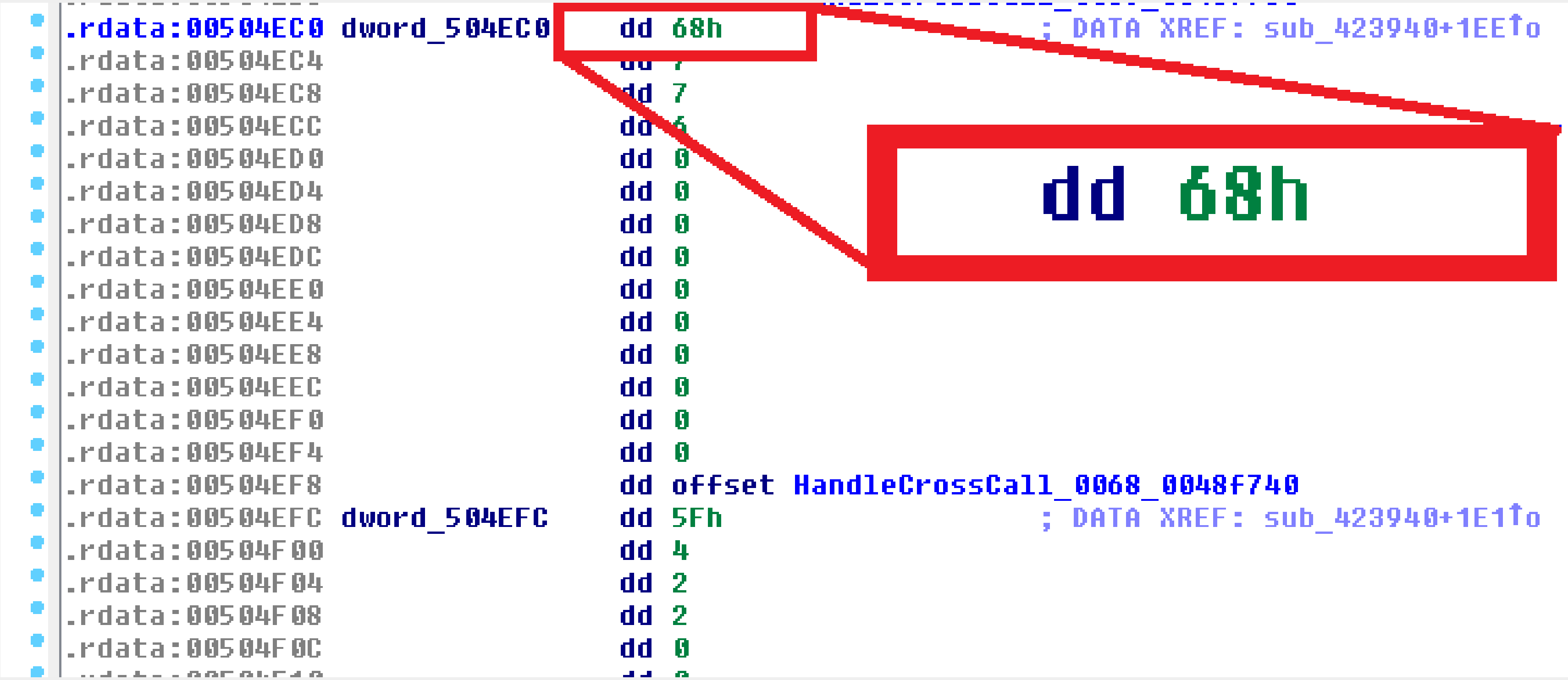
- Find Data Reference for the endpoint (only 1)



```
.rdata:00504EC0 dword_504EC0     dd 68h                  ; DATA XREF: sub_423940+1EE↑o
.rdata:00504EC4                  dd 7
.rdata:00504EC8                  dd 7
.rdata:00504ECC                  dd 6
.rdata:00504ED0                  dd 0
.rdata:00504ED4                  dd 0
.rdata:00504ED8                  dd 0
.rdata:00504EDC                  dd 0
.rdata:00504EE0                  dd 0
.rdata:00504EE4                  dd 0
.rdata:00504EE8                  dd 0
.rdata:00504EEC                  dd 0
.rdata:00504EF0                  dd 0
.rdata:00504EF4                  dd 0
.rdata:00504EF8                  dd offset HandleCrossCall_0068_0048f740
.rdata:00504EFC dword_504EFC     dd 5Fh                  ; DATA XREF: sub_423940+1E1↑o
.rdata:00504F00                  dd 4
.rdata:00504F04                  dd 2
.rdata:00504F08                  dd 2
.rdata:00504F0C                  dd 0
```

Cross Call Structure

- Cross Call ID

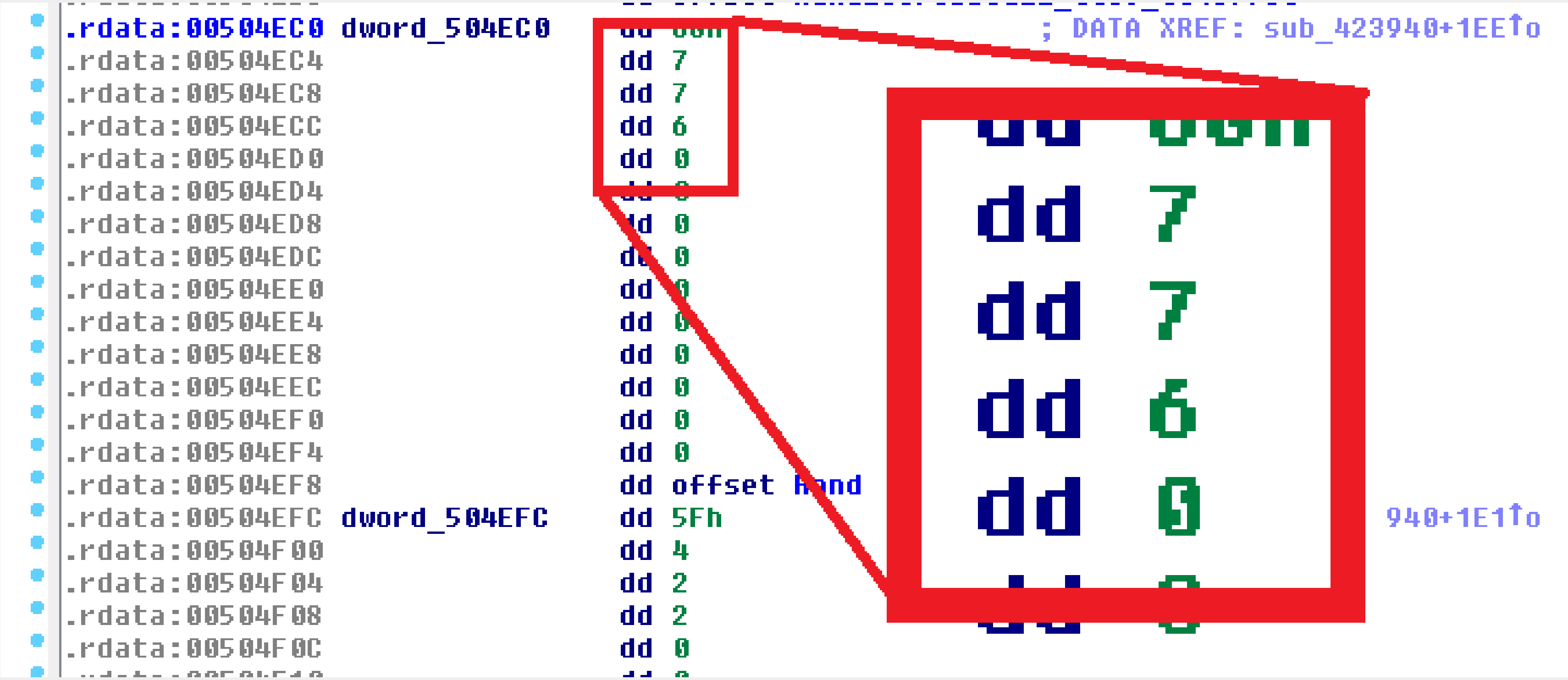


```
.rdata:00504EC0 dword_504EC0    dd 68h                         ; DATA XREF: sub_423940+1EE↑o
.rdata:00504EC4                 dd 7
.rdata:00504EC8                 dd 7
.rdata:00504ECC                 dd 6
.rdata:00504ED0                 dd 0
.rdata:00504ED4                 dd 0
.rdata:00504ED8                 dd 0
.rdata:00504EDC                 dd 0
.rdata:00504EE0                 dd 0
.rdata:00504EE4                 dd 0
.rdata:00504EE8                 dd 0
.rdata:00504EEC                 dd 0
.rdata:00504EF0                 dd 0
.rdata:00504EF4                 dd 0
.rdata:00504EF8                 dd offset HandleCrossCall_0068_0048f740
.rdata:00504EFC dword_504EFC    dd 5Fh                         ; DATA XREF: sub_423940+1E1↑o
.rdata:00504F00                 dd 4
.rdata:00504F04                 dd 2
.rdata:00504F08                 dd 2
.rdata:00504F0C                 dd 0
```

dd 68h

Cross Call Structure

- Parameters



```
.rdata:00504EC0 dword_504EC0     dd 0Bh                    ; DATA XREF: sub_423940+1EE↑o
.rdata:00504EC4                  dd 7
.rdata:00504EC8                  dd 7
.rdata:00504ECC                  dd 6
.rdata:00504ED0                  dd 0
.rdata:00504ED4                  dd 0
.rdata:00504ED8                  dd 0
.rdata:00504EDC                  dd 0
.rdata:00504EE0                  dd 0
.rdata:00504EE4                  dd 0
.rdata:00504EE8                  dd 0
.rdata:00504EEC                  dd 0
.rdata:00504EF0                  dd 0
.rdata:00504EF4                  dd 0
.rdata:00504EF8                  dd offset Hand
.rdata:00504EFC dword_504EFC     dd 5Fh                                          940+1E1↑o
.rdata:00504F00                  dd 4
.rdata:00504F04                  dd 2
.rdata:00504F08                  dd 2
.rdata:00504F0C                  dd 0
```

```
dd 0Bh
dd 7
dd 7
dd 6
dd 0
dd 0
```

Cross Call Parameters Types

Chromium code defines 6 valid Parameter types

```
enum ArgType {
  INVALID_TYPE = 0,
  WCHAR_TYPE,
  ULONG_TYPE,
  UNISTR_TYPE,
  VOIDPTR_TYPE,
  INPTR_TYPE,
  INOUTPTR_TYPE,
  LAST_TYPE
};
```

Adobe sandbox implementation adds two more

Cross Call Parameters Types

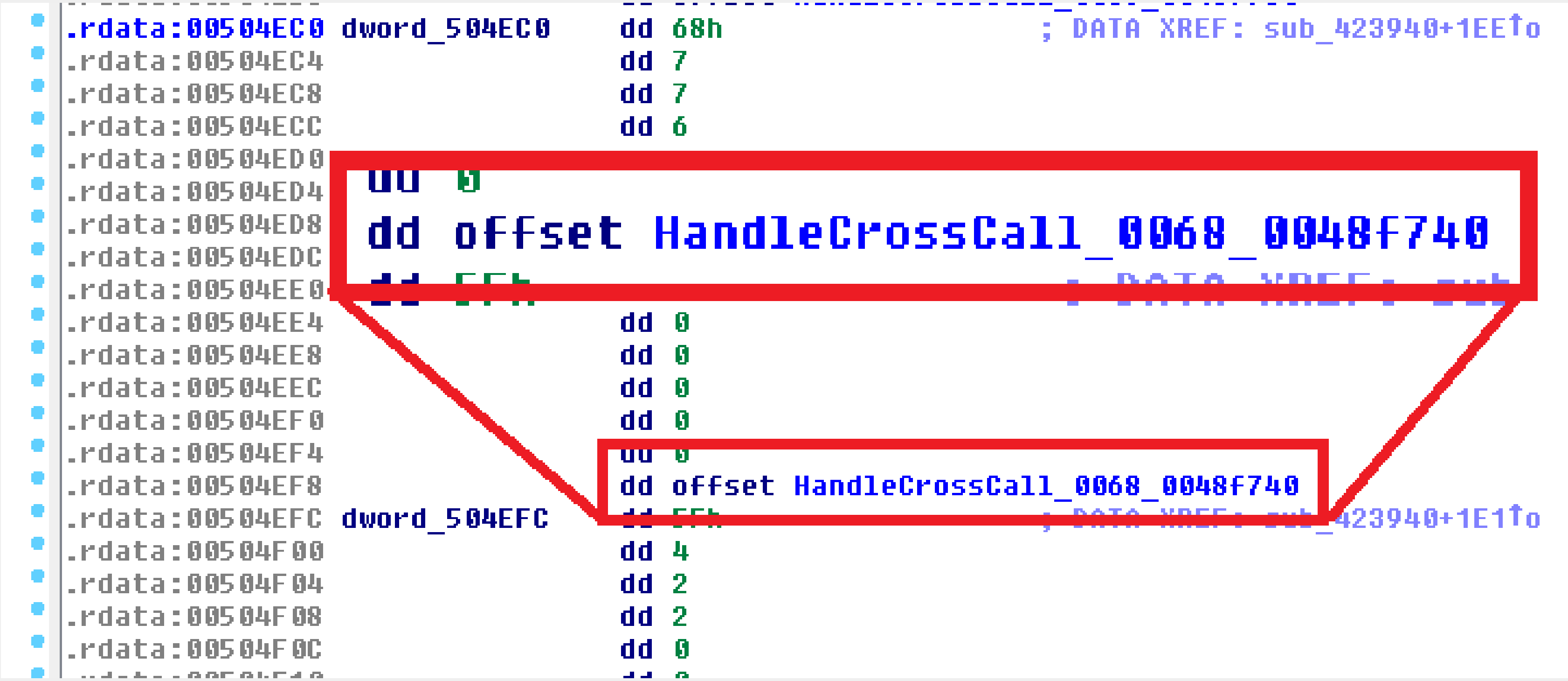
InternetGetCookie function (Windows)

```
BOOL InternetGetCookie(
  _In_     LPCTSTR lpszUrl,
  _In_     LPCTSTR lpszCookieName,
  _Out_    LPTSTR lpszCookieData,
  _Inout_  LPDWORD lpdwSize
);
```

We can now assume that Parameter Type 7 is a LPCTSTR

Cross Call Structure

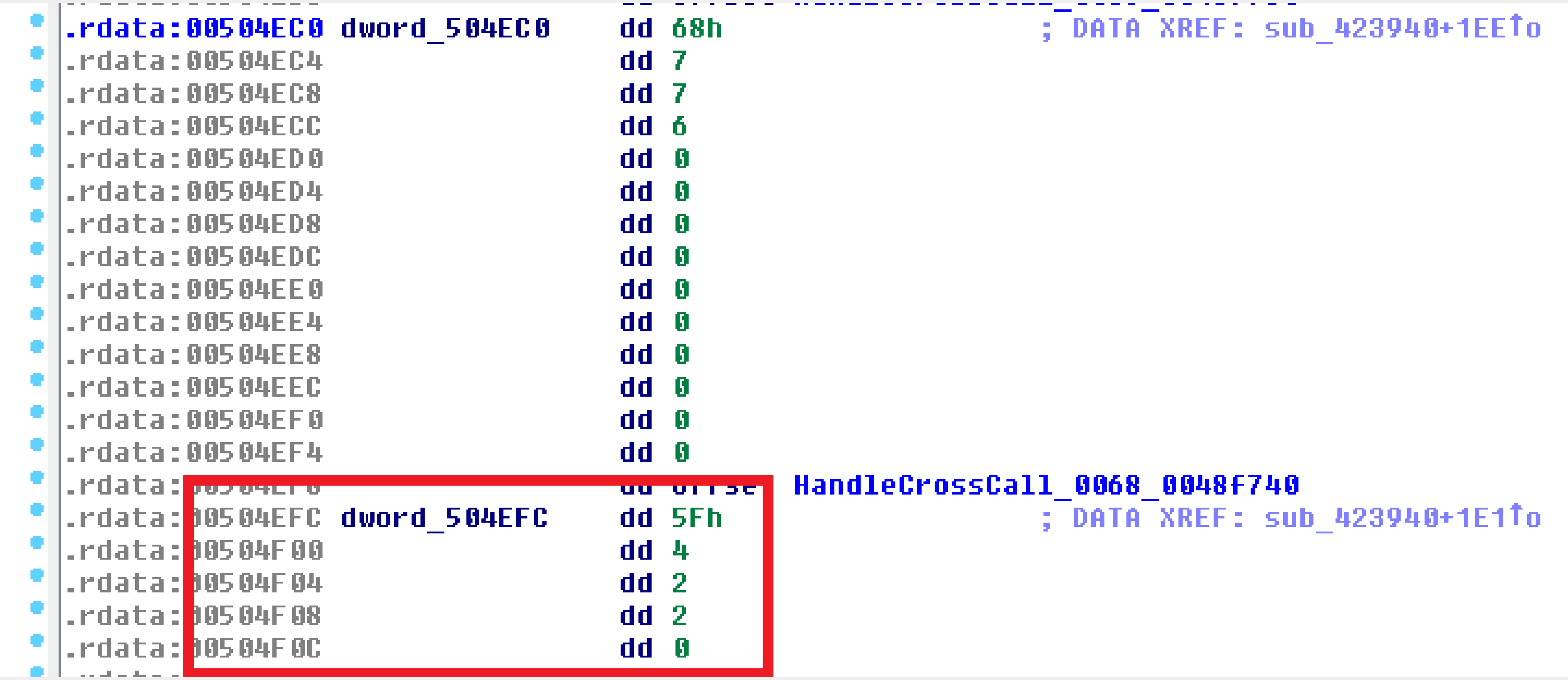
- Endpoint Function



```
.rdata:00504EC0 dword_504EC0    dd 68h                    ; DATA XREF: sub_423940+1EE↑o
.rdata:00504EC4                 dd 7
.rdata:00504EC8                 dd 7
.rdata:00504ECC                 dd 6
.rdata:00504ED0
.rdata:00504ED4
.rdata:00504ED8
.rdata:00504EDC
.rdata:00504EE0
.rdata:00504EE4                 dd 0
.rdata:00504EE8                 dd 0
.rdata:00504EEC                 dd 0
.rdata:00504EF0                 dd 0
.rdata:00504EF4                 dd 0
.rdata:00504EF8                 dd offset HandleCrossCall_0068_0048f740
.rdata:00504EFC dword_504EFC    dd FFh                    ; DATA XREF: sub_423940+1E1↑o
.rdata:00504F00                 dd 4
.rdata:00504F04                 dd 2
.rdata:00504F08                 dd 2
.rdata:00504F0C                 dd 0
```

```
dd 0
dd offset HandleCrossCall_0068_0048f740
dd FFh                    ; DATA XREF: sub
```

Step 3: Cross Call Adding Function



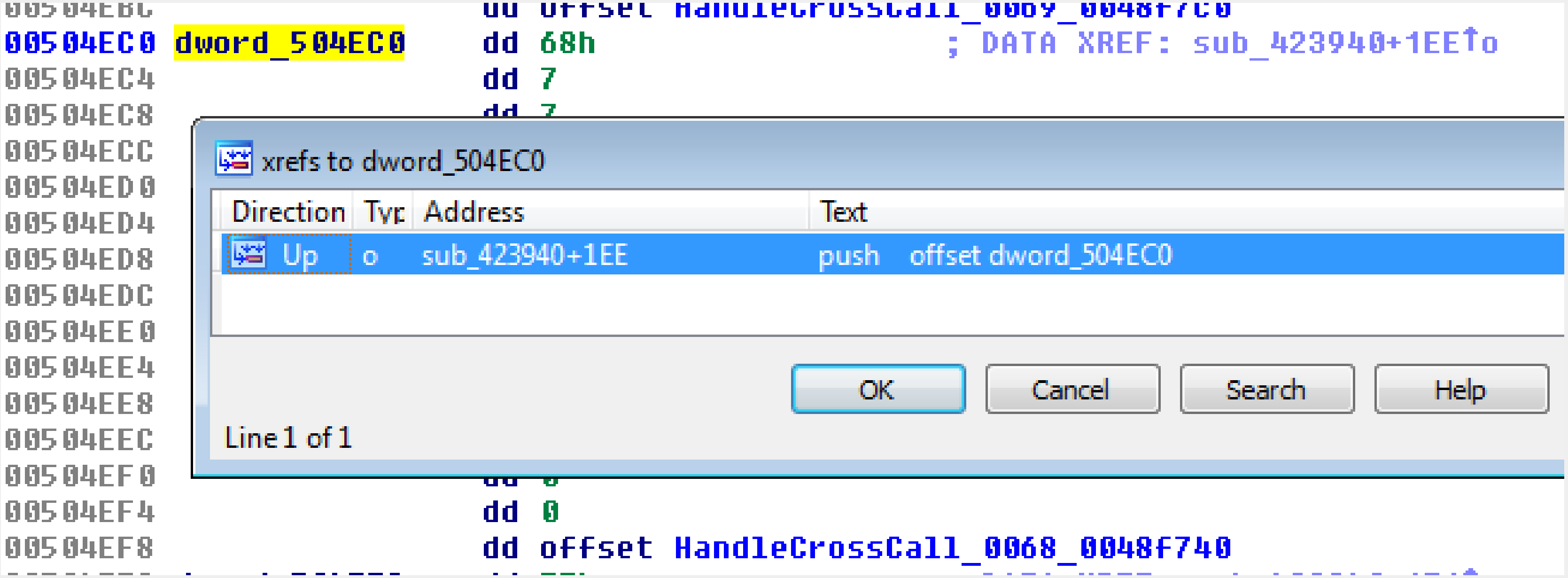
```
.rdata:00504EC0 dword_504EC0    dd 68h                                  ; DATA XREF: sub_423940+1EE↑o
.rdata:00504EC4                 dd 7
.rdata:00504EC8                 dd 7
.rdata:00504ECC                 dd 6
.rdata:00504ED0                 dd 0
.rdata:00504ED4                 dd 0
.rdata:00504ED8                 dd 0
.rdata:00504EDC                 dd 0
.rdata:00504EE0                 dd 0
.rdata:00504EE4                 dd 0
.rdata:00504EE8                 dd 0
.rdata:00504EEC                 dd 0
.rdata:00504EF0                 dd 0
.rdata:00504EF4                 dd 0
.rdata:00504EF8                 dd offset HandleCrossCall_0068_0048f740
.rdata:00504EFC dword_504EFC    dd 5Fh                                  ; DATA XREF: sub_423940+1E1↑o
.rdata:00504F00                 dd 4
.rdata:00504F04                 dd 2
.rdata:00504F08                 dd 2
.rdata:00504F0C                 dd 0
```

Finding Broker Endpoints

1. Finding one broker endpoint function
2. Find structure containing pointer to endpoint function
3. **Find function responsible for adding this Cross Call**
4. Find all Cross Call structures
5. Find all Cross Call endpoints and parameters

Step 3: Cross Call Adding Function

- Find the function adding Cross Calls

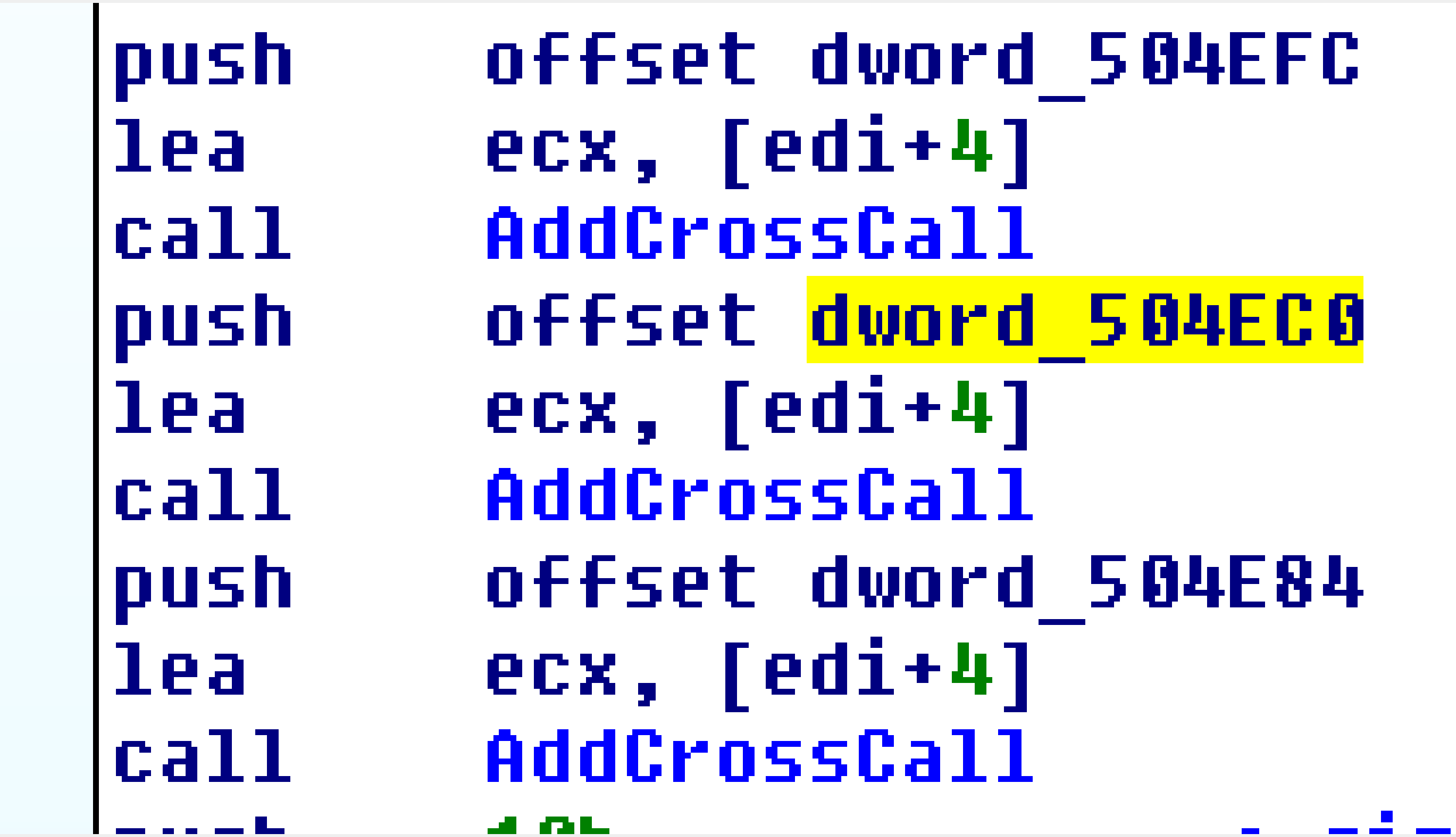


```
00504EBC                  dd offset HandleCrossCall_0069_0048F7C0
00504EC0 dword_504EC0     dd 68h                       ; DATA XREF: sub_423940+1EE↑o
00504EC4                  dd 7
00504EC8                  dd 7
00504ECC
00504ED0
00504ED4
00504ED8
00504EDC
00504EE0
00504EE4
00504EE8
00504EEC
00504EF0                  dd 0
00504EF4                  dd 0
00504EF8                  dd offset HandleCrossCall_0068_0048F740
```

xrefs to dword_504EC0

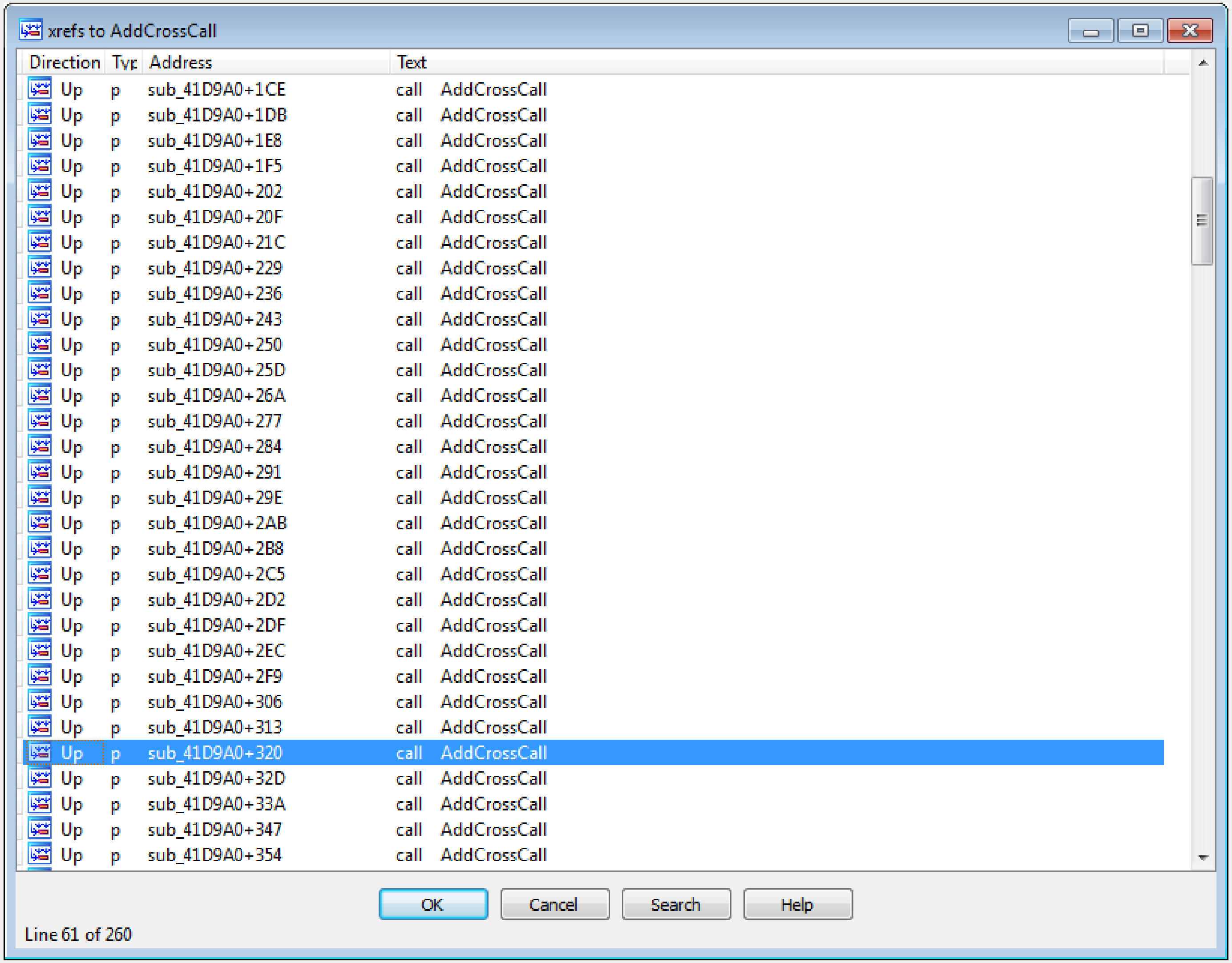
Direction	Typ	Address	Text
Up	o	sub_423940+1EE	push offset dword_504EC0

Step 3: Cross Call Adding Function



```
push    offset dword_504EFC
lea     ecx, [edi+4]
call    AddCrossCall
push    offset dword_504EC0
lea     ecx, [edi+4]
call    AddCrossCall
push    offset dword_504E84
lea     ecx, [edi+4]
call    AddCrossCall
```

Step 3: Cross Call Adding Function



xrefs to AddCrossCall

Direction	Typ	Address	Text
Up	p	sub_41D9A0+1CE	call AddCrossCall
Up	p	sub_41D9A0+1DB	call AddCrossCall
Up	p	sub_41D9A0+1E8	call AddCrossCall
Up	p	sub_41D9A0+1F5	call AddCrossCall
Up	p	sub_41D9A0+202	call AddCrossCall
Up	p	sub_41D9A0+20F	call AddCrossCall
Up	p	sub_41D9A0+21C	call AddCrossCall
Up	p	sub_41D9A0+229	call AddCrossCall
Up	p	sub_41D9A0+236	call AddCrossCall
Up	p	sub_41D9A0+243	call AddCrossCall
Up	p	sub_41D9A0+250	call AddCrossCall
Up	p	sub_41D9A0+25D	call AddCrossCall
Up	p	sub_41D9A0+26A	call AddCrossCall
Up	p	sub_41D9A0+277	call AddCrossCall
Up	p	sub_41D9A0+284	call AddCrossCall
Up	p	sub_41D9A0+291	call AddCrossCall
Up	p	sub_41D9A0+29E	call AddCrossCall
Up	p	sub_41D9A0+2AB	call AddCrossCall
Up	p	sub_41D9A0+2B8	call AddCrossCall
Up	p	sub_41D9A0+2C5	call AddCrossCall
Up	p	sub_41D9A0+2D2	call AddCrossCall
Up	p	sub_41D9A0+2DF	call AddCrossCall
Up	p	sub_41D9A0+2EC	call AddCrossCall
Up	p	sub_41D9A0+2F9	call AddCrossCall
Up	p	sub_41D9A0+306	call AddCrossCall
Up	p	sub_41D9A0+313	call AddCrossCall
Up	p	sub_41D9A0+320	call AddCrossCall
Up	p	sub_41D9A0+32D	call AddCrossCall
Up	p	sub_41D9A0+33A	call AddCrossCall
Up	p	sub_41D9A0+347	call AddCrossCall
Up	p	sub_41D9A0+354	call AddCrossCall

Line 61 of 260

Finding Broker Endpoints

1. Finding one broker endpoint function
2. Find structure containing pointer to endpoint function
3. Find function responsible for adding this Cross Call
4. **Find all Cross Call structures**
5. Find all Cross Call endpoints and parameters

Step 4: Find all Cross Call Structures

- Get all the Xrefs to the AddCrossCall function
- Find the parameter each time the function is called

Finding Broker Endpoints

1. Finding one broker endpoint function
2. Find structure containing pointer to endpoint function
3. Find function responsible for adding this Cross Call
4. Find all Cross Call structures
5. **Find all Cross Call endpoints and parameters**

Step 5: Done

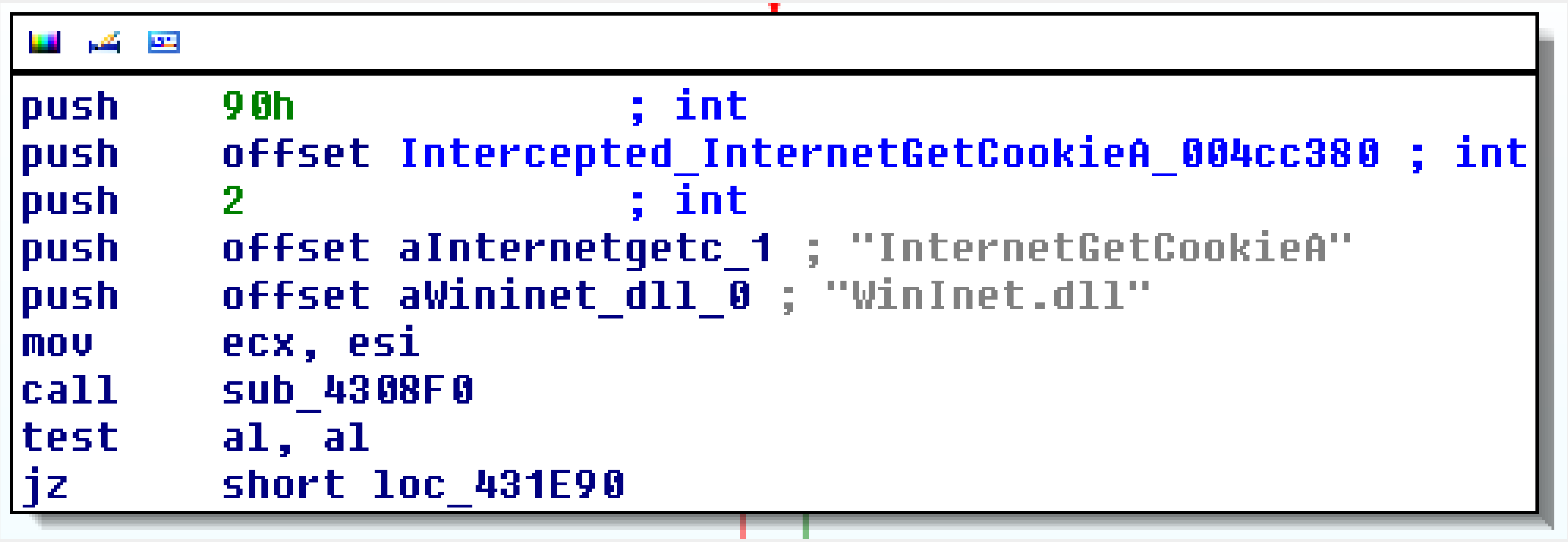
- You now have a list of 260 functions in AcroRd32.exe that handle Cross Calls inside the Broker
- You know the type of arguments to each function
- Time to reverse and find a working escape

Intercepted Windows API Functions

- AcroRD32.exe also intercepts a lot of default windows API functions
- Most of the intercepted functions are redirected to a Cross Call
- Matching intercepted functions with Cross Call IDs would make our work easier

Intercepted Windows API Functions

- One function responsible for enabling all API interceptions



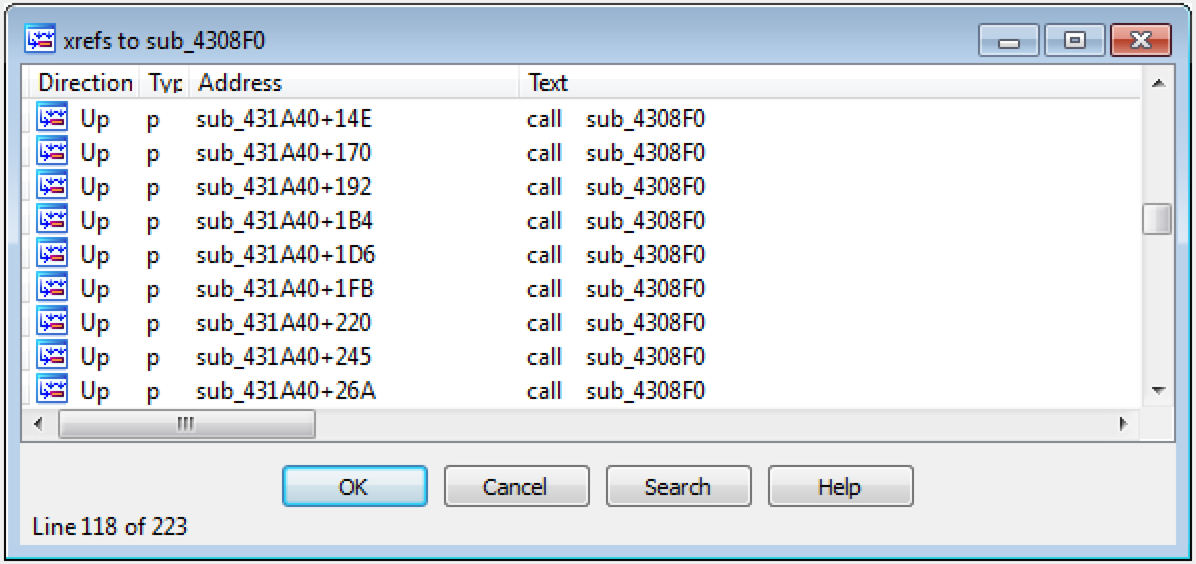
```
push    90h                     ; int
push    offset Intercepted_InternetGetCookieA_004cc380 ; int
push    2                       ; int
push    offset aInternetgetc_1 ; "InternetGetCookieA"
push    offset aWininet_dll_0 ; "WinInet.dll"
mov     ecx, esi
call    sub_4308F0
test    al, al
jz      short loc_431E90
```

Intercepted Windows API Functions

- Function parameters are
  - Name of the .dll file
  - Function Name
  - Interception type
  - Intercept Function
  - Unknown

Intercepted Windows API Functions

1. Find all calls to this function
2. Find all Intercepted Function Names
3. Link Intercept Function to Cross Call IDs



xrefs to sub_4308F0

Direction	Typ	Address	Text
Up	p	sub_431A40+14E	call sub_4308F0
Up	p	sub_431A40+170	call sub_4308F0
Up	p	sub_431A40+192	call sub_4308F0
Up	p	sub_431A40+1B4	call sub_4308F0
Up	p	sub_431A40+1D6	call sub_4308F0
Up	p	sub_431A40+1FB	call sub_4308F0
Up	p	sub_431A40+220	call sub_4308F0
Up	p	sub_431A40+245	call sub_4308F0
Up	p	sub_431A40+26A	call sub_4308F0

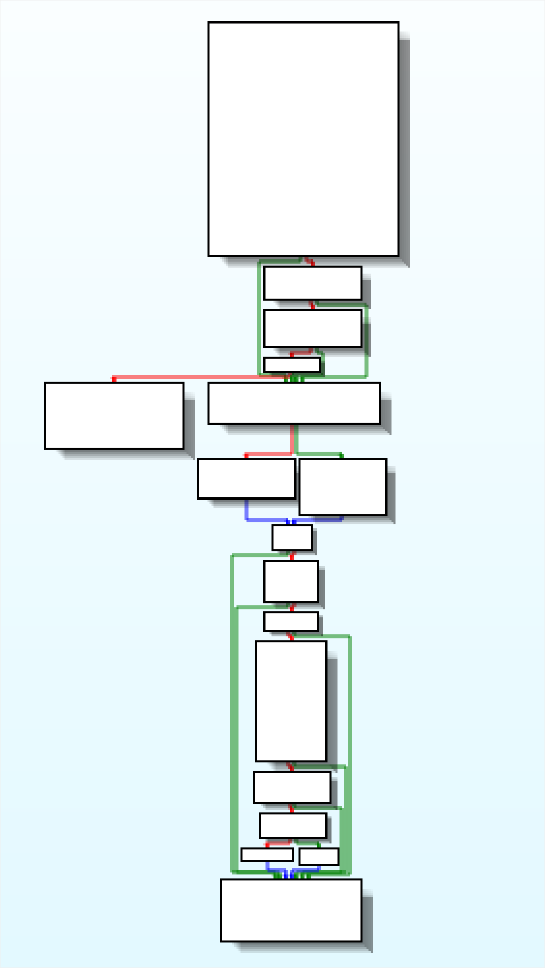
Line 118 of 223

Find Cross Call ID

- Most Intercept Functions go straight into a Cross Call
- Finding Cross Call ID can be (somewhat) automated
- Not all Intercept Function actually end in a Cross Call

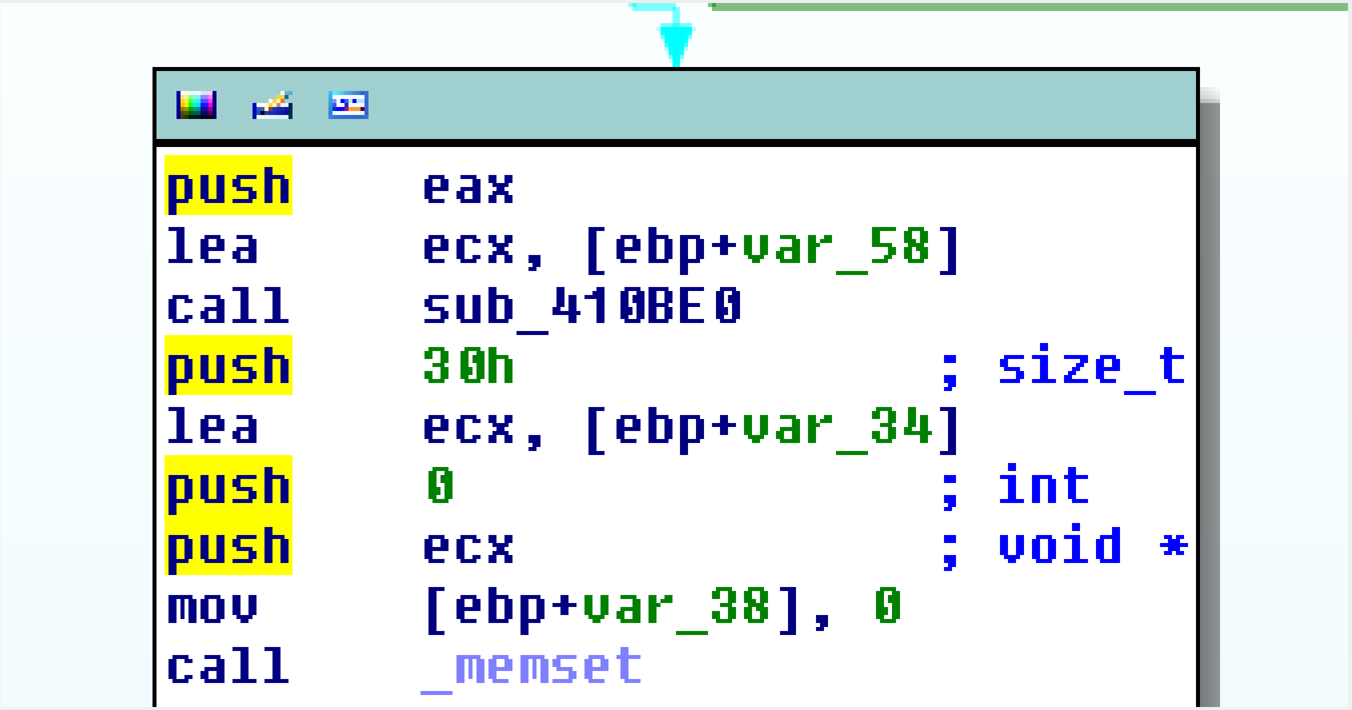
Intercept Functions

InternetOpenA



Finding the Cross Call ID

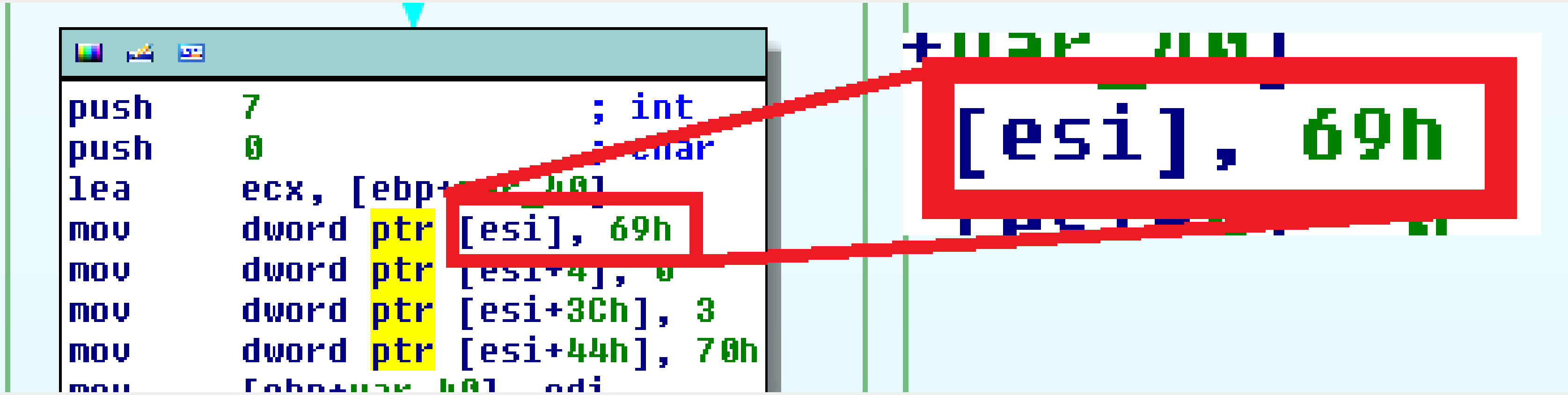
- A 0x30 sized structure is initialized



```
push    eax
lea     ecx, [ebp+var_58]
call    sub_410BE0
push    30h                 ; size_t
lea     ecx, [ebp+var_34]
push    0                   ; int
push    ecx                 ; void *
mov     [ebp+var_38], 0
call    _memset
```

Finding the Cross Call ID

- Cross Call ID is first Dword in the structure

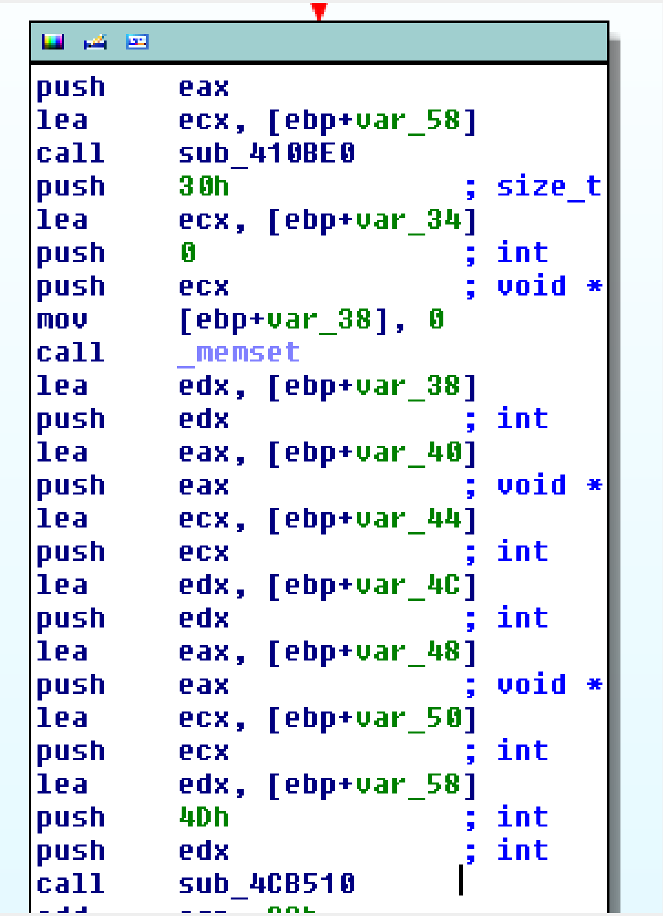


```
push    7                         ; int
push    0                         ; char
lea     ecx, [ebp+var_40]
mov     dword ptr [esi], 69h
mov     dword ptr [esi+4], 0
mov     dword ptr [esi+3Ch], 3
mov     dword ptr [esi+44h], 70h
mov     [ebp+var_40], edi
```

[esi], 69h

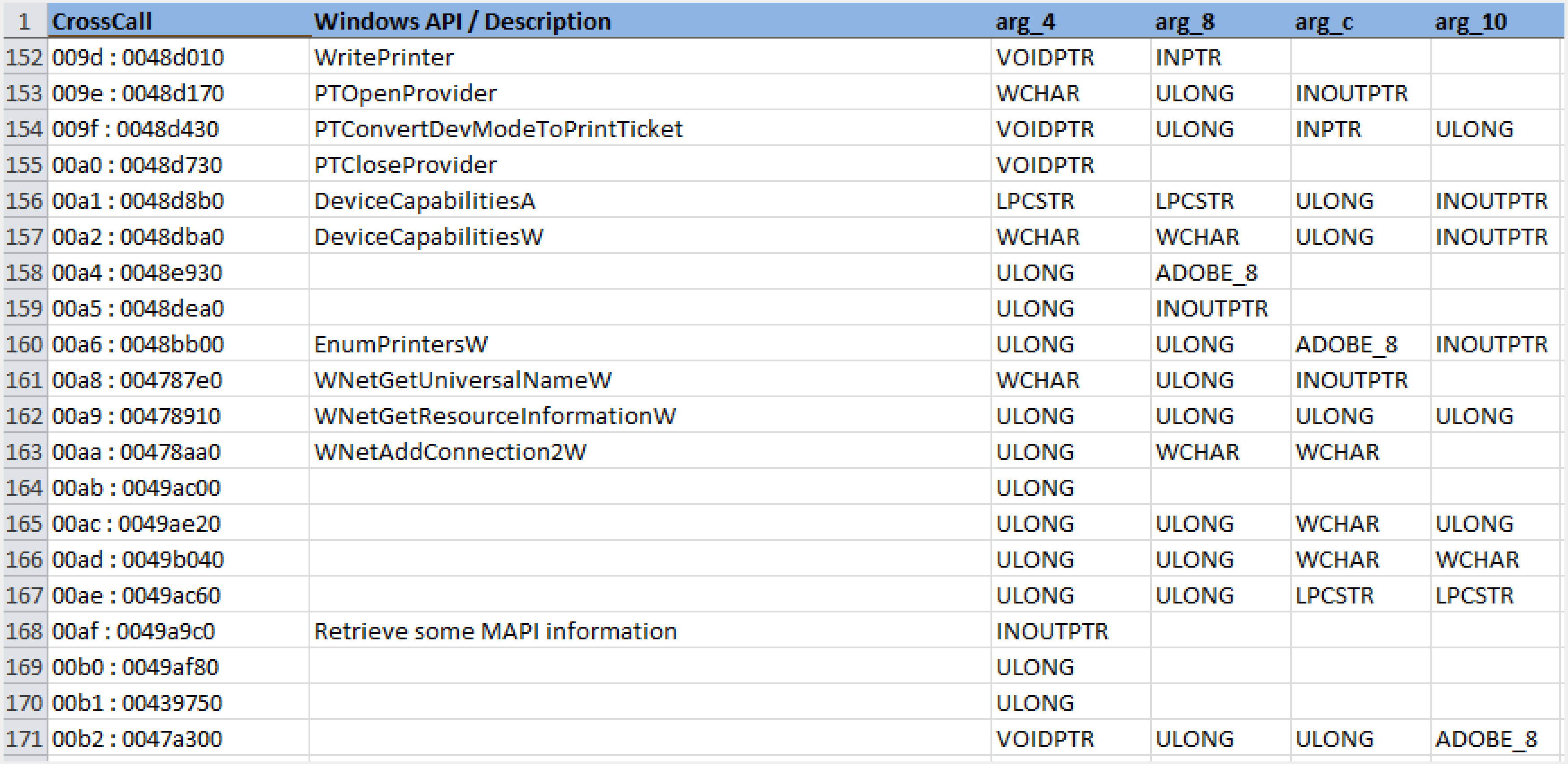
Finding the Cross Call ID

- OR Cross Call ID is pushed as 2nd Argument to another Function



```
push     eax
lea      ecx, [ebp+var_58]
call     sub_410BE0
push     30h                  ; size_t
lea      ecx, [ebp+var_34]
push     0                    ; int
push     ecx                  ; void *
mov      [ebp+var_38], 0
call     _memset
lea      edx, [ebp+var_38]
push     edx                  ; int
lea      eax, [ebp+var_40]
push     eax                  ; void *
lea      ecx, [ebp+var_44]
push     ecx                  ; int
lea      edx, [ebp+var_4C]
push     edx                  ; int
lea      eax, [ebp+var_48]
push     eax                  ; void *
lea      ecx, [ebp+var_50]
push     ecx                  ; int
lea      edx, [ebp+var_58]
push     4Dh                  ; int
push     edx                  ; int
call     sub_4CB510
```

Adobe Cross Call list



1	CrossCall	Windows API / Description	arg_4	arg_8	arg_c	arg_10
152	009d : 0048d010	WritePrinter	VOIDPTR	INPTR		
153	009e : 0048d170	PTOpenProvider	WCHAR	ULONG	INOUTPTR	
154	009f : 0048d430	PTConvertDevModeToPrintTicket	VOIDPTR	ULONG	INPTR	ULONG
155	00a0 : 0048d730	PTCloseProvider	VOIDPTR			
156	00a1 : 0048d8b0	DeviceCapabilitiesA	LPCSTR	LPCSTR	ULONG	INOUTPTR
157	00a2 : 0048dba0	DeviceCapabilitiesW	WCHAR	WCHAR	ULONG	INOUTPTR
158	00a4 : 0048e930		ULONG	ADOBE_8		
159	00a5 : 0048dea0		ULONG	INOUTPTR		
160	00a6 : 0048bb00	EnumPrintersW	ULONG	ULONG	ADOBE_8	INOUTPTR
161	00a8 : 004787e0	WNetGetUniversalNameW	WCHAR	ULONG	INOUTPTR	
162	00a9 : 00478910	WNetGetResourceInformationW	ULONG	ULONG	ULONG	ULONG
163	00aa : 00478aa0	WNetAddConnection2W	ULONG	WCHAR	WCHAR	
164	00ab : 0049ac00		ULONG			
165	00ac : 0049ae20		ULONG	ULONG	WCHAR	ULONG
166	00ad : 0049b040		ULONG	ULONG	WCHAR	WCHAR
167	00ae : 0049ac60		ULONG	ULONG	LPCSTR	LPCSTR
168	00af : 0049a9c0	Retrieve some MAPI information	INOUTPTR			
169	00b0 : 0049af80		ULONG			
170	00b1 : 00439750		ULONG			
171	00b2 : 0047a300		VOIDPTR	ULONG	ULONG	ADOBE_8

Endpoint Functions

- Arg_0 is IPCInfo structure

```
struct IPCInfo {
  int ipc_tag;
  const ClientInfo* client_info;
  CrossCallReturn return_info;
};

struct ClientInfo {
  HANDLE process;
  HANDLE job_object;
  DWORD process_id;
};
```

Restrictions

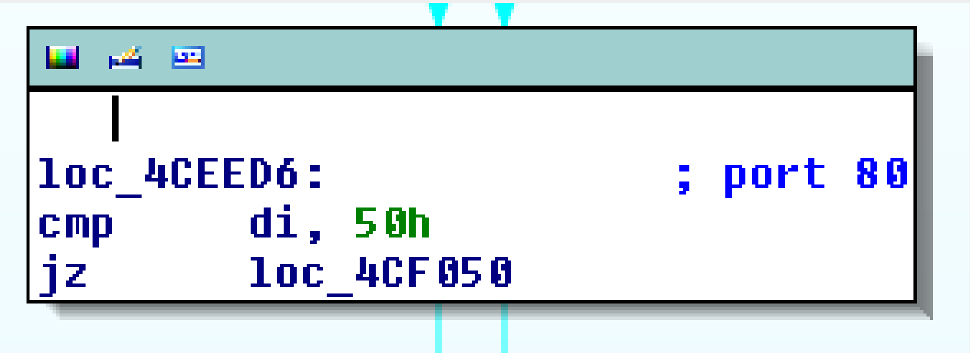
- The Broker performs a lot of sanity checks
  - Dialog boxes asking for permissions
  - Interesting API functions already ‘blocked’ (InternetSetStatusCallback for example)
  - File Policy tests
- Attack surface is still pretty big
- Adobe 0-Day used 2 Intercepted API Calls to trigger a heap buffer overflow

Testing Cross Calls

- We can fuzz the Endpoints
  - From Sandboxed process
  - From Broker process
- Need to be sure we have all structures correct

Testing Cross Calls

- Testing Intercepted API calls is easy
- Need a little reversing to make sure you end up at the actual Cross Call



```
loc_4CEED6:                     ; port 80
cmp     di, 50h
jz      loc_4CF050
```

InternetConnectA

- We can patch this in the Client Process for easy testing

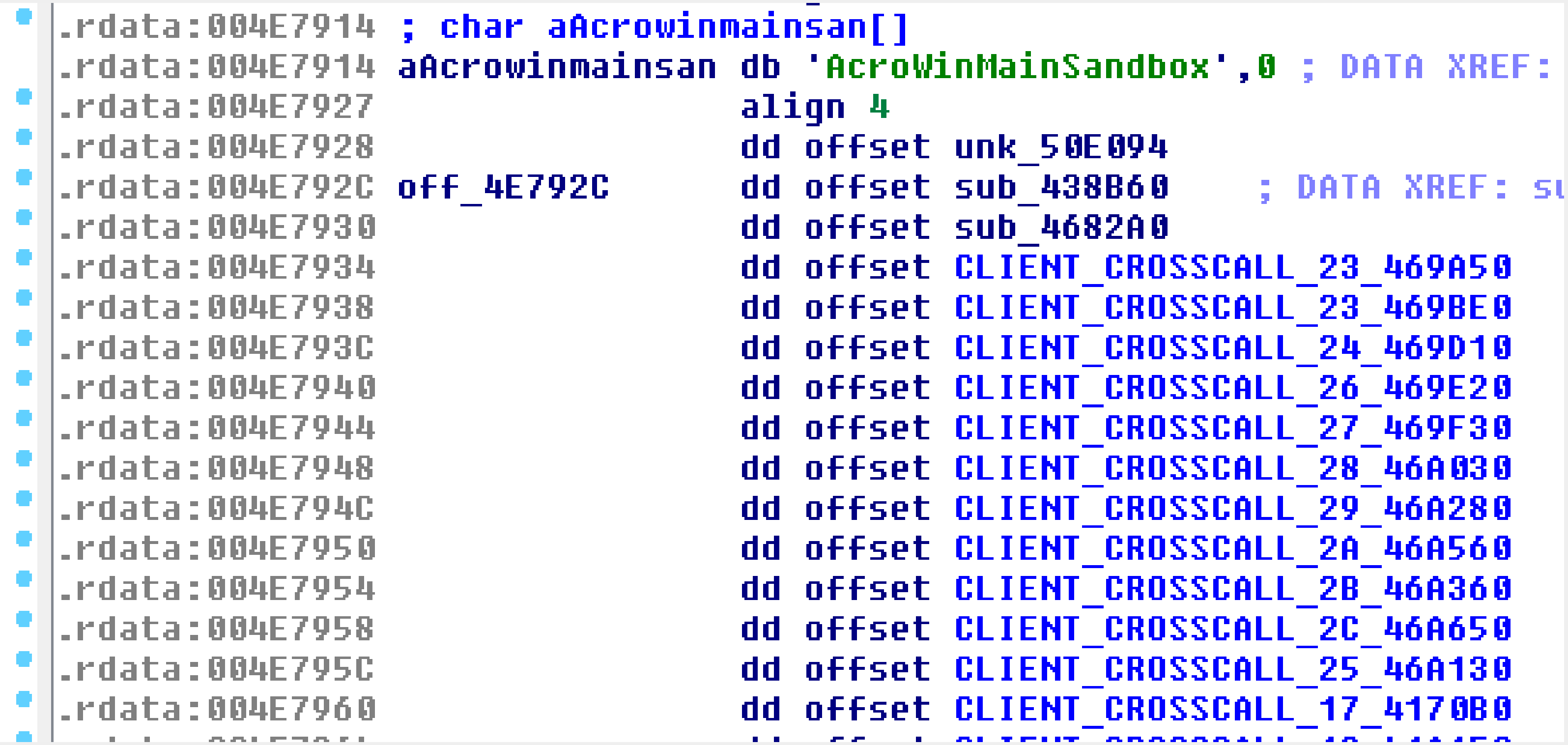
Testing Cross Calls

- Non Intercepted API Cross Calls have a wrapper function in AcroRD32.exe
- Wrapper functions do not require complex structures
- Might need some additional reversing to get the parameters correct

Testing Cross Calls

- String ‘AcroWinMainSandbox’ is just above a list of Cross Call Wrappers in ArcoRd32.exe
- Quick look through the functions gives away the Cross Call ID
- This can be linked back to the known parameters for the Cross Calls

Testing Cross Calls



```
.rdata:004E7914 ; char aAcrowinmainsan[]
.rdata:004E7914 aAcrowinmainsan db 'AcroWinMainSandbox',0 ; DATA XREF:
.rdata:004E7927                 align 4
.rdata:004E7928                 dd offset unk_50E094
.rdata:004E792C off_4E792C      dd offset sub_438B60     ; DATA XREF: su
.rdata:004E7930                 dd offset sub_4682A0
.rdata:004E7934                 dd offset CLIENT_CROSSCALL_23_469A50
.rdata:004E7938                 dd offset CLIENT_CROSSCALL_23_469BE0
.rdata:004E793C                 dd offset CLIENT_CROSSCALL_24_469D10
.rdata:004E7940                 dd offset CLIENT_CROSSCALL_26_469E20
.rdata:004E7944                 dd offset CLIENT_CROSSCALL_27_469F30
.rdata:004E7948                 dd offset CLIENT_CROSSCALL_28_46A030
.rdata:004E794C                 dd offset CLIENT_CROSSCALL_29_46A280
.rdata:004E7950                 dd offset CLIENT_CROSSCALL_2A_46A560
.rdata:004E7954                 dd offset CLIENT_CROSSCALL_2B_46A360
.rdata:004E7958                 dd offset CLIENT_CROSSCALL_2C_46A650
.rdata:004E795C                 dd offset CLIENT_CROSSCALL_25_46A130
.rdata:004E7960                 dd offset CLIENT_CROSSCALL_17_4170B0
```

Testing Cross Calls

- Injecting python interpreter into sandboxed process
  - Only injects into processes running with LOW Integrity
- Run python scripts inside the sandbox
- Allows for easy Cross Calls testing

Bypassing memory ASLR (heapspray)

- You can ‘heapspray’ from the Client into the broker
- Broker will call ReadProcessMemory to read large arguments from some Cross Calls
- 0-Day discovered in the wild used this to bypass memory ASLR
- Creating allocation bigger than 0x80000 will result in (partly) predictable location

Cross Call Demos

- Cross Call ID 0x49
- Arguments:
  - WChar

Demo Cross Call 0x49

- Not a sandbox escape
- Only opens .txt .pdf and .log files with the correct handler

Demo 1

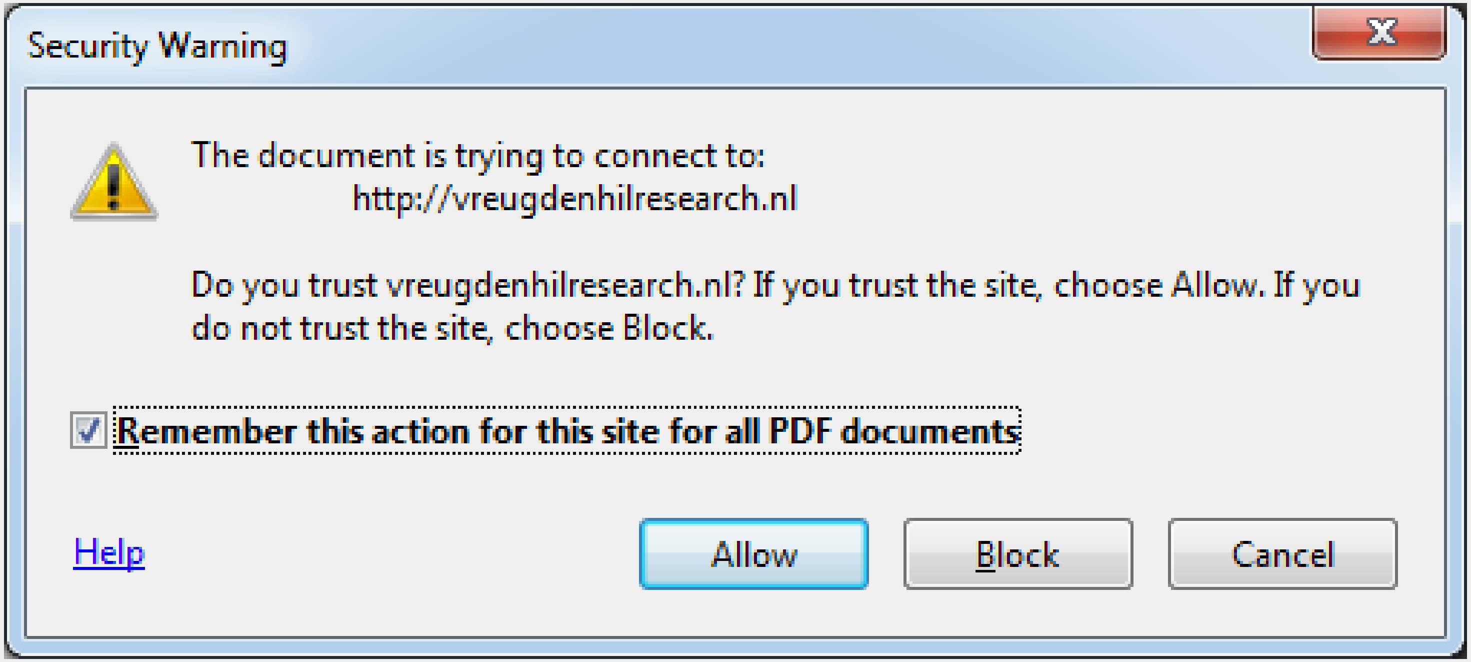
- This issue has been patched in the latest version

How did that work

- Uses Adobe Reader ability to open URLs
- Evades some restrictions
- Works best when Chrome or Firefox are set as the default browser
- Cross Call ID 0x46
- Parameters
  - WChar URL
  - ULong

Cross Call 0x46

- When trying to open a link from a pdf the following warning is shown



Security Warning

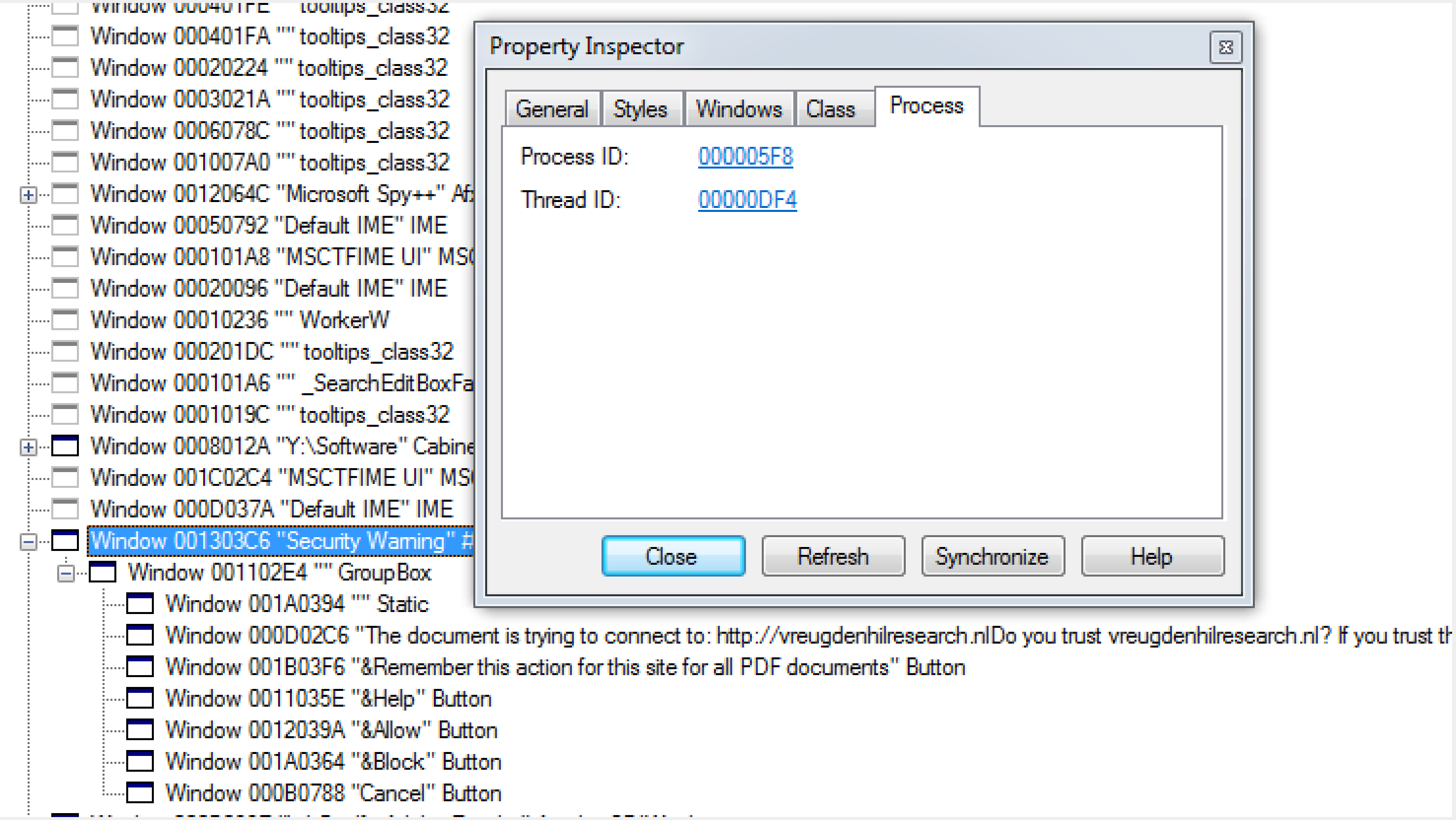
The document is trying to connect to:
http://vreugdenhilresearch.nl

Do you trust vreugdenhilresearch.nl? If you trust the site, choose Allow. If you do not trust the site, choose Block.

☑ Remember this action for this site for all PDF documents

Cross Call 0x46

- Microsoft Spy++ information on this window



```
Window 000401FE "" tooltips_class32
Window 000401FA "" tooltips_class32
Window 00020224 "" tooltips_class32
Window 0003021A "" tooltips_class32
Window 0006078C "" tooltips_class32
Window 001007A0 "" tooltips_class32
Window 0012064C "Microsoft Spy++" Afx
Window 00050792 "Default IME" IME
Window 000101A8 "MSCTFIME UI" MSC
Window 00020096 "Default IME" IME
Window 00010236 "" WorkerW
Window 000201DC "" tooltips_class32
Window 000101A6 "" _SearchEditBoxFa
Window 0001019C "" tooltips_class32
Window 0008012A "Y:\Software" Cabine
Window 001C02C4 "MSCTFIME UI" MSC
Window 000D037A "Default IME" IME
Window 001303C6 "Security Warning" #
  Window 001102E4 "" GroupBox
    Window 001A0394 "" Static
    Window 000D02C6 "The document is trying to connect to: http://vreugdenhilresearch.nlDo you trust vreugdenhilresearch.nl? If you trust th
    Window 001B03F6 "&Remember this action for this site for all PDF documents" Button
    Window 0011035E "&Help" Button
    Window 0012039A "&Allow" Button
    Window 001A0364 "&Block" Button
    Window 000B0788 "Cancel" Button
```

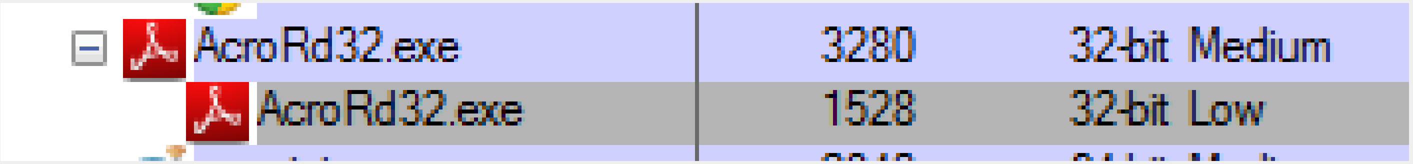
Property Inspector

Process ID: 000005F8

Thread ID: 00000DF4

Cross Call 0x46

- PID 0x5F8 = 1528



⊟ AcroRd32.exe	3280	32-bit Medium
AcroRd32.exe	1528	32-bit Low

- Dialog belongs to sandboxes process and can be circumvented
- Same with the URL escape, this happens in the sandboxed process

Cross Call 0x46

- We can send random strings to the Broker as argument for this Cross Call
- Sanity checks performed
  - PathIsURLW
  - Get default ‘open’ handler for ‘http’
  - ShellExecuteW
- Parameters are NOT quoted

Cross Call 0x46

- PathIsURLW doesn’t care
  - Anything that matches ^ASCII+:ASCII will pass
- Chrome.exe doesn’t care
  - Invalid parameters are ignore
  - Whitespace used as parameter delimiter
- Firefox.exe doesn’t care (enough)
- iexplore.exe does care
  - Code exec still possible but a lot harder

Cross Call 0x46

- Code exec with Chrome.exe

```
Chrome.exe
--a:b=1
--type=plugin
--plugin-path=c:\dr\evil.dll
```

- Code Exec with Firefox.exe

```
Firefox.exe
-a:b
-profile “profile”
```

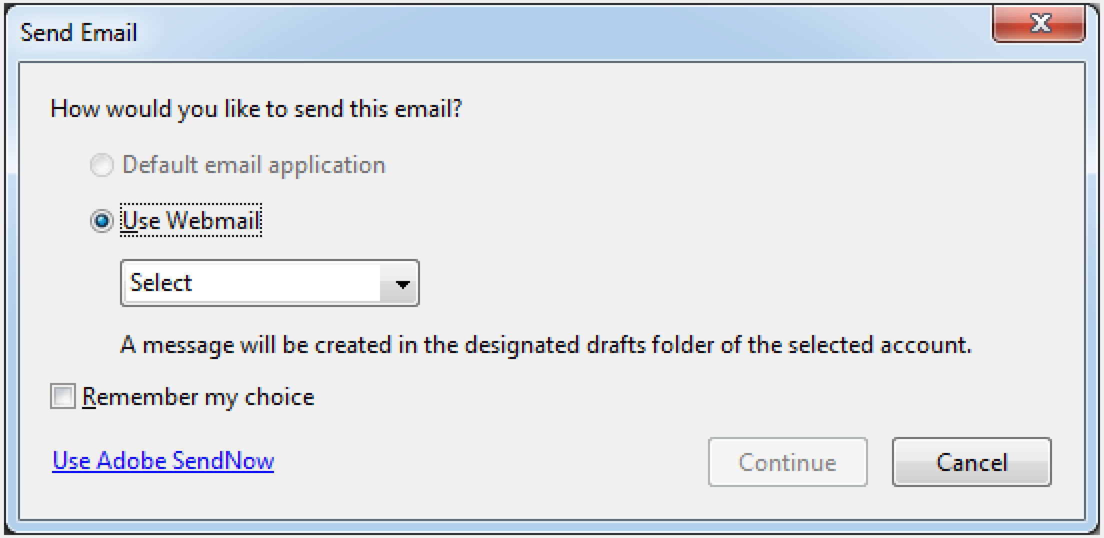
Cross Call 0x46

- This Issue has been patched
- Broker code now contains a call to UrlCanonicalize

Demo 2

What happened there?

- Cross Call 0x107 is being used
- This is normally used to login a webmail account



Send Email

How would you like to send this email?

○ Default email application

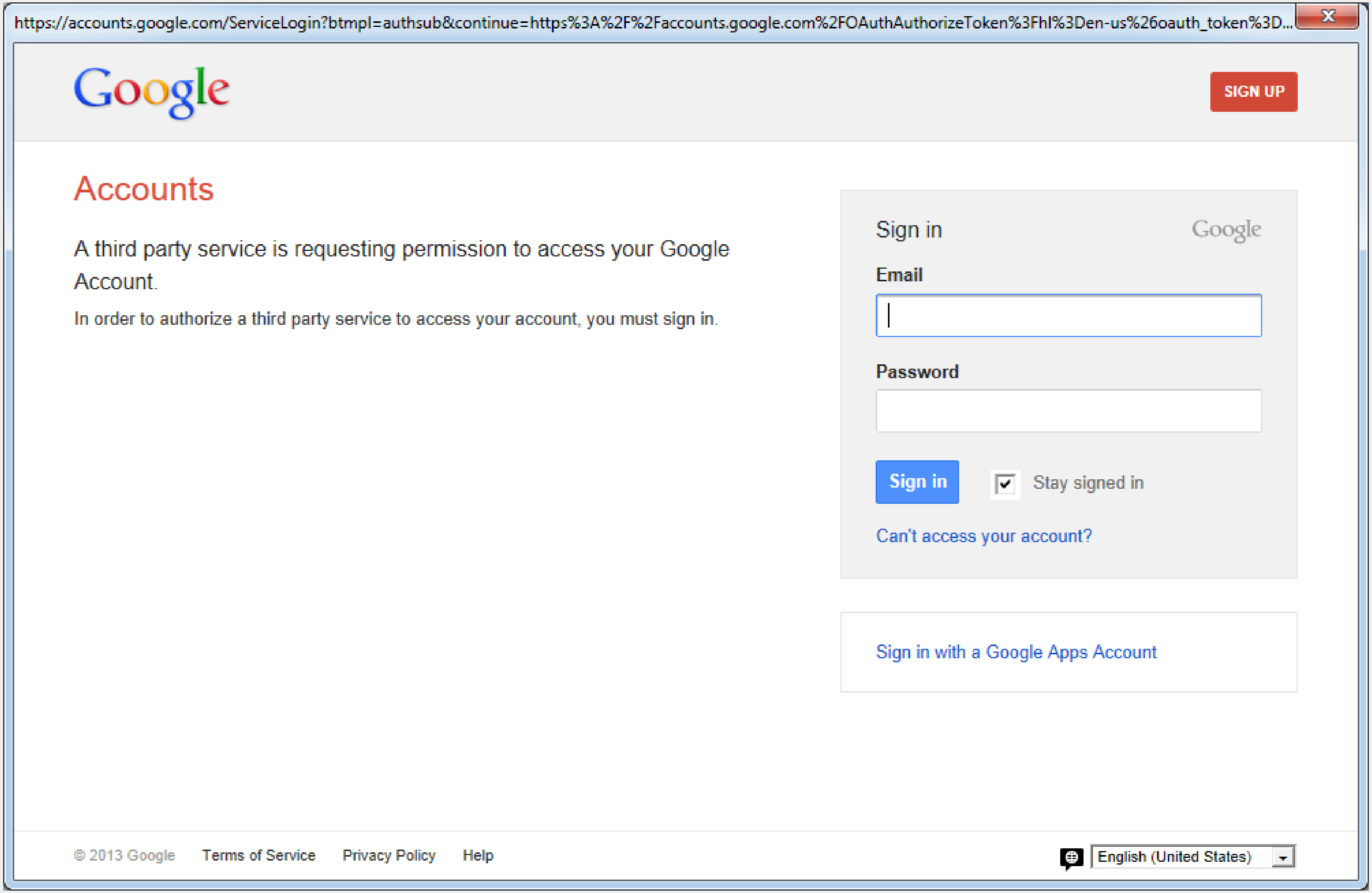
● Use Webmail

Select

A message will be created in the designated drafts folder of the selected account.

□ Remember my choice

Use Adobe SendNow



https://accounts.google.com/ServiceLogin?btmpl=authsub&continue=https%3A%2F%2Faccounts.google.com%2FOAuthAuthorizeToken%3Fhl%3Den-us%26oauth_token%3D...

Accounts

A third party service is requesting permission to access your Google Account.

In order to authorize a third party service to access your account, you must sign in.

Sign in

Email:

Password:

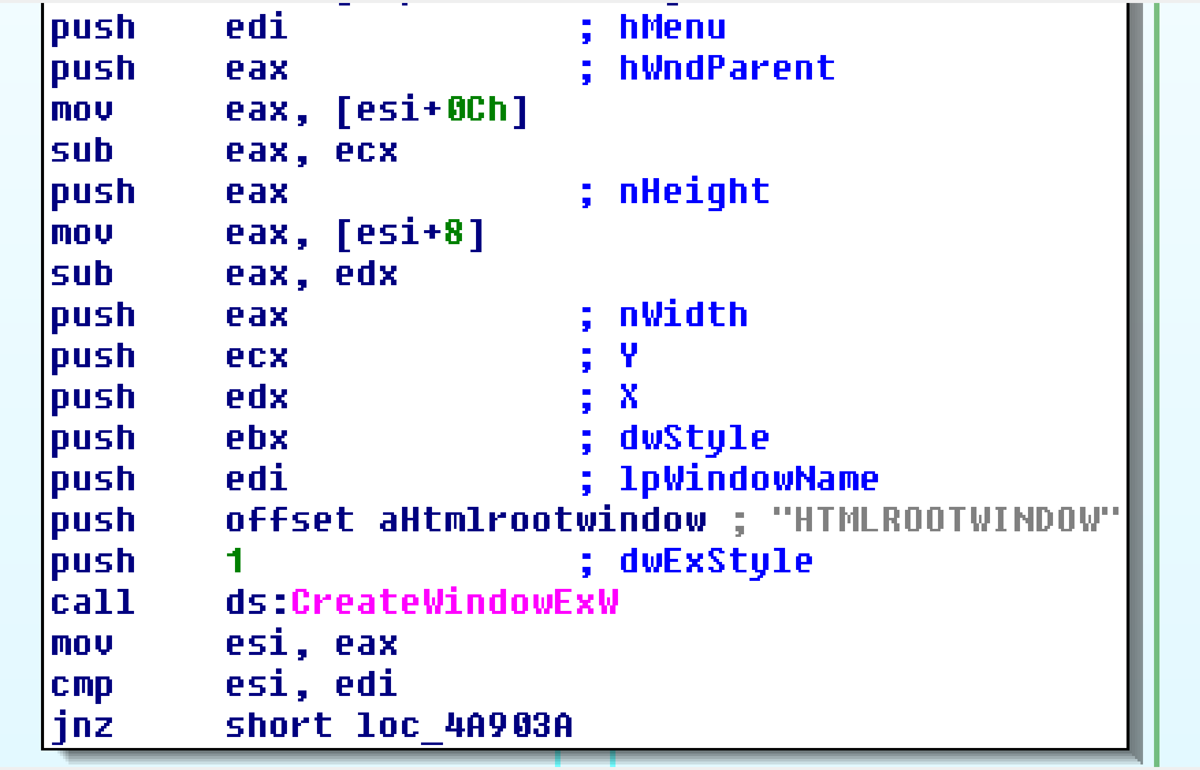
☑ Stay signed in

Cross Call 107

- This is not a browser
- This is a Window hosting ieframe.dll
- Basically the same as iexplore.exe running inside the Broker process
- But … NO Protected Mode
- Add an IE9 exploit and we’re done

Cross Call 107

- CreateWindowExW

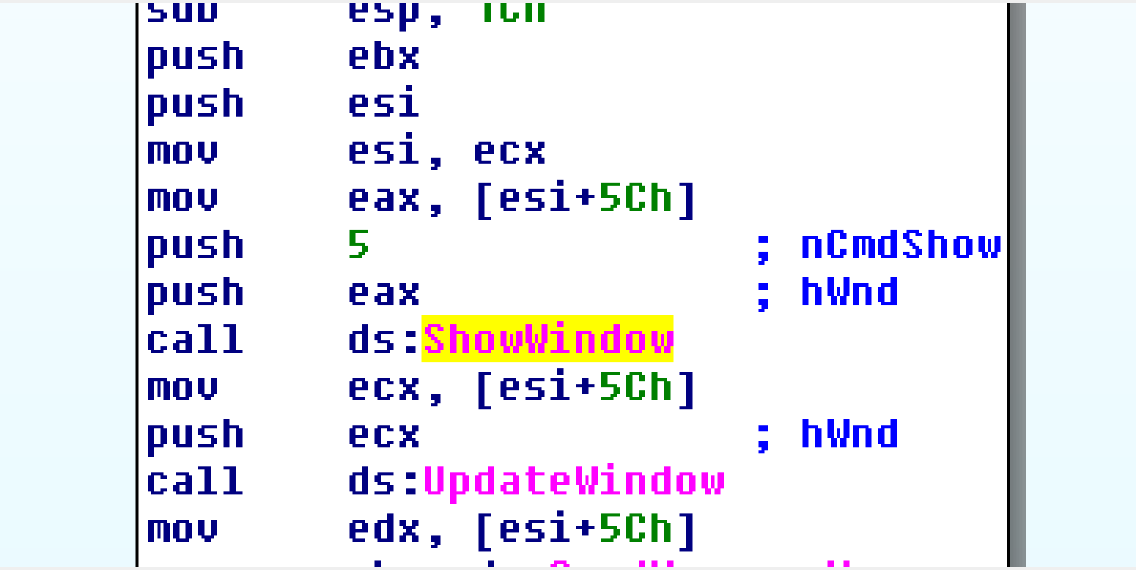


```
push    edi                 ; hMenu
push    eax                 ; hWndParent
mov     eax, [esi+0Ch]
sub     eax, ecx
push    eax                 ; nHeight
mov     eax, [esi+8]
sub     eax, edx
push    eax                 ; nWidth
push    ecx                 ; Y
push    edx                 ; X
push    ebx                 ; dwStyle
push    edi                 ; lpWindowName
push    offset aHtmlrootwindow ; "HTMLROOTWINDOW"
push    1                   ; dwExStyle
call    ds:CreateWindowExW
mov     esi, eax
cmp     esi, edi
jnz     short loc_4A903A
```

Cross Call 107

`ieframe!CWebBrowserOC::Navigate2`

- Show the Window



```
sub     esp, 1Ch
push    ebx
push    esi
mov     esi, ecx
mov     eax, [esi+5Ch]
push    5               ; nCmdShow
push    eax             ; hWnd
call    ds:ShowWindow
mov     ecx, [esi+5Ch]
push    ecx             ; hWnd
call    ds:UpdateWindow
mov     edx, [esi+5Ch]
```

Expanding the Attack Surface

- If you cannot find anything useful …
- Add more processes to communicate with

type=compute-only-renderer

- You can launch an additional Broker Client pair
- type=compute-only-renderer
- Both processes run as MED integrity
- Creates a Named Pipe for communication
- Sandboxed process can Read and Write to this Pipe

64BitsMAPIBroker.exe

- Cross Call 0xBE will Launch 64BitsMAPIBroker
- Creates a Named Pipe
  - Potential new attack surface
  - Did not test

?