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 🙄

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
Adobe Sandbox

• Child process command line arguments specify communication channel details and process type
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
Adobe Sandbox Job

Job Name:
<Unnamed Job>

Processes in Job:

<table>
<thead>
<tr>
<th>Process</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcroRd32.exe</td>
<td>2140</td>
</tr>
</tbody>
</table>

Job Limits:

<table>
<thead>
<tr>
<th>Limit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active Processes</td>
<td>1</td>
</tr>
<tr>
<td>Desktop</td>
<td>Limited</td>
</tr>
<tr>
<td>Display Settings</td>
<td>Limited</td>
</tr>
<tr>
<td>Exit Windows</td>
<td>Limited</td>
</tr>
<tr>
<td>Global Atoms</td>
<td>Limited</td>
</tr>
<tr>
<td>USER Handles</td>
<td>Limited</td>
</tr>
<tr>
<td>Read Clipboard</td>
<td>Limited</td>
</tr>
<tr>
<td>System Parameters</td>
<td>Limited</td>
</tr>
<tr>
<td>Write Clipboard</td>
<td>Limited</td>
</tr>
<tr>
<td>Administrator Access</td>
<td>Limited</td>
</tr>
</tbody>
</table>
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

Broker Client communication

We will focus on the Broker endpoints

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 ---\n// [ parameter 0 size 4 bytes]
// [ parameter 1 type 4 bytes]
// [ parameter 1 offset 4 bytes] -------------------|--\n// [ parameter 1 size 4 bytes]
// [ parameter 2 type 4 bytes]
// [ parameter 2 offset 4 bytes] ---------------------------\n// [ parameter 2 size 4 bytes]
// [ ]-----------------------------------| | |
// | value 0 (x bytes) | <-----------------------/ | |
// | value 1 (y bytes) | <-----------------------/ |
// | end of buffer | <-----------------------/ |
// |-----------------------------------| |

// |-----------------------------------| |

---

---delta to ---

-------------------|--

---------------------------

|------------------------------| | | |

---------------------------

"
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

```c
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

```cpp
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
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)
Cross Call Structure

- Cross Call ID

![Cross Call Structure Diagram]
Cross Call Structure

• Parameters
Cross Call Parameters Types

Chromium code defines 6 valid Parameter types

```c
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)

```c
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
Step 3: Cross Call Adding 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 3: Cross Call Adding Function

- Find the function adding Cross Calls
Step 3: Cross Call Adding Function

```
push    offset dword_504EFC
leax    ecx, [edi+4]
call    AddCrossCall
push    offset dword_504EC0
leax    ecx, [edi+4]
call    AddCrossCall
push    offset dword_504E84
leax    ecx, [edi+4]
call    AddCrossCall
```
Step 3: Cross Call Adding 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 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
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 Call IDs
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
Finding the Cross Call ID

- Cross Call ID is first Dword in the structure
Finding the Cross Call ID

• OR Cross Call ID is pushed as 2\textsuperscript{nd} Argument to another Function
## Adobe Cross Call list

<table>
<thead>
<tr>
<th>CrossCall</th>
<th>Windows API / Description</th>
<th>arg_4</th>
<th>arg_8</th>
<th>arg_c</th>
<th>arg_10</th>
</tr>
</thead>
<tbody>
<tr>
<td>009d : 0048d010</td>
<td>WritePrinter</td>
<td>VOIDPTR</td>
<td>INPTR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>009e : 0048d170</td>
<td>PTOpenProvider</td>
<td>WCHAR</td>
<td>ULONG</td>
<td>INOUTPTR</td>
<td></td>
</tr>
<tr>
<td>009f : 0048d430</td>
<td>PTConvertDevModeToPrintTicket</td>
<td>VOIDPTR</td>
<td>ULONG</td>
<td>INPTR</td>
<td>ULONG</td>
</tr>
<tr>
<td>00a0 : 0048d730</td>
<td>PTCloseProvider</td>
<td>VOIDPTR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00a1 : 0048d8b0</td>
<td>DeviceCapabilitiesA</td>
<td>LPCSTR</td>
<td>LPCSTR</td>
<td>ULONG</td>
<td>INOUTPTR</td>
</tr>
<tr>
<td>00a2 : 0048dba0</td>
<td>DeviceCapabilitiesW</td>
<td>WCHAR</td>
<td>WCHAR</td>
<td>ULONG</td>
<td>INOUTPTR</td>
</tr>
<tr>
<td>00a4 : 0048e930</td>
<td>DeviceCapabilitiesW</td>
<td>ULONG</td>
<td>ADOBE_8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00a5 : 0048dea0</td>
<td>DeviceCapabilitiesW</td>
<td>ULONG</td>
<td>INOUTPTR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00a6 : 0048bb00</td>
<td>EnumPrintersW</td>
<td>ULONG</td>
<td>ULONG</td>
<td>ADOBE_8</td>
<td>INOUTPTR</td>
</tr>
<tr>
<td>00a8 : 004787e0</td>
<td>WNetGetUniversalNameW</td>
<td>WCHAR</td>
<td>ULONG</td>
<td>INOUTPTR</td>
<td></td>
</tr>
<tr>
<td>00a9 : 00478910</td>
<td>WNetGetResourceInformationW</td>
<td>ULONG</td>
<td>ULONG</td>
<td>ULONG</td>
<td>ULONG</td>
</tr>
<tr>
<td>00aa : 00478aa0</td>
<td>WNetAddConnection2W</td>
<td>ULONG</td>
<td>WCHAR</td>
<td>WCHAR</td>
<td>ULONG</td>
</tr>
<tr>
<td>00ab : 0049ac00</td>
<td></td>
<td>ULONG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00ac : 0049ae20</td>
<td></td>
<td>ULONG</td>
<td>ULONG</td>
<td>WCHAR</td>
<td>ULONG</td>
</tr>
<tr>
<td>00ad : 0049b040</td>
<td></td>
<td>ULONG</td>
<td>ULONG</td>
<td>WCHAR</td>
<td>WCHAR</td>
</tr>
<tr>
<td>00ae : 0049ac60</td>
<td></td>
<td>ULONG</td>
<td>ULONG</td>
<td>LPCSTR</td>
<td>LPCSTR</td>
</tr>
<tr>
<td>00af : 00499c0</td>
<td>Retrieve some MAPI information</td>
<td>INOUTPTR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00b0 : 0049af80</td>
<td></td>
<td>ULONG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00b1 : 00439750</td>
<td></td>
<td>ULONG</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>00b2 : 0047a300</td>
<td></td>
<td>VOIDPTR</td>
<td>ULONG</td>
<td>ULONG</td>
<td>ADOBE_8</td>
</tr>
</tbody>
</table>
Endpoint Functions

- Arg_0 is IPCInfo structure

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

```c
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

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
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
Cross Call 0x46

- Microsoft Spy++ information on this window
Cross Call 0x46

- PID 0x5F8 = 1528

- 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 ignored
  - 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
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

Sign in
Stay signed in

Can't access your account?

Sign in with a Google Apps Account
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

```
iedrame!CWebBrowserOC::Navigate2

- Show the Window
```

```Assembly
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