UNDERSTANDING SANDBOXES

Paul Sabanal
IBM Security Systems, X-Force Advanced Research And Development
pv.sabanal[at]gmail.com
@polsab
UNDERSTANDING SANDBOXES

INTRODUCTION
INTRODUCTION

- Sandboxing talks in Blackhat
  - Adobe Reader X sandbox (BH USA 2011)
  - Flash sandboxes (BH USA 2012)
    - Flash Player Protected Mode for Firefox (Firefox Flash)
    - Flash Player Protected Mode for Chrome (Chrome Flash)
    - Flash Player Protected Mode for Chrome Pepper (Pepper Flash)
- This is a condensed version of these two talks!
UNDERSTANDING SANDBOXES

WINDOWS SANDBOXING
WHAT IS SANDBOXING?

- Running an application inside a confined environment
- Mitigates post-exploitation code execution
WHAT ARE THE GOALS OF A SANDBOX?

- Increase the cost of exploitation by
  - Restricting access to stuff you should NOT access
  - Restricting access to stuff you don’t NEED to access
HOW TO ACHIEVE THIS?

- Practical Windows Sandboxing recipe by David Leblanc
- Use the following Windows mechanisms to restrict the privileges and capabilities of a sandboxed process:
  - Restricted Tokens
  - Integrity levels
  - Job Objects
  - Alternate Desktop and Alternate Windows Station
HOW TO ACHIEVE THIS?

- Google Chrome first implemented it (Source code available at The Chromium Project)

- Other sandbox implementations re-used some code from Chrome
UNDERSTANDING SANDBOXES

SANDBOX MECHANISMS
UNDERSTANDING SANDBOXES

SANDBOX MECHANISM: SANDBOX RESTRICTIONS
MECHANISMS > SANDBOX RESTRICTIONS

- Restricted Tokens
- Windows Integrity Mechanism (Integrity Levels)
- Job Objects
- Alternate Desktop and Windows Station
MECHANISMS > SANDBOX RESTRICTIONS > RESTRICTED TOKENS

- Restricts access to securable objects
- Disables privileges
- Sandbox token still have access to some resources (e.g. those accessible to Everyone and Users group)
MECHANISMS > SANDBOX RESTRICTIONS > WINDOWS INTEGRITY MECHANISM

- Low Integrity sandbox process
- Prevents write access to most resources
- Most resources have a Medium or higher integrity level
Restrict additional capabilities
- Spawning new processes
- Clipboard access
- Modification to systems settings
- Active process limits
- CPU limits
**Mechanisms > Sandbox Restrictions > Alternate Desktop and Windows Station**

- Windows on a same desktop can send messages to each other
  - No security checks

- Windows on a separate desktop cannot send messages to each other
UNDERSTANDING SANDBOXES

SANDBOX MECHANISMS:
SANDBOX ARCHITECTURE
SANDBOX ARCHITECTURE
UNDERSTANDING SANDBOXES

SANDBOX MECHANISMS: STARTUP SEQUENCE
Mechanisms > Startup Sequence

1. The broker process is started
2. The broker process sets up the sandbox restrictions
3. The broker process sets up the policies
4. The sandbox process is spawned in a suspended state
5. The broker process sets up interceptions in the sandbox process
6. The sandbox process resumes execution
MECHANISMS > STARTUP SEQUENCE

Firefox Flash

<table>
<thead>
<tr>
<th>File Path</th>
<th>Create Time</th>
<th>Size</th>
<th>Process Name</th>
<th>Origin</th>
<th>Security Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>firefox.exe</td>
<td>3228</td>
<td>0.37</td>
<td>55,180 K Firefox</td>
<td>Mozilla Corporation</td>
<td>Medium</td>
</tr>
<tr>
<td>plugin-container.exe</td>
<td>3876</td>
<td>0.35</td>
<td>10,340 K</td>
<td>Mozilla Corporation</td>
<td>Medium</td>
</tr>
<tr>
<td>FlashPlayerPlugin_11.3_300_257.exe</td>
<td>3252</td>
<td>0.06</td>
<td>4,672 K Adobe Flash Player 11.3 r300</td>
<td>Adobe Systems, Inc.</td>
<td>Medium</td>
</tr>
<tr>
<td>FlashPlayerPlugin_11.3_300_257.exe</td>
<td>1468</td>
<td>15.70</td>
<td>64,252 K Adobe Flash Player 11.3 r300</td>
<td>Adobe Systems, Inc.</td>
<td>Low</td>
</tr>
</tbody>
</table>

Chrome Flash

<table>
<thead>
<tr>
<th>File Path</th>
<th>Create Time</th>
<th>Size</th>
<th>Process Name</th>
<th>Origin</th>
<th>Security Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>chrome.exe</td>
<td>1044</td>
<td>0.08</td>
<td>27,188 K Google Chrome</td>
<td>Google Inc.</td>
<td>Medium</td>
</tr>
<tr>
<td>chrome.exe</td>
<td>3508</td>
<td>0.02</td>
<td>16,276 K Google Chrome</td>
<td>Google Inc.</td>
<td>Untrusted</td>
</tr>
<tr>
<td>chrome.exe</td>
<td>3228</td>
<td>0.19</td>
<td>39,304 K Google Chrome</td>
<td>Google Inc.</td>
<td>Untrusted</td>
</tr>
<tr>
<td>rundll32.exe</td>
<td>4084</td>
<td>0.02</td>
<td>5,280 K Windows host process (Run...)</td>
<td>Microsoft Corporation</td>
<td>Medium</td>
</tr>
<tr>
<td>chrome.exe</td>
<td>3160</td>
<td>3.03</td>
<td>86,292 K Google Chrome</td>
<td>Google Inc.</td>
<td>Low</td>
</tr>
</tbody>
</table>

Pepper Flash

<table>
<thead>
<tr>
<th>File Path</th>
<th>Create Time</th>
<th>Size</th>
<th>Process Name</th>
<th>Origin</th>
<th>Security Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>chrome.exe</td>
<td>2508</td>
<td>2.35</td>
<td>42,340 K Google Chrome</td>
<td>Google Inc.</td>
<td>Medium</td>
</tr>
<tr>
<td>chrome.exe</td>
<td>3716</td>
<td>4.88</td>
<td>39,136 K Google Chrome</td>
<td>Google Inc.</td>
<td>Untrusted</td>
</tr>
<tr>
<td>chrome.exe</td>
<td>3244</td>
<td>5.83</td>
<td>45,360 K Google Chrome</td>
<td>Google Inc.</td>
<td>Untrusted</td>
</tr>
<tr>
<td>chrome.exe</td>
<td>1836</td>
<td>13.32</td>
<td>75,072 K Google Chrome</td>
<td>Google Inc.</td>
<td>Untrusted</td>
</tr>
<tr>
<td>PEPPER.EXE</td>
<td>4640</td>
<td>5.67</td>
<td>41,912 K Google Chrome</td>
<td>Google Inc.</td>
<td>Untrusted</td>
</tr>
</tbody>
</table>

DIGGING DEEP INTO THE FLASH SANDBOXES
UNDERSTANDING SANDBOXES

SANDBOX MECHANISMS: INTERCEPTION MANAGER
**MECHANISMS > INTERCEPTION MANAGER**

- Transparently forwards API calls from the sandboxed process to the broker or browser process via IPC
- API calls are evaluated by the policy engine against sandbox policies
- Done via API interception (API hooking)
MECHANISMS > INTERCEPTION MANAGER > EXAMPLE INTERCEPTION TYPES

- **INTERCEPTION_SERVICE_CALL** – NTDLL API patching
  
  ```
  MOV EAX, <ServiceID>
  MOV EDX, <ThunkCodeAddress>
  JMP EDX
  ```

- **INTERCEPTION_SIDESTEP** – API entry point patching
  
  ```
  JMP <ThunkCodeAddress>
  <original API code>
  <original API code>
  <. . .>
  ```

- **INTERCEPTION_EAT** – Export Address Table patching
UNDERSTANDING SANDBOXES

SANDBOX MECHANISMS: INTER-PROCESS COMMUNICATION (IPC)
M E C H A N I S M S  >  I P C

- Used for communication between the sandbox processes

- 3 IPC implementations were used:
  - Sandbox IPC
  - Chromium IPC
  - Simple IPC (Chrome Flash only)

- See our BH papers for the IPC message structure details
UNDERSTANDING SANDBOXES

SANDBOX MECHANISMS: POLICY ENGINE
MECHANISMS > POLICY ENGINE

- Responsible for evaluating the API calls against the sandbox policies
- Allows the broker to specify exceptions to the default restrictions in the sandbox
- These whitelist rules grant the sandbox specific access to certain objects, overriding the sandbox restrictions
Policy rules are added programmatically, using the sandbox::PolicyBase::AddRule() function:

```
AddRule(subsystem, semantics, pattern)
```

- **subsystem** – indicates the Windows subsystem the rule apply
- **semantics** – indicates the permission that will be applied
- **pattern** – expression to match the object name the policy will be applied to
## Examples of Subsystems

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUBSYS_FILES</td>
<td>Creation and opening of files and pipes.</td>
</tr>
<tr>
<td>SUBSYS_NAMED_PIPES</td>
<td>Creation of named pipes.</td>
</tr>
<tr>
<td>SUBSYS_PROCESS</td>
<td>Creation of child processes.</td>
</tr>
<tr>
<td>SUBSYS_REGISTRY</td>
<td>Creation and opening of registry keys.</td>
</tr>
</tbody>
</table>

## Examples of Semantics

<table>
<thead>
<tr>
<th>Semantics</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILES_ALLOW_ANY</td>
<td>Allows open or create for any kind of access that the file system supports.</td>
</tr>
<tr>
<td>NAMEDPIPES_ALLOW_ANY</td>
<td>Allows creation of a named pipe.</td>
</tr>
<tr>
<td>REG_ALLOW_ANY</td>
<td>Allows read and write access to a registry key.</td>
</tr>
</tbody>
</table>
MECHANISMS > POLICY ENGINE > ADDING POLICY RULES

Examples

\begin{verbatim}
AddRule(SUBSYS_FILES, FILES_ALLOW_ANY, "C:\Users\p01\AppData\Roaming\Macromedia\Flash Player\*"")
\end{verbatim}

\begin{verbatim}
AddRule(SUBSYS_REGISTRY, REG_ALLOW_ANY, "HKEY_CURRENT_USER\Software\Macromedia\FlashPlayer*"")
\end{verbatim}
Mechanisms > Policy Engine > Admin-Configurable Policies

- Reader X and Firefox Flash allow custom policies through a configuration file.

- The policy file is named ProtectedModeWhitelistConfig.txt and is placed in:
  - Reader X
    - Reader install directory
  - Firefox Flash
    - %WINDIR%\System32\Macromed\Flash (32-bit Windows)
    - %WINDIR%\SysWow64\Macromed\Flash (64 bit Windows)
Mechanisms > Policy Engine > Firefox Flash > Admin-Configurable Policies

- Policy rules take the following format:

\[
\text{POLICY\_RULE\_TYPE} = \text{pattern string}
\]

- POLICY\_RULE\_TYPE is a subset of semantics and indicates the permission that will be applied.

- Example

\[
\text{FILES\_ALLOW\_ANY} = "c:\logs\*"
\]
UNDERSTANDING SANDBOXES

SANDBOX LIMITATIONS
“What can a malicious code do once it is running within a sandbox?”
SANDBOX LIMITATIONS > FILE SYSTEM READ ACCESS

- Reader X, Firefox Flash, and Chrome Flash allow read access to all files that are accessible from the user’s account.
  - The sandbox process token still has access to some files (such as those accessible to the Everyone and Users group)
  - There are some hard-coded policy rules that allow read access to all files

```
SubSystem=SUBSYS_FILES
Semantics=FILES_ALLOW_READONLY
Pattern="*"
```
SANDBOX LIMITATIONS > FILE SYSTEM READ ACCESS

- Chrome browser and Pepper Flash do not allow any read access of files
- Implication: Sensitive files (documents, source codes, etc.) can be stolen
**SANDBOX LIMITATIONS > REGISTRY READ ACCESS**

- Reader X, Firefox Flash, and Chrome Flash allow read access to registry keys that are accessible from the user’s account.
  - The sandbox process token still has access to some keys (such as those accessible to the Everyone and Users group)
  - There are some hard-coded policy rules that allow read access to major registry hives:

```plaintext
SubSystem=SUBSYS_REGISTRY
Semantics=REG_ALLOW_READONLY
Pattern="HKEY_CLASSES_ROOT*"
```
Disclaimer: This content has been generated from the image and may not be accurate or complete. For any queries, please refer to the original document.

SANDBOX LIMITATIONS > REGISTRY READ ACCESS

- Chrome browser and Pepper Flash do not allow any read access of registry keys.
- Implication: Disclosure of system configuration information and potentially sensitive application data from the registry.
**SANDBOX LIMITATIONS > NETWORK ACCESS**

- Reader X, Firefox Flash, and Chrome Flash do not restrict network access.
- Chrome browser and Pepper Flash do not allow socket creation.

**Implications:**
- Allows transfer of stolen information to a remote attacker.
- Allows attack of internal systems not accessible from the outside.
SANDBOX LIMITATIONS > POLICY ALLOWED WRITE ACCESS TO FILES/FOLDERS

- Reader X and Firefox Flash contain default policy rules that grant the sandbox process write access to certain folders and files
- Some are third party applications
- Implication: Control the behavior of the sandboxed application itself or other applications
SANDBOX LIMITATIONS > CLIPBOARD READ/WRITE ACCESS

- Reader X, Firefox Flash, and Chrome Flash do not have clipboard access restrictions set in their job objects.
- Reader X and Firefox Flash’s SandboxClipboardDispatcher also provides clipboard services.
- Chrome browser and Pepper Flash do not allow clipboard access.
- Implication: Disclosure of possibly sensitive information.
SANDBOX LIMITATIONS > WRITE ACCESS TO FAT/FAT32 PARTITIONS

- FAT/FAT32 partitions have no security descriptors
- Limitation of all sandboxes
- Implication: Propagation capabilities
  – Dropping of malicious files into FAT/FAT32 partitioned USB flash drives
SANDBOX LIMITATIONS > SUMMARY

- Limitations and weaknesses still exist
- Still possible to carry out information theft
- Chrome and Pepper Flash are the most restrictive
UNDERSTANDING SANDBOXES

SANDBOX ESCAPE
Sandbox Escape > Local Elevation of Privilege (EoP) Vulnerabilities

- Particularly those that result in kernel-mode code execution

- Multiple interface to kernel-mode code are accessible to the sandboxed process

- See “There's a party at Ring0, and you're invited” by Tavis Ormandy and Julien Tinnes.
SANDBOX ESCAPE > NAMED OBJECT SQUATTING ATTACKS

- Crafting a malicious named object that is trusted by a higher-privileged process
- Tom Keetch demonstrated named object squatting against Protected Mode IE on “Practical Sandboxing on the Windows Platform”
First code running in a privileged context to touch untrusted data

Code that parses the IPC message and code that deserializes parameters are interesting

All IPC implementations are open source

Example: SkBitmap deserialization bug discovered by Mark Dowd in Chrome
Policies that allow write access are potential vectors for sandbox escape

Scenario: Registry key
- Does it contain configuration entries used by higher-privileged applications?

Scenario: Folders
- Can you overwrite executable files?
- Does it contain configuration data used by higher-privileged applications?
Decides what potentially security-sensitive action to allow/deny

Policy engine vulnerabilities can be used to evade policy checks

Example: REG_DENY policy in Adobe Reader X can be bypassed due to lack of canonicalization (CVE-2011-1353)
  – Bug we discovered and demoed at BH USA 2011
  – Also independently discovered by Zhenhua Liu of Fortinet's Fortiguard Labs
SANDBOX ESCAPE > POLICY ENGINE VULNERABILITIES > CVE-2011-1353

- Registry entry to disable/enable the Reader X sandbox:
  
  \texttt{HKEY\_CURRENT\_USER\Software\Adobe\Acrobat\ Reader\10.0\Privileged} bProtectedMode = 0 (disabled), non-zero (enabled)

- There is an allow-any policy for "HKCU\Software\Adobe\Acrobat Reader\10.0\*" but there is a deny-access policy for the Privileged key:
  
  \texttt{Semantics: REG\_DENY}  
  \texttt{Pattern: HKEY\_CURRENT\_USER\Software\Adobe\Acrobat\ Reader\10.0\Privileged*}

- However, the deny-access policy can be bypassed:
  
  \texttt{HKEY\_CURRENT\_USER\Software\Adobe\Acrobat\ Reader\10.0\Privileged}
Services exposed by higher-privileged processes are a large attack surface for sandbox escape

Example: Untrusted pointer dereference in Chrome Flash broker (CVE-2012-0724, CVE-2012-0725)
– 2 bugs we discovered last March 2012
– Also independently discovered by Fermin J. Serna of the Google Security Team
SANDBOX ESCAPE > SERVICE VULNERABILITIES > CVE-2012-0724, CVE-2012-0725

- 2 service handlers in Chrome Flash broker accept a SecurityFunctionTableA pointer (1\textsuperscript{st} parameter)

<table>
<thead>
<tr>
<th>Simple IPC Message ID</th>
<th>Parameters</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2B</td>
<td>VOIDPTR sec_func_table</td>
<td>Broker a call to \texttt{AcquireCredentialHandlesA()}</td>
</tr>
<tr>
<td>0x2D</td>
<td>VOIDPTR sec_func_table, ULONG32 cred_handle_lower, ULONG32 cred_handle_upper</td>
<td>Broker a call to \texttt{FreeCredentialsHandle()}</td>
</tr>
</tbody>
</table>

- The pointer is fully trusted and dereferenced inside the service handlers in a call instruction:

```
Service_0x2B_AcquireCredentialsHandleA:
...
mov reg, [sec_func_table] ; sec_func_table is fully controllable
...
call [reg+0Ch] ; sec_func_table->AcquireCredentialsHandleA()
; reg+0Ch is fully controllable!
```
SANDBOX ESCAPE > SUMMARY

- Involves exploiting a weakness in a higher-privileged application
- Permissive policies and improper handling of untrusted data are prime examples of weaknesses that can lead to a sandbox escape
- The sandbox mechanisms used by higher-privileged processes such as the IPC, policy engine and services are potential vectors for sandbox escape
UNDERSTANDING SANDBOXES

DEMO
**Sandbox Bypass On Reader X Demo**

- RCE + Sandbox Escape for Adobe Reader 10.0.1

- Remote Exploit
  - CVE-2011-0609 remote code execution

- Sandbox Bypass Exploit
  - CVE-2011-1353 for Adobe Reader X sandbox bypass
SANDBOX ESCAPE ON CHROME FLASH DEMO

- RCE + Sandbox Escape for Chrome Flash 11.1.102.55

- Remote Exploit
  - CVE-2012-0769 for Flash info leak
  - CVE-2012-0779 for Flash EIP control
    https://community.rapid7.com/community/metasploit/blog/2012/06/22/the-secret-sauce-to-cve-2012-0779-adobe-flash-object-confusion-vulnerability

- Escape Exploit
  - CVE-2012-0725 for Chrome Flash Broker info leak and EIP control
UNDERSTANDING SANDBOXES

CONCLUSION
**CONCLUSION**

- Attackers now need an additional sandbox escape vulnerability to fully compromise a system
- Sandboxes are proven to be effective but limitations still exist
- Chrome and Pepper Flash are the most restrictive
UNDERSTANDING SANDBOXES

Thank You!

- Paul Sabanal
  IBM X-Force Advanced R&D
  pv.sabanal[at]gmail.com
  @polsab