A Castle Made of Sand

Adobe Reader X Sandbox

Richard Johnson

rjohnson@sourcefire.com

Adobe Acrobat

- “Adobe Reader is free software that lets you open, view, search, digitally sign, verify, and print PDF files. To date, more than 600 million copies of Adobe Reader have been distributed worldwide on 23 platforms and in 33 languages”

  http://www.adobe.com/products/reader/faq.html

Agenda

- Why Adobe needs a sandbox
- What's in a Sandbox
- Windows Sandboxing
- Adobe Reader Sandbox Architecture
- Attacking Sandboxes
- Conclusion

Internet Usage Statistics

- As of June, 2010 there were 2 billion internet users http://www.internetworldstats.com/stats.htm
  - 600 million Reader downloads = 30% market
- Chrome market share was 23.8% in January, 2011 http://www.w3schools.com/browsers/browsers_stats.asp
  - Roughly 476 million users

Adobe Acrobat Security History

- Acrobat Reader CVE Vulnerabilities

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	XSS	Http Response Splitting	Bypass something	Gain Privileges	CSRF	# of exploits
1999	1		1	1							
2000	1		1	1							
2001	1										
2002	1										
2003	3		2	1							
2004	6		5	4							
2005	9	4	5	3							
2006	7	2	3		1				2		
2007	9	3	3		1	2	1			1	1
2008	11	2	8	4	1						3
2009	39	14	30	17	10			1	1		4
2010	68	35	60	33	29	2		3	1		4
2011	28	10	22	8	8	2			4		
Total	184	70	140	72	50	6	1	4	8	1	12
% Of All		38.0	76.1	39.1	27.2	3.3	0.5	2.2	4.3	0.5	

http://www.cvedetails.com/product/497/Adobe-Acrobat-Reader.html?vendor_id=53

Adobe Acrobat Security History

- Acrobat CVE Vulnerabilities

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	XSS	Bypass something	Gain Privileges	CSRF	# of exploits
2000	1		1	1						
2003	3		2							
2004	3		2	2						
2005	2	1	1	1						
2006	4		2	1	1			2		
2007	4	1	1			2			1	
2008	15	1	10	4	2			1		3
2009	49	18	39	22	14		2			4
2010	65	33	58	33	28	2	3	1		3
2011	28	10	22	8	8	2		4		
Total	174	64	138	72	53	6	5	8	1	10
% Of All		36.8	79.3	41.4	30.5	3.4	2.9	4.6	0.6	

http://www.cvedetails.com/product/921/Adobe-Acrobat.html?vendor_id=53

Adobe Acrobat Security History

- Adobe CVE Vulnerabilities
  - 358 Vulnerabilities
  - 278 Vulnerabilities lead to code execution
  - 22 Exploits in the wild
  - 15 Exploits achieve code execution

- "During the Q1 2010, 48 percent of all exploits involved malicious PDFs, making Adobe Reader the most exploited software."
  http://www.esecurityplanet.com/article.php/3925701/RSA-New-Frontiers-in-Threat-Research.htm

Google Chrome Security History

- Chrome CVE Vulnerabilities
  - 244 Vulnerabilities
  - 36 Vulnerabilities lead to code execution
  - 12 Exploits in the wild
  - 3 Exploits achieve code execution

Adobe Acrobat X

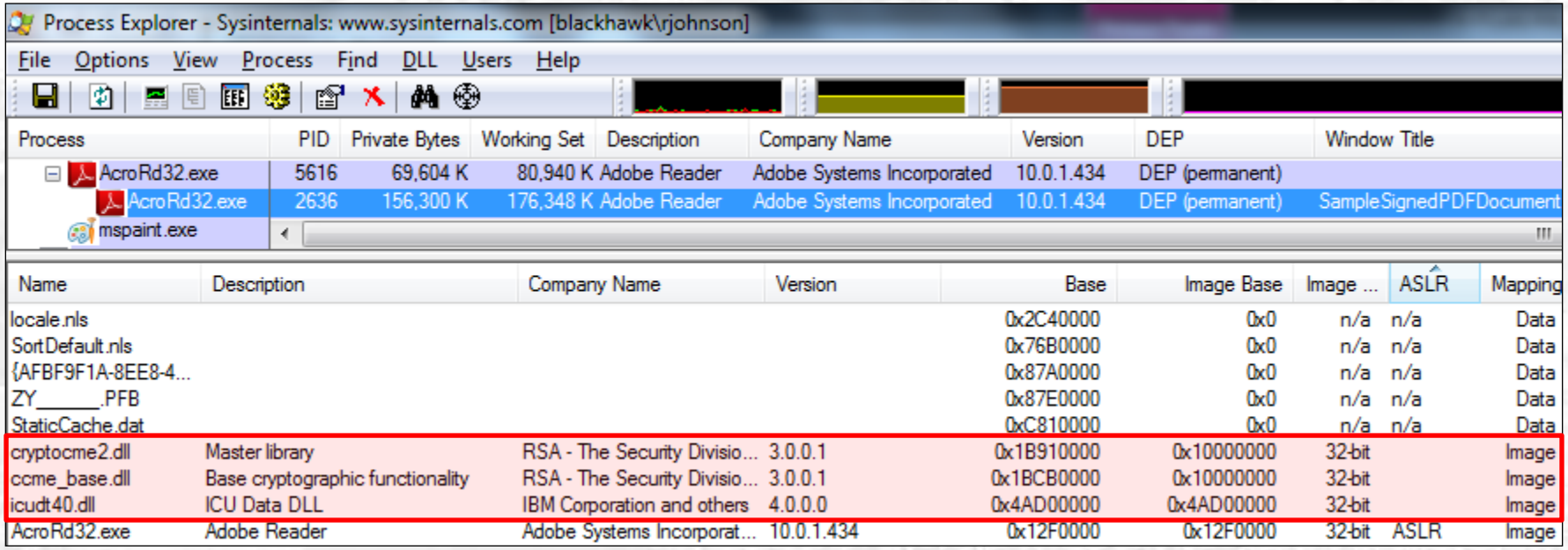
- These statistics prompted a security push to make the next version of Adobe Acrobat significantly more resilient to hacking attempts
- Adobe Acrobat X products have been hardened to utilize operating system provided mitigations on the Windows Platform
- In addition, a new sandbox designed to limit the impact of successful exploitation attempts has been implemented

Use of Windows Mitigations

- Address Space Layout Randomization
  - Adobe has modified all internal code to take advantage of random image mappings
- Data Execution Prevention
  - Enabled with PERMENENT flag
- "…Q2 of last year, PDF attacks fell to 30 percent..."
  http://www.esecurityplanet.com/article.php/3925701/RSA-New-Frontiers-in-Threat-Research.htm

Windows Mitigations Fail

- Sadly, 3rd party libraries that do not support ASLR can be forced to load via PDF



http://blogs.adobe.com/security/SampleSignedPDFDocument.pdf

The Sandbox Concept

- A sandbox is a mitigation strategy centered around the concept of isolating complex code into a lower privileged process which is managed by a higher privileged process

- The higher privileged process is less prone to attack due to reduced attack surface and can restrict resources from a compromised lower privileged process

Sandbox Architecture Requirements

- Sandbox mitigations require the ability to:
  - Create a child process with restricted access to resources
  - Communicate between the processes to broker request access to resources

Sandbox Architecture on Windows

- Process Restrictions
  - Restricted process tokens
  - Restricted process job object
- IPC Mechanisms for System Call brokering
  - Sockets, Pipes, Shared Memory, Files, etc

Sandbox Architecture on Windows

- Restricted process tokens
  - Create processes with restricted privileges

```
BOOL CreateRestrictedToken(
        HANDLE ExistingTokenHandle,
        DWORD Flags,
        DWORD DisableSidCount,
        PSID_AND_ATTRIBUTES SidsToDisable,
        DWORD DeletePrivilegeCount,
        PLUID_AND_ATTRIBUTES PrivilegesToDelete,
        DWORD RestrictedSidCount,
        PSID_AND_ATTRIBUTES SidsToRestrict,
        PHANDLE NewTokenHandle
);
```

```
BOOL WINAPI CreateProcessAsUser(
        HANDLE hToken,
        LPCTSTR lpApplicationName,
        LPTSTR lpCommandLine,
        LPSECURITY_ATTRIBUTES lpProcessAttributes,
        LPSECURITY_ATTRIBUTES lpThreadAttributes,
        BOOL bInheritHandles,
        DWORD dwCreationFlags,
        LPVOID lpEnvironment,
        LPCTSTR lpCurrentDirectory,
        LPSTARTUPINFO lpStartupInfo,
        LPPROCESS_INFORMATION lpProcessInformation
);
```

Sandbox Architecture on Windows

- Restricted job object

```
HANDLE WINAPI CreateJobObject(
                LPSECURITY_ATTRIBUTES lpJobAttributes,
                LPCTSTR lpName
);
```

```
typedef struct _SECURITY_ATTRIBUTES {
                DWORD  nLength;
                LPVOID lpSecurityDescriptor;
                BOOL   bInheritHandle;
} SECURITY_ATTRIBUTES, *LPSECURITY_ATTRIBUTES;
```

```
BOOL WINAPI AssignProcessToJobObject(
                HANDLE hJob,
                HANDLE hProcess
);
```

```
BOOL CreateCustomDACL(SECURITY_ATTRIBUTES * pSA) {
    //     Built-in guests are denied all access.
    //     Anonymous logon is denied all access.
    //     Administrators are allowed full control.
    // Modify these values as needed to generate the proper
    // DACL for your application.
    TCHAR * szSD = TEXT("D:")       // Discretionary ACL
      TEXT("(D;OICI;GA;;;BG)")      // Deny access to
                                    // built-in guests
      TEXT("(D;OICI;GA;;;AN)")      // Deny access to
                                    // anonymous logon
      TEXT("(A;OICI;GA;;;BA)");     // Allow full control
                                    // to administrators

   if (NULL == pSA)
      return FALSE;

    return ConvertStringSecurityDescriptorToSecurityDescriptor(
            szSD,
            SDDL_REVISION_1,
            &(pSA->lpSecurityDescriptor),
            NULL);
}
```

Adobe Reader X Sandbox Design

- Adobe enables the sandbox through a configuration option called ‘Protected Mode’

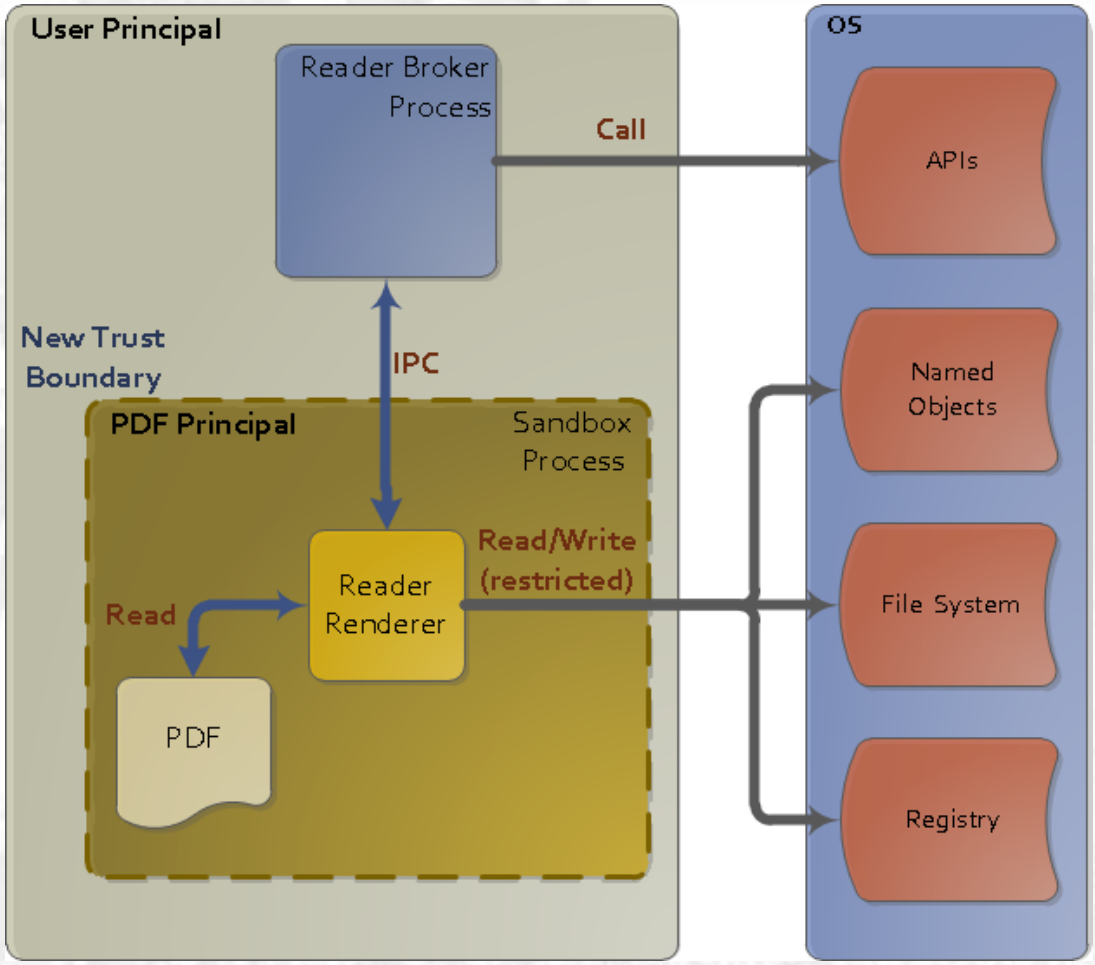
- Separation of rendering code from basic process initialization and management code
  - 25mb broker process
  - 200mb rendering process

Adobe Reader X Sandbox Design

- Rendering process has restricted tokens which disallow writing to the file system or executing new processes
- Requests for system resources are denied and then requested from the broker process via a shared memory protocol
- Requests are validated against internal policy

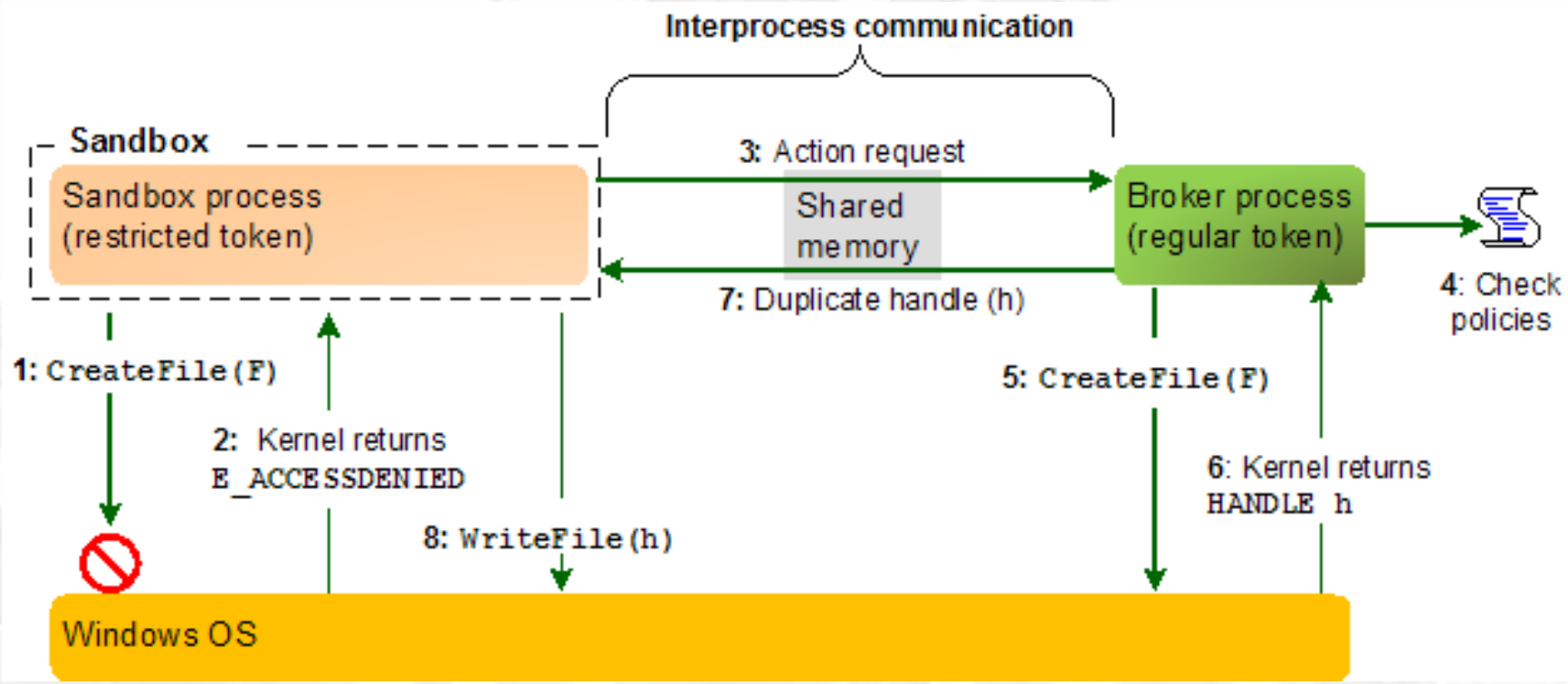
Adobe Reader X Sandbox Design

- OS denies requests to resources
- Broker gets request and checks ACLs
- Broker gets resource and duplicates the handle



http://blogs.adobe.com/asset/2010/10/inside-adobe-reader-protected-mode-part-1-design.html

Adobe Reader X Sandbox Design



http://blogs.adobe.com/asset/2010/11/inside-adobe-reader-protected-mode-part-3-broker-process-policies-and-inter-process-communication.html

Adobe Reader X Sandbox Config

- Configuration settings
  - JavaScript enabled by default
  - JavaScript global object security policy
  - JavaScript blacklist
  - ACLs for file, registry, process access
  - Log file disabled by default

JavaScript Blacklist

- Blacklist is stored in the registry
- Blacklist is capable of blocking API names
  - Withstands obfuscation methods
  - Does not come with any blocked by default
- Blacklist cannot pattern match or prevent generic algorithms for spraying
- More: http://vrt-blog.snort.org/2010/01/acrobat-javascript-blacklist-framework.html

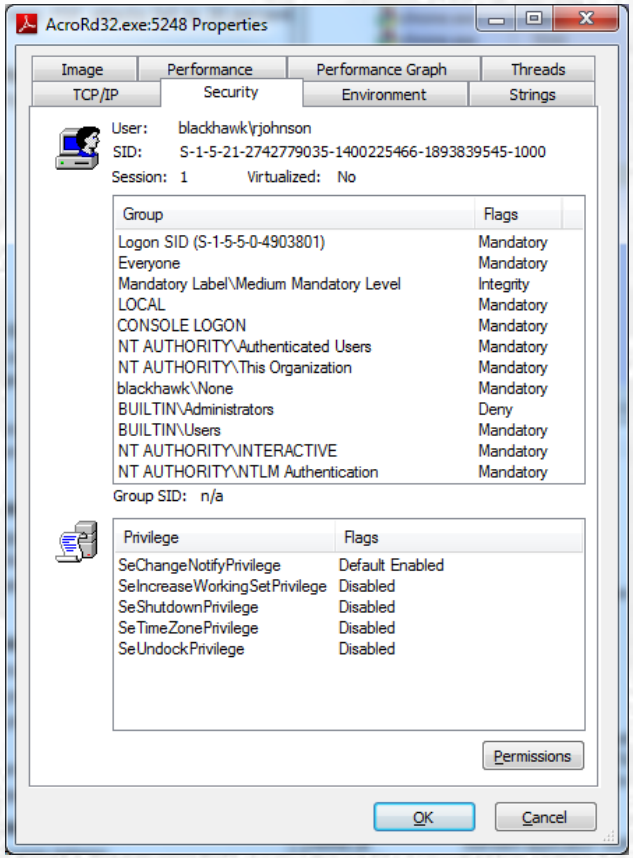
Sandbox Analysis

- Determine rights of separate processes
- Determine IPC mechanisms in use
- Validate resource requests are denied
- Fuzz or audit broker resource request parser

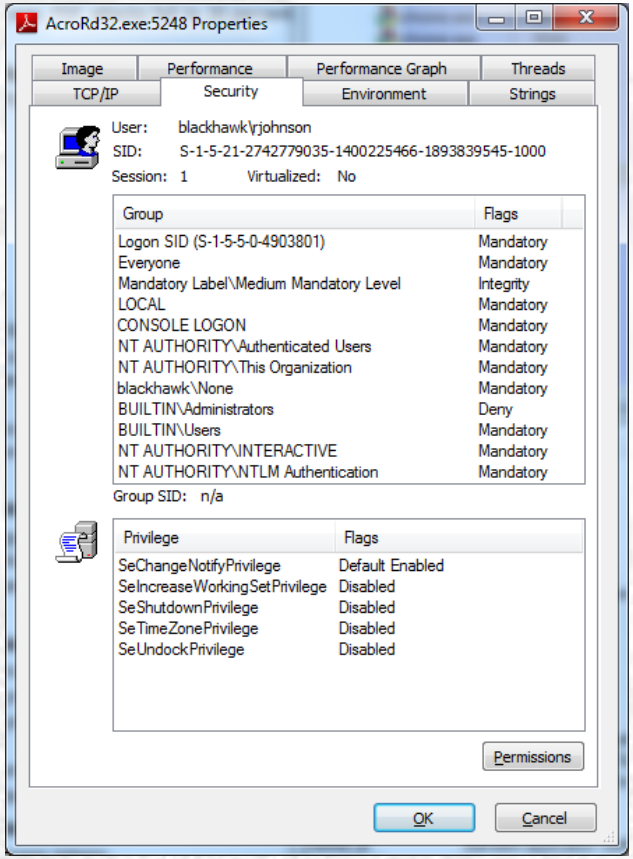
Sandbox Analysis

- Token restriction

Adobe Reader 9



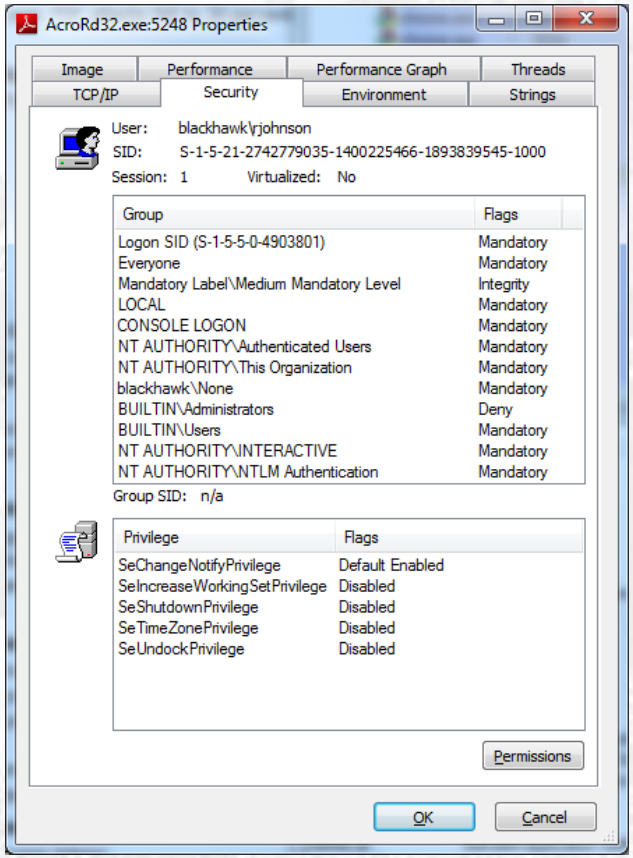
Adobe Reader X Broker



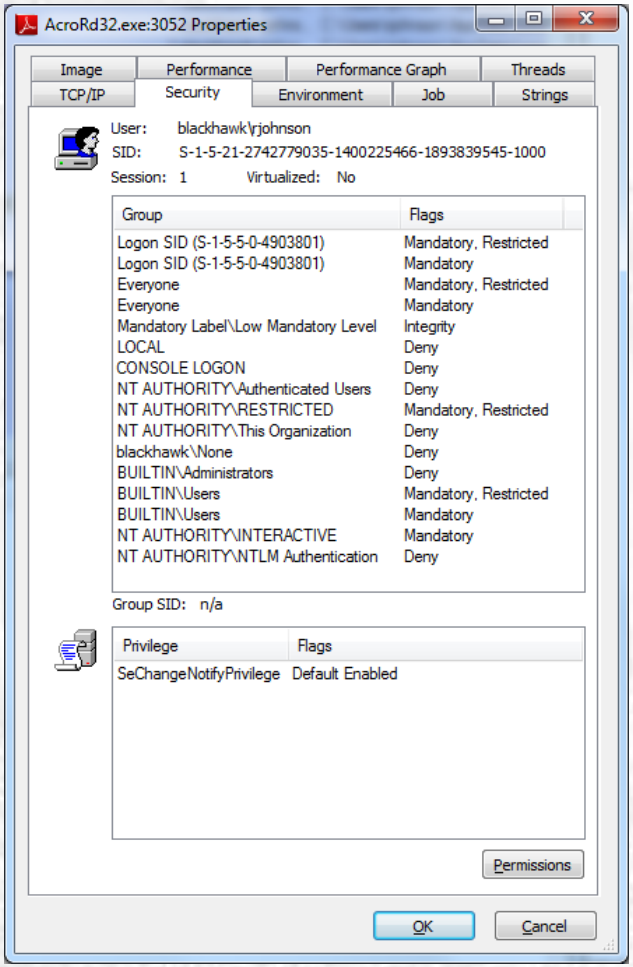
Sandbox Analysis

- Token restriction

Adobe Reader X Broker



Adobe Reader X Renderer



Sandbox Analysis

- Job limits
  - Limit of one ActiveProcess
  - No changing or creating desktops
  - Cannot use handles associated with another job
  - Denied access to ChangeDisplaySettings
  - Denied access to ExitWindows
  - Denied access to SystemParametersInfo

Sandbox Analysis

- Determine IPC mechanisms in use
  - Trace APIs related to various IPC mechanisms
  - Show Windbg method of catching creation of IPC endpoints

Sandbox Analysis

- Determine IPC mechanisms in use
  - Clipboard
  - COM
  - Data Copy
  - DDE
  - File Mapping
  - Mailslots
  - Pipes
  - RPC
  - Windows Sockets

Sandbox Analysis

- Memory mappings are backed to pagefile and may be named or unnamed

- If unnamed, the handle must be passed to the child process via DuplicateHandle

Sandbox Analysis

- Windbg can trace mappings for you

```
r $t0 = 0;

bp KERNELBASE!CreateFileMappingW ".if (poi(@esp + 4)) = -1 { .echo ; kn 5 ; .printf
\"\\nCreateFileMappingW\\nHandle: %x\\n\", poi(@esp + 4) ; ddu esp + 24 l1 ; gu ; .printf \"Mapped
Memory Handle: %x\\n\\n\", @eax ; r $t0 = @eax ; g } .else { g } "

bp KERNELBASE!MapViewOfFile ".if (poi(@esp + 4)) = $t0 {  r $t1 = poi(@esp + 24) ; .echo ; kn 5 ; gu
; .printf \"\\nMapViewOfFile\\nMapped Address: %x   Size: %d\\nSetting memory breakpoint\\n\\n\",
@eax, @$t1 ; ba r 4 @eax \".echo Mapped Memory Access ; kn 4 ; ub ; g\" ; g } .else { g } "

bp KERNELBASE!OpenFileMappingW "kn 5 ; .echo ; .printf \"OpenFileMappingW\nPath: [%mu]\", poi(@esp +
c) ; .if(poi(@esp + 4)) & 2 { .printf \" FILE_MAP_WRITE\" } ; .if(poi(@esp + 4)) & 4 { .printf \"
FILE_MAP_READ\" } ; .echo ; .echo ; g"

bp DuplicateHandle ".echo ; .printf \"DuplicateHandle: %x\", poi(@esp + 8) ; .echo ; .echo ; g"

bp ConnectNamedPipe
bp CreateNamedPipeW

bp AcroRd32Exe+0xc08f ".echo Attach to client
```

Sandbox Analysis

- Windbg can trace mappings for you

```
 # ChildEBP RetAddr
00 0041ec44 7700ac7e KERNELBASE!OpenFileMappingW
01 0041ec60 7700ac11 SHLWAPI!SHCreateSharedSection+0x16
02 0041ec90 7700acf6 SHLWAPI!OpenGlobalCounterFileMappingAndMapMemory+0x3d
03 0041eca8 7700e9de SHLWAPI!GetGlobalCounterMemoryAddress+0x3d
04 0041ecb4 75dac572 SHLWAPI!SHGlobalCounterGetValue+0xd
OpenFileMappingW Path: [windows_shell_global_counters] FILE_MAP_WRITE FILE_MAP_READ
--------------------
DuplicateHandle: 1e4
--------------------
 # ChildEBP RetAddr
00 0041f0f0 00f2f824 KERNELBASE!CreateFileMappingW
01 0041f118 00f3023b AcroRd32Exe+0x1f824
02 0041f138 00f2e438 AcroRd32Exe+0x2023b
03 0041f230 00f4bf6b AcroRd32Exe+0x1e438
04 0041f360 00f1bdfa AcroRd32Exe+0x3bf6b
CreateFileMappingW
Handle: ffffffff
0041f118  0041f138 ".A.ò콠.A찔.꼨.戇ø"
Mapped Memory Handle: 220
--------------------
DuplicateHandle: 220
```

Sandbox Analysis

- Windbg can trace mappings for you

```
 # ChildEBP RetAddr
00 0041f0f0 00f2f870 KERNELBASE!MapViewOfFile
01 0041f118 00f3023b AcroRd32Exe+0x1f870
02 0041f138 00f2e438 AcroRd32Exe+0x2023b
03 0041f230 00f4bf6b AcroRd32Exe+0x1e438
04 0041f360 00f1bdfa AcroRd32Exe+0x3bf6b
--------------------
MapViewOfFile
Mapped Address: a4a0000   Size: 4321592     Setting memory breakpoint
--------------------
Mapped Memory Access
 # ChildEBP RetAddr
00 0041f0f8 00f2f963 AcroRd32Exe+0x237ac
01 0041f118 00f3023b AcroRd32Exe+0x1f963
02 0041f138 00f2e438 AcroRd32Exe+0x2023b
03 0041f230 00f4bf6b AcroRd32Exe+0x1e438
AcroRd32Exe+0x23795:
00f33797 8d0480          lea     eax,[eax+eax*4]
00f3379a 8d148508000000  lea     edx,[eax*4+8]
00f337a1 8b4508          mov     eax,dword ptr [ebp+8]
00f337a4 53              push    ebx
00f337a5 8907            mov     dword ptr [edi],eax
00f337a7 8955fc          mov     dword ptr [ebp-4],edx
00f337aa 8908            mov     dword ptr [eax],ecx
```

Attacking IPC Message Format

- Adobe uses a shared memory structure to request resources from the broker process
- This additional attack surface deserves a critical look from a code quality perspective
- We can inject a DLL to request resources in a loop with corrupt values

Attacking IPC Message Format

- Inject a DLL for fuzzing

```
int InjectDLL(HANDLE hProcess, char *moduleName)
{
   unsigned char *remoteBuffer;
   LPTHREAD_START_ROUTINE loadLibraryAddr;
   HANDLE hThread;
   DWORD moduleNameLen, ret;

   moduleNameLen = strlen(moduleName) + 1;

   remoteBuffer = (unsigned char *)VirtualAllocEx(
                        hProcess, NULL, moduleNameLen, MEM_COMMIT, PAGE_READWRITE);

   WriteProcessMemory(hProcess, remoteBuffer , moduleName, moduleNameLen, NULL);

   loadLibraryAddr = (LPTHREAD_START_ROUTINE)GetProcAddress(
                        GetModuleHandleA("kernel32.dll"), "LoadLibraryA");

   hThread = CreateRemoteThread(
                        hProcess, NULL, 0, loadLibraryAddr, (void *)remoteBuffer, 0, NULL);

   ret = WaitForSingleObject(hThread, 5 * 1000);
...
}
```

Attacking IPC Message Format

- Fuzz from within the DLL

```
BOOL APIENTRY DllMain(HANDLE hModule, DWORD  dwReason, LPVOID lpReserved)
{
    if(dwReason == DLL_PROCESS_ATTACH )
    {
        MessageBoxA(NULL, "Dll injected!", "Fuzzer Dll", MB_OK);
        if((hFuzzThread = CreateThread(
            NULL,                   // default security attributes
            0,                      // use default stack size
            FuzzerFunction,        // thread function name
            NULL,                   // argument to thread function
            0,                      // use default creation flags
            &dwFuzzThreadId)) == NULL)   // returns the thread identifier
        {
            MessageBoxA(NULL, "Failed to create fuzzing thread", "Fuzzer Dll", MB_OK);
        }

        ...
    }
    return TRUE;
}
```

Attacking IPC Message Format

- Fuzz from within the DLL

```
DWORD WINAPI FuzzerFunction(LPVOID lpParam)
{
    DWORD iteration = 0;
    FILE *file;

    do
    {
        char *path = GenFuzzedString();
        file = fopen(path, "r");
        if(file != NULL)
            fclose(file);

        file = fopen(path, "w");
        if(file != NULL)
        fclose(file);

        ...
    } while (iteration++ < ITERATIONS);

    return 0;
}
```

Unrestricted Access

- Socket and Handle use is not restricted
  - Could use PDF exploit as a pivot point into a sensitive network using less sophisticated attacks to achieve persistence
- Reading of the file system is not restricted
  - Combined with above flaw, file system may be dumped over a socket

Unrestricted Access

- Reading from Clipboard is not restricted

- Log file is disabled by default
  - When it is enabled, it is stored in one of the few writable directories by default

Future Potential

- Network Sandboxing (LeBlanc)
  - A solution is outlined in http://blogs.msdn.com/b/david_leblanc/archive/2007/11/02/more-on-sandboxing-network-implications.aspx
  - tl;dr – Use Windows Firewall to limit connections to and from the acrord32.exe process

Future Potential

- File I/O Sandboxing (rjohnson)
  - On launch copy required resources to a temp directory
  - Limit all reads to the temp directory rather than allowing global read access

Future Potential

- Utilize 64-bit process advantages (anti-spray)
- Javascript blacklist could be utilized to prevent loading of generic spray code
  - Currently only blacklist APIs rather than allow a fingerprinting mechanism

Conclusion

- Adobe is moving in the right direction
- Improvements need to be implemented on other platforms
- Offering configuration that includes the ability to enable available solutions would lead to a more secure sandbox

Questions?

Thank you!

rjohnson@sourcefire.com

http://vrt-blog.snort.org/