Postscript Pat and His Black and White Hat

Steven Seeley (mr_me) of Source Incite

whoami

- Independent Security Researcher
- Fitness enthusiast / body builder
- ZDI platinum researcher
- Sharing n-day writeup’s & exploits at srcincite.io
- Teaching **Full Stack Web Attack**
  - https://srcincite.io/training/
- Forever trying to learn Spanish!

SOURCE INCITE

Agenda

- Project Outline
- Postscript Essentials
- Prior Research
- Understanding Adobe’s Postscript Engine
- Attack Vectors
- Postscript Auditing Toolkit - PAT
    - Design and Architecture

Agenda

  - Mutators
  - Limitations
- Results
- Exploitation Primitives
- Conclusion
- Future Work
- References

Project Outline

- Duration: 2 months part-time (21 hours a week)
- Difficulty: Moderate
- Target: acrodistdll.dll
  - Adobe Acrobat's postscript parser
  - Version 19.10.20064.48846
  - ~ 7 Mb of code to target

Postscript Essentials

- Created by Adobe between 1982-1984 by John Warnock, Charles Geschke, Doug Brotz, Ed Taft and Bill Paxton
- Postscript Level 1 - Released 1984
    - First set of operands introduced
- Postscript Level 2 - Released 1991
    - Introduced image parsing, composite fonts
- Postscript Level 3 - Released 1997
    - Introduced more compression/decompression filters, better color handling and more operands

Postscript Essentials

Postscript is based upon LaTeX (.tex) and LaTeX is just a way to describe data. Think markdown.

```
Cartesian closed categories and the price of eggs
Jane Doe
September 1994

Hello world!
```

```
\documentclass{article}
\title{Cartesian closed categories and the price of eggs}
\author{Jane Doe}
\date{September 1994}
\begin{document}
   \maketitle
   Hello world!
\end{document}
```

Postscript Essentials

Uses: Document Structuring Conventions (DSC)

```
%!PS-Adobe-3.0
...Document header comments...
%%BoundingBox: (atend)
%%EndComments
...Rest of the document...
%%Trailer
%%BoundingBox: 0 0 157 233
...Document clean up...
%%EOF
```

semicolon

camelcase

postscript datatypes

double comment

Postscript Essentials

- A complete Virtual Machine
  - Interpreted page description language, from top to bottom
  - Stack based “backwards” syntax. It’s also known as postfix notation or reverse polish notation.
  - Vector based image generation
  - Three major implementations:
    - Ghostscript
    - PSNormalizer
    - Adobe Postscript

Postscript Essentials

- Types of stacks
    - Operand - all operands with their arguments
    - Dictionary - dictionaries with keys and values
    - Execution - procedures in order of execution
    - Graphics state - graphics coordinates and line positions

Postscript Essentials

Type	Example 1	Example 2	Example 3
Literal Name	/s1	/$1	/s1 /proc1 def
Procedure	{ }	/p1 { -3 1 roll } def	/p1 { /p2 {} def } def
Dictionary	<< >>	/d1 3 dict def	/d2 << /k1 (v1) >> def
Array	[ ]	/a1 3 array def	/a2 [1 2 3] def
String	( )	/s1 (AAAA) def	/s2 <41414141> def
Number	1337	/n1 1337 def	/negn1 -1337 def
Real	0.1337	/re1 0.1337 def	/negre1 -0.1337 def
Radix Number	16#4141	/ra1 32#41414141 def	/ra2 2#1000 /def
Comment	%	%!PS	%%Page: 243 23
Boolean	true	false	/b1 true def

Postscript Essentials

Getting and Setting

```
/d1 3 dict def % define a dictionary
d1 /k0 1 put   % put a Number into the 1st key
d1 /k1 get     % get a value from the 2nd key
```

- This is how we manipulate the dictionary stack
- There are a number of built in dictionaries that we can read (but not always write to)

Postscript Essentials

- Dictionaries
  - systemdict - read only
  - userdict - read only
  - errordict - read/write
  - Undocumented dictionaries
    - internaldict - read only

Postscript Essentials

Reference vs Execution

```
/proc1 { /arg1 exch def } def
```

- Now referencing the procedure is just using /proc1
- Executing the procedure is just using proc1
- Nested procedures are possible
- This is how we manipulate the execution stack

Postscript Essentials

Stack Manipulation <arg2> <arg1> <operand> <return>

- save/restore - pushes/pops VM memory state on and off the stack
- gsave/grestore - pushes/pops the graphics state on and off the stack
- grestoreall - keeps popping the graphics state off the stack until its the last one
- clipsave/cliprestore - pushes/pops only clipping data of the graphics state on and off the stack

Postscript Essentials

Stack Manipulation <arg2> <arg1> <operand> <return>

- roll - reverses the stack order by *n* depth
- index - copies the *n* index to the top of the stack
- exch - exchanges the top two elements on the stack
- dup - copies the top element and places it on the stack
- pop - deletes a value from the stack

Postscript Essentials

A “hello world” example:

```
%!PS
/Courier findfont  % find the font
36 scalefont       % set our font scale
setfont            % set the font
/ar [ (hola) ] def % define an array with 1 string
72 684 moveto      % set our starting point on the page
ar {               % procedure entry for the array
30 string cvs      % convert each item to a string sizeof 30
show               % display it
10 0 rmoveto       % now we relatively move to the next line
} forall           % loop
showpage           % print to the printer
```

Well, it will just print “hola” to the screen

Prior Research

Prior Research

- Ruxcon presentation “A Ghost from Postscript” by Yu Hong (redrain) of Qihoo 360CERT and @SparkZheng of Blue-lotus. They also found the .findlibfile SAFER bypass.
- Adobe Acrobat Distiller .ps OOB Write (CVE-2018-12758) by Zhiyuan Wang of Chengdu Qihoo360
- Various CVE’S (type confusions / SAFER bypasses) by Tavis Ormandy of Google Project Zero
- Various Postscript CharString bugs in ATMFD by Mateusz Jurczyk

Prior Research

- Ghostbutt (CVE-2017-8291) which is a type confusion found by HD Moore
- Adobe Acrobat Distiller .joboptions Font Name Heap Overflow (unknown CVE) found by Paul Craig of Security Assessment
- Buffer Overflow in Distiller (CVE-2006-3453) by Adobe PSIRT

Prior Research

In summary…

- Very little public research targeting Adobe’s postscript parser, acrodistdll.dll.
- Decent amount of public work targeting Ghostscript though.
- Literally, only two CVE’s I could find for postscript parsing bugs in Distiller. CVE-2006-3453 and CVE-2018-12758.
- Adobe’s postscript parser is a closed source target *without*, symbols makes code auditing much harder.

Understanding Adobe's Postscript Engine

Adobe Acrobat Distiller (acrodist.exe)

- We can fuzz via the command line, traditional file format fuzzing:

```
C:\path\to\acrodist.exe C:\path\to\sample.ps
```

- Need full path to the postscript file
- Need to launch GUI on every iteration
- We could use in memory fuzzing and hook parsing functions after a ReadFile call.
  - Maybe hard to reproduce crash cases cleanly

Understanding Adobe's Postscript Engine

Adobe Acrobat Distiller (acrodist.exe)

- We can use the /F flag to allow acrodist.exe to access the filesystem
- Could be a nice security boundary to find bypasses
  - Much *harder* since we don't have source
- Need a way to remove GUI overhead
- Turns out acrodist.exe uses window messaging

Understanding Adobe's Postscript Engine

We can actually build our own client and send window messages to acrodist.exe !

```
DISTILLRECORD dr;
COPYDATASTRUCT cds;
CWnd *hDistillerCWnd = FindWindow("Distiller", NULL);
if (hDistillerCWnd != NULL){
    strcpy(dr.outputFile, "C:\\sample.pdf");
    strcpy(dr.fileList, "C:\\sample.ps");
    dr.param = EQ_NO_SAVE_DIALOG;
    cds.dwData = DM_DISTILL;
    cds.cbData = sizeof(DISTILLRECORD);
    cds.lpData = (PVOID)&dr;
    ok = (BOOL)hDistillerCWnd->SendMessage(WM_COPYDATA,
      (WPARAM)m_hWnd, (LPARAM)&cds);
    if (ok)
        hDistillerCWnd->PostMessage(WM_TIMER, ID_TIMER, 0L);
}
```

Understanding Adobe's Postscript Engine

After some googling, I found distctrl.h which gives some of the definitions

```
    #define DM_CMDLINE  0x4C646D43
    #define DM_DISTILL  0x44696E73
    #define DM_DONE     0x64616C65
```

Still missing some typedef's (for example DISTILLRECORD), but they can be reversed out by hooking SendMessage.

But there are several structures and this could get complicated fast.

Understanding Adobe's Postscript Engine

Adobe Acrobat Distiller (acrodist.exe)

- Found an easier way though. Inside of acrodistdll.dll there is an exported function called: _DistMain@16

- No structures, etc. Just the filename for processing and it will handle all the window messaging for us.

```
distmain = (DistMain)GetProcAddress(hlib, "_DistMain@16");
distmain(0, 0, filename, 4);
```

- This is just the WindowProc prototype!

Understanding Adobe's Postscript Engine

Adobe Acrobat Distiller (acrodist.exe)

- The WindowProc callback into acrodistdll.dll from acrodist.exe is not exported.
- No way to fuzz the target without a GUI called “Distiller”.
- However, we can fuzz using a client vs server model due to window messaging.
- Build our client.exe to send sample.ps to acrodist.exe which is monitored for exceptions.

Understanding Adobe's Postscript Engine

Adobe Acrobat Distiller (acrodist.exe)

- No need for window clickers, failure happens with a log file of the filename sample.log for sample.ps
- We can avoid a log (minimize filesystem interaction) using the command line argument `--deletelog:off`
- Another useful command line param is /O which means you can specify a path to the output file.

Attack Vectors

Attack Vectors

Many file formats that are defined within the bounds of Postscript, we are to covering them all!

- Postscript implementation
    - Filters/Operands
- Encapsulated Postscript File (eps)
- Postscript Fonts
    - Type 1 - Predecessor to OpenType
    - Type 3 - Type 1 postscript without the encryption layer
    - Type 42 - TrueType in Postscript

Attack Vectors

Filters - Used for decompressing user supplied data

- DCTDecode
- CCITTFaxDecode
- FlateDecode
- RunLengthDecode
- LZWDecode
- etc..

Attack Vectors

```
/dctdecode-test
{ /input
    (/path/to/poc.jpg) (r) file
    /ASCIIHexDecode filter /DCTDecode filter
  def
  360 72 translate
  175 47 scale
  500 133 8
  [500 0 0 -133 0 133]
  input
  false
  3
  colorimage
} bind def
dctdecode-test
```

FUZZ

TARGET

I used file to avoid some odd window bug in acrodist.exe

Attack Vectors

```
/flatedecode-test
{{ /input currentfile
      0 (%EndMask) /SubFileDecode filter
      /ASCIIHexDecode filter /FlateDecode filter
    def
    /DeviceGray setcolorspace
  <<
    /Decode [0 1] /BitsPerComponent 8
    /Width 256 /ImageType 1
    /DataSource input
    /ImageMatrix [256 0 0 256 0 0] /Height 256
  >> image
  } stopped pop } bind def
flatedecode-test
41414141...
%EndMask
```

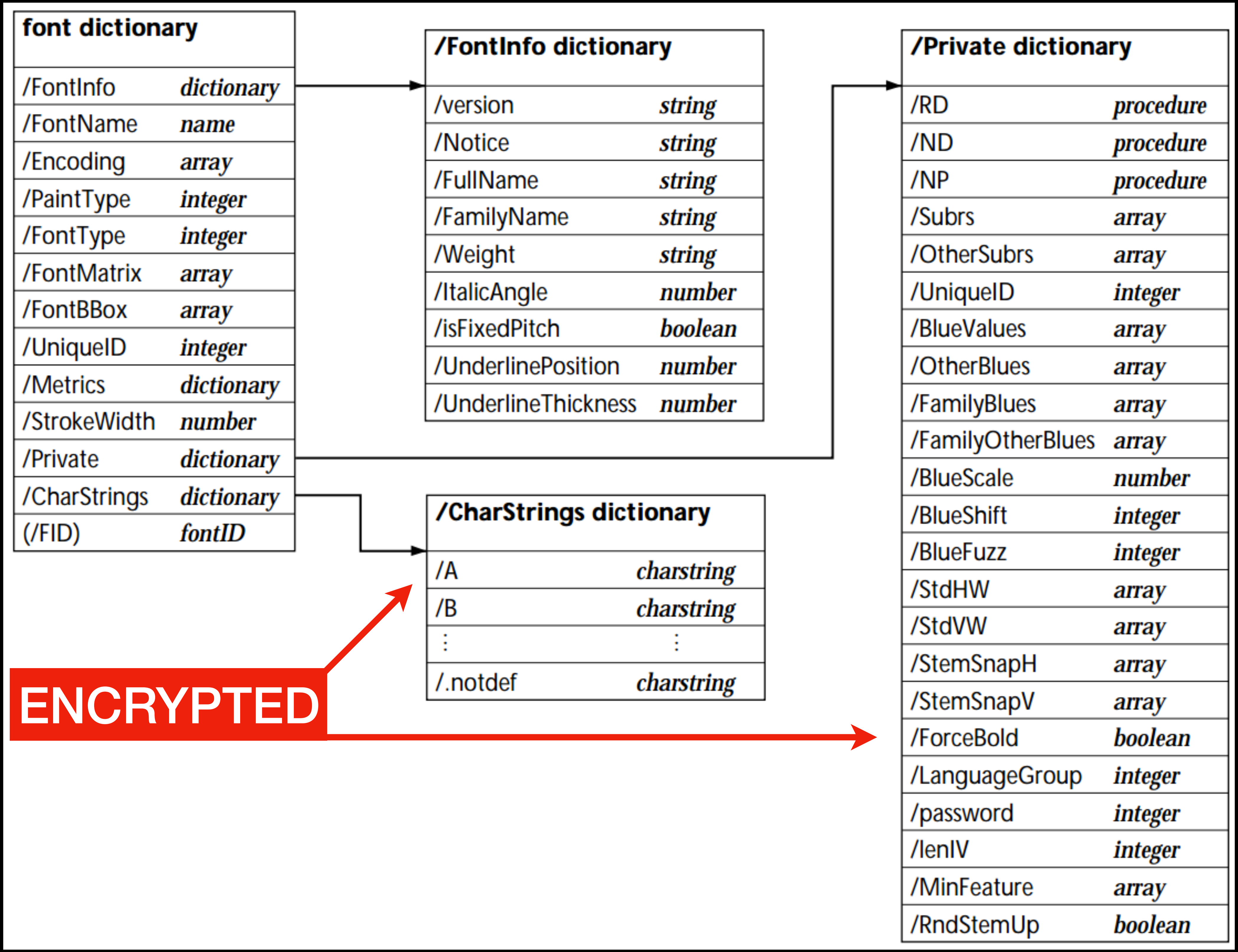
TARGET

FUZZ

Type 1 Font

OpenType predecessor (w/ encryption)

Attack Vectors



font dictionary	
/FontInfo	*dictionary*
/FontName	*name*
/Encoding	*array*
/PaintType	*integer*
/FontType	*integer*
/FontMatrix	*array*
/FontBBox	*array*
/UniqueID	*integer*
/Metrics	*dictionary*
/StrokeWidth	*number*
/Private	*dictionary*
/CharStrings	*dictionary*
(/FID)	*fontID*

/FontInfo dictionary	
/version	*string*
/Notice	*string*
/FullName	*string*
/FamilyName	*string*
/Weight	*string*
/ItalicAngle	*number*
/isFixedPitch	*boolean*
/UnderlinePosition	*number*
/UnderlineThickness	*number*

/CharStrings dictionary	
/A	*charstring*
/B	*charstring*
⋮	⋮
/.notdef	*charstring*

/Private dictionary	
/RD	*procedure*
/ND	*procedure*
/NP	*procedure*
/Subrs	*array*
/OtherSubrs	*array*
/UniqueID	*integer*
/BlueValues	*array*
/OtherBlues	*array*
/FamilyBlues	*array*
/FamilyOtherBlues	*array*
/BlueScale	*number*
/BlueShift	*integer*
/BlueFuzz	*integer*
/StdHW	*array*
/StdVW	*array*
/StemSnapH	*array*
/StemSnapV	*array*
/ForceBold	*boolean*
/LanguageGroup	*integer*
/password	*integer*
/lenIV	*integer*
/MinFeature	*array*
/RndStemUp	*boolean*

Attack Vectors

```
%!PS-AdobeFont-1.1: CMMI10 1.100
%%CreationDate: 1996 Jul 23 07:53:57
11 dict begin
/FontInfo 7 dict dup begin
...
end readonly def
/FontName /CMMI10 def
/PaintType 0 def
/FontType 1 def
/FontMatrix [0.001 0 0 0.001 0 0] readonly def
/Encoding 256 array
0 1 255 {1 index exch /.notdef put} for
...
dup 121 /y put
dup 122 /z put
readonly def
/FontBBox{-32 -250 1048 750}readonly def
/UniqueID 5087385 def
currentdict end
currentfile eexec
D9D66F633B846A97B686A97E45A3D0AA0529731C99A784CCBE85B4993B2EEBDE
3B12D472B7CF54651EF21185116A69AB1096ED4BAD2F646635E019B6417CC77B
```

ENCRYPTED

Attack Vectors

```
static uint16_t cr_default = 4330;

static void
decrypt_charstring(unsigned char *line, int len)
{
  int i;
  int32_t val;
    byte plain;
    ...
    for (i = 0; i < len; i++) {
      byte cipher = line[i];
      plain = (byte)(cipher ^ (cr >> 8));
      cr = (uint16_t)((cipher + cr) * c1 + c2);
      line[i] = plain;
    }
```

STATIC

XOR FTW!!1

Attack Vectors

```
currentfile eexec
dup
/Private 19 dict dup begin
/RD{ ... }executeonly def
/ND{ ... }executeonly def
/NP{ ... }executeonly def
/MinFeature{16 16}ND
/password 5839 def
/UniqueID 5087385 def
/BlueValues [ ... ] ND
/OtherBlues [ -205 -194 ] ND
/BlueScale 0.04379 def
/BlueShift 7 def
/BlueFuzz 1 def
/StdHW [ 31 ] ND
/StdVW [ 72 ] ND
/ForceBold false def
/StemSnapH [ 25 31 ] ND
/OtherSubrs
...
```

```
2 index /CharStrings 41 dict dup begin
/.notdef {
    0 500 hsbw
    endchar
    } ND
/delta {
    42 444 hsbw
    -12 22 hstem
    679 32 hstem
    0 62 vstem
    162 27 vstem
    289 69 vstem
    222 437 rmoveto
    -125 -30 -97 -130 0 -121 rrcurveto
    -96 64 -72 94 vhcurveto
    117 83 157 138 hvcurveto
    0 91 -40 50 -34 45 rrcurveto
    -36 45 -59 75 0 44 rrcurveto
    22 20 24 35 vhcurveto
    30 0 20 -13 21 -14 rrcurveto
    20 -12 20 -13 15 0 rrcurveto
    ...
```

DECRYPTED

TYPE 2

Type 3 Font

OpenType Predecessor (w/o encryption)

Attack Vectors

```
%!PS-AdobeFont-1.0: ALSandra
...
%%EndComments

11 dict begin
/FontType 3 def
...
dup 252/udieresis put
dup 255/ydieresis put
readonly def
/BuildChar { ... /BuildGlyph get exec } bind def
/BuildGlyph { ... /CharProcs ... } bind def
/CharProcs 183 dict def
CharProcs begin
  /exclam { 339 0 106 -47 217 866 setcachedevice
    213 776 moveto
     213 750 212.333 711 211 659 curveto
     209.667 607 209 568 209 542 curveto
     209 509.333 205.833 459.333 199.5 392 curveto
     193.167 324.667 190 274.333 190 241 curveto
     190 232.333 192.167 221.167 196.5 207.5 curveto
     200.833 193.833 203 184 203 178 curveto
     203 170 195.667 159 181 145 curveto
```

UNENCRYPTED

Type 42 Font

Postscript TrueType

Attack Vectors

```
%!PS-TrueTypeFont
...
%%EndComments
12 dict begin
  /FontName /ALSandra def
  /FontType 42 def
  /FontMatrix [1 0 0 1 0 0] def
  /PaintType 0 def
  /FontBBox {-0.449 -0.941281 1.779 1.132 }readonly def
/FontInfo 10 dict dup begin
 /version (Macromedia Fontographer 4.1.5 5/24/04) readonly def
 /Notice (\050c\051 Copyright 2004 Autumn Leaves. All rights reserved.) readonly
 /FullName (AL Sandra) readonly def
 /FamilyName (AL Sandra) readonly def
 /Weight (Book) readonly def
 /FSType 1 def
 /ItalicAngle 0 def
 /isFixedPitch false def
 /UnderlinePosition -0.143 def
 /UnderlineThickness 0.02 def
end readonly def
...
```

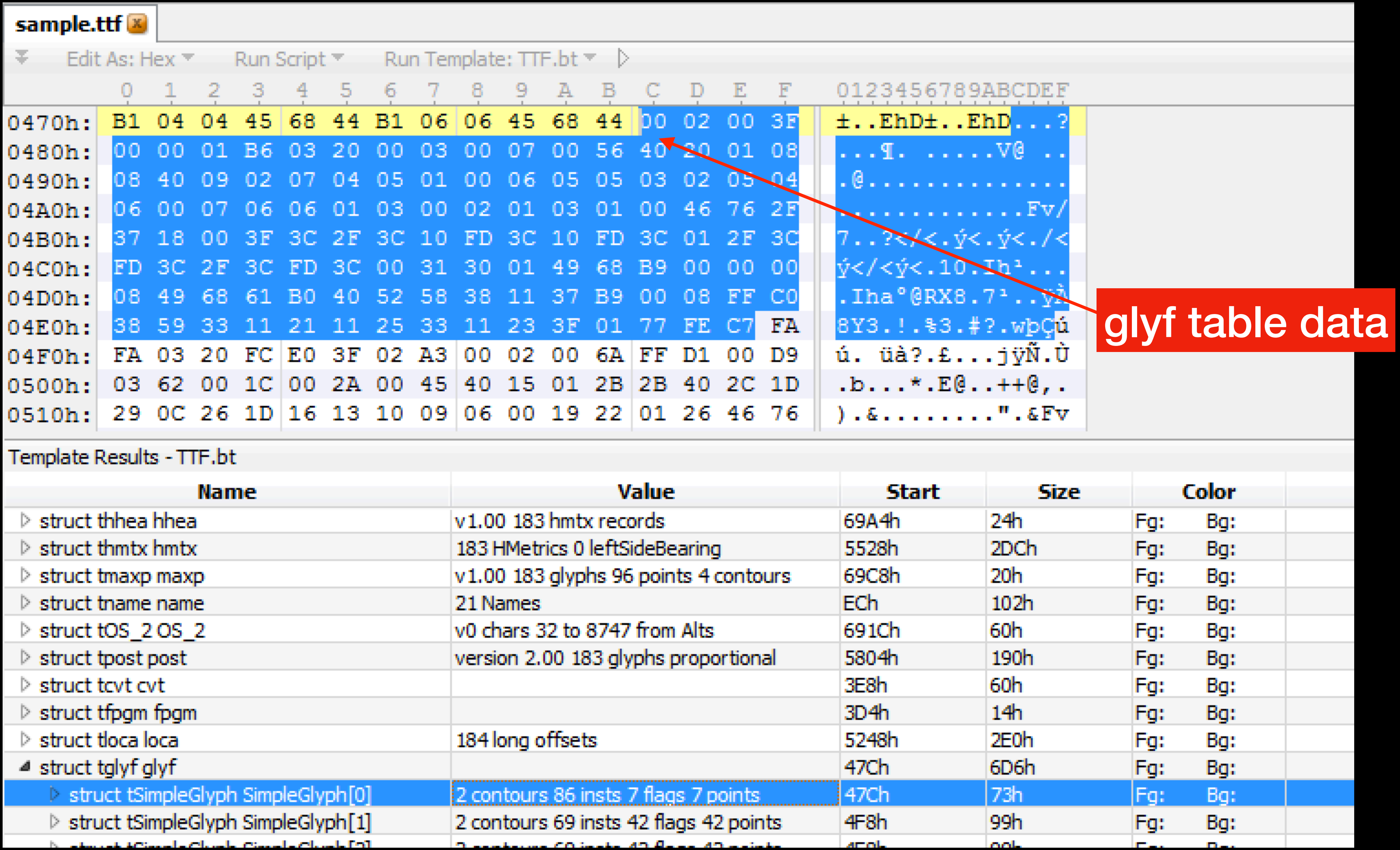
DETAILS

Attack Vectors

glyf table data

```
    ...
    dup 251/ucircumflex put
    dup 252/udieresis put
    dup 255/ydieresis put
  readonly def
  /sfnts [
  ...
<
0002003F000001B603200003000700564020010808400902070405010
05
0503020504060007060601030002010301004676 2F3718003F3C2F3C1
3C
...
>
```

Attack Vectors



```
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0470h: B1 04 04 45 68 44 B1 06 06 45 68 44 00 02 00 3F  ±..EhD±..EhD...?
0480h: 00 00 01 B6 03 20 00 03 00 07 00 56 40 20 01 08  ...¶. .....V@ ..
0490h: 08 40 09 02 07 04 05 01 00 06 05 05 03 02 05 04  .@..............
04A0h: 06 00 07 06 06 01 03 00 02 01 03 01 00 46 76 2F  .............Fv/
04B0h: 37 18 00 3F 3C 2F 3C 10 FD 3C 10 FD 3C 01 2F 3C  7..?</<.ý<.ý<./<
04C0h: FD 3C 2F 3C FD 3C 00 31 30 01 49 68 B9 00 00 00  ý</<ý<.10.Ih¹...
04D0h: 08 49 68 61 B0 40 52 58 38 11 37 B9 00 08 FF C0  .Iha°@RX8.7¹..ÿÀ
04E0h: 38 59 33 11 21 11 25 33 11 23 3F 01 77 FE C7 FA  8Y3.!.%3.#?.wþÇú
04F0h: FA 03 20 FC E0 3F 02 A3 00 02 00 6A FF D1 00 D9  ú. üà?.£...jÿÑ.Ù
0500h: 03 62 00 1C 00 2A 00 45 40 15 01 2B 2B 40 2C 1D  .b...*.E@..++@,.
0510h: 29 0C 26 1D 16 13 10 09 06 00 19 22 01 26 46 76  ).&........".&Fv
```

Template Results - TTF.bt

Name	Value	Start	Size	Color	
struct thhea hhea	v1.00 183 hmtx records	69A4h	24h	Fg:	Bg:
struct thmtx hmtx	183 HMetrics 0 leftSideBearing	5528h	2DCh	Fg:	Bg:
struct tmaxp maxp	v1.00 183 glyphs 96 points 4 contours	69C8h	20h	Fg:	Bg:
struct tname name	21 Names	ECh	102h	Fg:	Bg:
struct tOS_2 OS_2	v0 chars 32 to 8747 from Alts	691Ch	60h	Fg:	Bg:
struct tpost post	version 2.00 183 glyphs proportional	5804h	190h	Fg:	Bg:
struct tcvt cvt		3E8h	60h	Fg:	Bg:
struct tfpgm fpgm		3D4h	14h	Fg:	Bg:
struct tloca loca	184 long offsets	5248h	2E0h	Fg:	Bg:
struct tglyf glyf		47Ch	6D6h	Fg:	Bg:
struct tSimpleGlyph SimpleGlyph[0]	2 contours 86 insts 7 flags 7 points	47Ch	73h	Fg:	Bg:
struct tSimpleGlyph SimpleGlyph[1]	2 contours 69 insts 42 flags 42 points	4F8h	99h	Fg:	Bg:

Attack Vectors

- Type 42 Postscript fonts are parsed and rasterized!
- This means we have a large attack surface for just TrueType char-strings to attack the vm

```
SRP0[ ]         /* SetRefPoint0 */
MIRP[11101]     /* MoveIndirectRelPt */
ALIGNRP[ ]      /* AlignRelativePt */
SRP0[ ]         /* SetRefPoint0 */
MIRP[11101]     /* MoveIndirectRelPt */
ALIGNRP[ ]      /* AlignRelativePt */
SVTCA[1]        /* SetFPVectorToAxis */
MDAP[1]         /* MoveDirectAbsPt */
ALIGNRP[ ]      /* AlignRelativePt */
```

PAT

Postscript Auditing Toolkit

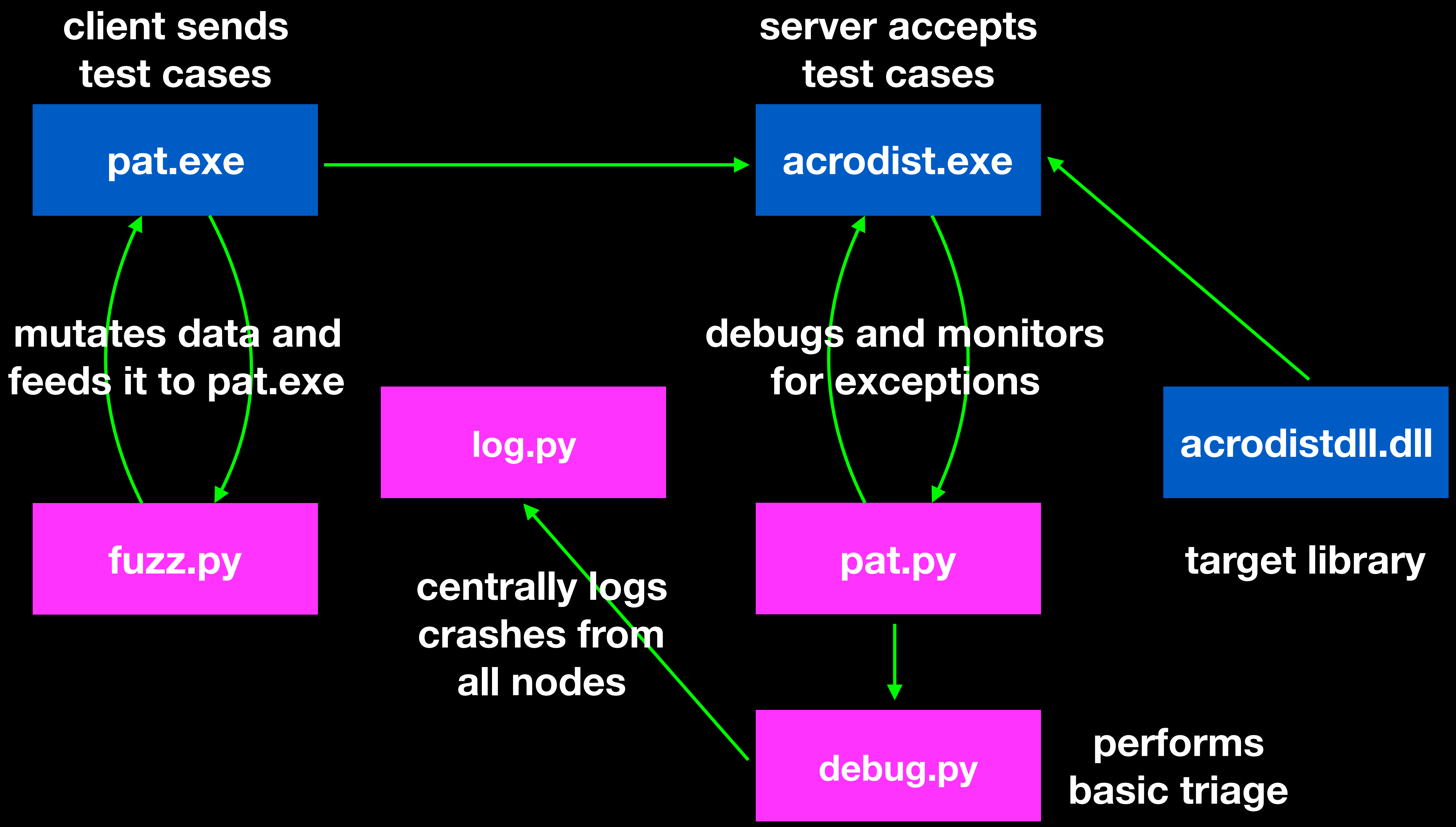
Design and Architecture

I wrote a vulnerability scanner that abstracts all the predicates in a binary, traverses the callgraph and generates phormulaes to run then with a SMT solver. I found 1 vuln in 3 days with this tool.
He wrote a dumb ass fuzzer and found 5 vulns in 1 day.
Good thing I'm not a n00b like that guy.

Design and Architecture

1. Start and monitor acrodist.exe for exceptions
2. Send mutated data to acrodist.exe via window messaging
3. Configuration is controlled in fuzz.ini
    - TYPE - the fuzzing file format type. Valid values are ps, jpg and lzw
    - INPUT_DIRECTORY - directory of samples to use (not always needed)
    - TEST_TIMEOUT - how fast we should send test cases to the server. Default is 0.5 seconds
    - MUTATOR - the mutation engine to use, specified in the documentation
    - IGNORE_CRASHES - the crashes for the debugger to ignore
    - RESTART_TIMEOUT - how often to restart the acrodist.exe server. Default is 5 minutes.

Design and Architecture


client sends test cases
pat.exe
server accepts test cases
acrodist.exe
mutates data and feeds it to pat.exe
debugs and monitors for exceptions
log.py
acrodistdll.dll
fuzz.py
pat.py
target library
centrally logs crashes from all nodes
debug.py
performs basic triage

Lexers

Postscript Auditing Toolkit

Lexers

In order to parse postscript, I built several lexers for specific datatypes in postscript based on Adobes specifications

They answer the questions:

1. How many operands/arrays/strings/etc are in the file ?
2. Where are they positioned?
3. What are the values?

This allows us to perform targeted, semi-smart mutation fuzzing

Lexers

For speed, I built a lexer for each data type

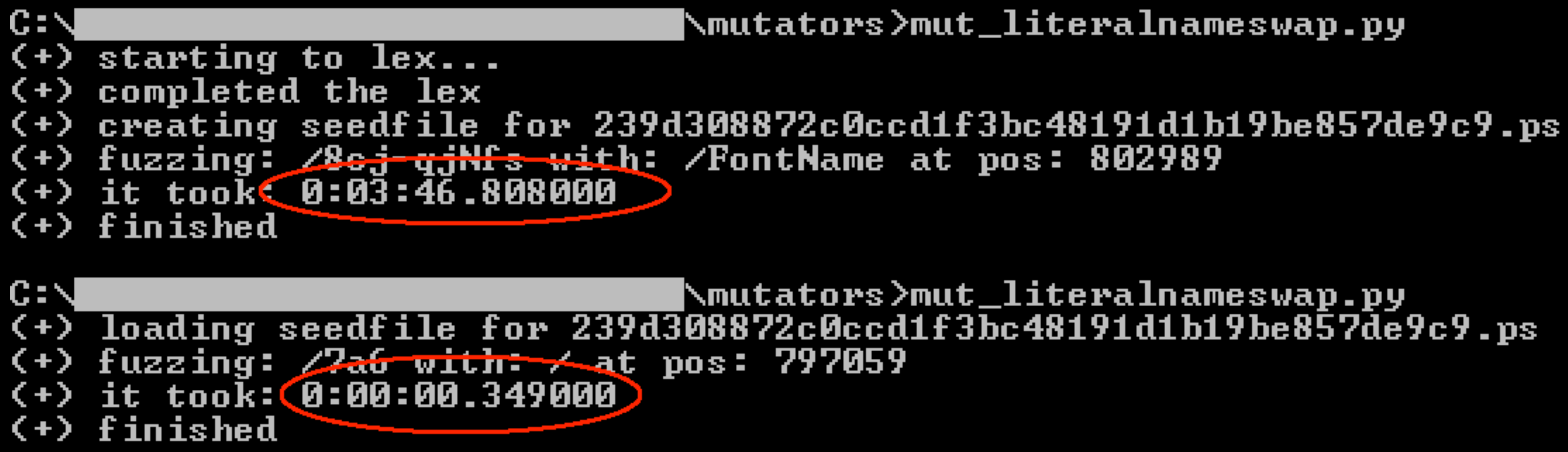
```
if self.t == GENERIC:
    from lexers.pslexergeneric import GenericLexer
    m = GenericLexer()
    m.build()
    self.tokens = m.tokenize(data)
elif self.t == COMMENT:
    ...
elif self.t == NUMBER:
    ...
elif self.t == STRING:
    ...
elif self.t == PROCEDURE:
    ...
elif self.t == ARRAY:
    ...
elif self.t == LITERAL_NAME:
    ...
elif self.t == OPERATOR:
    ...
```

Lexers

Speed is a factor, depending on the size of the postscript file, the lexing process can take well over a minute!

Solution:

Use a caching mechanism.



```
C:\[redacted]\mutators>mut_literalnameswap.py
(+) starting to lex...
(+) completed the lex
(+) creating seedfile for 239d308872c0ccd1f3bc48191d1b19be857de9c9.ps
(+) fuzzing: /8cj-qjNfs with: /FontName at pos: 802989
(+) it took: 0:03:46.808000
(+) finished

C:\[redacted]\mutators>mut_literalnameswap.py
(+) loading seedfile for 239d308872c0ccd1f3bc48191d1b19be857de9c9.ps
(+) fuzzing: /7a6 with: / at pos: 797059
(+) it took: 0:00:00.349000
(+) finished
```

Grammers

Postscript Auditing Toolkit

Postscript Grammer

Generic Postscript Generator - fuzzgenps.py

Pretty much calls all different postscript operators with different postscript arguments and values.

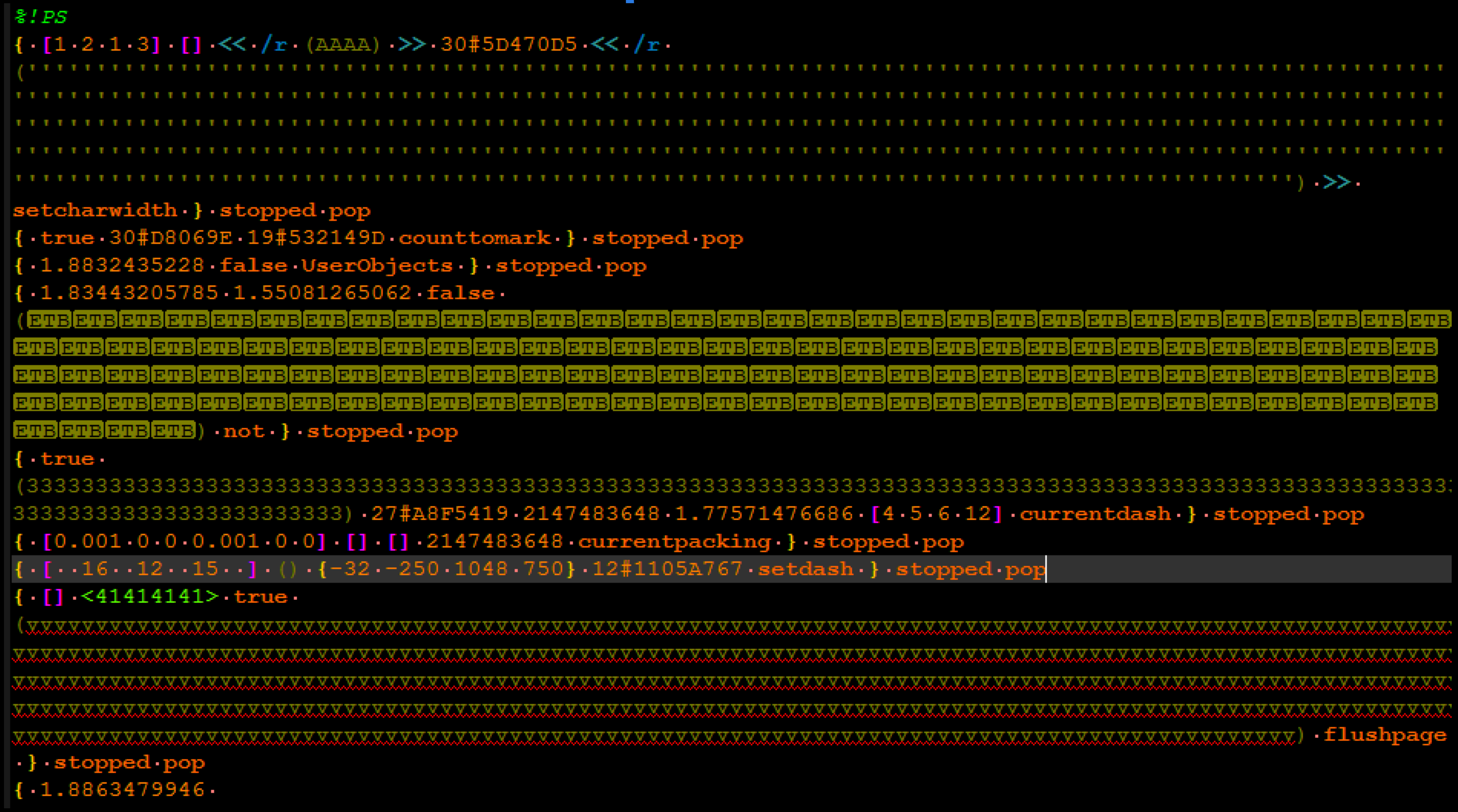
It’s possible to have a try/catch in Postscript via:

```
{ <FUZZ> } stopped {} if

{ <FUZZ> } stopped pop
```

stopped will take a procedure and execute it and place a boolean on the stack. True if it stopped. So we can pop it.

Postscript Grammer



```
%!PS
{·[1·2·1·3]·[]·<<·/r·(AAAA)·>>·30#5D470D5·<<·/r·
(''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''
''''''''''''''''''''''''''''''''''''''''''''''''''''
'''''''''''''''''''''''''''''''''''''''''''''''')·>>·
setcharwidth·}·stopped·pop
{·true·30#D8069E·19#532149D·counttomark·}·stopped·pop
{·1.8832435228·false·UserObjects·}·stopped·pop
{·1.83443205785·1.55081265062·false·
(ETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETB
ETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETB
ETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETB
ETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETBETB
ETBETBETBETB)·not·}·stopped·pop
{·true·
(3333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333333
333333333333333333333333)·27#A8F5419·2147483648·1.77571476686·[4·5·6·12]·currentdash·}·stopped·pop
{·[0.001·0·0·0.001·0·0]·[]·[]·2147483648·currentpacking·}·stopped·pop
{·[··16··12··15··]·()·{-32·-250·1048·750}·12#1105A767·setdash·}·stopped·pop
{·[]·<41414141>·true·
(yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy
yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy)·flushpage
·}·stopped·pop
{·1.8863479946·
```

Mutators

Postscript Auditing Toolkit

"Designing the mutation engine for a new fuzzer has more to do with art than science"

–Michal Zalewski (@lcamtuf)

Dumb Mutators

Since I am targeting several filters I use some generic flippers - byteflip/ byteflipascii

The byteflip is primarily for targeting display operands and the byteflipascii is primarily for targeting filters.

- Display operand fuzzing

Fuzzing a format (such as jpeg or zlib) before using it in a postscript template which is then parsed to pat.exe. Targets operands such as colorimage

- Post-insertion

Fuzzing the eps file format after a format has already been inserted which is then parsed to pat.exe

Operand Mutators

Manipulates the postscript operands - operandswap

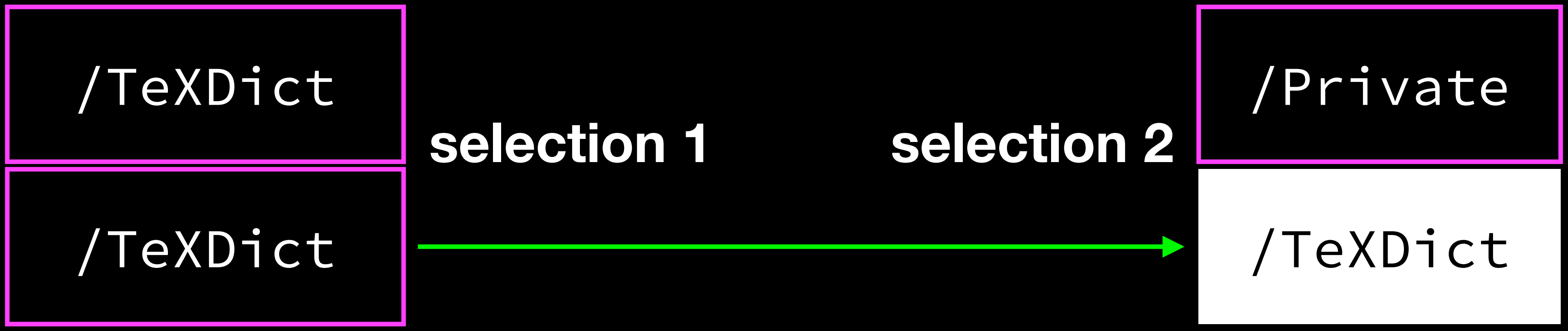
The operandswap mutator actually chooses at runtime between swapping in file operands or using an operand from a pre-defined list.

- Using a predefined list, this is essentially a single change
- Using an in file operand, it essentially is two changes

Datatype Mutators

So far I have only a single datatype mutator:
literalnameswap

The first essentially swaps literal names such as /TeXDict with /Private, but *not* /Private with /TextDict.



Private becomes /TeXDict but the original /TeXDict stays the same. Parsing is top down :->

Charstring Mutators

Type 1 Font

The FontForge application has built in python bindings to convert binary fonts to several postscript font standards

1. Convert the TTF to Type 1 using FontForge

```
C:\PROGRA~1\FontForgeBuilds\bin\ffpython.exe ttf2t1.py
```

```
import fontforge
font = fontforge.open("sample.ttf")
font.generate("sample.t1")
```

Charstring Mutators

Type 1 Font

The t1utils package provides the ability to decrypt a type 1 font, no need to implement this myself!

2. Decrypt the type 1 postscript font

```
t1disasm sample.t1 sample.t1.decrypted.ps
```

3. Modify the decrypted file to insert/replace char-string dicts

4. Re-encrypt the modified type 1 postscript font

```
t1asm sample.t1.decrypted.ps fuzzed.t1.ps
```

Charstring Mutators

Type 3 Font

This is the easiest, nothing is encrypted or encoded

1. Convert the TTF to Type 3 using FontForge

```
C:\PROGRA~1\FontForgeBuilds\bin\ffpython.exe ttf2t3.py
```

```
import fontforge
font = fontforge.open("sample.ttf")
font.generate("sample.t3")
```

2. Modify the file to insert/replace char-string dicts

Charstring Mutators

Type 42 Font

1. Use font-tools to get a TTX file

```
from fontTools.ttLib import TTFont
font = TTFont('sample.ttf')
font.saveXML('sample.ttx')
```

2. Modify the TTX file to insert/replace char-string dicts

These use the full TrueType instruction set which gives us a huge attack surface

Charstring Mutators

```
<TTGlyph name="a" xMin="19" yMin="-19" xMax="518" yMax="230">
    <contour>
        <pt x="518" y="17" on="1"/>
        ...
    </contour>
    <instructions>
    <assembly>
    ...
    MDAP[0]      /* MoveDirectAbsPt */
    MDAP[0]      /* MoveDirectAbsPt */
    MDAP[0]      /* MoveDirectAbsPt */
    MDAP[0]      /* MoveDirectAbsPt */
    SVTCA[0]     /* SetFPVectorToAxis */
    ...
```

Charstring Mutators

Type 42 Font

3. Convert the TTF to Type 42 using FontForge

```
C:\PROGRA~1\FontForgeBuilds\bin\ffpython.exe ttf2t42.py
```

```
import fontforge
font = fontforge.open("sample.ttf")
font.generate("sample.t42")
```

4. Now use the font by adding some postscript

```
/ALSandra findfont 12 scalefont
setfont newpath 50 700 moveto
(font fuzzing) show showpage
```

Limitations

Limitations

- Speed
  - Bottlenecked via ps input processing speeds and disk I/O
  - I kept hitting an OOB bug when sending multiple ps files using colorimage. This slowed down filter fuzzing.
  - Even though tokens were cached, we still had some overhead here
  - Runs on Windows, and I built my fuzzer in python…
- Scaling
  - All the tests were performed with literally 4 VM’s running on ram-disk.

Results

“You don’t have a fuzzing result until you have …one billion iterations”

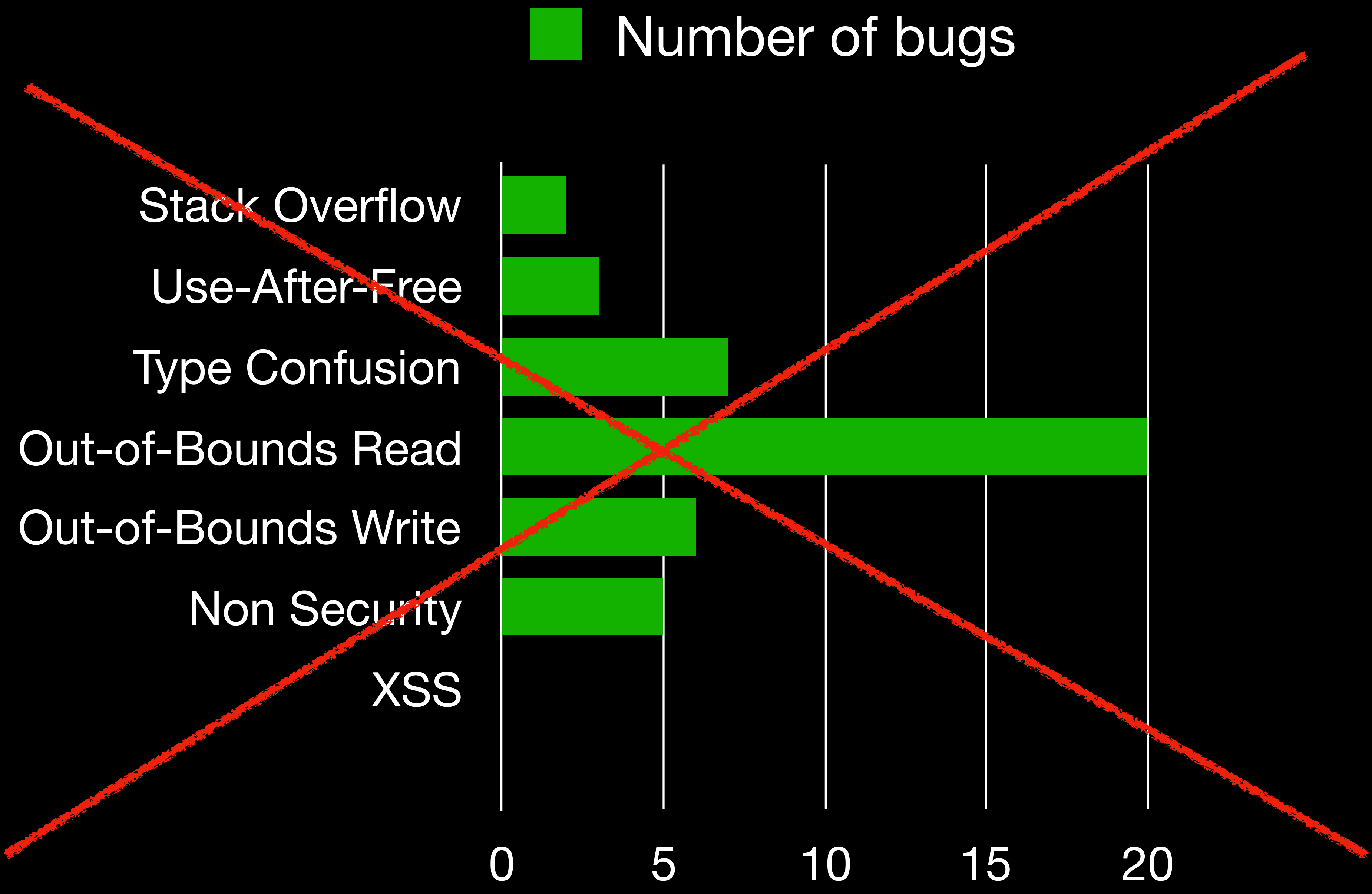
–Ben Nagy (@rantyben)

Results

Corpus Distilling

- When fuzzing the /DCTDecode filter I used an initial seed corpus of ~ 50,000 1Mb valid jpegs
- This was reduced using DynamoRIO drcov.exe and some custom tooling
- Reduced set was ~ 10% of total initial corpus
- This all was done on those same 4 VM's

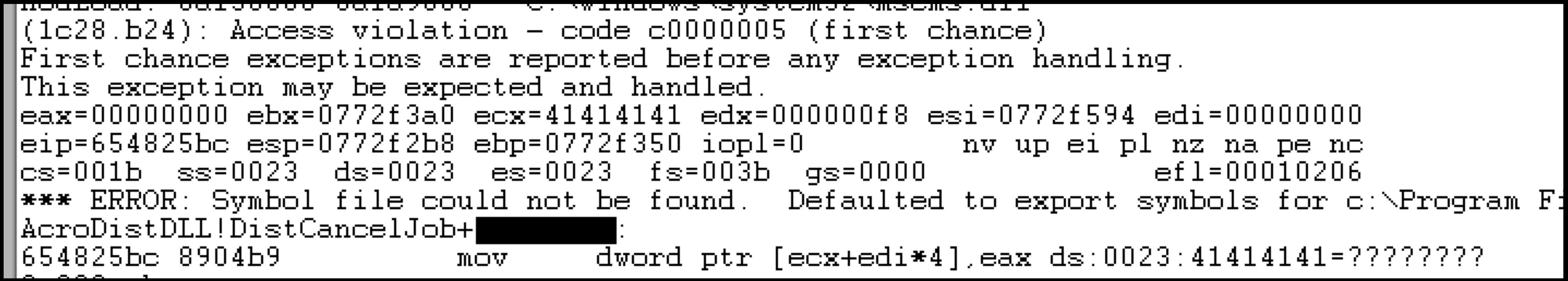
Results*



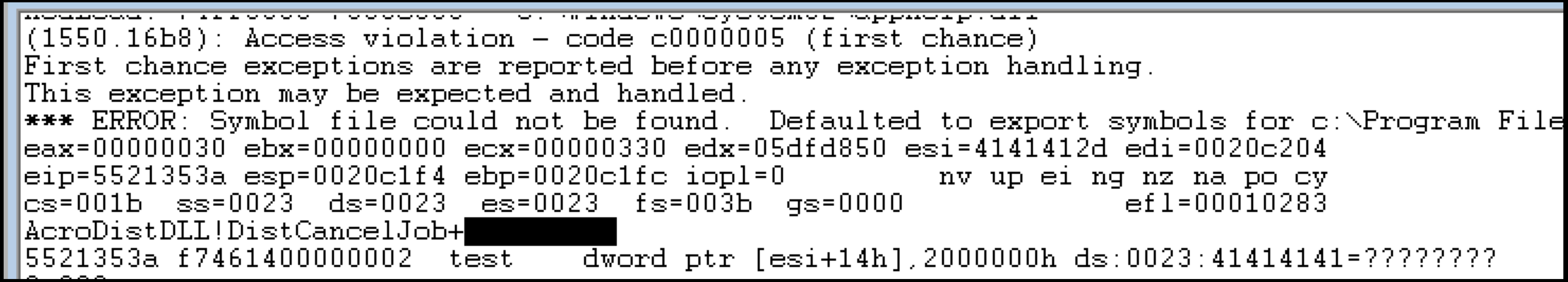
A total of 80+ security vulnerabilities uncovered

*** Too many to count**

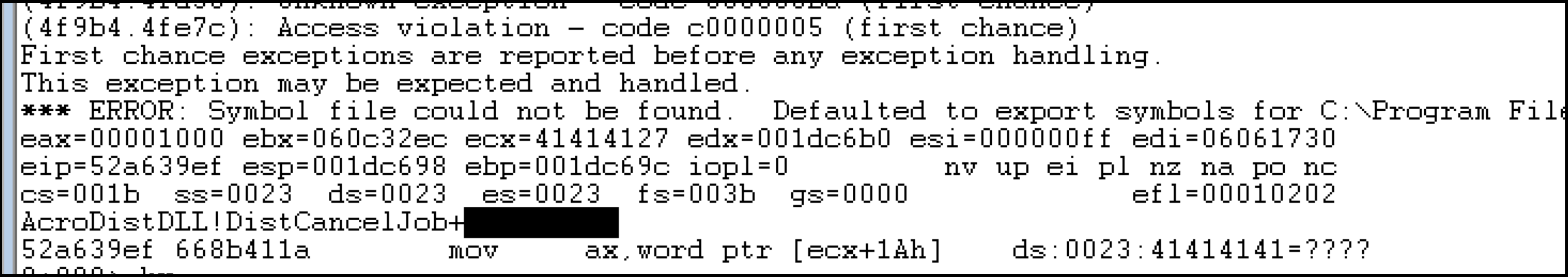
Exploitation primitives



```
(1c28.b24): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=0772f3a0 ecx=41414141 edx=000000f8 esi=0772f594 edi=00000000
eip=654825bc esp=0772f2b8 ebp=0772f350 iopl=0         nv up ei pl nz na pe nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010206
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for c:\Program Fi
AcroDistDLL!DistCancelJob+[redacted]:
654825bc 8904b9          mov     dword ptr [ecx+edi*4],eax ds:0023:41414141=????????
```



```
(1550.16b8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for c:\Program File
eax=00000030 ebx=00000000 ecx=00000330 edx=05dfd850 esi=4141412d edi=0020c204
eip=5521353a esp=0020c1f4 ebp=0020c1fc iopl=0         nv up ei ng nz na po cy
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010283
AcroDistDLL!DistCancelJob+[redacted]
5521353a f7461400000002  test    dword ptr [esi+14h],2000000h ds:0023:41414141=????????
```



```
(4f9b4.4fe7c): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
*** ERROR: Symbol file could not be found.  Defaulted to export symbols for C:\Program File
eax=00001000 ebx=060c32ec ecx=41414127 edx=001dc6b0 esi=000000ff edi=06061730
eip=52a639ef esp=001dc698 ebp=001dc69c iopl=0         nv up ei pl nz na po nc
cs=001b  ss=0023  ds=0023  es=0023  fs=003b  gs=0000             efl=00010202
AcroDistDLL!DistCancelJob+[redacted]
52a639ef 668b411a        mov     ax,word ptr [ecx+1Ah]    ds:0023:41414141=????
```

Conclusion

- Postscript is a HUGE attack surface and is very hard to secure due to the nature of scripting environments
- There exist no virtual machine specific mitigations to prevent/slow down exploitation of specific bug classes
- Not touched by many researchers probably due to the nature and complexity of postscript
- Project is still in an execution state, please come back in 4 months

Future Work

Honestly I have hardly scratched the surface:

- Mutators
    - Attack more datatypes
- Grammers
    - Use a proper grammar engine!
- More reversing for attack surface
- More fuzzing
- More custom tooling
- Targeting non-postscript vectors

References

- PostScript Language Document Structuring Conventions Specification - https://www-cdf.fnal.gov/offline/PostScript/5001.PDF
- Encapsulated PostScript File Format Specification - https://www-cdf.fnal.gov/offline/PostScript/5002.PDF
- The Postscript Level 2 Language Specification - https://www-cdf.fnal.gov/offline/PostScript/PLRM2.pdf
- The Postscript Level 3 Language Specification - https://www-cdf.fnal.gov/offline/PostScript/PLRM3.pdf
- The Postscript 3 Core Font Set - https://www-cdf.fnal.gov/offline/PostScript/PL3corefont.pdf
- Adobe Type 1 Font Format - https://www-cdf.fnal.gov/offline/PostScript/T1_SPEC.PDF

References

- Adobe Type 1 Font Format Supplement - https://www-cdf.fnal.gov/offline/PostScript/5015.Type1_Supp.pdf
- Adobe Type 42 Font Format Specification - https://www-cdf.fnal.gov/offline/PostScript/5012.pdf
- The Postscript Language Tutorial and Cookbook (bluebook) - https://www-cdf.fnal.gov/offline/PostScript/BLUEBOOK.PDF
- The Postscript Language Program Design (greenbook) - https://www-cdf.fnal.gov/offline/PostScript/GREENBK.PDF
- Filters and Reusable Streams - https://www.adobe.com/content/dam/acom/en/devnet/postscript/pdfs/TN5603.Filters.pdf
- Acrobat Distiller API Reference - https://www.adobe.com/content/dam/acom/en/devnet/acrobat/pdfs/Distiller9APIReference.pdf

References

- A Ghost from Postscript - https://ruxcon.org.au/assets/2017/slides/hong-ps-and-gs-ruxcon2017.pdf
- A year of Windows kernel font fuzzing #2: the techniques - https://googleprojectzero.blogspot.com/2016/07/a-year-of-windows-kernel-font-fuzzing-2.html
- One font vulnerability to rule them all #1: Introducing the BLEND vulnerability - https://googleprojectzero.blogspot.com/2015/07/one-font-vulnerability-to-rule-them-all.html
- Ben Nagy quote - https://twitter.com/rantyben/status/755575547460059136
- Michal Zalewski quote - https://lcamtuf.blogspot.com/2014/08/binary-fuzzing-strategies-what-works.html
- Malformed .joboptions File Effecting Adobe Acrobat Distiller v8 - https://security-assessment.com/files/documents/advisory/2008-05-15_Acrobat_Distiller_Malformed_joboptions_File.pdf

References

- Adobe Acrobat Distiller PostScript Arbitrary Code Execution Vulnerability - https://tools.cisco.com/security/center/viewAlert.x?alertId=58614
- Ghostscript shell command execution in SAFER mode - https://lgtm.com/blog/ghostscript_CVE-2018-19475
- The Type 2 Charstring Format https://www.adobe.com/content/dam/acom/en/devnet/font/pdfs/5177.Type2.pdf

Thanks! Questions?

esteban@srcincite.io / @steventseeley