bØrken fonts

The Story of Naïve Parsers and attacker controlled reboots

Marc Schönefeld, Security Researcher

- Working daytime for Red Hat, busy with Java bugs
- Did talks on Security and RE topics on several conferences before (CSW, PacSec, HITB, SyScan, Blackhat, …)
- Most vulnerabilities published until end of 2009 were java-centric or web application flaws
- The results presented in this talk presents research done „after hours“, hence this is no official Red Hat talk

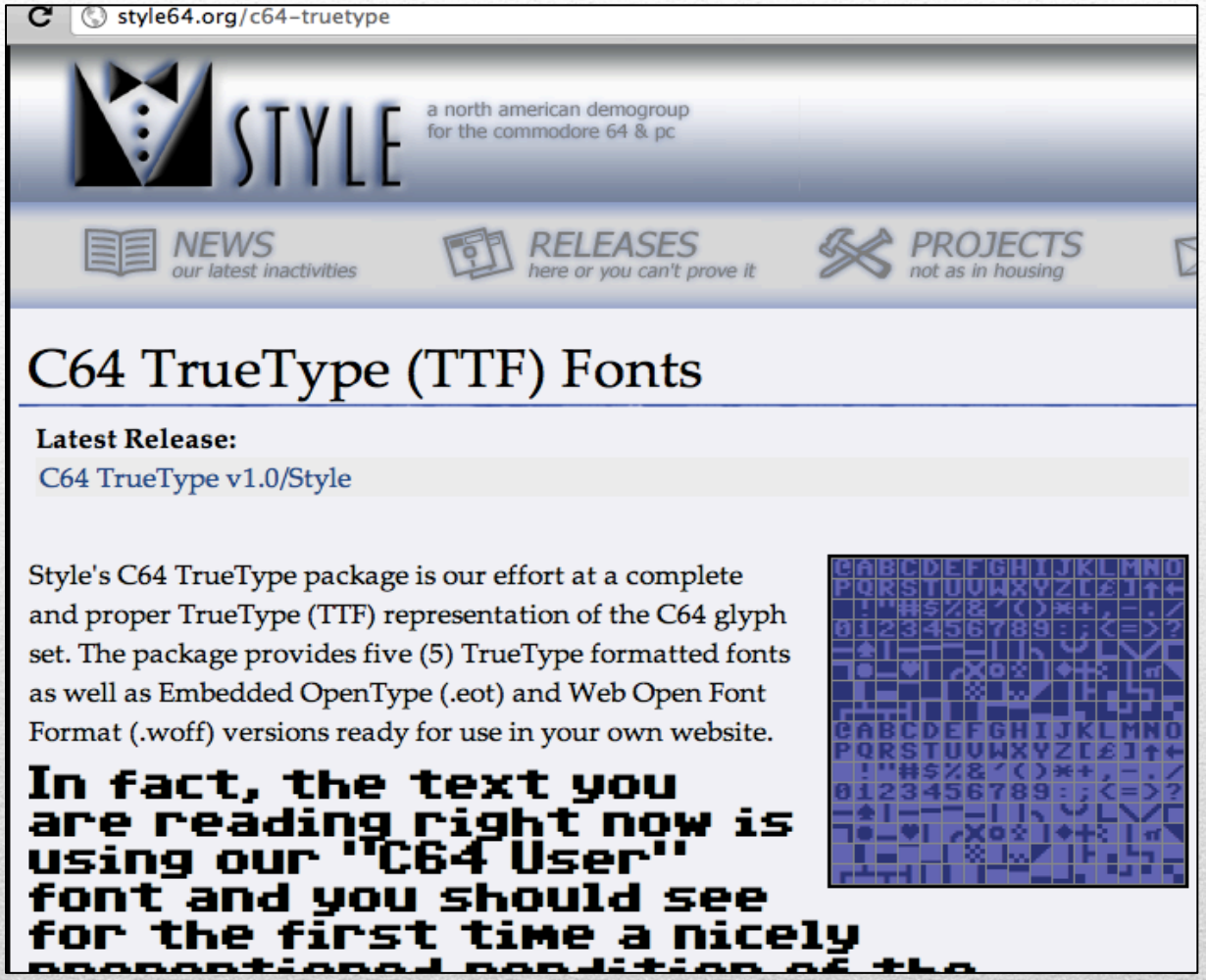
The Speaker

- Initial goal was to broaden my horizon on web technologies and tools
- End of 2009 I thought like there must be some shorter way to find software bugs than by reading code ☺
  - Ideally find a cross-OS attack surface
  - Not too many vulnerabilities reported yet
  - A decent potential to write some new tool
- Hmm, what technology fits this criteria ?

On the search

- The web is a large playground for new technologies
- New features often introduced in products with eye on functionality only, who cares for security if it looks nice
- So did HTML5 by introducing the webfont feature
  - http://www.w3.org/TR/css3-webfonts/#font-descriptions
- From an attackers perspective grabbing fonts from an untrusted site for local rendering is a tempting invitation

New technologies new bugs



Fonts in browsers

CAN SEC WEST

Font-Face Syntax

```
@font-face {
 font-family: ‘CBM';
 src: url(’CBM.eot');
 src: local('☺'),
 url("CBM.woff") format("woff"),
 url(”CBM.ttf") format("truetype"),
 url(”CBM.otf") format("opentype"),
 url(”CBM.svg") format("svg");
}
```

The small print

- A font is represented by logical name
- To be used in CSS family declaration
- Once the browser discovered a compatible font it is used for rendering
- Font origin != page origin (font-kit, google fonts)
- So could catch one from malware.com too (XSS)

Fonts in browsers

Fonts in browsers

- Playing around with **zzuf** and pango-view
  - Displaying some unicode chars (parsing and rendering)
  - In a simple for loop
- That manual approach brought up a memory lookup bug

CVE-ID	
CVE-2010-0421 (under review)	Learn more at National Vulnerability Database (NVD) • Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings

Description

Array index error in the hb_ot_layout_build_glyph_classes function in pango/opentype/hb-ot-layout.cc in Pango before 1.27.1 allows context-dependent attackers to cause a denial of service (application crash) via a crafted font file, related to building a synthetic Glyph Definition (aka GDEF) table by using this font's charmap and the Unicode property database.

References

- CONFIRM:http://ftp.gnome.org/pub/GNOME/sources/pango/1.27/pango-1.27.1.tar.bz2

The first discovery

- Browsers delegate the font rendering work to the underlying OS library, such as Pango
- OS font libs are originally designed for the happy-go-lucky cases, assuming local fonts are trusted
- The threat model changed drastically with the release of browsers that support web fonts (Firefox 3.6 , Chrome 5, …),
  - Pango renders for FFx, 3.0.x => 3.6.x s/local/remote/
  - On OSX you have Apple Type Services, on Windows ATMFD and Uniscribe
- Browsers have to protect against direct attacks against os libs, but we'll see they don't do a good job
- First line of defence would be a sanitizer for rogue font data (which can have bugs too), we'll come back to that

The new threat model

- Being not a font expert I learnt that all font formats are Glyph data + meta data , stored in specific tables
- Additional a good reference guide is invaluable :
  - Like “Fonts & Encodings” from Yannis Haralambous,
  - on over 1000 pages fonts every aspect of fonts is dissected
- A great tool to explore fonts is the ttx tool set
  - http://sourceforge.net/projects/fonttools/files/
  - Written in Python, easily extensible, very helpful when trying to fuzz only specific tag ranges

GASP, CMAP, what‘s that ? Learning about Fonts

CAN SEC WEST

```
]ttx -l CBM.ttf
Listing table info for ".//CBM.ttf":
    tag     checksum   length   offset
    ----  ----------  -------  -------
    FFTM  0x48151085        28    11708
    OS/2  0x51e82100        86      376
    cmap  0x1d5eff85       330      864
    cvt   0x00440511         4     1196
    feat  0xc00e0454        44    11736
    gasp  0xfffe000f         8    11700
    glyf  0xef4054b9      8460     1592
    head  0xe961f826        54      252
    hhea  0x057a029b        36      308
    hmtx  0x1e9719cf       398      464
    loca  0x73577b9e       392     1200
    maxp  0x01080055        32      344
    morx  0x010ca7b3       368    11780
    name  0xe1407369      1168    10052
    post  0xaf11f0e6       480    11220
```

A font is more than just vectors

Tag	Usage
OS/2	Metrics
CMAP	char to glyph mapping
cvt	Control Value table
feat	Layout feature table
gasp	Grid fitting and scan conversion
glyf	Glyf outline table
Head	Font header table
hhea	Horizontal header table
fpgm	Font program table (*bytecode*)
CFF	Compact Font Program (*bytecode*)
[...]	

Fonts = more than just vectors

- In order to automate font fuzzing we need the following:
  - A fuzzing engine
    - Dynamically Serve Content , generate font data on the fly
  - Browser integration
    - Start the browser in a subprocess for each test case
    - Run the fuzzer in an external CGI / JSP script
    - Run the fuzzer inside browser process
  - Structure awareness
    - Dumb fuzzing or
    - Ability to fuzz a certain structure (like the range of cmap only)
  - Crash analysis
    - Valgrind , Crashwrangler, !exploitable

Testing methodology

- Jan / Feb 2010
- Fuzzing Engine:
  - Dumb fuzzing with external zzuf process
  - Not structure aware
- Browser integration:
  - Browser start for every iteration
- Fuzzing method not structure aware
- Summary:
  - Bjarne Stroustrup rule #1: Learn from the prototype, but throw it away, so I did, because
  - terribly slow, hard to reproduce and continue interrupted
  - Missed a lot of cases due to caching effects

Fuzzer generation zero

- Zzuf
  - Written by Sam Hocevar , released under the WTFPL
  - http://caca.zoy.org/attachment/wiki/zzuf/zzuf-20070225.pdf
  - Fine-granular control over fuzzing parameters
- Zzuf supports flexible fuzzing parameter
  - Seed = The config param for the random generator (-s)
  - Ratio = The density of fuzzed bits within file (-r 0.001 = 0.1%)
  - Range = The fuzzed area within the file (--bytes = from – to)
- zzuf code was used in the first iterations of the font fuzzer, but later versions re-implemented the necessary parts in other languages (Python, Java, JavaScript)

Engine: Role model Zzuf

- Around March 2010
- Fuzzing Engine:
  - Server-based with Apache Tomcat,
  - Used a JSP to proxy calls to zzuf
  - structure awareness prepared via range support
- Browser integration:
  - Browser calls JSP, refresh with updated seed
  - Utilize data URLs to prevent caching effects (Version 1.b)
- Stroustrup rule #1: Learn from prototype, and throw away:
  - slow, hard to reproduce and continue interrupted
  - necessary to know tomcat internals to tweak performance

Fuzzer generation one

- Around June 2010
- Fuzzing Engine:
  - Server-based with python, using BaseHTTPServer
  - structure awareness via range support and ttx integration
- Browser integration:
  - Browser calls python service, refresh with updated seed
  - Utilize data URLs to prevent caching effects
  - Export standalone reproducer
- Stroustrup rule #1: Learn from prototype, and throw away:
  - Big minus, http interaction slows down business

Fuzzer generation two

- Findings with Generation 2 fuzzer:
  - Google Chrome
  - Mozilla Firefox (& SeaMonkey)
  - Opera
  - Microsoft Uniscribe Processor
  - Microsoft Windows Kernel

Fuzzer generation two

- About Google Chrome and fonts:
  - Chrome comes with the Open Type Sanitizer since version 5, so blocks the most forged fonts not to touch the OS level
  - OTS has blind spots such as in the TTF Bytecode sanitization (http://code.google.com/p/ots/wiki/DesignDoc)

Strictly speaking the following bugs are no Chrome bugs,

- Instead places where OTS allowed a malicious font hit a vulnerable system library function

Chrome bugs

- July 2010, Chrome Bug #48283 (CVE-2010-2897)
  - Dumb fuzzing Chrome 5 with **Generation 2** didn‘t result in many bugs on OSX and Linux
  - Next step was with browsers inside a Windows XP/SP3, making sure KB979559 font fix applied
  - After a longer fuzzing run , the machine suddenly **rebooted**,
  - a retry still did, so the bug looked stable
- Google security team investigated this to be a bug in windows ATMFD (Adobe Type Manager) stumbling over broken CFF table offset sizes
  - According to CFF spec only values 1, 2, 3, or 4 are allowed
- To protect Chrome users from this windows bug , OTS in Chrome 6 was hardened to catch b0rken offsets
- Microsoft confirmed to fix at a later point in time (Dec 2010)

Chrome bugs

- August 2010, Chrome Bug #51070 (CVE-2010-3111)
  - After cr#48283 was fixed I went to verify with **Generation 2**
  - Tried some other fonts too on XP/SP3
  - And again, the machine **rebooted**, a either incomplete fix or new bug
- Same game: Google security team investigated this to be a different bug
  - in windows ATMFD, having problems with malicious font hinting code using an oversized stack
- To protect Chrome users from this windows bug OTS was hardened to block those malicious hinting information to harm
- Microsoft confirmed to fix at a later point in time (Dec 2010)

Chrome bugs

- Firefox initially didn‘t come with OTS , so with a broken font it was easy to hit the browser core or the underlying system font lib
- I (and most probably other researchers too) asked Mozilla to address the problem on the root cause instead of the instance level (hint hint OTS)
- As a great leap for firefoxes the December 2010 update (3.6.13) introduced OTS as an additional line of defence

Firefox bugs

September 2010, Mozilla Bug #583520 (CVE-2010-2770)

- **Generation 2** was able to find real-life bugs, so I deferred to throw it away, instead did more fuzzing runs
- This time primarily on OSX
- After a while crashwrangler reported a double-free issue
- Firefox security team confirmed this to be an exploitable bug and refined the patch over multiple iterations

```
Faulty glyph (id:38) outline detected - replacing with a space/null
glyph - in memory font kind
Fri Jul 30 12:20:24 maeckes2.local firefox-bin[10483] <Error>:
CGBitmapContextInfoCreate: unable to allocate 10584 bytes for bitmap dat
objc[10483]: FREED(id): message autorelease sent to freed object=0x1f2de

[..]

---
exception=EXC_BAD_INSTRUCTION:signal=4:is_exploitable=yes:
instruction_disassembly=:instruction_address=0x0000000097db24b4:
access_type=:access_address=0x0000000000000000:
Illegal instruction at 0x0000000097db24b4, probably a exploitable issue.
```


December 2010, Mozilla Bug #583520 (CVE-2010-3768)

- I gave **Generation 2** to Mozilla security team, they found out a series of numerous other bugs
- Additionally I reported the following '**invalid write**' one:

```
exception=EXC_BAD_ACCESS:signal=11:is_exploitable=yes:instruction_disassemb
ly=movl %eax,(%esi):instruction_address=0x000000009011404d: access_type=
write:access_address=0x00000000fedd02b4:
Crash accessing invalid address.
```

- And could not resist the following question:
  - *"Is your future strategy to handle the font bugs case wise, or probably introducing stricter acceptance rules via a (better be sandboxed) font sanitizer ?"*
- Mozilla added protection by integrating OTS against malicious fonts with MSFA-2010-78

Firefox bugs

- In July 2010 Opera was informed of a rebooting crash
  - found with **Generation 2**
  - similar to the first Chrome crash mentioned earlier
  - Opera left the issue unpatched until Dec 2010, and released a text-only bulletin , http://www.opera.com/support/kb/view/980/
  - Until today Opera still does not apply font sanitizing

Opera

- Mid of 2010, CVE-2010-1833
  - found with **Generation 2**
  - *Viewing or downloading a document containing a maliciously crafted embedded font may lead to arbitrary code execution A memory corruption issue exists in Apple Type Services' handling of embedded fonts.*
  - *Viewing or downloading a document containing a maliciously crafted embedded font may lead to arbitrary code execution. This issue is addressed through improved bounds checking.*
  - OTS is not included with Safari

Safari

- September 2010, MS10-063, to fix CVE-2010-2738
  - Several fonts fuzzed with fuzzer **Generation 2** caused the Uniscribe Processor to fail in usp10.dll,
  - For one of those case !exploitable reported to be harmful
  - So the issue was then first reported to Mozilla and
    - Confirmed it was not a problem with Firefox using Uniscribe, rather an inner Uniscribe processor
    - it could be later reproduced with an eot font on IE8, thx to taviso for ttf2eot
- In Sep 2010 (2 months after the report) Microsoft released an update
- There still seem some NP-derefs with the original reproducer, but not regarded as a security issue

Windows Uniscribe Processor (Ffx & IE8)

- December 2010, MS10-091 was released, to fix CVE-2010-3956 and CVE-2010-3957
- It took about half a year for the Chrome workarounds to become obsolete
  - CVE-2010-3956 fixed the OpenType Font Index issue
  - CVE-2010-3957 fixed the OpenType Font Double Free bug

ATMFD (Win Kernel)

- Since Nov 2010, I am hacking on Generation 3
  - Switching to optional Lightweight web server (python CGIHTTPServer)
  - Fuzzing engine is ported to javascript
  - DOM and CSS is updated on the fly
  - Fuzzed content replaced in element, no page reload
  - Ability to dump simplified reproducer case
  - Ability to use different html templates (to test interaction with CSS effects, shadows, text stroke, etc.)
  - Will be released under GPL soon

Fuzzer 3rd generation

- February 2011, APSB 11-02 was released, fixing CVE-2010-0577
  - Fonts in Flash since the early days , DefineFont , DefineFont2 and DefineFont3 tags used flash-specific glyph shape tables
  - Flash 10 introduced the DefineFont4 (id=91) tag , allowing to embed complete Compact font file (CFF) structures
- Used **zzuf** and my flash parser to go for range fuzzing

```
python parseflash.py clays/clay.swf  | grep DefineFont
115549:115555:137887:91:DefineFont4(10):22332:{'fontname':
'Windsong', 'reserved': 0, 'fontFlagsBold': 0, 'fontFlagsItalic':
0, 'fontdata': 'OTTO\x00\n\x00\x80\x00\x03\x00 CFF \x95\xa3B
\xb8\x00\x00\x00\xac\x00\x00N\xf0OS/2 .....
```

Font bugs in Flash

- The browers using OTS are more stable against broken fonts (however still holes with TTF bytecode)
- Dumb fuzzing still smart enough to find bugs in fontlibs
- Not only browser core functionality affected
  - Fonts are in Flash, Shockwave, Java, PDF, etc. too
- IE will fully join browser deathmatch with version 9
- This research hasn't covered mobile devices at all
- Vendors should fix the non-security crashers too, to allow better fuzzing without interruptions

Resumee

- Be prepared for bugs in the next most vulnerable wave of active content-types
- We had graphics formats, we have/had fonts
- Put your hope in the “functionality first” mentality of the web , so my personal estimation is that WebGL and other HTML5 gimmicks hold a large arsenal of exploitable bugs

Outlook

- http://www.adobe.com/content/dam/Adobe/en/devnet/font/pdfs/5176.CFF.pdf The CFF specification
- http://www.typetester.org/ Testing fonts
- http://www.fontmaster.nl/pdf/OT_docs/OT_Development.pdf Information about Font Rendering Details
- http://code.google.com/p/ots/wiki/DesignDoc OTS Design information

Random useful references

Acknowledgements

- Thanks for working with me, fixing the reported bugs:
  - Adobe PSIRT
  - Apple Security Team
  - Behdad Esfahbod (Pango project lead)
  - Chrome Security Team
  - Microsoft Security team
  - Mozilla Security Team
  - My colleagues at Red Hat Security Response Team
  - Opera Security Team

Questions ?

Marc /at/ illegalaccess /dot/ org

CAN SEC WEST