Project Zero

News and updates from the Project Zero team at Google

Monday, June 27, 2016

A year of Windows kernel font fuzzing #1: the results

Posted by Mateusz Jurczyk of Google Project Zero

This post series is about how we used at-scale fuzzing to discover and report a total of 16 vulnerabilities in the handling of TrueType and OpenType fonts in the Windows kernel during the last year. In part #1 here, we present a general overview of the font security area, followed by a high-level explanation of the fuzzing effort we have undertaken, including the overall results and case studies of two bug collisions. In the upcoming part #2, we will share the specific technical details of the project, and how we tried to optimize each part of the process to the maximum extent, and go beyond the current state of the art in Windows kernel font fuzzing. Read on!

Background

To most readers of this blog, the fact that fonts are a very significant attack vector does not have to be reiterated. There are a number of different file formats in active use. These formats are extremely complex, both structurally and semantically. As a result, they are correspondingly difficult to implement correctly, which is further amplified by the fact that a majority of currently used font rasterizers date back to (early) 90's, and were written in native languages such as C or C++. Controlled font files are also deliverable through a variety of remote channels – documents, websites, spool files etc. Last but not least, the two powerful virtual machines executing programs describing glyph outlines in the TrueType and OpenType formats have proven vastly useful for creating reliable exploitation chains, thanks to the ability to perform arbitrary arithmetic, bitwise and other operations on data in memory. For all of these reasons, fonts have been an attractive source of memory corruption bugs.

Font processing vulnerabilities were put to use "in the wild" on many occasions, ranging from a Duqu malware 0-day TTF exploit for the Windows kernel (and a number of other such 0-days patched in emergency fixes), comex' iOS jailbreak via a FreeType Type 1 vulnerability, to successful pwn2own competition entries (Joshua Drake - Java 7 SE - 2013, Keen Team - Windows kernel - 2015). Microsoft alone has released several dozen security bulletins for their font engine during the last decade, and other vendors and projects have not been much better in this regard. Security conferences have been filled with talks discussing font fuzzers, or details of the specific bugs the researchers had found. From the perspective of user security, this is a really disadvantageous situation. If there is a family of software so fragile and yet widely deployed and accessible, that most security professionals can just hop in and easily find a convenient 0-day bug and use it in targeted attacks or mass campaigns, something is clearly wrong.

Addressing the font software security posture problem

As pictured, the situation felt like it required addressing on a more general level, instead of adding one or two more vulnerabilities to the global font track record and somehow feeling safer. Let's face it – the currently used implementations are not going away any time soon, as performance is still a big factor in font rasterization, and the code bases have reached a high level of maturity over the years. One generic approach is to limit the privileges of font processing code in their respective environments, such as enforcing sandboxing of the FreeType library, or moving the font engine out of the kernel in Windows (which Microsoft has done starting with Windows 10). However, that is mostly beyond our reach.

What is within our reach, though, is significantly raising the bar for finding security bugs in the relevant code, thus increasing the cost of such operations and eliminating some of the actors from the equation entirely. Ever since early 2012, we have been using the internal fuzzing infrastructure and available resources to fuzz-test the FreeType project at scale. Until this day, this has resulted in over 50 bug reports, many of which were likely exploitable memory corruption flaws (see lists on the Savannah and Project Zero bug trackers). Some manual code auditing was also involved in the discovery of a few of the issues. We hope that the effort has cleared out most or all easily fuzzable, low-hanging fruit.

In the context of vulnerability hunting, however, FreeType is a relatively easy target – its open-source nature enables very convenient source code manual auditing with full understanding of the underlying logic, makes it possible to employ static analysis algorithms, and allows us to compile in any kind of instrumentation into the final binary, with low runtime overhead (as compared to DBI). For example, we have extensively used the AddressSanitizer, MemorySanitizer and SanitizerCoverage instrumentations, which drastically improved the error detection ratio and provided us with code coverage information, which could be used for coverage-driven fuzzing.

Conversely, the Windows kernel and its font implementation can be considered a harder than average target. The source code is not available, and debugging symbols are only public for parts of the engine (bitmap and

Search This Blog

Search

Labels

- antivirus

Archives

▼ 2016 (10)
► July (1)
▼ June (3)
How to Compromise the Enterprise Endpoint
A year of Windows kernel font fuzzing #1: the resu...
Exploiting Recursion in the Linux Kernel
► March (3)
► February (2)
► January (1)
► 2015 (33)
► 2014 (11)

TTF handling in win32k.sys, but not Type 1 and OTF handling in ATMFD.DLL). This already makes any manual work more demanding, as it must involve reverse engineering, which may prove especially difficult for parts of the code operating on font data in an indirect manner (as opposed to code which maps 1:1 to parts of the specification, such as the glyph outline virtual machine). Furthermore, the code executes in the kernel, in one module shared with the rest of the graphical subsystem, which makes any kind of interaction (e.g. instrumentation) non-trivial, to say the least. There are of course mechanisms to enhance bug discovery (such as *Special Pools*), but there are also obstacles standing in the way, like generic exception handling potentially masking some error indicators.

In early 2015, we started off by manually taking apart the Type 1 / CFF virtual machine in ATMFD, which turned out to be the perfect auditing target. Fully self-contained, sufficiently complex but reasonably limited in size, full of legacy code and seemingly without a proper review in the past – a mixture which cannot be underestimated. The audit resulted in 8 vulnerabilities reported to Microsoft in the Windows kernel, some of them extremely critical. For a detailed write up on that research and the most interesting BLEND vulnerability, check out the "One font vulnerability to rule them all" blog post series, starting with part #1.

The CharString implementation could indeed be efficiently audited as a whole, but the same strategy couldn't be applied to the entire win32k.sys and ATMFD.DLL font-related code base. The volume of code and different program states makes it hardly possible to comprehend them all, not to mention keeping the big picture in mind and thinking of all the potential misbehaviors and corner cases. The other option was of course fuzzing – a method which doesn't provide as much confidence in the results and code / program state coverage, but scales really well, only needs time for the initial set up, and has proven highly effective in the past. In fact, our guesstimate – based on the public track record – is that historically, more than 90% font bugs (similarly to the entirety of security issues) have been found with fuzzing. This has the additional advantage that reporting fuzzed-out bugs is in line with the goal of *raising the bar for other bughunters*, as they use similar techniques, and wouldn't be able to find the cleared out low hanging fruit anymore.

With this in mind, we started a Windows kernel font fuzzing effort in May 2015, in an attempt to take what was previously known on the subject, and push each part of the process a bit forward, optimizing them or trying to achieve maximum effectiveness. After roughly a year's time and many bulletins released since then, the kernel is now fuzz-clean against the techniques we used, and as we believe that both the results and the methods may be interesting for a wider audience, this post series is to wrap up and summarize the initiative.

Results

Without further ado, below is a list of all vulnerabilities found by fuzzing the Windows kernel in the last year:

Tracker ID	Type	Format (driver)	SFNT table(s)	Reported	Fixed
368	Pool buffer overflow	TTF (win32k.sys)	glyf	21 May 2015	11 August 2015
369	Pool buffer overflow	OTF (atmfd.dll)	GPOS	21 May 2015	20 July 2015
370	Pool buffer overflow	TTF (win32k.sys)	hmtx, maxp	21 May 2015	11 August 2015
382	Pool out-of-bound reads	OTF (atmfd.dll)	CFF	21 May 2015	11 August 2015
383	Use-after-free	OTF (atmfd.dll)	CFF	21 May 2015	11 August 2015
384	Use-after-free	OTF (atmfd.dll)	CFF	21 May 2015	11 August 2015
385	Use of uninitialized memory	OTF (atmfd.dll)	CFF	21 May 2015	11 August 2015
386	Pool out-of-bound reads	OTF (atmfd.dll)	CFF	21 May 2015	11 August 2015
392	Pool out-of-bound reads	OTF (atmfd.dll)	CFF	21 May 2015	11 August 2015
401	Pool buffer overflow	TTF (win32k.sys)	glyf	21 May 2015	11 August 2015
402	Pool buffer overflow	TTF (win32k.sys)	fpgm, hmtx, maxp	21 May 2015	11 August 2015
506	Pool buffer overflow	TTF (win32k.sys)	OS/2	18 August 2015	10 November 2015
507	Pool buffer overflow	TTF (win32k.sys)	glyf	18 August 2015	10 November 2015
682	Stack buffer overflow	OTF (atmfd.dll)	CFF	22 December 2015	8 March 2016
683	Pool buffer overflow	OTF (atmfd.dll)	CFF	22 December 2015	8 March 2016
684	Pool buffer overflow	TTF (win32k.sys)	EBLC, EBSC	22 December 2015 (25 January 2016)	12 April 2016

The linked bug entries include brief descriptions of the crashes, example crash logs from Windows 7 x86 with Special Pools enabled, and obligatory proof of concept files. In order to reproduce some of the crashes, it might also be necessary to use a dedicated font loading program, which was provided privately to Microsoft (but is also discussed in detail in the next post).

As shown in the table, the crashes were reported in three iterations: the first one obviously contained the bulk of the issues, as the fuzzer was hitting a lot of different states and code paths right from the start. The second and third iterations were run for a longer time, in order to shake out any crashes which might have been masked by other, more frequently hitting bugchecks. The time periods between each run (3-4 months) are the times Microsoft took to release patches for the reported bugs, and are associated with the Project Zero 90-day disclosure deadline (which Microsoft met in all cases). One case (#684) had an updated report time, as we had to figure out with Microsoft the system settings necessary to reproduce the system crash.

The flaws represented a variety of programming errors in the handling of both TTF and OTF files, in the code responsible for processing varied SFNT tables. The vast majority of the issues could be used for local privilege escalation (sandbox escape etc.) and even remote code execution (for applications which pass user-controlled files directly to GDI), which is consistent with Microsoft's "Critical" assessment of these bugs. Even though the fuzzing was run on Windows 7, nearly all of the reported bugs were still present in newer versions of the system. It's also worth noting that while the elevation of privileges scenario is mitigated in Windows 10 by the architectural shift to performing font rasterization in a user-mode process with restricted privileges, an RCE in the context of that process is still a viable option (although much more limited than directly compromising the ring-0 security context).

Collisions

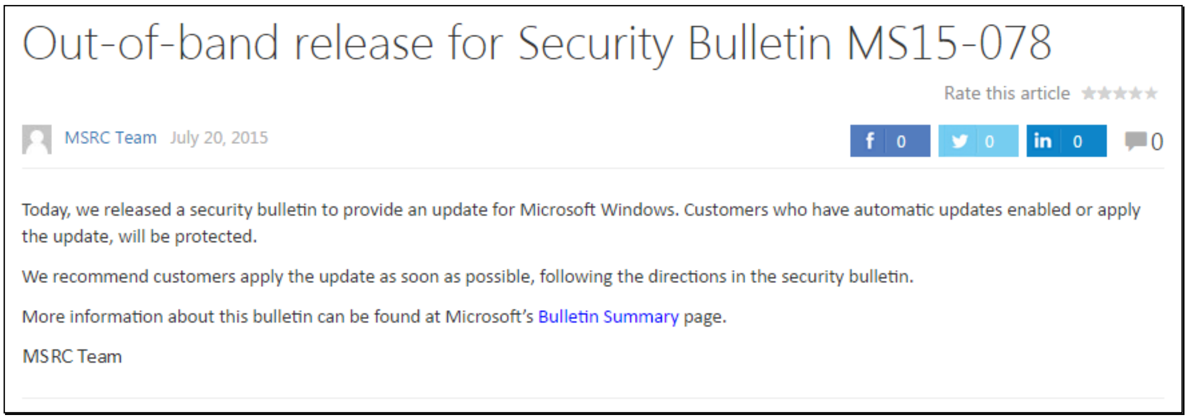
There is no better validation of the value of any particular defensive bug hunting effort, than observing your reported bugs colliding with weaponized exploits used in the wild (bonus points for 0-days utilized to harm users). While there was an extensive track record of such bugs and their usage in the past, the question of whether any new discoveries would collide with exploits still circulating in 2015 would remain to be answered.

We didn't have to wait long. As it turned out, exactly the two first bugs that we filed in the tracker (issues #368 and #369) very quickly proved to be the same as the TTF vulnerability used by the Keen Team for privilege escalation during pwn2own 2015, and an OTF vulnerability found in the Hacking Team data leak in July 2015 (with a fully weaponized exploit), subsequently fixed by Microsoft in an emergency bulletin.

Interestingly, the two colliding bugs were in fact the most trivial ones to find. Our fuzzer was constantly hitting the corresponding crashes during the first iteration, and upon some brief analysis we determined that both conditions could indeed be triggered by performing trivial changes (single bit flips, single byte swaps) in many legitimate fonts. This appears to align well with our quest to raise the bar by clearing out the low hanging fruit, as it confirms that exactly such bugs are typically discovered and exploited by other researchers.

The HackingTeam 0-day bug collision

After reporting the first batch of crashes to Microsoft in May 2015 (7 OpenType and 4 TrueType bugs), we patiently waited for the vendor to release corresponding patches. Very soon, we were informed that they managed to reproduce all of the problems, and scheduled the fixes for the August Patch Tuesday. However on July 20, in the middle of my vacation, I noticed that an out-of-band MS15-078 security bulletin was released that day:


Out-of-band release for Security Bulletin MS15-078

Rate this article

MSRC Team July 20, 2015

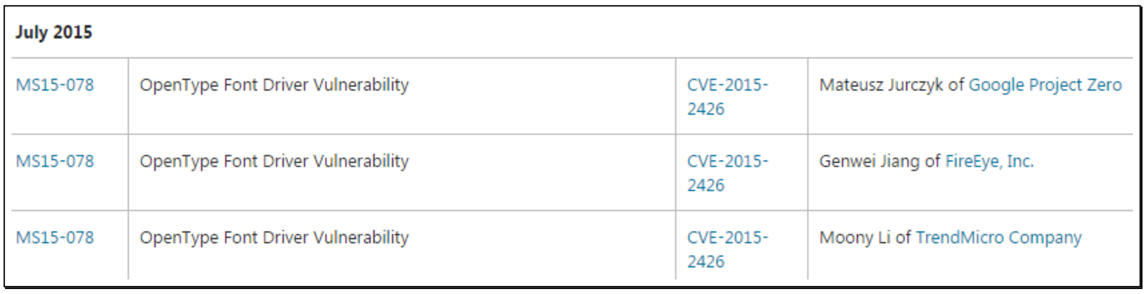
Today, we released a security bulletin to provide an update for Microsoft Windows. Customers who have automatic updates enabled or apply the update, will be protected.

We recommend customers apply the update as soon as possible, following the directions in the security bulletin.

More information about this bulletin can be found at Microsoft's Bulletin Summary page.

MSRC Team

Even more interestingly, I was one of the three people credited in the Acknowledgements section:



July 2015			
MS15-078	OpenType Font Driver Vulnerability	CVE-2015-2426	Mateusz Jurczyk of Google Project Zero
MS15-078	OpenType Font Driver Vulnerability	CVE-2015-2426	Genwei Jiang of FireEye, Inc.
MS15-078	OpenType Font Driver Vulnerability	CVE-2015-2426	Moony Li of TrendMicro Company

As it quickly turned out, there was a collision of one of our reported crashes (issue #369) with a weaponized exploit discovered by two independent researchers in the Hacking Team leak. As a quick reminder, on July 5 2015, a link to a .torrent file hosting gigabytes of private Hacking Team company data was announced on their (hacked) Twitter account. Most of the security industry rushed to investigate the suddenly available resources, finding not only the source code of surveillance products and controversial information on business operations, but also multiple exploits for 0-day vulnerabilities in software such as Flash Player (4 exploits), Windows kernel (2 exploits), Internet Explorer, SELinux, and so on. Until July 20, however, there was no public knowledge of the existence of CVE-2015-2426, indicating that the bug was reported to Microsoft privately.

The exploit took advantage of the vulnerability to elevate the code's privileges in the system on Windows platforms up to 8.1. A detailed analysis of the root cause of the bug and exploitation techniques can be found in the 360 Vulcan Team blog post (in Chinese), but in essence, it was caused by an invalid assumption made in the ATMFD.DLL OpenType driver, as illustrated in the pseudo-code below:

```
LPVOID lpBuffer = EngAllocMem(8 + GPOS.Class1Count * 0x20);
if (lpBuffer != NULL) {
 // Copy the first element.
 memcpy(lpBuffer + 8, ..., 0x20);

 // Copy the remaining Class1Count - 1 elements.
}
```

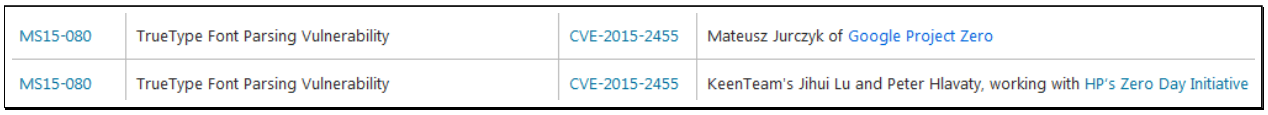
Here, the code assumed that `Class1Count` (a 16-bit field in the GPOS table) would always be non-zero, and copied the first table item regardless of the actual value. As a result, if the field was equal to 0x0000, the dynamically allocated buffer was overflown by 32 (0x20) bytes. With proper massaging of the kernel pools, it was possible to place a win32k.sys `CHwndTargetProp` object directly after the overwritten memory region, and have its vtable corrupted with a user-mode address. From there, it was only a matter of leaking the base address of the win32k.sys module, and constructing a ROP chain to disable SMEP and execute the privilege escalation shellcode.

The important point is that in order to trigger the bug through fuzzing, it was sufficient to set just two subsequent bytes at a specific location in the file (corresponding to the `Class1Count` field) to 0. In other words, it was trivial to discover with a dumb fuzzer, and it's surprising that such a bug could even survive until 2015, with so much work being supposedly put into the security of font processing.

For the curious, the original HT exploit was later ported to Windows 8.1 64-bit by Cedric Halbronn of NCC Group (see this article).

The pwn2own bug collision

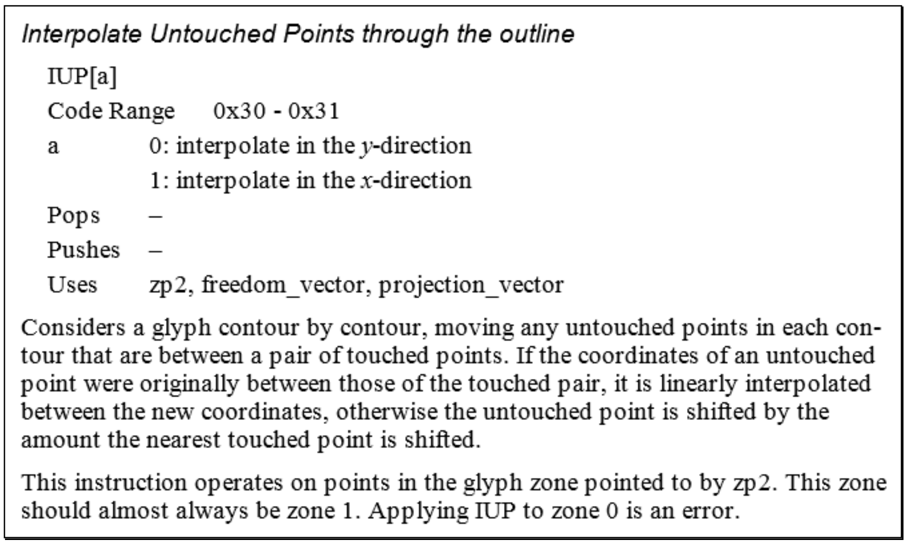
Three weeks later, on August 11 2015, patches for the rest of the crashes from the first batch were released as planned. Again though, I was surprised with the Acknowledgements section, which also gave credit for reporting one of the vulnerabilities to other researchers:



MS15-080	TrueType Font Parsing Vulnerability	CVE-2015-2455	Mateusz Jurczyk of Google Project Zero
MS15-080	TrueType Font Parsing Vulnerability	CVE-2015-2455	KeenTeam's Jihui Lu and Peter Hlavaty, working with HP's Zero Day Initiative

This time, Keen Team! If you look up CVE-2015-2455, one of the results will be a ZDI-15-388 advisory, referring to the bug as "(Pwn2Own)" in the title, and mentioning that it was related to the "IUP" TrueType instruction. Yes, this is exactly issue #368. The vulnerability was indeed leveraged to achieve a sandbox escape and a full system compromise in one of the "Adobe Reader" or "Adobe Flash Player" categories (it's not clear, as two distinct TTF bugs were used during the competition), during the pwn2own contest which took place on March 19-20, 2015. For the curious, the exploitation of the other Team Keen's TTF flaw was explained during the "This Time Font hunt you down in 4 bytes" talk at REcon 2015. Notably, Microsoft took almost 5 months to fix the bugs since they were reported through ZDI, but still fit in the 90-day Project Zero disclosure deadline (as the start date was May 21).

As for the technical details, the bug existed in the implementation of the "IUP" instruction, which translates to *Interpolate Untouched Points through the outline* in the TrueType instruction set specification:



Interpolate Untouched Points through the outline

IUP[a]	
Code Range	0x30 - 0x31
a	0: interpolate in the *y*-direction
	1: interpolate in the *x*-direction
Pops	–
Pushes	–
Uses	zp2, freedom_vector, projection_vector

Considers a glyph contour by contour, moving any untouched points in each contour that are between a pair of touched points. If the coordinates of an untouched point were originally between those of the touched pair, it is linearly interpolated between the new coordinates, otherwise the untouched point is shifted by the amount the nearest touched point is shifted.

This instruction operates on points in the glyph zone pointed to by zp2. This zone should almost always be zone 1. Applying IUP to zone 0 is an error.

If you look closely, there is an important note at the end of the description: ***Applying IUP to zone 0 is an error***. Yes, you guessed it, that was the bug. The instruction handler at `win32k!itrp_IUP` was missing verification that the current zone was not zone 0, and thus operated on zone 0 as if it was zone 1, ultimately leading to a pool-based buffer overflow with largely controlled contents and length. The three TrueType instructions below were sufficient to trigger a crash:

```
PUSH[ ]  /* 1 value pushed */
0
SZP2[ ]  /* SetZonePointer2 */
IUP[0]   /* InterpolateUntPts */
```

More importantly, the crash was also very easy to come by when mutating legitimate fonts – a single bit flip was enough to change the `SZP2` / `SZPS` instruction argument from 1 to 0. Accordingly, this was the top 1 hitting crash in the first run of my dumb fuzzing. When I later added a TrueType program generator to the mix, the issue also manifested when loading every second test case, which forced us to account for this in the generator and avoid the specific construct until the bug was fixed.

On an unrelated note, the case is a great example of how just reading the official specification carefully may hand you a critical vulnerability with very little effort and no auditing or fuzzing involved whatsoever. :)

Closing thoughts

If nothing else, the effort and its results are evidence that fuzzing, if done correctly, is still a very effective approach to vulnerability hunting, even with theoretically "mature" and heavily tested code bases. Furthermore, the two bug collisions prove that Windows kernel font bugs are still alive and kicking, or at least were actively used in the wild in 2015. In the second post of the series, we will discuss the *meaty* parts of the research: how we prepared the input corpus, mutated and generated interesting font samples, fuzzed the Windows kernel at scale, reproduced the crashes and minimized them.

In the meanwhile, just last week (following last week's Microsoft Patch Tuesday) we opened up issue #785 in the Project Zero tracker, which discusses the details of an ATMFD.DLL pool corruption vulnerability in the "NamedEscape" attack surface, the same that Hacking Team's second Windows kernel 0-day (CVE-2015-2387) was located at. Happy reading!

Posted by Ben at 9:19 AM   G+1 +42 Recommend this on Google

2 comments:

GrimTim June 27, 2016 at 3:11 PM

Where can I find known bad font files for testing detection by AV products and other investigations including testing the quick and dirty method described here: http://security.stackexchange.com/questions/41652/how-do-i-know-if-a-font-is-malicious

Reply

Yuhong Bao June 27, 2016 at 3:15 PM

What is fun is the MS15-077/MS15-078 fiasco where one ATMFD patch was released for Server 2003 and another about a week later was not.

Reply

Enter your comment...

Comment as: Google Account

Publish Preview

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)