Inside Cloudbleed
John Graham-Cumming
A little about Cloudflare
What was Cloudbleed?
“Our edge servers were running past the end of a buffer and returning memory that contained private information such as HTTP cookies, authentication tokens, HTTP POST bodies, and other sensitive data.

And some of that data had been cached by search engines.”
Don’t leak random chunks of memory onto the Internet.
Don’t have those chunks of memory cached by search engines.
The Night Of
Could someone from cloudflare security urgently contact me.

12:11 AM - 18 Feb 2017

259 Retweets 920 Likes
Received details of bug from Tavis Ormandy
Cross functional team assembled in Room 403 in San Francisco
Email Obfuscation feature disabled globally

Project Zero confirm they no longer see leaking data
London team awake and online
San Francisco actively fuzzing parsing code to look for additional problems
Manual inspection for additional issues
Automatic HTTPS Rewrites disabled worldwide
Discovered we didn’t have a kill switch for Server-Side Excludes

At was an ancient feature that predated all the engineers on the team

Implemented a global kill switch and redeployed Cloudflare NGINX Lua code globally
The Technical Cause
Had to have one of
- Email Obfuscation
- Automatic HTTPS Rewrites
- Server-Side Excludes

Page had to end with something like
- `<script type="`
- `<IMG HEIGHT="50px" WIDTH="200px" SRC="`
<script type="" script_consume_attr := ((unquoted_attr_char)* :>> (space|'/'|'>'')) ) >{ ddctx("script
consume_attr"); } } @{ fhold; fgoto script_tag_parse; } $lerr{ dd("script consume_attr failed"); fgoto script_consume_attr; };
<script type="" />

script_consume_attr := ((unquoted_attr_char)* :>> (space|'|'>'))
}>{ ddctx("script consume_attr"); } }

 @{$ fhold; fgoto script_tag_parse; }

$lerr{ dd("script consume_attr failed");
    fgoto script_consume_attr; };

<script type=""
<script type="">

script_consume_attr := ((unquoted_attr_char)* :>> (space|'/'|'>')) ) >{ ddctx("script consume_attr"); } 

@{ fhold; fgoto script_tag_parse; } 

$lerr{ dd("script consume_attr failed"); fgoto script_consume_attr; };
<script type="">

script_consume_attr := ((unquoted_attr_char)*
:>> (space|'/'|'>'')) >{ ddctx("script
consume_attr"); }  

}@{ fhold; fgoto script_tag_parse; }

$lerr{ dd("script consume_attr failed"); 
      fgoto script_consume_attr; };

/* generated code */
if ( ++p == pe )
    goto _test_eof;
ngx_int_t
ngx_http_email_parse_email(ngx_http_request_t *r, ngx_http_email_ctx_t *ctx) {
    u_char *p = ctx->pos;
    u_char *pe = ctx->buf->last;
    u_char *eof = ctx->buf->last_buf ? pe : NULL;
}

Irony: we caused this because we were migrating away from the buggy parser
(gdb) p *in->buf
$6 = {
pos = 0x558a238e94f7 "<script type="",
  last = 0x558a238e9504 "",

  [...]

  last_buf = 0,

  [...]
}
Old and new parsers present

(gdb) p *in->buf
$8 = {
    pos = 0x558a2f58be30 "<script type="",
    last = 0x558a2f58be3e "",

    [...]

    last_buf = 1,

    [...]
}
Past the end of the buffer

/* #line 877 "ngx_http_email_filter_parser.rl" */
{ dd("script consume_attr failed");
  {goto st1266;} }
  goto st0;
 [...]

st1266:
  if ( ++p == pe )
    goto _test_eof1266;
Good, Bad, Ugly
Good: stemmed the leak in 47 minutes

NOT SURE IF SINCERE YAY

OR SARCASTIC YAY?
Bad: we had been leaking sensitive stuff

- Private key used to secure internal connectivity between our machines
- Some internal authentication secrets
- But also...
  - HTTP headers for requests to our customers’ websites (including cookies)
  - POST data (passwords, credit card numbers, SSNs could have been present)
  - URI parameters
  - JSON blobs for API calls
  - API authentication secrets, OAuth keys,
Worked with Google, Bing, Yahoo, Yandex, Baidu, DuckDuckGo, and many others to remove cached data

Found a total of 770 unique URIs across 161 unique domains with sensitive, cached data
Going Public
Programme of events

Arrival and drinks
16:30

Dinner
18:45

Guest Talk
John Graham-Cumming,
Cloudflare
19:00

Carriages
20:30

CONTACT
Julia Zanhieri 020 7590 259 633

South Kensington Club
36-42 Harrington Rd
Kensington
London SW7 3ND

John Graham-Cumming

Head of Engineering, Cloudflare

John Graham-Cumming is a computer programmer and author. He studied mathematics and computer science at Oxford and stayed for a year to study computer security. As a programmer he has worked in Silicon Valley and New York, the UK, Germany and France and currently works at Cloudflare. His open-source SIPHash program won a State Productivity Award in 2010.

Incident report on memory leak caused by Cloudflare parser bug

23 Feb 2017 by John Graham-Cumming.

Last Friday, Tavis Ormandy from Google’s Project Zero contacted Cloudflare to report a security problem with our edge servers. He was seeing corrupted web pages being returned by some HTTP requests run through Cloudflare.

It turned out that in some unusual circumstances, which I’ll detail below, our edge servers were running past the end of a buffer and returning memory that contained private information such as HTTP cookies, authentication tokens, HTTP POST bodies, and other sensitive data. And some of that data had been cached by search engines.

For the avoidance of doubt, Cloudflare customer SSL private keys were not leaked. Cloudflare has always terminated SSL connections through an isolated instance of NGINX that was not affected by this bug.

We quickly identified the problem and turned off three minor Cloudflare features (email obfuscation, Server-side Excludes and Automatic HTTPS Rewrites) that were all using the same HTML parser chain that was causing the leakage. At that point it was no longer possible for memory to be returned in an HTTP response.
Software Bug at Internet-Service Provider Sparks Privacy Concerns

Bug at Cloudflare caused some of its servers to leak information that should have remained private.
Impact Statistics
Estimated Instances of Data Leakage

- September 22, 2017 -> February 13, 2017 605,307
- February 13, 2017 -> February 18, 2017 637,034
Based on data cached by search engines

- Each leak contained

  67.54 Internal Cloudflare Headers
  0.44 Cookies
  0.04 Authorization Headers / Tokens

No Passwords, credit cards, SSNs
Impact by customer size

<table>
<thead>
<tr>
<th>Requests per Month</th>
<th>Anticipated Leaks</th>
</tr>
</thead>
<tbody>
<tr>
<td>200B – 300B</td>
<td>22,356 – 33,534</td>
</tr>
<tr>
<td>100B – 200B</td>
<td>11,427 – 22,356</td>
</tr>
<tr>
<td>50B – 100B</td>
<td>5,962 – 11,427</td>
</tr>
<tr>
<td>10B – 50B</td>
<td>1,118 – 5,926</td>
</tr>
<tr>
<td>1B – 10B</td>
<td>112 – 1,118</td>
</tr>
<tr>
<td>500M – 1B</td>
<td>56 – 112</td>
</tr>
<tr>
<td>250M – 500M</td>
<td>25 – 56</td>
</tr>
<tr>
<td>100M – 250M</td>
<td>11 – 25</td>
</tr>
<tr>
<td>50M – 100M</td>
<td>6 – 11</td>
</tr>
<tr>
<td>10M – 50M</td>
<td>1 – 6</td>
</tr>
<tr>
<td>&lt;10M</td>
<td>&lt; 1</td>
</tr>
</tbody>
</table>
The Really Ugly Truth
This had been going on for months

- September 22, 2016
  Automatic HTTP Rewrites enabled (uses new parser)

- January 30, 2017
  Server-Side Excludes migrated to new parser

- February 13, 2017
  Email Obfuscation partially migrated to new parser

- February 18, 2017 and leak is stopped
  Google reports problem to Cloudflare
Roughly 180 sites had flaw plus one of these two:
- September 22, 2016: Automatic HTTP Rewrites enabled (uses new parser)
- January 30, 2017: Server-Side Excludes migrated to new parser

Around 6,500 sites had flaw plus:
- February 13, 2017: Email Obfuscation partially migrated to new parser
- February 18, 2017: Google reports problem to Cloudflare and leak is stopped

This had been going on for months
This had been going on for months
Core War
Zero Tolerance for crashes on metals

John Graham-Cumming <jgc@cloudflare.com>

Mar 3

Guys,

Based on the problems of the last two weeks and the fact that we had quite a lot of information about crashes in production that could have indicated what was happening I think we need to get back to having a zero tolerance policy towards crashes of software on metals.

From today I'd like the following policy to be implemented: every crash of software on a metal machine is to be logged as a Jira and cc:ed to myself and (redacted). An engineer must be assigned by the relevant manager immediately and finding and fixing the problem becomes job 1.

Any new software put into production that starts crashing will be immediately rolled back.
This is not the same as the luajit heap exhaustion aborts.

But it is very strange. The segfault is due to a bad value (0x809c95d0) in rdx (which holds the third parameter to lj_cconv_ct_ct). But I can't see how the instructions in lj_cconv_ct_tv leading up to the call to lj_cconv_ct_ct can ever produce that value (they correctly compute &cts[9]). It's not that the interpreter state is corrupted. The bad value just seems to come out of nowhere.

Shrug.

Cosmic rays 😞 ?

Sounds like a cop-out, I know. But given the context, after lj_cconv_ct_tv there shouldn't be any writes to memory other than the stack, so it's possible to reconstruct the instruction flow and register values leading up to the segfault. Except that the %rdx value is wrong.
“The bad value just seems to come out of nowhere”
Mysteries

- Instructions & memory state could not explain the register state
- Instructions & register state could not explain the memory state
- RIP does not point to a valid instruction
- Instruction & register state did not match the signal information
On average, ~1 mystery core dump a day

Scattered over all servers, all colos

Per server, 1 in 10 years
  - Can’t reproduce
  - Hard to try any potential fix
Core dump corruption?

```
program:kernel message:nginx-fl[40163]: segfault at 48 ip 0000564f0786dddf1 sp 00007ffe8b571138 error 6 in nginx-fl[564f075cf000+e19000] service:kernel @timestamp:2017-06-16T06:43:47.000Z @version:1 host:71m158 input:syslog-journald _id:AVyvpL0mpNLF2t2aG106_type:logs _index:services-2017.06.16 _score:
```
Bugs in the kernel code that controls virtual memory can lead to mysterious effects, but:

- No bugs reported upstream
- Couldn’t find any bugs in the relevant code
- No correlation between the core dumps and relevant metrics
During further investigation, suggested to check if the generations of hardware could be relevant. Surprisingly all 18 nginx-cache SIGSEGV happened on broadwell metals. Given broadwell metals (1325) are only less than 1/3 of our entire fleet, we suspect these core dumps might be related to hardware.

The next step is to check if there is anything else that are shared among these metal different memory layouts, we want to check if all the core dumps are under the same

CC: John Graham-Cumming

Dates
Created: 23/Jul/17 1:39 AM
Updated: 11/Jul/17 11:24 PM
Resolved: 23/Jul/17 6:54 PM
Intel® Xeon® Processor E5-2600 v4 Product Family

Specification Update

December 2016
BDF76  An Intel® Hyper-Threading Technology Enabled Processor May Exhibit Internal Parity Errors or Unpredictable System Behavior

Problem: Under a complex series of microarchitectural events while running Intel Hyper-Threading Technology, a correctable internal parity error or unpredictable system behavior may occur.

Implication: A correctable error (IA32_MC0_STATUS.MCACOD=0005H and IA32_MC0_STATUS.MSCOD=0001H) may be logged. The unpredictable system behavior frequently leads to faults (e.g. #UD, #PF, #GP).

Workaround: It is possible for the BIOS to contain a workaround for this erratum.

Status: For the Steppings affected, see the Summary Tables of Changes.
<table>
<thead>
<tr>
<th>Microcode Update</th>
<th>Customer Release Date</th>
<th>Intended Stepping</th>
<th>Revision ID</th>
<th>Workaround for Errata</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEF406F1_0B000005</td>
<td>7/13/2015</td>
<td>B0/M0/R0</td>
<td>0B000005</td>
<td>BDF56, BDF64</td>
</tr>
<tr>
<td>MEF406F1_0B000006</td>
<td>7/17/2015</td>
<td>B0/M0/R0</td>
<td>0B000006</td>
<td>BDF57, BDF51, BDF7, BDF54</td>
</tr>
<tr>
<td>MEF406F1_0B000007</td>
<td>8/7/2015</td>
<td>B0/M0/R0</td>
<td>0B000007</td>
<td>BDF44</td>
</tr>
<tr>
<td>MEF406F1_0B000008</td>
<td>9/15/2015</td>
<td>B0/M0/R0</td>
<td>0B00000B</td>
<td>BDF63</td>
</tr>
<tr>
<td>MEF406F1_0B000014</td>
<td>3/9/2016</td>
<td>B0/M0/R0</td>
<td>0B000014</td>
<td>BDF63, BDF68</td>
</tr>
<tr>
<td>MEF406F1_0B000017</td>
<td>3/09/2016</td>
<td>B0/M0/R0</td>
<td>0B000017</td>
<td>BDF67, BDF69, BDF70, BDF75, BDF77</td>
</tr>
<tr>
<td>MEF406F1_0B00001A</td>
<td>5/16/2016</td>
<td>B0/M0/R0</td>
<td>0B00001A</td>
<td>BDF76</td>
</tr>
<tr>
<td>MEF406F1_0B00001B</td>
<td>6/17/2016</td>
<td>B0/M0/R0</td>
<td>0B00001B</td>
<td>BDF80</td>
</tr>
<tr>
<td>MEF406F1_0B00001D</td>
<td>8/05/2016</td>
<td>B0/M0/R0</td>
<td>0B00001D</td>
<td>BDF78, BDF79</td>
</tr>
<tr>
<td>MEF406F1_0B00001E</td>
<td>9/02/2016</td>
<td>B0/M0/R0</td>
<td>0B00001E</td>
<td>BDF83</td>
</tr>
<tr>
<td>MEF406F1_0B00001F</td>
<td>10/14/2016</td>
<td>B0/M0/R0</td>
<td>0B00001F</td>
<td>BDF81</td>
</tr>
</tbody>
</table>
Rolled out new BIOS fleet wide on July 29, 2017
Thoughts

- Be as transparent as possible
- Co-operate with security researchers
- Start taking notes immediately
- Over communicate
- Go back and see if hindsight was 20/20
Thank you