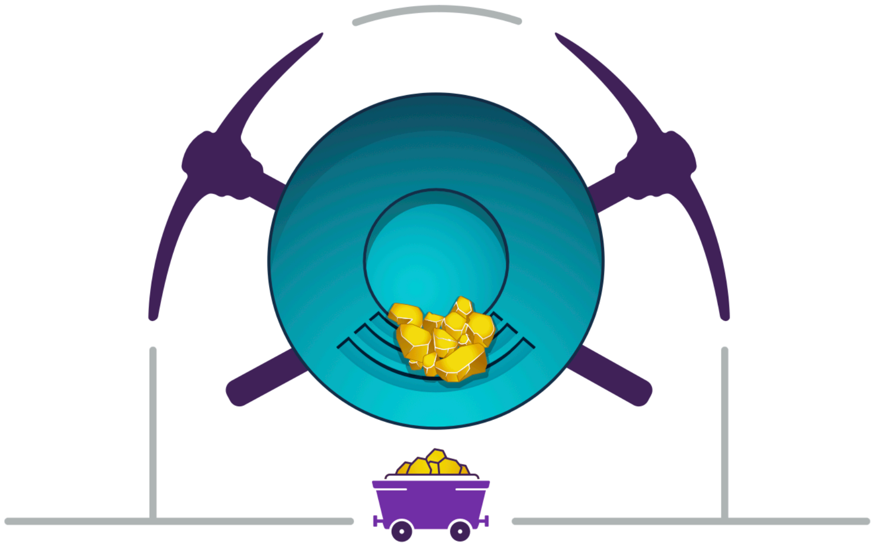


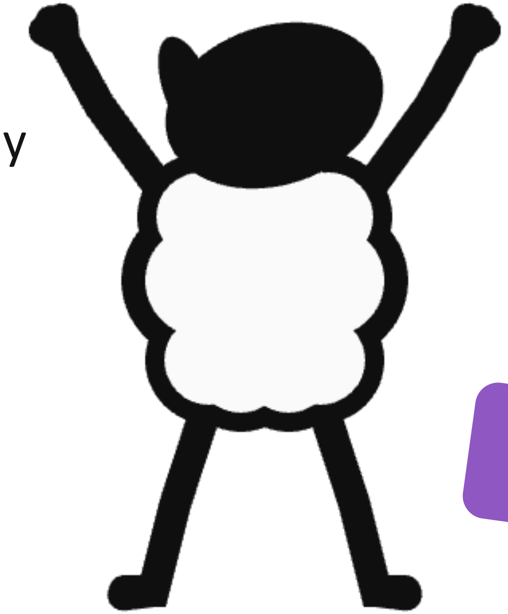
JSLUICE

There’s Gold In Them Thar Files!

Presented by: **Tom Hudson, Senior Security Engineer, Bishop Fox** | Date: 2023-06-24

Hello, BSides :)

- ✦ I'm Tom(NomNom)
- ✦ It's been a while! Hello! 👋
- ✦ I make open-source tools (gron, anew, meg, fff, unfurl, gf, waybackurls, httprobe, assetfinder, qsrepla…
- ✦ I like questions, so have 'em ready!
- ✦ I do security tooling R&D stuff at Bishop Fox
  - That means this slide-deck is branded and in light-mode
  - …and also lacks legally-questionable use of watermarked stock photography



The return of light-mode sheepy (:

Crawling Used To Be Easy

- The *Old Web* was pretty easy to crawl
- Links were links, marquees scrolled, and HTML was unsullied by JavaScript
- When JavaScript arrived it mostly made a trail of kitten gifs follow your cursor

```
<a href=/guestbook.html>Sign my guestbook!</a>
```

A guestbook is like a comments section, but for your whole site

2001: A Cyberspace Odyssey

- In about 2001 JavaScript got a new superpower: `XMLHttpRequest`
  - At the time you might have known it as: `ActiveXObject("Microsoft.XMLHTTP")`
- Now JavaScript could fetch new data and stuff it into the page without a page reload
- Fast-forward a couple of decades and we have *ReangularJSQuery*

Honestly, felt kind of magical to not hear the reload "click" every time a page changed

Get $10 off any purchase of $30 or more this holiday season at wine.com.

Slide show: Hubble telescope images

Scenic views from space

Also today
- Top 10 tax deductions
- What manly men read
- Save: off-season travel
- Take Barbie on a trip

Make the connection
- Top wireless phones
- Anxiety at social events?
- Send a holiday e-card

The shy man's dating guide

Hungry for freebies?

MSNBC News

Yellow Pages

Dealing With The New Web

- One way to deal with JavaScript is to use a (headless) browser – a sort of *dynamic analysis*
  - It’s kinda slow and resource intensive
  - You only find out about things that are actually executed
- To do *static analysis* you could use regular expressions
  - Something something, then you have two problems…

```
fetch('/api/v2/guestbook', {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({msg: "..."})
})
```

'fetch' is a modern alternative to XMLHttpRequest

Irregularly Regular

- Using regular expressions *seems* simple enough
- You have to deal with nested and escaped quotes, differing whitespace, *random* variance etc
  - **At scale, edge-cases become commonplace**
- Running several-dozen complex regular expressions across multi-megabyte-files isn't great
  - Maintaining several-dozen complex regular expressions is worse :(

```
'/api/v2/guestbook' => /fetch\('([^']+)'/
"/api/v2/guestbook" => /fetch\(['"]([^'"]+)['"]/
"/api/user/o'neill" => /fetch\((['"])([^\1]+)\1/
```

I stole this one from somewhere, but it's a real regex for finding URLs in JavaScript!

```
(?:"|'|\s)(((https?://[A-Za-z0-9_\-\.]+(:\d{1,5})?)+([\.]{1,2})?/[A-Za-z0-9/\-_\.\\%]+([\?|#][^"']+)?)|((\.{1,2}/)?[a-
zA-Z0-9\-_/\\%]+\.(aspx?|js(on|p)?|html|php5?|html|action|do)([\?|#][^"']+)?)|((\.{0,2}/)[a-zA-Z0-9\-_/\\%]+(/|\\)[a-
zA-Z0-9\-_]{3,}([\?|#][^"|']+)?)|((\.{0,2})[a-zA-Z0-9\-_/\\%]{3,}/))(?:"|'|\s)
```

Context could be another name for an SMS scam 🤔

- Extracting URLs and paths by themselves is nice
- Extracting the context around them is nicer
- We can do that with the power of **Tree-sitter** (https://tree-sitter.github.io/tree-sitter/)
  - Shout-out to **@LewisArdern** and **@Semgrep** for inspiration :)

```
fetch('/api/v2/guestbook', {
  method: "POST",
  headers: {
    "Content-Type": "application/json"
  },
  body: JSON.stringify({msg: "..."})
})
```

Sitting In A Tree: P, A, R, S, I, N, G

- Raw JavaScript source code is difficult to understand for humans, doubly so for programs
- Tree-sitter parses JavaScript (and dozens of other languages) into *syntax trees*
  - It's meant for tasks like syntax highlighting so it's tolerant of minor errors <3
- **`jsluice`** can show you the syntax tree for any JavaScript file

We're 8 slides in and **jsluice** has finally showed up (:

```
$ cat hello.js
console.log("Hello, world!")

$ jsluice tree hello.js
hello.js:
program
  expression_statement
    call_expression
      function: member_expression
        object: identifier (console)
        property: property_identifier (log)
      arguments: arguments
        string ("Hello, world!")
```

Meet jsluice: Extracting URLs

- There's a `jsluice` Go package, and also a command-line tool
  - We're going to focus mainly on the command-line tool :)
- The `urls` mode can extract URLs, paths, and (where possible) HTTP methods, headers, body data etc
  - From calls to `fetch`, uses of `XMLHttpRequest`, assignments to `document.location`, calls to jQuery's `$.get`, `$.post`, and `$.ajax`, and a handful of other places

jsluice outputs JSONLines; you might want to pipe it to **jq** :)

```
$ jsluice urls fetch.js
{
  "url": "/api/v2/guestbook",
  "method": "POST",
  "headers": {
    "Content-Type": "application/json"
  },
  "type": "fetch"
}
```

XMLHttpRequest is tricksy

- ✦ XMLHttpRequest is especially annoying to deal with
  - – The data we want is spread out between multiple function calls
- ✦ Note that **`jsluice`** understands string concatenation :)

```
function callAPI(method, callback){
    var xhr = new XMLHttpRequest();
    xhr.onreadystatechange = callback;
    xhr.open('GET', '/api/' + method + '?format=json');
    xhr.setRequestHeader('Accept', 'application/json');

    if (window.env != 'prod'){
        xhr.setRequestHeader('X-Env', 'staging')
    }
    xhr.send();
}
```

```
{
  "url": "/api/EXPR?format=json",
  "queryParams": ["format"],
  "method": "GET",
  "headers": {
    "Accept": "application/json",
    "X-Env": "staging"
  },
  "type": "XMLHttpRequest.open"
}
```

'EXPR' is the default placeholder, but you can change it with **--placeholder**

Secret Sauce

- Modern web apps talk to lots of APIs, run in The Cloud™, and need secrets for stuff like that
- Sometimes those secrets end up in JavaScript files
- You can find secrets with `jsluice` too!

```
$ jsluice secrets awskey.js
{
  "kind": "AWSAccessKey",
  "data": {
    "key": "AKIAIOSFODNN7EXAMPLE",
    "secret": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
  },
  "filename": "awskey.js",
  "severity": "high",
  "context": {
    "awsKey": "AKIAIOSFODNN7EXAMPLE",
    "awsSecret": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "bucket": "examplebucket",
    "server": "someserver.example.com"
  }
}
```

Look at that sweet context
that was extracted!

Custom Secrets

- There are built-in matchers for AWS, GCP, GitHub, and a few other types of secrets
- The internet is awash with different secrets types, and your target might use an obscure vendor
- You can provide your own patterns in a JSON file :)

```
[
  {
    "name": "genericSecret",
    "key": "(secret|private|apikey)",
    "value": "[%a-zA-Z0-9+/]+"
  },
  {
    "name": "firebaseConfig",
    "object": [
      {"key": "apiKey", "value": "^AIza.+"},
      {"key": "storageBucket"}
    ]
  }
]
```

You can specify a severity too, to make triage easier

```
$ jsluice secrets --patterns=custom.json firebase.js
{
  "kind": "firebaseConfig",
  "data": {
    "apiKey": "AIzaSyB47WKzDu9kkmFAsAYFlagkuJxdEXAMPLE",
    "appId": "1:586572527435:web:14c624679103dc3e74b755",
    "authDomain": "someauthdomain.firebaseapp.com",
    "projectId": "someprojectid",
    "storageBucket": "somebucketthatisnotthere.appspot.com"
  },
  "filename": "firebase.js",
  "severity": "info",
  "context": null
}
```

JSLUICE

Queries

- Tree-sitter is super cool, it has its own query language for querying syntax trees
- The **query** mode lets you run queries, and massages the results into valid JSON
- Use the **tree** mode we saw earlier to help you write queries
  - Also the docs: https://tree-sitter.github.io/tree-sitter/using-parsers#query-syntax

```
$ jsluice query -q '(object) @m' fetch.js | jq
{
  "body": "JSON.stringify({id: 123})",
  "headers": {
    "Content-Type": "application/json"
  },
  "method": "POST"
}
{
  "Content-Type": "application/json"
}
{
  "id": 123
}
```

If **jsluice** can't convert something directly to JSON it makes it a string

A Neat Trick: Finding Common Keys

- ✦ Need a word-list for the most common object keys?
- ✦ Try out this *one-liner* :)

```
$ find . -type f -name '*.js' |           # Find JavaScript files
  jsluice query -q '(object) @m' |        # Extract the objects
  jq -r 'to_entries[] | .key' |           # Extract the keys
  sort | uniq -c | sort -nr               # Sort and rank them
5 method
4 headers
3 url
3 server
3 secret
3 data
3 Content-Type
...
```

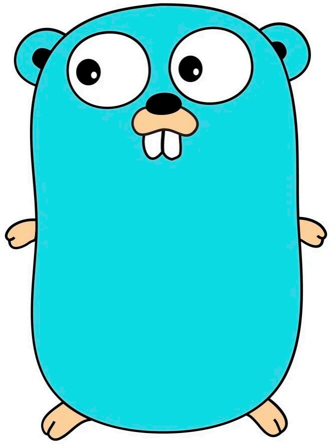
Maybe my *testdata* directory doesn't make for the most representative object keys (:

Where Good Things Come

- The command-line tool is nice, and you can use it for automation in shell scripts
- But if you want to get serious, use the Go package…

```
analyzer := jsluice.NewAnalyzer(sourceCode)

analyzer.AddURLMatcher(
  jsluice.URLMatcher{"string", func(n *jsluice.Node) *jsluice.URL {

    val := n.DecodedString()
    if !strings.HasPrefix(val, "mailto:") {
      return nil
    }

    return &jsluice.URL{URL: val, Type: "mailto"}
  }},
)

for _, match := range analyzer.GetURLs() {
  fmt.Println(match.URL)
}
```

You can make custom matchers using the full power of Tree-sitter :)



One Last One-liner

- Sometimes the most interesting things are in *inline JavaScript*
- Use **htmlq** to extract them, and some shell trickery to process them :)
  - https://github.com/mgdm/htmlq

```
$ find . -type f -exec file {} \; |   # Find files and check what type they are
  grep 'HTML document' |              # Take just the HTML files
  cut -d: -f1 |                       # Remove everything after the filename
  while read htmlfile; do             # Loop over each filename
    # Use htmlq to extract inline JavaScript
    jsluice secrets <(htmlq -f $htmlfile script --text)
  done
```

Maybe **jsluice** will get native support for HTML files soon :)

THANK YOU <3

Questions? :)

BISHOPFOX.COM

Presented by: **Tom Hudson, Senior Security Engineer, Bishop Fox** | Date: 2023-06-24