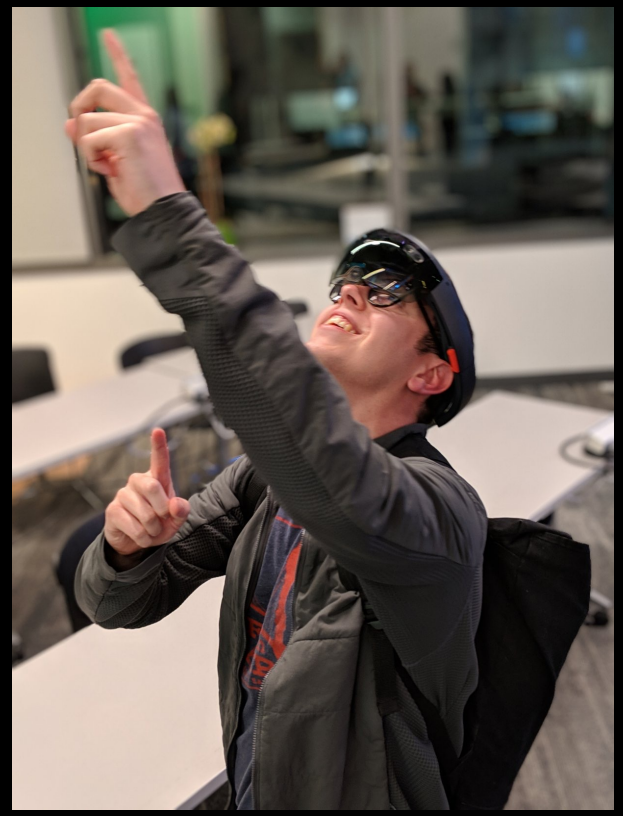
Manual JavaScript Analysis is a Bug

LASCON 2021

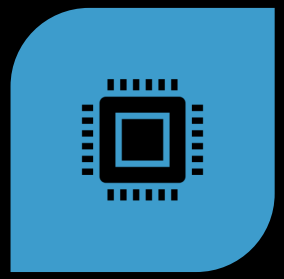
About Me

- Lead Security Engineer - Salesforce
- OWASP SF Organizer
- B.Sc. in Computer Security and Ethical Hacking
- SecuriTEA & Crumpets Podcast Host
  - https://www.youtube.com/c/karlbaskin
  - https://twitter.com/securitnc



Agenda

MANUAL WORK IS A BUG



WHAT CAN/SHOULD WE AUTOMATE?

WHAT ALREADY EXISTS

PROOF OF CONCEPT APP

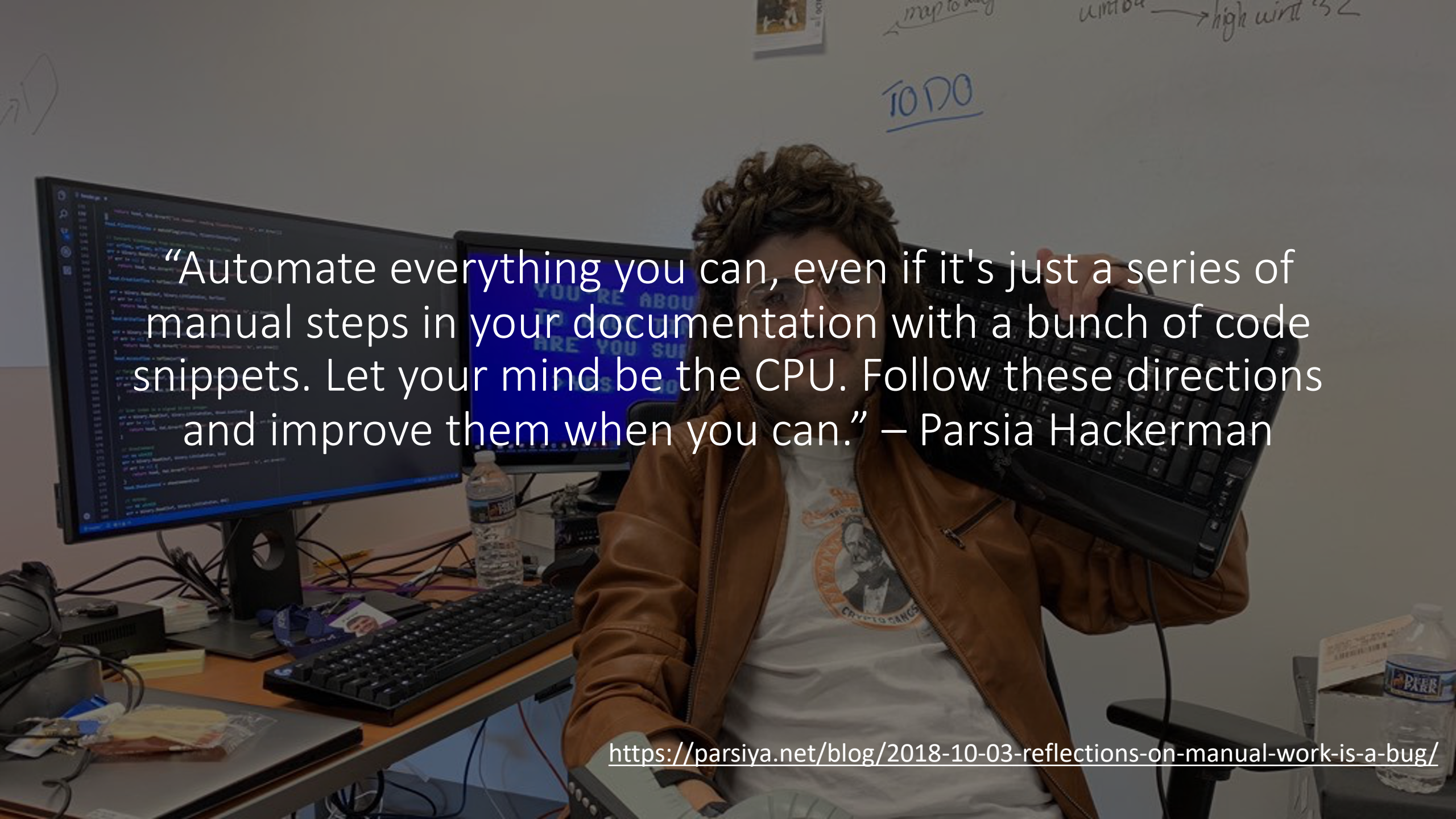
Manual Work Is a Bug

A.B.A: always be automating – Thomas A. Limoncelli

- Automation can be used to help improve productivity and reduce repetitive work

- Four Phases
  1. Document the steps
  2. Create automation equivalents
  3. Create automation
  4. Self-service and autonomous systems

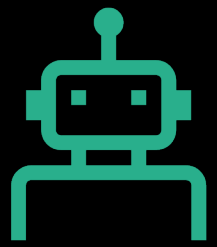
https://queue.acm.org/detail.cfm?id=3197520
https://parsiya.net/blog/2018-10-03-reflections-on-manual-work-is-a-bug/



"Automate everything you can, even if it's just a series of manual steps in your documentation with a bunch of code snippets. Let your mind be the CPU. Follow these directions and improve them when you can." – Parsia Hackerman

https://parsiya.net/blog/2018-10-03-reflections-on-manual-work-is-a-bug/

Automation will allow you to spend your time focusing on things which require your energy such as business logic flaws, or identifying more complex attack chains



What ~~can/~~should we automate?

We should not have to manually detect ‘basic’ sources and sinks

Detecting Issues

Semgrep

Burp Suite

Detecting Issues: Semgrep

- Semgrep is free/customizable and can identify issues in source-code
  - Supports JavaScript/TypeScript

SEMGREP RULE

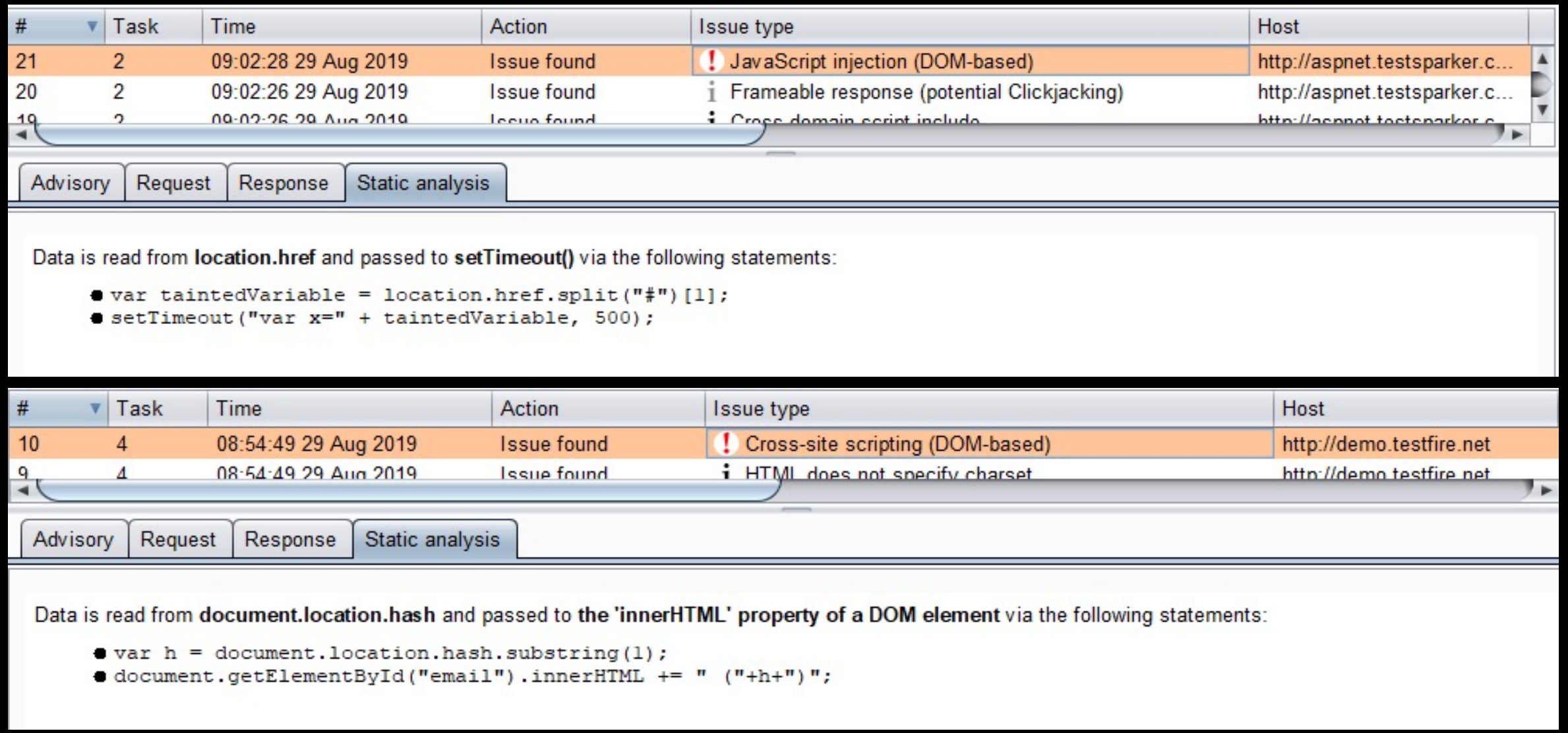
Simple Advanced

```
rules:
- id: xss
  message:  DOM XSS
  languages: [js]
  severity: INFO
  patterns:
    - pattern-either:
      - pattern:  |
          $($QUERYSELECTOR).attr("href",(new URLSearchParams(window.location.search)).get($PARAM));
```

TEST CODE

```
1  $(function(){
2      $('#backLink').attr("href",(new URLSearchParams(window.location.search)).get('returnUrl'));
3  });
4
```

https://parsiya.net/blog/2021-06-22-semgrep-the-surgical-static-analysis-tool/
https://semgrep.dev

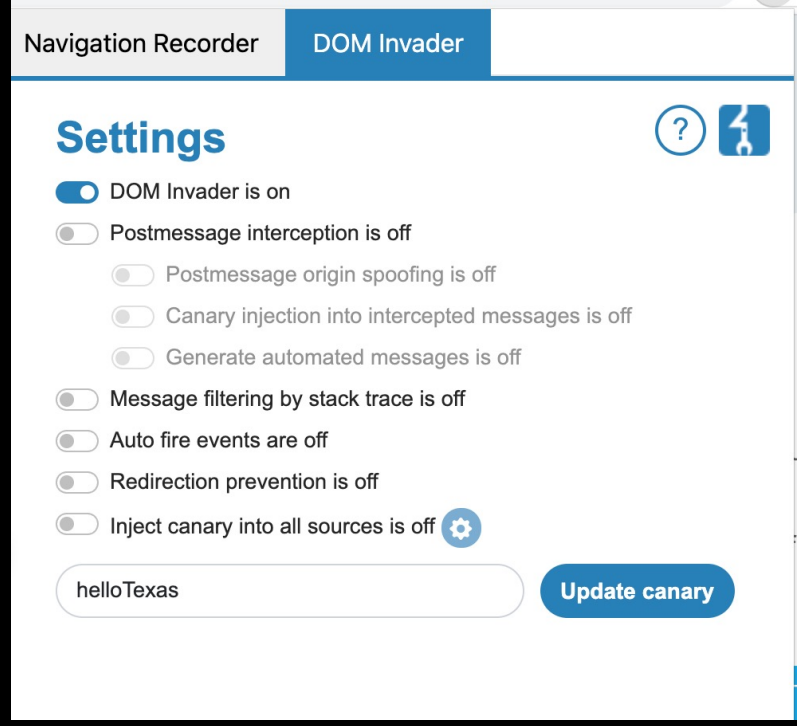


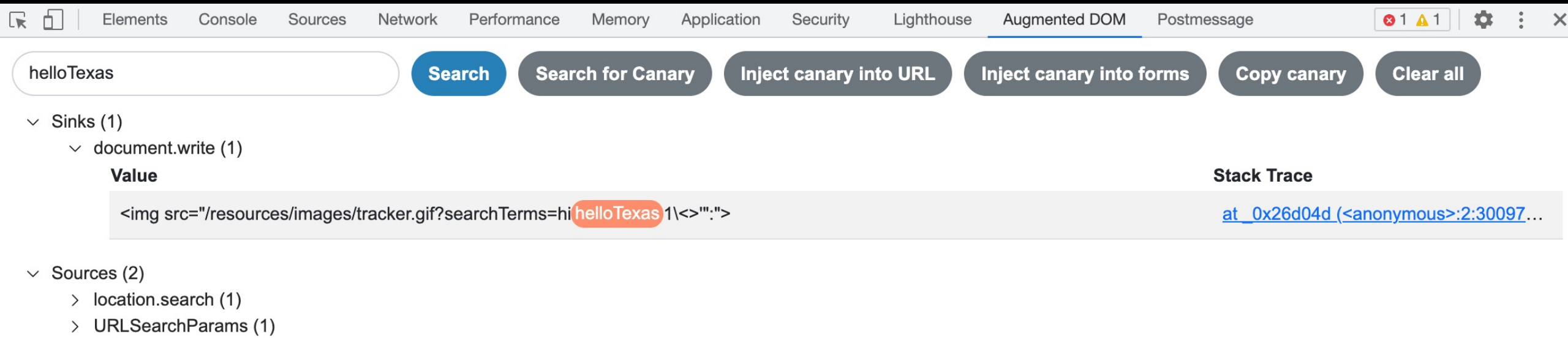
Detecting Issues: Burp Suite

https://portswigger.net/blog/dynamic-analysis-of-javascript

DOM Invader

- The browser should be leveraged to accurately detect DOM Issues
- DOM invader works with the community version!





Detecting Out-Of-Bound Responses

- Burp Collaborator is **king** for Out-Of-Bound Detection
  - XXE
  - SSRF
  - bXSS
  - SQLi

- Setting up your own server is a good exercise

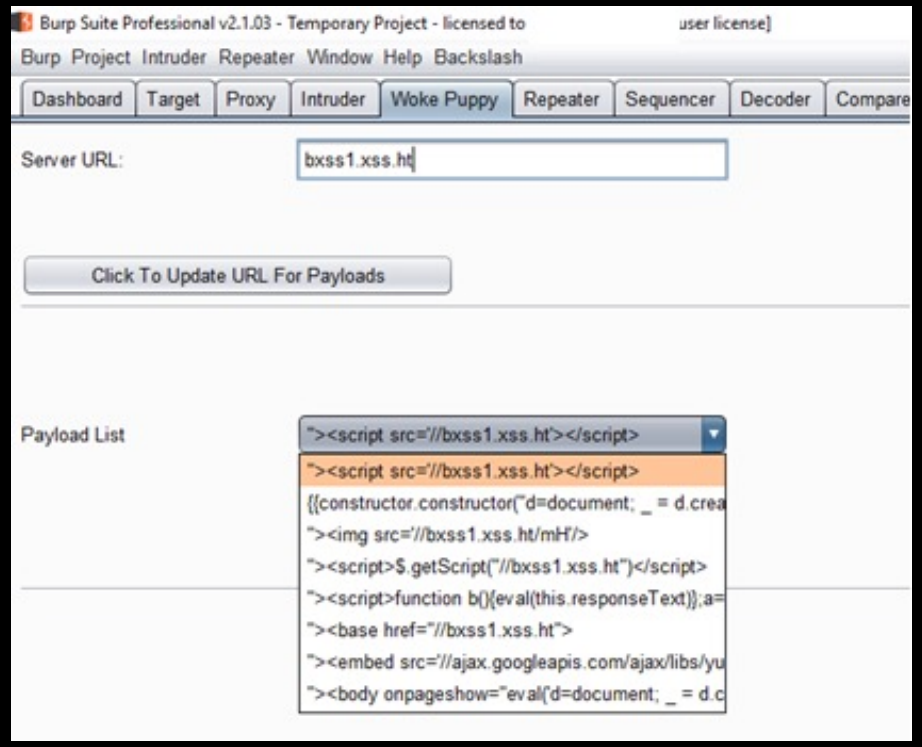
Detecting Out-Of-Bound Responses: bXSS

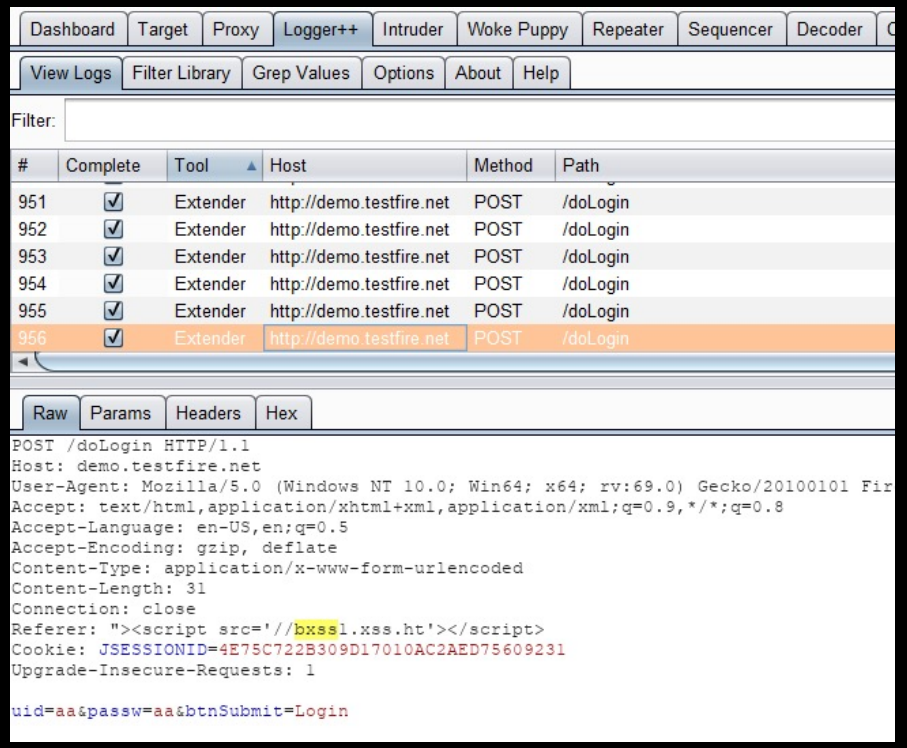
- Burp Collaborator does detect bXSS, but..
- Sleepy puppy was the initial ‘bXSS’ tool (now deprecated)
- Current contenders:
  - XSS Hunter (Top Dog)
  - bXSS (Hot Dog)

https://github.com/Netflix-Skunkworks/sleepy-puppy
https://github.com/LewisArdern/bXSS
https://xsshunter.com

Detecting Out-Of-Bound Responses: bXSS

- Modifying SP Burp Extension to detect bXSS





https://github.com/LewisArdern/sleepy-puppy
https://portswigger.net/web-security/cross-site-scripting/cheat-sheet

We should not have to manually identify what URLs, and assets exist in an application!

Endpoint Discovery:

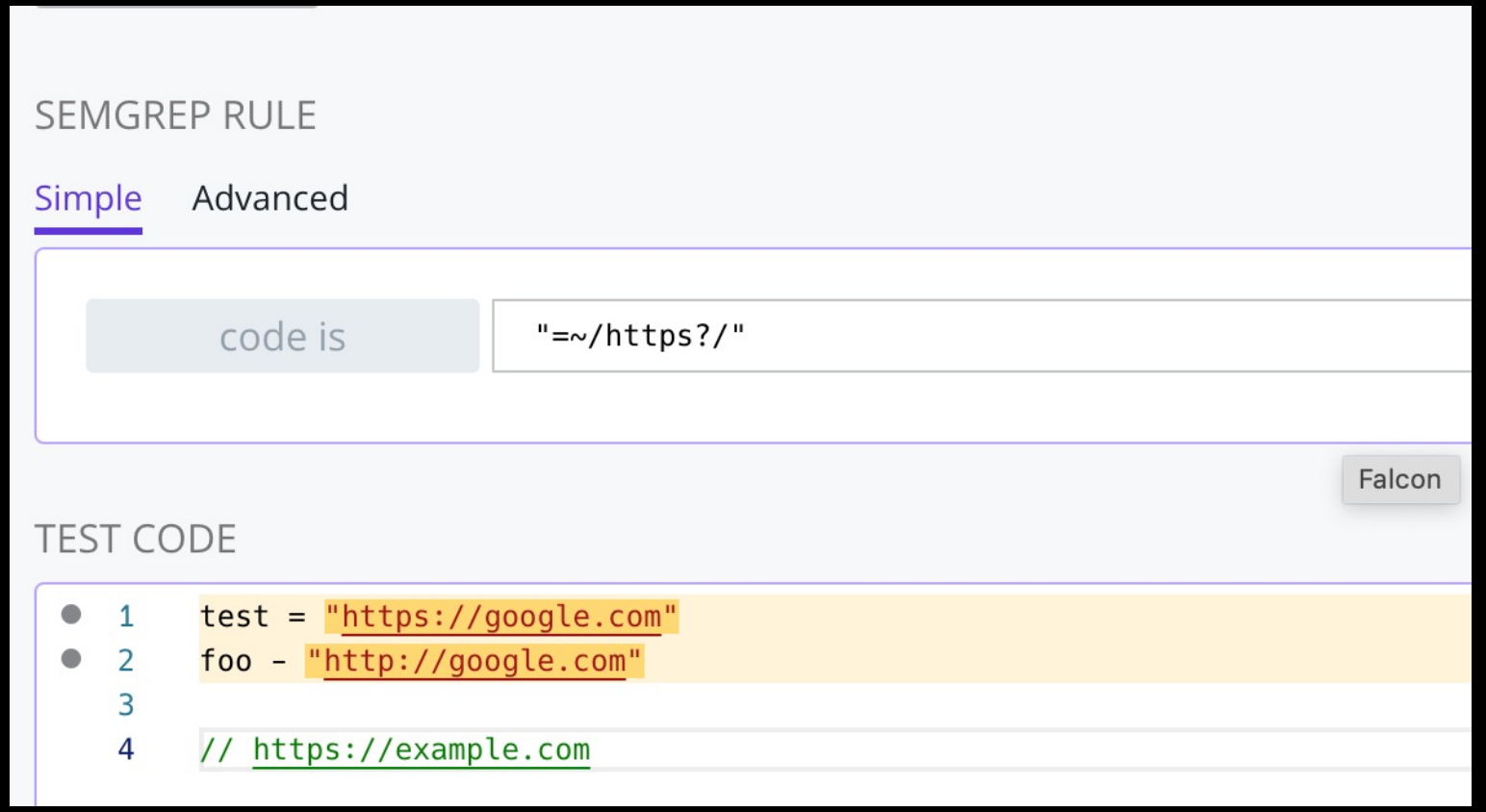
https://github.com/PortSwigger/js-link-finder

LinkFinder Log:

```
38 - //learn.one.aol.com
39 - //help.one.aol.com
40 - http://nexage-dev.demohoster.com
41 - http://qa-ge-iqservice001.us-ec.adtech.com:8080/h2/index.do
42 - http://mydev.aol.com:9003/
43 - http://mydev.aol.com:9004/
44 - https://onevideo.aol.com
45 - http://m-dev-aop-devint05.advertising.aol.com:8083/aop-aux/
46 - http://m-prd-aopadminui001.advertising.aol.com/admin/
47 - http://uk.admin.adlearnop.advertising.aol.com/
48 - http://jp.admin.adlearnop.advertising.aol.com/
49 - http://sj-qa03.sj.adap.tv:4310/platform/index.html#/logon
50 - http://mydev.aol.com:9010/analytics/
51 - https://portal.vidible.tv
52 - http://mydev.aol.com:9002/
53 - https://oneapi-dev.aol.com/one-central/
54 - https://one-dev.aol.com/one-central/
55 - https://id-uat.b2b.oath.com/identity/oauth2/authorize
56 - https://id-uat.b2b.oath.com/identity/XUI/#logout/
57 - https://one-dev.aol.com/one-central/oidc.html
```

Endpoint Discovery:

https://semgrep.dev/s/nKyZ

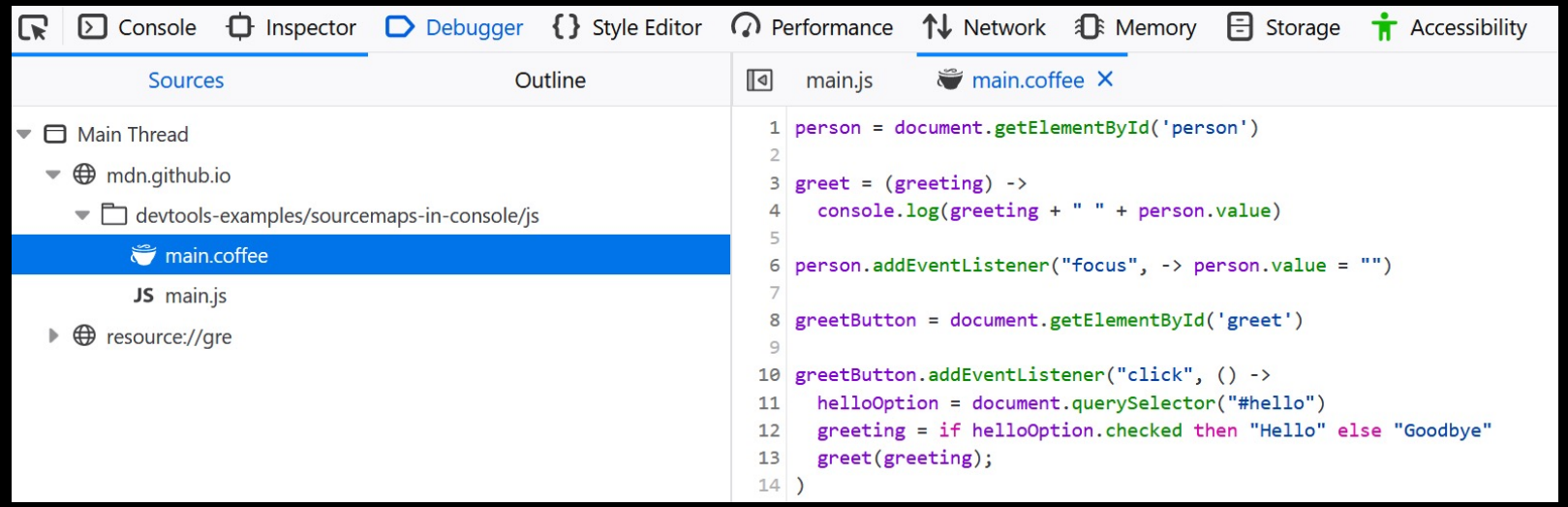


Source Maps (*.js.map)

- Production minified JavaScript is intimidating to review:
  - Computer Transpiled: TypeScript, Coffee Script
  - Minified: Webpack, rollup, Pracel

Source Maps (*.js.map)

- Tools to reverse source maps
  - 99% Time Can be Viewed directly in the browser
  - https://github.com/denandz/sourcemapper
  - https://github.com/pavloko/source-map-unpack
  - https://github.com/mozilla/source-map



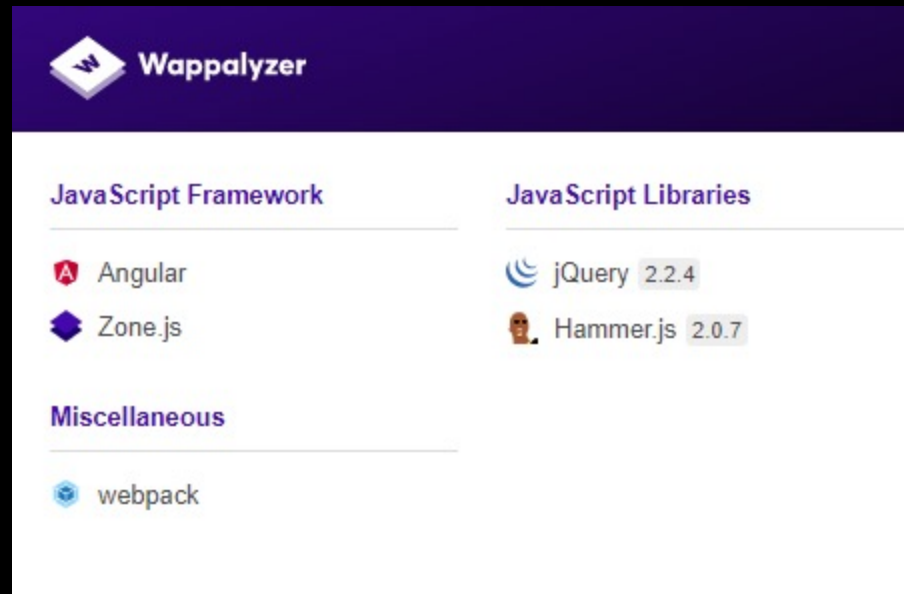
- Discovery:
  - https://github.com/righettod/burp-piper-custom-scripts#extract-spa-low-hanging-fruits

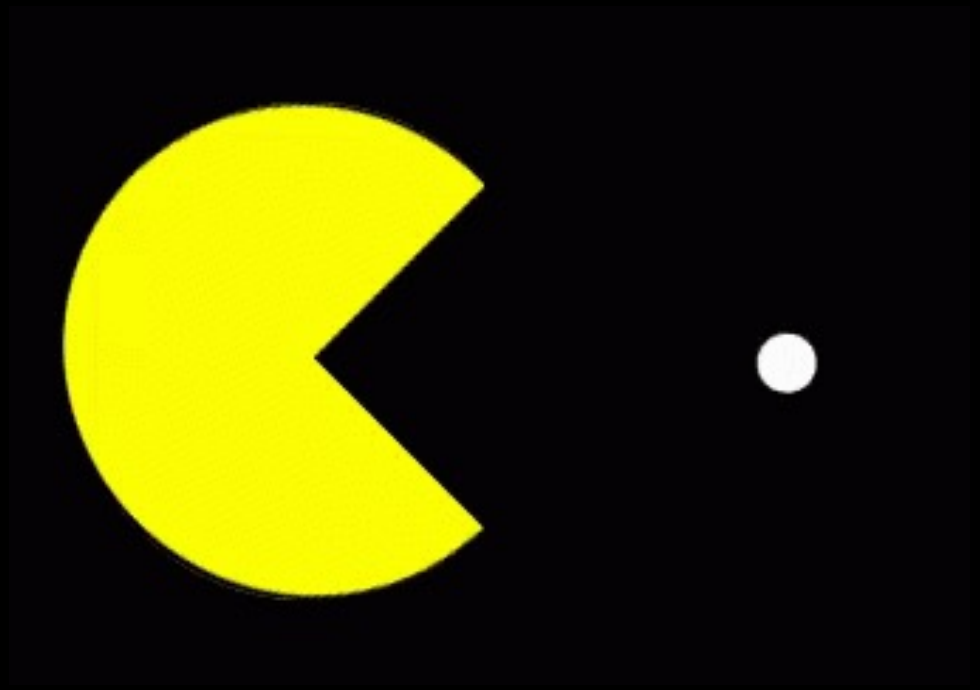
Understanding Technologies (Fingerprinting)

- Extensions
  - Asset Discover
    - https://github.com/redhuntlabs/BurpSuite-Asset_Discover
  - Software Version
    - https://github.com/zaproxy/zap-extensions/tree/main/addOns/wappalyzer/

- Browser Plugins
  - https://www.wappalyzer.com/

- Source Code
  - https://docs.npmjs.com/files/package.json





We should not have to manually find ‘hungry’ regular expressions

Detecting ReDoS

- Certain vulnerable regular expressions loop exponentially when validating specific strings
  - The expression *^(a+)+$* takes 65536 steps to check the string *aaaaaaaaaaaaaaaaX*
  - The number of steps doubles for each additional “a”
  - An attacker can exploit vulnerable expressions to consume server resources and create a DoS condition

```
const regularExpression = /^(a+)+$/

const string = 'aaaaaaaaaaaaaaaaX'

regularExpression.test(string)
```

Detecting ReDoS

- Useful reading:
  - https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-staicu.pdf

- Useful tools for detecting ReDoS
  - https://github.com/davisjam/vuln-regex-detector
  - https://www.npmjs.com/package/redos
  - https://github.com/substack/safe-regex
  - https://github.com/MakeNowJust-Labo/recheck

Detecting ReDoS

- Personal approach:
  - Extract Regex patterns with Semgrep
  - Pass expressions to recheck
  - Combine Semgrep/recheck output

```
ran 1 rules on 1 files: 1 findings
{
  '0': {
    status: 'vulnerable',
    regex: '/([a-z]+)+$/',
    path: '/Users/lardern/Documents/Projects/Personal/Example/test.js',
    line: 59
  }
}
```

We should not have to manually find API keys or secrets in source code

```
Secret found: AWS = 'amzn.mws.00a19111-8311-5311-5391-5e211cbfcd5c'
Secret found: aws_account_id = 123456789012
```

Detecting Secrets

- TruffleHog
  https://github.com/dxa4481/truffleHog
- Ripgrep
  https://github.com/BurntSushi/ripgrep
- git-secrets
  https://github.com/awslabs/git-secrets
- Semgrep
  https://semgrep.dev/p/secrets

https://edoverflow.com/2019/ci-knew-there-would-be-bugs-here/

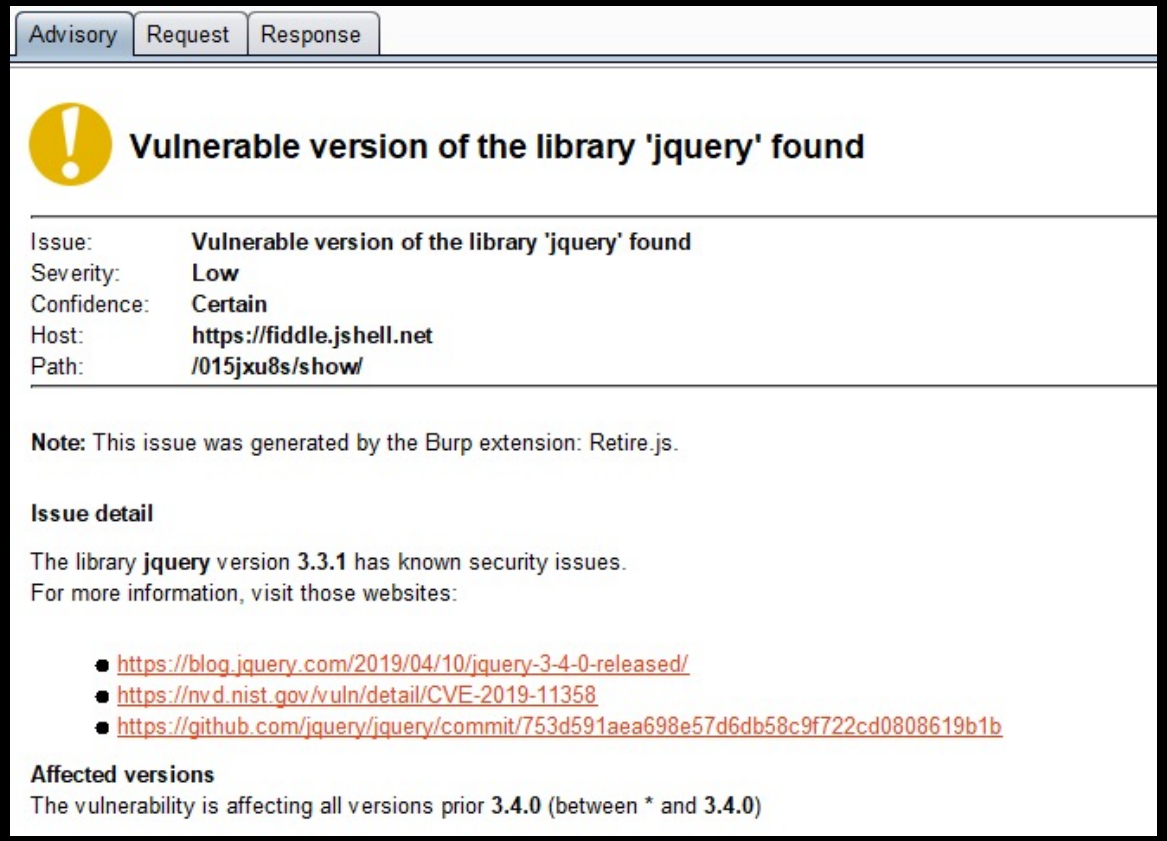
We should not have to manually identify known security issues in third-party components

Detecting Vulnerable Third-Party Components

Example	Command
npm	npm audit
yarn	yarn audit
bower	auditjs --bower bower.json
Client-Side JavaScript	retire --js /path/
Node.js Open-Source	snyk test

https://docs.npmjs.com/cli/audit
https://retirejs.github.io/retire.js/

NPM Audit && RetireJS




```
C:\Users\lewis\Documents\Projects\metasecjs>npm audit

                        === npm audit security report ===

found 0 vulnerabilities
 in 1391 scanned packages
```

What if we had a tool such as npm audit that could help drive secure-development and issue discovery?

What would this look like?

Expertise:

1. Review popular dependencies for:
   a) Possible misconfigurations
   b) Possible security issues
2. Add them to a repository to retrieve later

Tool Creation:

1. Review a package.json file for its dependencies
   a) Retrieve the name, and version
2. Look-up repository and match version and name
3. Return list of recommendations to investigate in a readable format

Tool Output

	A	B	C	D	E	F	G
1	Package	Version	Guidance	Outcome	Assessor Comments	References	
2	express-session	^1.62.2	Ensure that the secret is not hard-coded				
3			Ensure that the httpOnly flag is set for applications that do not need access to the cookie, e.g Angular				
4			Ensure the 'secure' flag is set to prevent sessions sent over HTTP				
5			Ensure the sameSet flag is set with 'strict' or 'relaxed'				
6			Ensure the path is set to '/'				
7			Ensure the cookie name beings with a cookie prefix __Host or __Secure				
8			Ensure that the store is not the default 'Memory' store as this can be problematic for memory exhaustion and other security issues				
9	another-package	v9001	It's over 9000!				

Example: express-session

```
session = require('express-session')
app.use(session({
    secret: config.SESSION_SECRET,
    name: '__Host-auth-cookie',
    cookie: { httpOnly: true, secure: true, sameSite: ‘strict’, path: ‘/’ },
    store: new MySQLStore({}, sqlConnection)
}));
```

Never Store The Secret In Source Control

```
session = require('express-session')
app.use(session({
    secret: config.SESSION_SECRET,
    name: '__Host-auth-cookie',
    cookie: { httpOnly: true, secure: true, sameSite: 'strict', path: '/' },
    store: new MySQLStore({}, sqlConnection)
}));
```

Enforce The Browser To Only Transmit Over HTTPs

```
session = require('express-session')
app.use(session({
    secret: config.SESSION_SECRET,
    name: '__Host-auth-cookie',
    cookie: { httpOnly: true, secure: true, sameSite: 'strict', path: '/' },
    store: new MySQLStore({}, sqlConnection)
}));
```

Prevent Inclusion In Cross-Origin Requests

```
session = require('express-session')
app.use(session({
    secret: config.SESSION_SECRET,
    name: '__Host-auth-cookie',
    cookie: { httpOnly: true, secure: true, sameSite: ‘strict’, path: ‘/’ },
    store: new MySQLStore({}, sqlConnection)
}));
```

Prevent Memory Exhaustion With A Store

```
session = require('express-session')
app.use(session({
    secret: config.SESSION_SECRET,
    name: '__Host-auth-cookie',
    cookie: { httpOnly: true, secure: true, sameSite: 'strict', path: '/' },
    store: new MySQLStore({}, sqlConnection)
}));
```

Triaged Tool Output

	A	B	C	D	E	F	G
1	Package	Version	Guidance	Outcome	Assessor Comments	References	
2	express-session	^1.62.2	Ensure that the secret is not hard-coded	Good	Uses config		
3			Ensure that the httpOnly flag is set for applications that do not need access to the cookie, e.g Angular	Good	Uses httpOnly		
4			Ensure the 'secure' flag is set to prevent sessions sent over HTTP	Good	Uses secure		
5			Ensure the sameSet flag is set with 'strict' or 'relaxed'	Good	Uses sameSite strict		
6			Ensure the path is set to '/'	Good	Sets path to '/'		
7			Ensure the cookie name beings with a cookie prefix __Host or __Secure	Good	Sets __Host on cookie name		
8			Ensure that the store is not the default 'Memory' store as this can be problematic for memory exhaustion and other security issues	Good	Uses MySQL Database!		
9	another-package	v9001	It's over 9000!	Yes	It is over 900!		

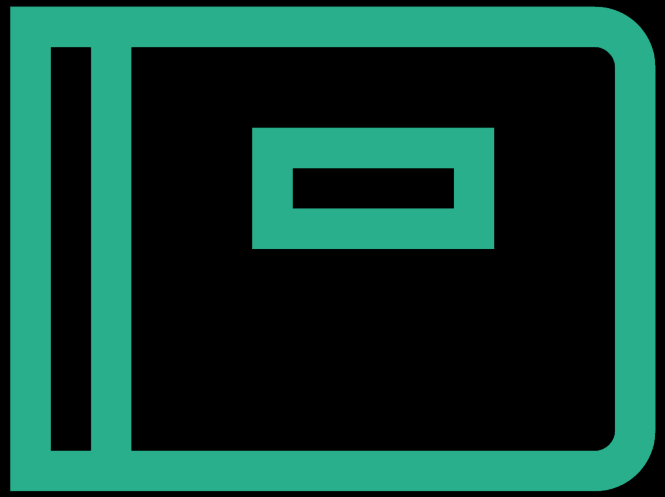
What if we had a tool which combines all the open source projects which help identify security issues and drive secure development?

- Metasec.js is a **meta analysis** tool to review **JavaScript** Applications using **open-source** tools:

- Experimental Functionality:
  - Help drive secure development based on package.json dependencies
  - Identifies issues with third-party dependencies
  - Looks for secrets
  - Looks for ReDoS
  - Performs security linting against an application
  - Audits the application for electron issues with doyensec/electronegativity if electron is in the package.json file

Metasec.js

https://github.com/lewisardern/metasecjs



Demo

Conclusion

- Automation helps streamline work processes
- To get the best results you should automate from both a static and dynamic perspective
- Automation can be used to help identify security issues and drive secure development in modern JavaScript applications
- Automate and be AWESOME!

Thank you!

Email: lewis@ardern.io
Website: https://ardern.io
Twitter: https://twitter.com/LewisArdern
GitHub: https://github.com/LewisArdern