

Web Application Route Security

Finding authentication and authorization security bugs in web application routes

CS-6727 Final Presentation
April 9th, 2023

Matt Schwager
mschwager3@gatech.edu

What is the problem?

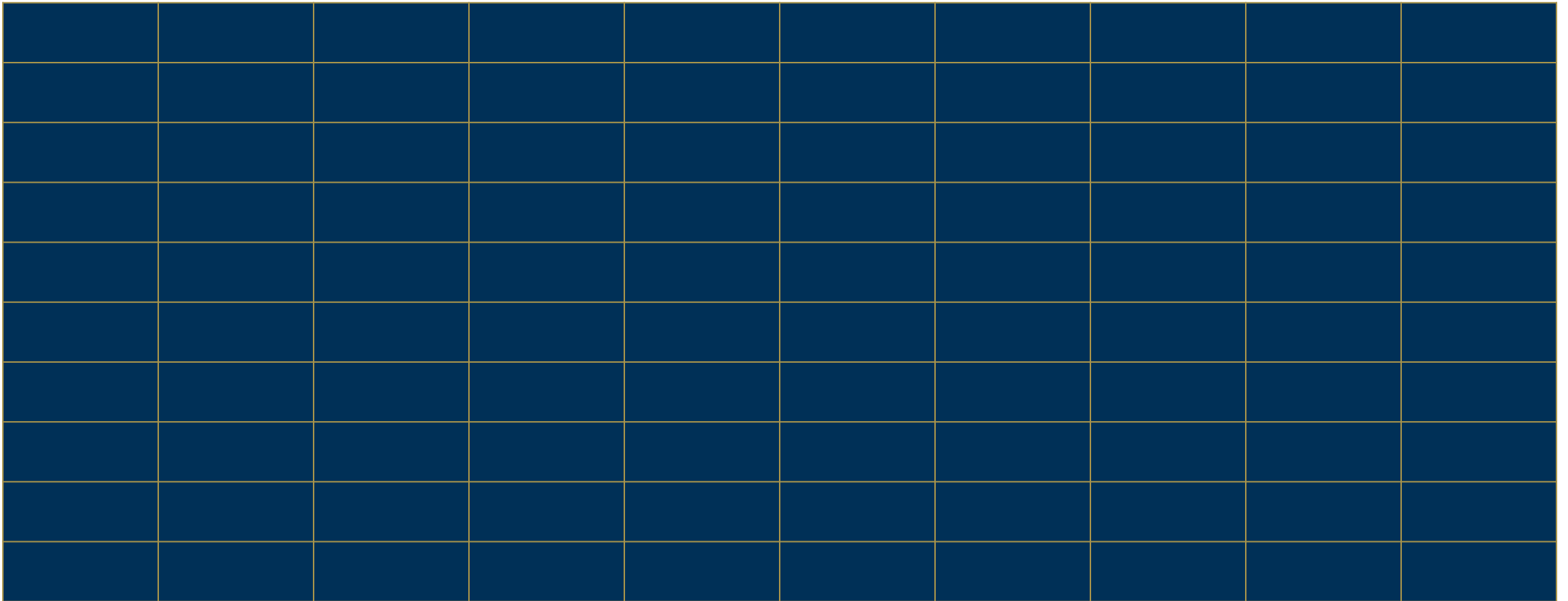
- Insecure routes in application code
- **Routes**: connect URL paths to application code responsible for handling that web request
- **Insecure**: improper authentication (authn) or authorization (authz) logic applied to web request
- **Authn**: validate who you are
- **Authz**: validate what you can access
- **Roles**: access levels specifying what actions you may perform
- Consider a web request to **<https://example.com/users/create>**
 - Endpoint is publicly available (no authn)
 - E.g. missing a **@RequiresAuthentication** annotation
 - Endpoint is accessible by guest accounts (improper authz)
 - E.g. using **@RolesAllowed(ROLE_GUEST)** instead of **@RolesAllowed(ROLE_ADMIN)**

Why is it a problem?

- Modern web application have hundreds or thousands of routes
- Simple programmer error, forgetfulness, or unfamiliarity with codebase
- Complex authz schemes with dozens of user roles and access controls
- Evidenced by industry standard resources
 - **2022 CWE Top 25 #14** - CWE-287: Improper Authentication
 - **2022 CWE Top 25 #16** - CWE-862: Missing Authorization
 - **2022 CWE Top 25 #18** - CWE-306: Missing Authentication for Critical Function
 - **#21 most CVEs by CWE** - CWE-284: Access Control (Authorization) Issues
 - **#47 most CVEs by CWE** - CWE-639: Access Control Bypass Through User-Controlled Key
 - **2021 OWASP Top 10 #1** - Broken Access Control
 - **2021 OWASP Top 10 #7** - Identification and Authentication Failures (formerly Broken Authentication)
 - **2019 OWASP API Top 10 #2** - Broken User Authentication
 - **2019 OWASP API Top 10 #5** - Broken Function Level Authorization

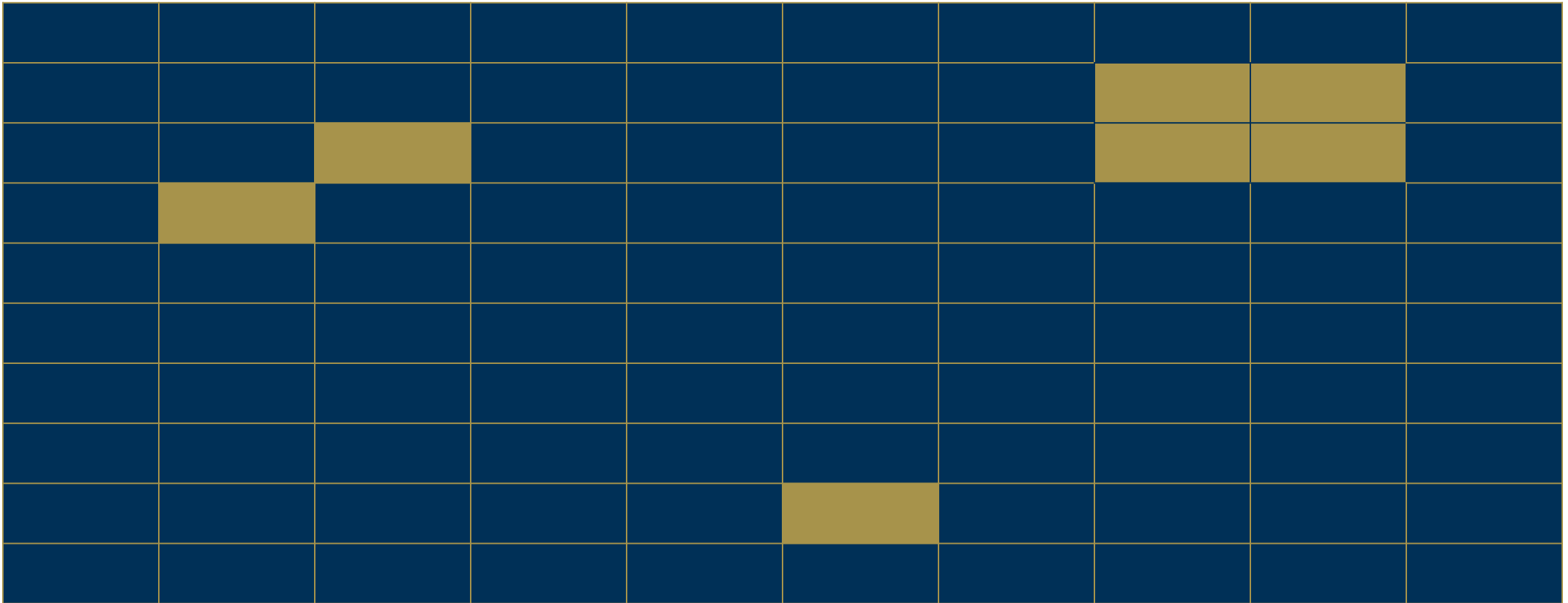
Needle in a haystack

Find the insecure route:



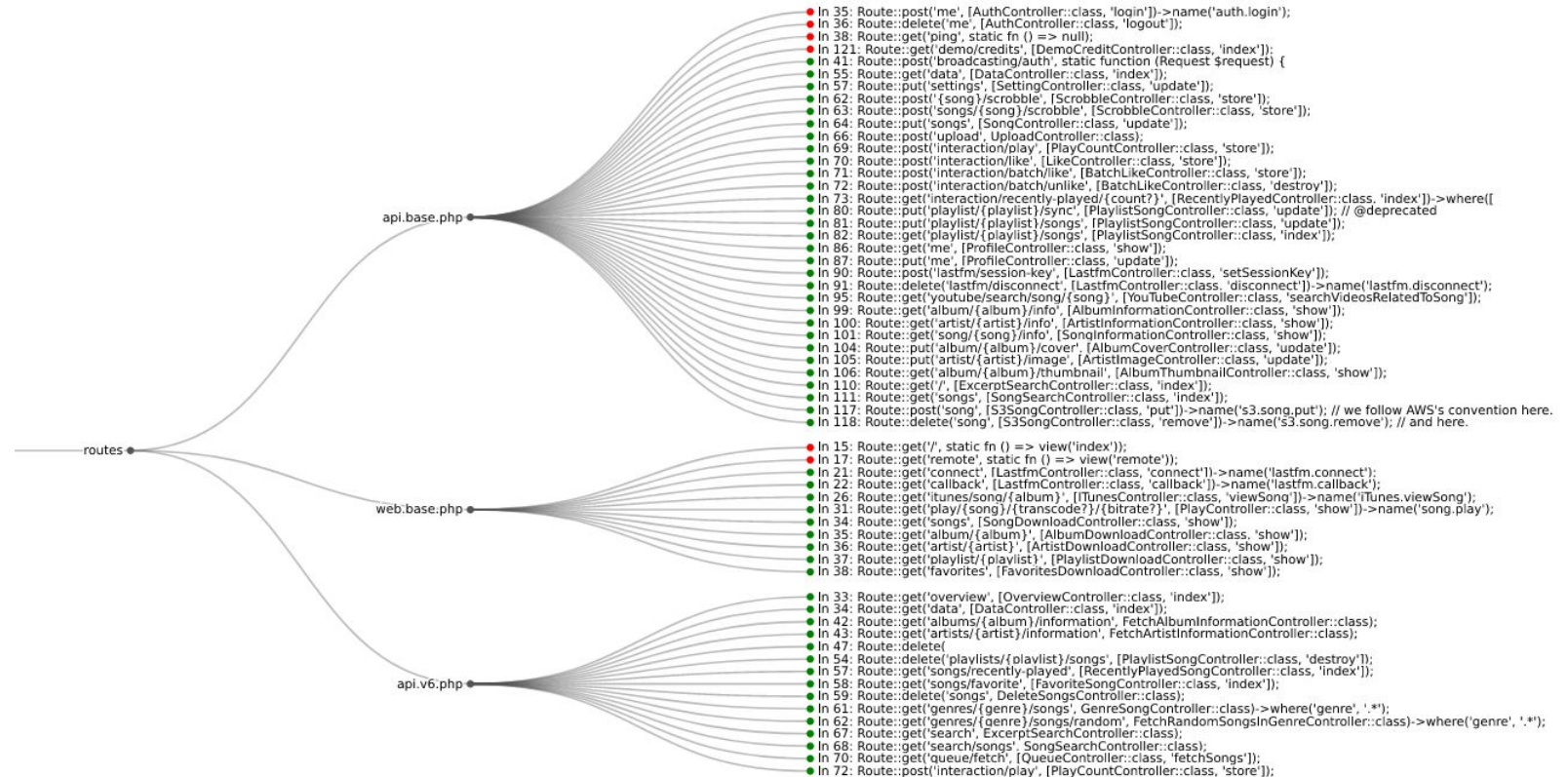
What if needles glowed in the dark?

Find the insecure route:



Introducing: route-detect

- Uses **static analysis** to find web application routes and their authn and authz properties
- Enables security researchers and engineers to quickly analyze and diagnose codebases for route security misconfigurations
- Supports **6 programming languages**, **17 web application frameworks**, and **61 authn/authz libraries**
- Favors breadth over depth



Routes from koel streaming server

How does it work?

- Provides an **easily installable, open source, command-line application**
- Builds on [Semgrep](#) for code analysis and command-line findings
- Builds on [D3.js](#) and local HTML files for visualizations
- Uses Python for “glue” code, basic interprocedural/interfile analysis, and distribution mechanism
- Makes heavy use of **automated testing to prevent false positives** and detect regressions
 - E.g. create test code that looks like real findings, then ensure route-detect finds it
- Cross-reference against **basic code search with regular expressions to minimize false negatives**
 - E.g. search for “route” and “path”, and look for common authn/authz libraries

Demo

Evaluating route-detect

- Could not find any static analysis tools focusing on web application route authn/authz
- Similar technologies
 - Commercial solution: Contrast Security's [Route Coverage](#)
 - Runtime Application Self-Protection (RASP)
 - More like a firewall than code analysis
 - [Authorize](#) Burp Suite extension
 - Dynamic analysis, not static analysis
 - Requires a running application, and manual tool configuration
- Closest competitors
 - Regular expressions (e.g. grep) – can often find routes, but not authn/authz
 - Manual code analysis – slow, costly, and error-prone
- **Found 18 unintentionally unauthorized routes** at my employer

Evaluating against regular expressions

- Regular expression takeaways:
 - Can often find routes, but not authn/authz properties
 - Inaccuracies due to lack of semantic code understanding (e.g. commented out code)
 - Whitespace, newlines, and variable/module names lead to missing authn/authz properties
 - Multiline queries are ineffective
 - Overcounting due to common terms like “get”, “post”, and “handle”
 - Semgrep allows semantic analysis like type checking and module import detection
- **Regular expressions are okay for route detection, but not authn/authz detection**

Hypothetical manual code analysis

- Engage security consultancy to perform a web application assessment
- 1 person, 1 week engagement, 40 hours, \$20k
- Manual code audit of web application authn/authz properties
 - 1 day without route-detect, 8 hours, \$4k
 - ½ day with route-detect, 4 hours, \$2k
- Hypothetical, but **demonstrates efficiency gains with route-detect**

Real-world manual code analysis

- Analyze [koel](#) streaming server codebase
- Without route-detect
 - Check primary language (PHP): 1 min
 - Check framework (package.json, Laravel): 5 min
 - Understand routing and authn/authz (read Laravel docs): 15 min
 - Investigate authn and authz on 60 codebase routes: 30 min
 - Total: 51 min
- With route-detect
 - Run route-detect in autodetect mode: 1 min
 - Check browser visualization for unauthn/unauthz routes: 2 min
 - Investigate 6 codebase unauthn/unauthz routes: 3 min
 - Total: 6 min
- **51 min to 6 min is an 88% improvement**

Limitations

- Three primary ways to specify authn/authz route properties
 - Intraprocedural (logically close to route, same file) – **route-detect excels here**
 - Interprocedural/interfile (logically far from route, separate files) – **route-detect is weak here**
 - Cross-file class inheritance, “by convention” association, implicit association
 - Common in Python Django, and Ruby Rails and Grape frameworks
 - Global configuration (all in one global location) – **route-detect is weak here**
- Frameworks where authn/authz logic may be specified in many different ways
 - Combinatorial explosion in Semgrep rule size
 - Difficult to effectively manage and test rules
 - Common in Go middleware-based frameworks

Future work

- Anomaly detection
 - E.g. a file contains one route that is unauthenticated and the rest are authenticated
 - E.g. a file contains one route with role `ROLE_GUEST` and the rest are `ROLE_ADMIN`
- Support for additional programming languages, frameworks, and authn/authz libraries
- Improved support for interprocedural/interfile and global configuration authn/authz specifications

Conclusion

Questions, comments, feedback?

mschwager3@gatech.edu

<https://github.com/mschwager/route-detect>