black hat®
ARSENAL 2023

AUGUST 9-10
MANDALAY BAY/LAS VEGAS

black hat ARSENAL 2023

route-detect

Find authentication and authorization security bugs in web application routes

$ whoami

- Matt Schwager
- Senior Product Security Engineer at Red Canary
- Background in software and security engineering
- Interested in automated program analysis
  - Fuzzing, static analysis, dynamic analysis, etc.
  - Making the computer sweat, so you don't have to
- https://github.com/mschwager

Information Classification: General

What is the problem?

- Insecure routes in web application code
- **Routes**: connect URL paths to app code responsible for handling that web request
- **Insecure**: improper authentication (authn) or authorization (authz) logic
  - **Authn**: validate who you are
  - **Authz**: validate what you can access
- **Roles**: access levels specifying what actions you may perform

- Endpoint publicly available (no authn)
- E.g. missing **@RequiresAuthentication** annotation

- Endpoint accessible by guest accounts (improper authz)
- E.g. using **@RolesAllowed(ROLE_GUEST)** instead of **ROLE_ADMIN**

Information Classification: General

Why is it a problem?

- Complexity
  - Modern web applications have **hundreds or thousands of routes**
  - Authz schemes with **dozens of user roles** or access controls
- Opt-in
  - Authn and authz are typically opt-in vs. opt-out
  - Does **not** follow **secure by default** property
  - Programmer error, forgetfulness, or unfamiliarity with codebase

Information Classification: General

Evidence

- 2021 OWASP Top 10
  - #1 - Broken **Access Control**
  - #7 - Identification and **Authentication** Failures (formerly Broken Authentication)
- 2019 OWASP API Top 10
  - #2 - Broken User **Authentication**
  - #5 - Broken Function Level **Authorization**
- 2023 CWE Top 25
  - #11 - CWE-862: Missing **Authorization**
  - #13 - CWE-287: Improper **Authentication**
  - #20 - CWE-306: Missing **Authentication** for Critical Function
  - #24 - CWE-863: Incorrect **Authorization**

Information Classification: General

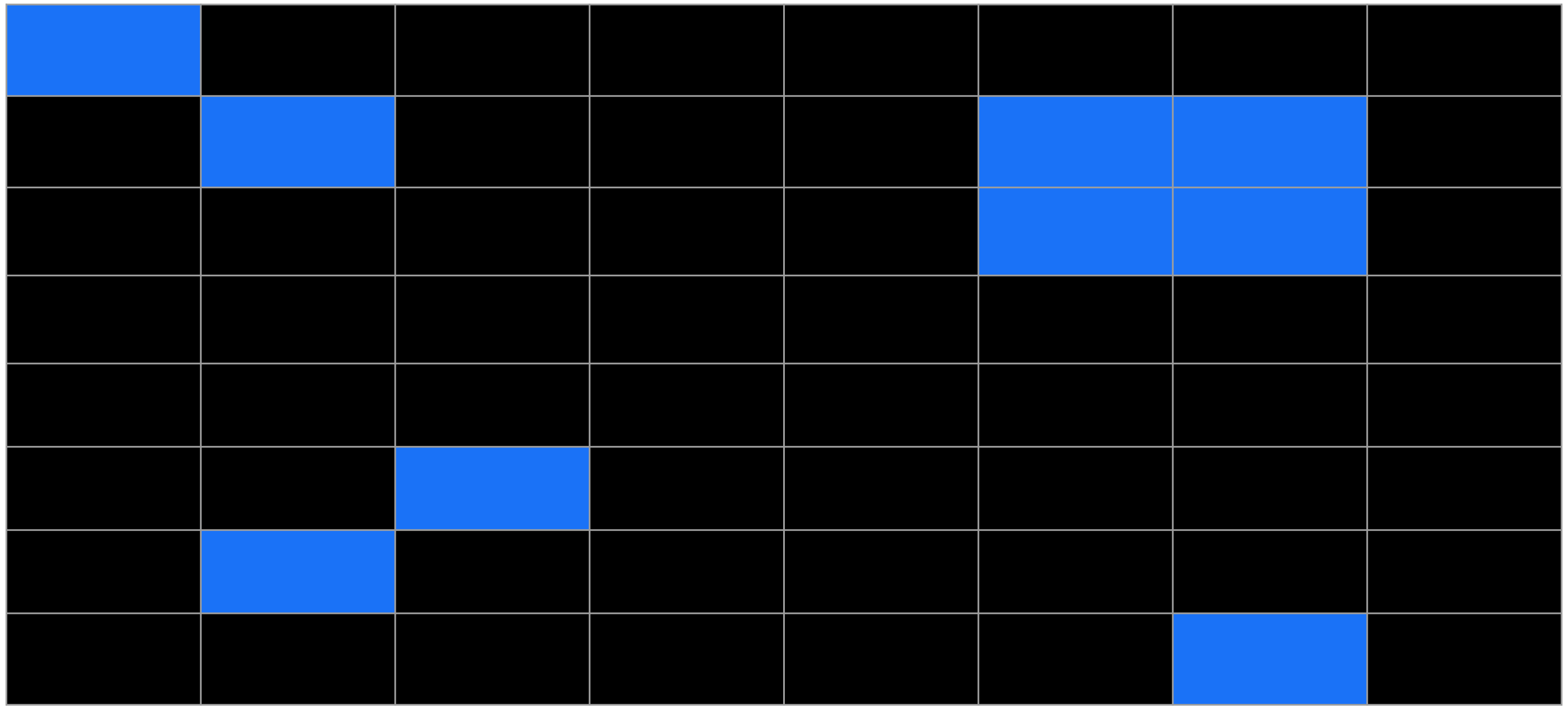
Needle in a haystack problem

Find the insecure routes:

[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

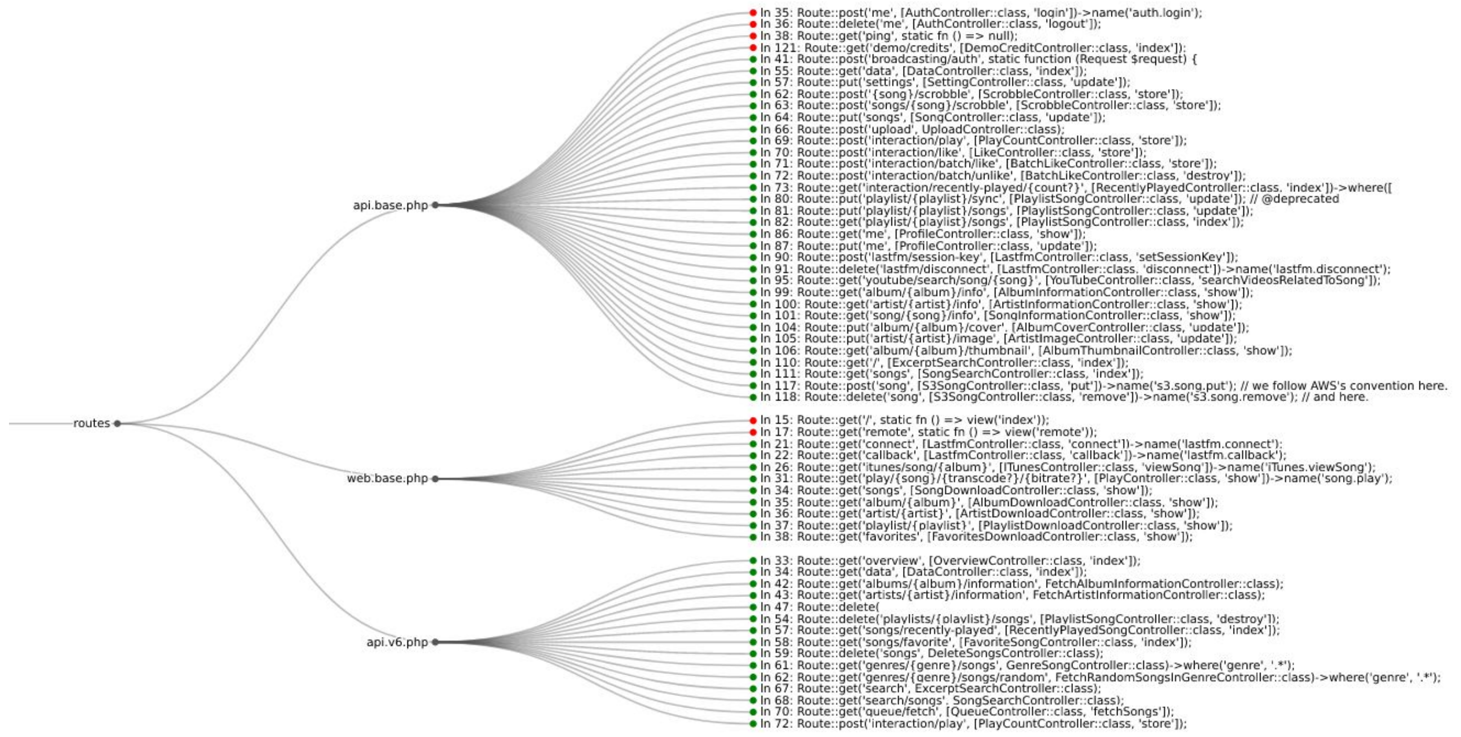
What if needles glowed in the dark?

Find the insecure routes:



Introducing: route-detect

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



```
ln 35: Route::post('me', [AuthController::class, 'login'])->name('auth.login');
ln 36: Route::delete('me', [AuthController::class, 'logout']);
ln 38: Route::get('ping', static fn () => null);
ln 121: Route::get('demo/credits', [DemoCreditController::class, 'index']);
ln 41: Route::post('broadcasting/auth', static function (Request $request) {
ln 55: Route::get('data', [DataController::class, 'index']);
ln 57: Route::put('settings', [SettingController::class, 'update']);
ln 62: Route::post('{song}/scrobble', [ScrobbleController::class, 'store']);
ln 63: Route::post('songs/{song}/scrobble', [ScrobbleController::class, 'store']);
ln 64: Route::put('songs', [SongController::class, 'update']);
ln 66: Route::post('upload', UploadController::class);
ln 69: Route::post('interaction/play', [PlayCountController::class, 'store']);
ln 70: Route::post('interaction/like', [LikeController::class, 'store']);
ln 71: Route::post('interaction/batch/like', [BatchLikeController::class, 'store']);
ln 72: Route::post('interaction/batch/unlike', [BatchLikeController::class, 'destroy']);
ln 73: Route::get('interaction/recently-played/{count?}', [RecentlyPlayedController::class, 'index'])->where([
ln 80: Route::put('playlist/{playlist}/sync', [PlaylistSongController::class, 'update']); // @deprecated
ln 81: Route::put('playlist/{playlist}/songs', [PlaylistSongController::class, 'update']);
ln 82: Route::get('playlist/{playlist}/songs', [PlaylistSongController::class, 'index']);
ln 86: Route::get('me', [ProfileController::class, 'show']);
ln 87: Route::put('me', [ProfileController::class, 'update']);
ln 90: Route::post('lastfm/session-key', [LastfmController::class, 'setSessionKey']);
ln 91: Route::delete('lastfm/disconnect', [LastfmController::class, 'disconnect'])->name('lastfm.disconnect');
ln 95: Route::get('youtube/search/song/{song}', [YouTubeController::class, 'searchVideosRelatedToSong']);
ln 99: Route::get('album/{album}/info', [AlbumInformationController::class, 'show']);
ln 100: Route::get('artist/{artist}/info', [ArtistInformationController::class, 'show']);
ln 101: Route::get('song/{song}/info', [SongInformationController::class, 'show']);
ln 104: Route::put('album/{album}/cover', [AlbumCoverController::class, 'update']);
ln 105: Route::put('artist/{artist}/image', [ArtistImageController::class, 'update']);
ln 106: Route::get('album/{album}/thumbnail', [AlbumThumbnailController::class, 'show']);
ln 110: Route::get('/', [ExcerptSearchController::class, 'index']);
ln 111: Route::get('songs', [SongSearchController::class, 'index']);
ln 117: Route::post('song', [S3SongController::class, 'put'])->name('s3.song.put'); // we follow AWS's convention here.
ln 118: Route::delete('song', [S3SongController::class, 'remove'])->name('s3.song.remove'); // and here.

ln 15: Route::get('/', static fn () => view('index'));
ln 17: Route::get('remote', static fn () => view('remote'));
ln 21: Route::get('connect', [LastfmController::class, 'connect'])->name('lastfm.connect');
ln 22: Route::get('callback', [LastfmController::class, 'callback'])->name('lastfm.callback');
ln 26: Route::get('itunes/song/{album}', [ITunesController::class, 'viewSong'])->name('iTunes.viewSong');
ln 31: Route::get('play/{song}/{transcode?}/{bitrate?}', [PlayController::class, 'show'])->name('song.play');
ln 34: Route::get('songs', [SongDownloadController::class, 'show']);
ln 35: Route::get('album/{album}', [AlbumDownloadController::class, 'show']);
ln 36: Route::get('artist/{artist}', [ArtistDownloadController::class, 'show']);
ln 37: Route::get('playlist/{playlist}', [PlaylistDownloadController::class, 'show']);
ln 38: Route::get('favorites', [FavoritesDownloadController::class, 'show']);

ln 33: Route::get('overview', [OverviewController::class, 'index']);
ln 34: Route::get('data', [DataController::class, 'index']);
ln 42: Route::get('albums/{album}/information', FetchAlbumInformationController::class);
ln 43: Route::get('artists/{artist}/information', FetchArtistInformationController::class);
ln 47: Route::delete(
ln 54: Route::delete('playlists/{playlist}/songs', [PlaylistSongController::class, 'destroy']);
ln 57: Route::get('songs/recently-played', [RecentlyPlayedSongController::class, 'index']);
ln 58: Route::get('songs/favorite', [FavoriteSongController::class, 'index']);
ln 59: Route::delete('songs', DeleteSongsController::class);
ln 61: Route::get('genres/{genre}/songs', GenreSongController::class)->where('genre', '.*');
ln 62: Route::get('genres/{genre}/songs/random', FetchRandomSongsInGenreController::class)->where('genre', '.*');
ln 67: Route::get('search', ExcerptSearchController::class);
ln 68: Route::get('search/songs', SongSearchController::class);
ln 70: Route::get('queue/fetch', [QueueController::class, 'fetchSongs']);
ln 72: Route::post('interaction/play', [PlayCountController::class, 'store']);
```

Routes from koel streaming server

Demo

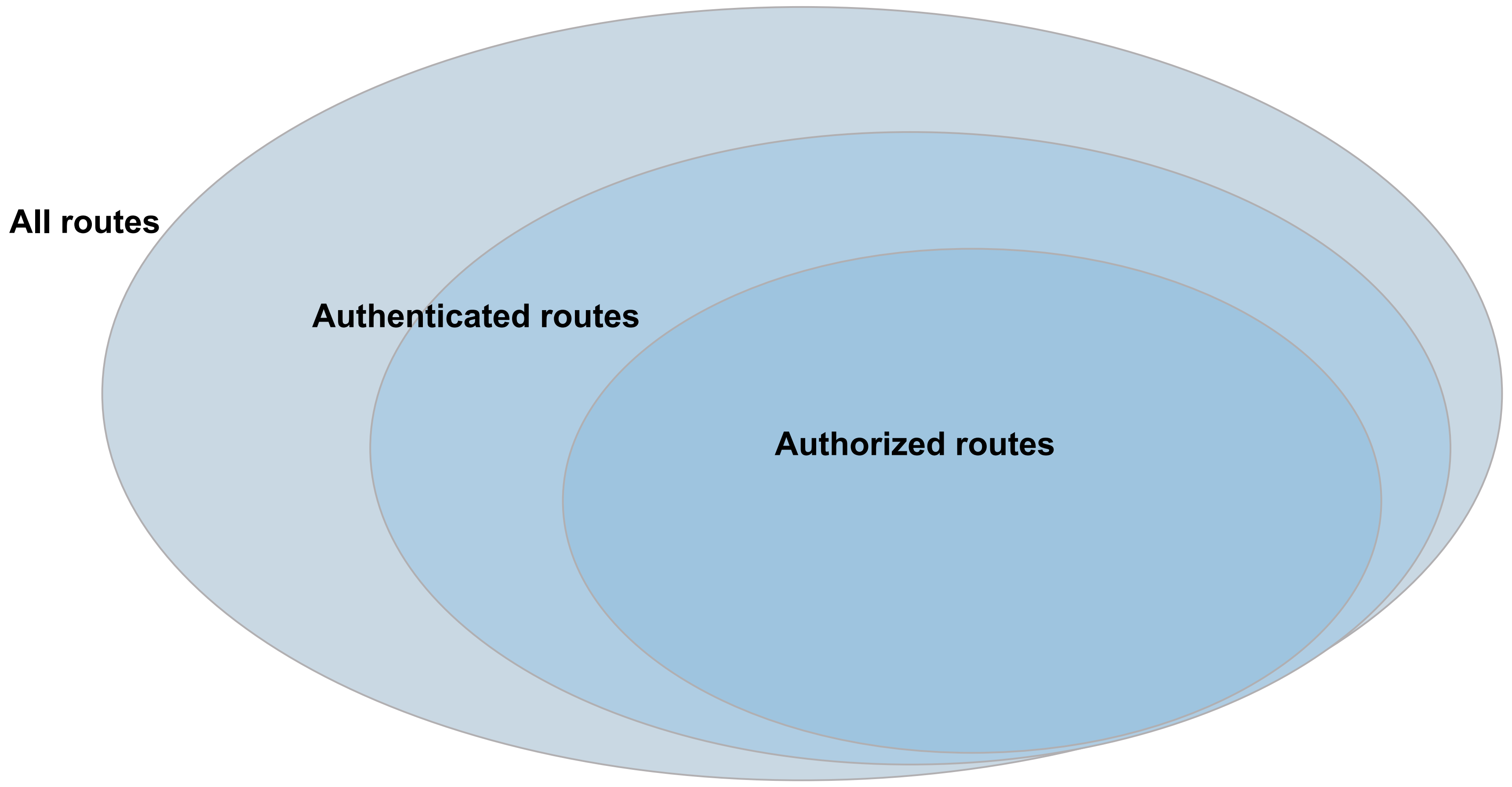
When should you use route-detect?

- Searching web application code for route authn and authz issues
- Precursor to deeper code security assessments
    - Understand the **attack surface** of the application
    - Understand the web application architecture and available functionality
    - Find **entry points for exploitation**
- Basic software project documentation
    - **Onboard new developers** with the project layout and available functionality

How does it work?

- Provides an **easily installable, open source, CLI application**
- Builds on Semgrep for code analysis and CLI findings
- Builds on D3.js and local HTML files for tidy-tree visualization
- Uses Python for "glue" code and application interface
- Heavy use of **automated testing to prevent false positives**
  - E.g. create test code like real findings, ensure route-detect finds it
- Cross-reference **basic regex code search to minimize false negatives**
  - E.g. search for "route", "path", etc, and improve route-detect rules

Finding routes and authn/authz



Example: Python Flask route authn

```
rules:
- id: flask-route-unauthenticated
  patterns:
    - pattern: |
        @$APP.route($PATH, ...)
        def $FUNC(...):
            ...
    - pattern-not: |
        @$APP.route($PATH, ...)
        @login_required(...)
        def $FUNC(...):
            ...
  message: Found unauthenticated Flask route
  languages: [python]
  severity: INFO
```

```
rules:
- id: flask-route-authenticated
  pattern: |
    @$APP.route($PATH, ...)
    @login_required(...)
    def $FUNC(...):
        ...
  message: Found authenticated Flask route
  languages: [python]
  severity: INFO
```

Example: Java Spring route authz

```
rules:
- id: spring-route-unauthorized
  patterns:
    - pattern: |
        @PostMapping(...)
        $RETURNTYPE $FUNC(...) { ... }
    - pattern-not: |
        @PostMapping(...)
        @RolesAllowed($AUTHZ)
        $RETURNTYPE $FUNC(...) { ... }
  message: Found unauthorized Spring route
  languages: [java]
  severity: INFO
```

```
rules:
- id: spring-route-authorized
  pattern: |
    @PostMapping(...)
    @RolesAllowed($AUTHZ)
    $RETURNTYPE $FUNC(...) { ... }
  message: Found authorized Spring route
  languages: [java]
  severity: INFO
```

Limitations

- Convention over configuration, i.e. **implicit** code relationships
    - Ruby Rails
- **Interprocedural** authn/authz information
    - Route information is logically far from authn/authz information
    - Python Django, Ruby Rails
- Middleware-based authn/authz information
    - Combinatorial explosion in number of ways authn/authz may be specified
    - Golang Gin, Golang Gorilla

What's next?

- Expand horizontally
  - Support more languages, frameworks, and authn/authz libraries
- Expand vertically
  - Deeper analysis, reduce false positives, address limitations, etc.
- Anomaly detection
  - What if all routes in a source code file are authn except one?
  - What if all routes in a directory have the same authz role except one?

Information Classification: General

black hat ARSENAL 2023

Conclusion

Questions, comments, rants?

https://github.com/mschwager/route-detect/issues