Securing GraphQL APIs with STACKHAWK

GraphQL Galaxy Conference

$ whoami

Scott Gerlach

- CSO/Co-Founder StackHawk, Inc
- CISO @SendGrid - 3 years
- Sr. Security Arch @GoDaddy - 9 years
- Husband, Dad, Brewer, Golfer, tinkerer
- @sgerlach
- linkedin.com/in/scott-gerlach-kaakaww

Advantages of GraphQL v. Rest API

Data Aggregation

Single Endpoint / Many Calls

Validation and Type Checking

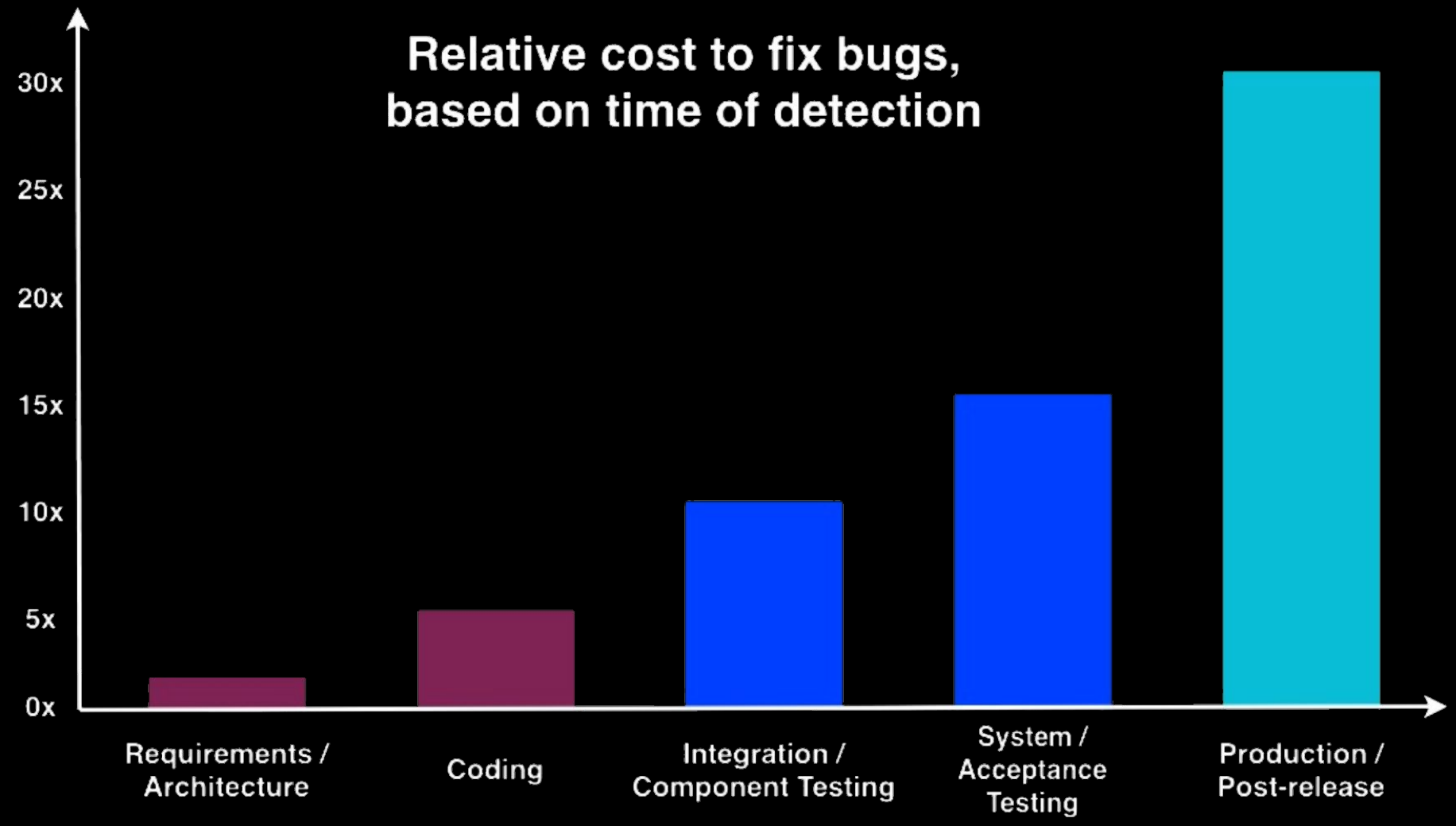
Gateway and Micro-Services

What GraphQL Doesn’t Do

Write Code for You

Magically Secure Code You Write

The Cost of Fixing Security Bugs



https://deepsource.io/blog/exponential-cost-of-fixing-bugs/

Umm, isn’t that what testing is for

YES!

That is exactly why you write functional, unit and integration tests!

Failing tests you can check before you commit code, and having those tests back you up in CI/CD is the cheap way!

But how do you test for security bugs?

How can you use this Awesome New Tech

And avoid the trappings of Security Bugs

STACKHAWK

Dynamic Application Security Testing (DAST)

Why DAST

- **No Language Dependency:** Works on the running app without regard to what language in which it's written
- **Context of how the application works:** Test the running application with bad inputs to see how it behaves
- **Lower False Positive Rates:** If DAST finds a security bug in the application, it almost certainly exists.

Find/Triage/Fix Security Bugs

StackHawk allows you to:

Find Security Bugs in your application

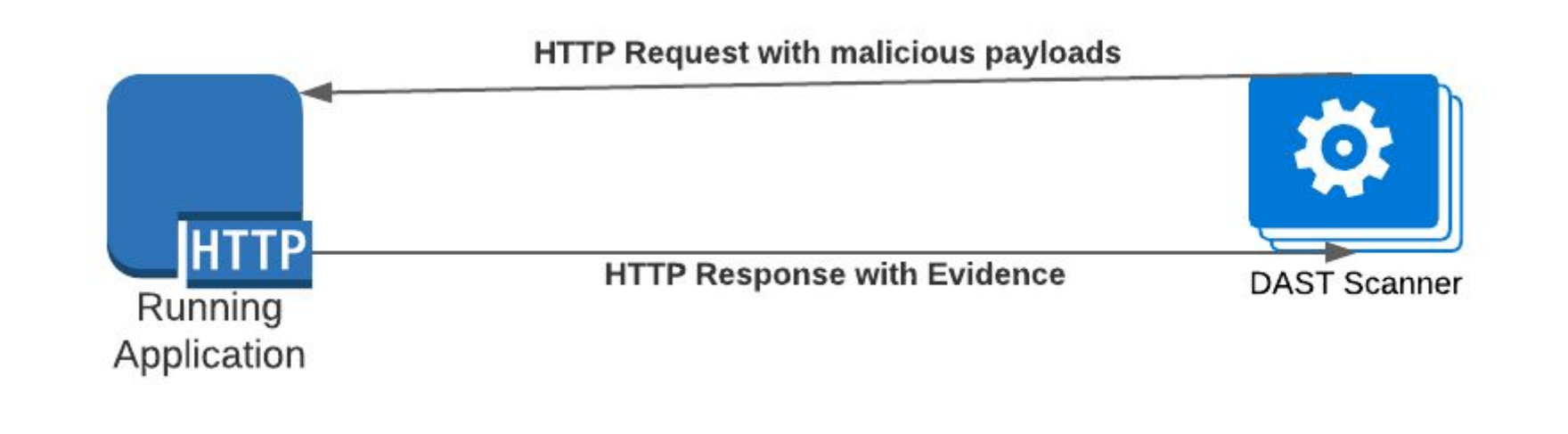
Triage those findings and be able to reproduce them

Manage those findings and have the scanner remember settings between scans

Bring developer attention to “New” findings in the app.

```
Results for scan id 72fe2e44-f1b5-4f8a-b542-539fcf415011
Scan results for http://localhost:3000
------------------------------------------------------------
Criticality: New/Triaged
   High: 0/2    Medium: 22/0    Low: 49/0
------------------------------------------------------------
1) Remote OS Command Injection
   Risk: High
   Cheatsheet: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/OS_Comma
   Paths (1):
     [False Positive] POST /graphql
2) SQL Injection
   Risk: High
   Cheatsheet: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Inje
   Paths (1):
     [Assigned] POST /graphql
3) Cross-Domain Misconfiguration
   Risk: Medium
   References: http://www.hpenterprisesecurity.com/vulncat/en/vulncat/vb/html5_overly_pe
   Paths (19):
     [New] POST /graphql
     [New] POST /graphql
     [New] POST /graphql
     [New] POST /graphql
     [New] POST /graphql
     ... 14 more in details
4) HTTP Only Site
   Risk: Medium
```

How it Works



Finds things like:

SQL and OS Injection

Cross-Site Scripting Problems

Cookie Safety

Content Security Policy Problems

and other Web Application Security issues

StackHawk Config

Simple YAML Configuration

Highlighted lines is all it takes

Stored with the project so CI can use it

Highly repeatable results

```yaml
app:
  applicationId: a919a264-de99-4087-b3b4-5939104caf9f
  env: Development
  host: http://localhost:3000

  graphqlConf:
    enabled: true
    schemaPath: /graphql
    requestMethod: POST
    uriMaxLength: "4000"
    maxDepth: "7"
    introspection:
      requestsPerCycle: "10"
      requestDelay: "1000"
    batchQueries: false
    operation: ALL
    filePath: ""
```

https://docs.stackhawk.com/hawkscan/configuration/

StackHawk in CI/CD

Integrations

- Atlassian Bamboo
- AWS Code Services
- Azure Pipelines
- CircleCI
- Concourse CI
- GitHub Actions
- GitLab
- Jenkins
- Travis CI

StackHawk in CI/CD - Today GitHub Actions

Integrations

Atlassian Bamboo

AWS Code Services

Azure Pipelines

CircleCI

Concourse CI

GitHub Actions

GitLab

Jenkins

Travis CI

Example - Vulnerable Code

```
post.js
user.js
seeders
static
assets
1
secret
app.ts
LICENSE
package.json
README.md
run.sh
tsconfig.json
yarn.lock
.dockerignore
.gitignore
docker-compose.yml
docker-entrypoint.sh
Dockerfile
README.md
stackhawk.yml
```

```
13          },
14          stderr: {
15              type: GraphQLString
16          }
17      }
18  })
19
20  export var SecretMutation: GraphQLFieldConfig<any,any,any> = {
21      type: CommandOutputType,
22      args: {
23          command: {
24              type: GraphQLString
25          }
26      },
27      resolve: async (_root: any, args: any, _info: any) => {
28          let command = args.command;
29          let results = await exec(command);
30          return results;
31      }
32  }
33
```

Example - GitHub Actions

```yaml
name: HawkScan
on:
  push:
  pull_request:
jobs:
  hawkscan:
    name: HawkScan
    runs-on: ubuntu-latest
    steps:
      - name: Clone repo
        uses: actions/checkout@v2
      - name: Docker Build
        run: SERVER_PORT=3000 docker-compose build
      - name: Run GraphAPI
        run: SERVER_PORT=3000 docker-compose up -d
      - name: Run HawkScan
        env:
          API_KEY: ${{secrets.HAWK_API_KEY}}
        run: >
          docker run -v $(pwd):/hawk -t --network vuln-graphql_default
          -e API_KEY="${API_KEY}"
          stackhawk/hawkscan stackhawk.yml stackhawk-github.yml
```

Example - The Scan Results

```
  {"query":"query  user($id:ID) { user(id:$id) { id username firstName lastName  } }","variables":{"id":"1"}}
  {"query":"query  post($id:ID) { post(id:$id) { id title content public  } }","variables":{"id":"1"}}
  {"query":"query  search($query:String) { search(query:$query) { id title content public  } }","variables":{"query":"KaaaKaww"}}
  ... 1 additional URLs
Spider complete
Spider Crawled 4 URLs:
http://localhost:3000
  http://localhost:3000/graphql
  http://localhost:3000/robots.txt
  http://localhost:3000/sitemap.xml
Active Scan {'policy': 'c45e2369-6dd4-4b0f-bbd2-b9cc0c3cd228', 'target': 'http://localhost:3000'} complete
Hawk Scanned 5 URLs:
https://localhost
  http://localhost:3000
  http://localhost:3000/graphql
  http://localhost:3000/robots.txt
  http://localhost:3000/sitemap.xml
Results for scan id 5d073bff-7d2d-4906-b90a-d49c2662146d
Scan results for http://localhost:3000
-----------------------------------------------------------
Criticality: New/Triaged
   High: 1/1    Medium: 22/0    Low: 46/0
-----------------------------------------------------------
1) Remote OS Command Injection
   Risk: High
   Cheatsheet: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/OS_Command_Injection_Defense_Cheat_Sheet.md
   Paths (1):
     [New] POST /graphql
2) SQL Injection
   Risk: High
   Cheatsheet: https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.md
   Paths (1):
     [Assigned] POST /graphql
```

Example - Now with a GraphQL-like Interface

Cheatsheet
https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/OS_Command_Injection_Defense_Cheat_Sheet.md

Actions

Display All 1 - 1 of 1

Path	Status	Method
/graphql	New	POST

Request Headers

```
POST http://localhost:3000/graphql HTTP/1.1
User-Agent: HawkScan/2.0; StackHawk, Inc. (https://www.stackhawk.com)
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 184
Accept: application/json
Content-Type: application/json
Host: localhost:3000
```

Cookie Parameters

```
connect.sid=s%3AzsFmi0SKelNa8KaE2KCqXRHzTVYKL8zp.Y3Dzzup21htdWZivRlg7cUkGuYO
```

Query

```
mutation superSecretPrivateMutation($command: String) {
  superSecretPrivateMutation(command: $command) {
    stdout
    stderr
  }
}
```

Variables

```
{
  "command": "KaaaKaww&cat /etc/passwd&"
}
```

Example - Let’s Fix It!

```
20  export var SecretMutation: GraphQLFieldConfig<any,any,any> = {
21      type: CommandOutputType,
22      args: {
23          command: {
24              type: GraphQLString
25          }
26      },
27      resolve: async (_root: any, args: any, _info: any) => {
28          let command = '';
29          switch(args.command){
30            case 'disk':
31              command = 'df -h';
32              break;
33            case 'log':
34              command = 'tail -15 /var/log/dpkg.log';
35              break;
36            default:
37              command = 'echo sorry, you have to pick a command';
38          }
39          let results = await exec(command);
40          return results;
41      }
42  }
```

Example - Good To Go!

Scans
Applications
Integrations
Invite Users

All Applications | All Environments | Display 10 | 1 - 10 of 175

Scan	Paths	High (Triaged)	Med (Triaged)	Low (Triaged)	Completed
GraphyGrapherson Development	5	0 — (1)	22 ▲ (0)	46 ▲ (0)	Dec 07, 2020 at 15:16 MST
GraphyGrapherson Development	5	1 ▲ (1)	22 ▲ (0)	46 ▲ (0)	Dec 07, 2020 at 14:40 MST
GraphyGrapherson Development	5	1 ▲ (1)	22 ▲ (0)	48 ▲ (0)	Dec 07, 2020 at 14:29 MST

Recap

- Easy to Get Started - A few YAML lines and up and running
- Works in almost every CI system. Check on EVERY PR/MR
- Intuitive triage screens - Multiple ways to address an issue
- Scanner remembers - Don’t re-address findings every time
- Gets out of you way - If there’s no new issues, everything proceeds as normal
- Confidence - You are not making security mistakes in the code you write

Useful Links

- graphQL Example - https://github.com/sgerlach/vuln-graphql/
- StackHawk Documentation - https://docs.stackhawk.com
- Signup for a Free StackHawk Developer Account @ https://www.stackhawk.com/free-plan/
- Twitter - @stackhawk, @sgerlach

Thanks!