# Introduction to CodeQL (formerly known as Semmle QL)

👋 by Thanat Sirithawornsant, Senior Security Engineer, Citrix 🥳

# What is CodeQL?

- CodeQL is advertised as a variant analysis tool.
- It is a query language which has its own syntax.
- It can be used to find a piece of code in a code base, not necessarily bugs or defects.
- It can be used as part of you CI/CD pipelines to detect bad smells.
- Free for research and open source

```
1   import java
2
3   from Variable v
4   where v.toString().regexpMatch(".*pass.*")
5   select v
```

## I am comfortable with grep, so I could just stick with that 🤔

```
vulnado/src on  master [?]
[I] ➜ grep -rn pass .
./main/java/com/scalesec/vulnado/Postgres.java:38:            stmt.executeUpdate("CREATE TABLE
 IF NOT EXISTS users(user_id VARCHAR (36) PRIMARY KEY, username VARCHAR (50) UNIQUE NOT NULL,
password VARCHAR (50) NOT NULL, created_on TIMESTAMP NOT NULL, last_login TIMESTAMP)");
./main/java/com/scalesec/vulnado/Postgres.java:90:    private static void insertUser(String us
ername, String password) {
./main/java/com/scalesec/vulnado/Postgres.java:91:       String sql = "INSERT INTO users (user
_id, username, password, created_on) VALUES (?, ?, ?, current_timestamp)";
./main/java/com/scalesec/vulnado/Postgres.java:97:          pStatement.setString(3, md5(passwo
rd));
./main/java/com/scalesec/vulnado/User.java:53:        String password = rs.getString("password
");
./main/java/com/scalesec/vulnado/User.java:54:        user = new User(user_id, username, passw
ord);
./main/java/com/scalesec/vulnado/LoginController.java:21:    if (Postgres.md5(input.password).
equals(user.hashedPassword)) {
./main/java/com/scalesec/vulnado/LoginController.java:31:  public String password;
```

## Or even ripgrep 💭

```
vulnado/src on  master [?]
[I] → rg pass
main/java/com/scalesec/vulnado/LoginController.java
21:    if (Postgres.md5(input.password).equals(user.hashedPassword)) {
31:  public String password;

main/java/com/scalesec/vulnado/User.java
53:        String password = rs.getString("password");
54:        user = new User(user_id, username, password);

main/java/com/scalesec/vulnado/Postgres.java
38:            stmt.executeUpdate("CREATE TABLE IF NOT EXISTS users(user_id VARCHAR (36) PRIMA
RY KEY, username VARCHAR (50) UNIQUE NOT NULL, password VARCHAR (50) NOT NULL, created_on TIME
STAMP NOT NULL, last_login TIMESTAMP)");
90:    private static void insertUser(String username, String password) {
91:       String sql = "INSERT INTO users (user_id, username, password, created_on) VALUES (?,
 ?, ?, current_timestamp)";
97:          pStatement.setString(3, md5(password));
```

# well, how about unsafe deserialisation vulnerabilities? 😟

```
import java
import semmle.code.java.dataflow.FlowSources
import UnsafeDeserialization

class UnsafeDeserializationConfig extends TaintTracking::Configuration {
  UnsafeDeserializationConfig() { this = "UnsafeDeserializationConfig" }

  override predicate isSource(DataFlow::Node source) { source instanceof RemoteUserInput }

  override predicate isSink(DataFlow::Node sink) { sink instanceof UnsafeDeserializationSink }
}

from UnsafeDeserializationSink sink, RemoteUserInput source, UnsafeDeserializationConfig conf
where conf.hasFlow(source, sink)
select sink.getMethodAccess(), "Unsafe deserialization of $@.", source, "user input"
```

# Example Vulnerable Java Application 😑

https://github.com/ScaleSec/vulnado

- written in Java based on spring framework
- MVC model
- with some writeup

# Vulnado - Intentionally Vulnerable Java Application

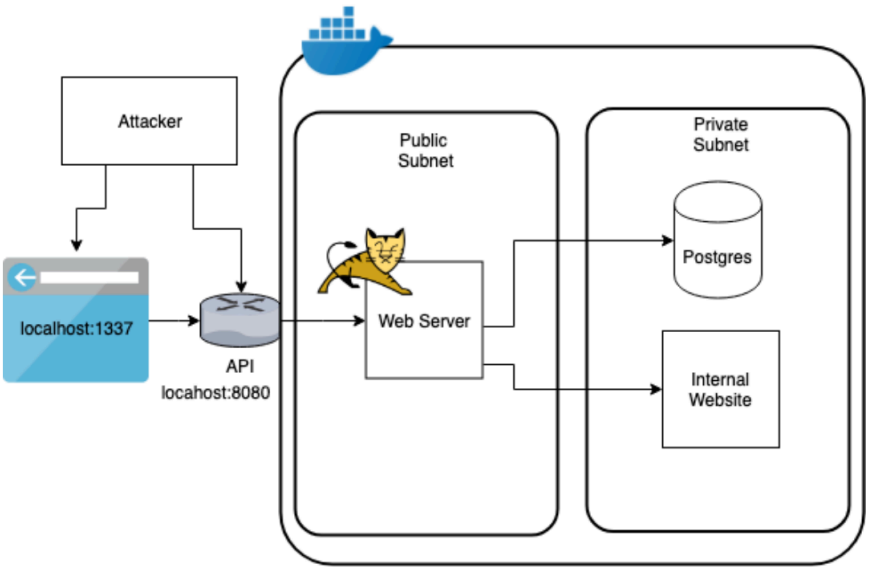
This application and exercises will take you through some of the OWASP top 10 Vulnerabilities and how to prevent them.

## Up and running

1. Install Docker for MacOS or Windows. You'll need to create a Docker account if you don't already have one.
2. `git clone git://github.com/ScaleSec/vulnado`
3. `cd vulnado`
4. `docker-compose up`
5. Open a browser and navigate to the client to make sure it's working: http://localhost:1337
6. Then back in your terminal verify you have connection to your API server: `nc -vz localhost 8080`

## Architecture

The docker network created by `docker-compose` maps pretty well to a multi-tier architecture where a web server is publicly available and there are other network resources like a database and internal site that are not publicly available.




## Exercises

- SQL Injection
- XSS - Cross Site Scripting
- SSRF - Server Side Request Forgery
- RCE - Remote Code Execution & Reverse Shell

# Getting Started 🧐

CodeQL offers 2 ways to use their tool

1. CodeQL CLI for engineers
2. LGTM.com -> CodeQL Platform

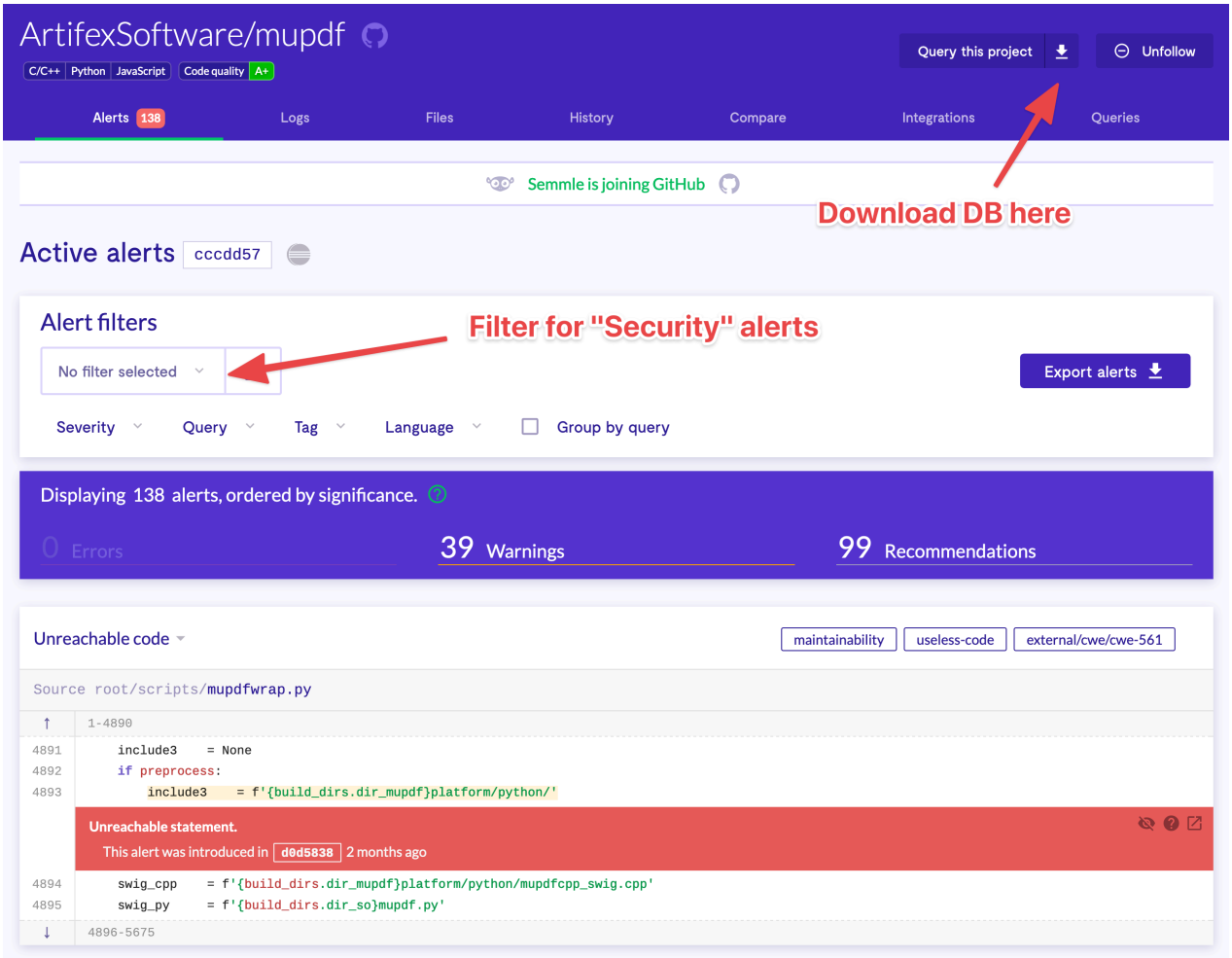
# CodeQL CLI 🥂

Instructions: https://help.semmle.com/codeql/codeql-cli/procedures/get-started.html

- can compile a database for CodeQL locally
    - allowing engineers to do experiments and research locally
- can easily run batches on queries with ease

# LGTM.com 🖐

- can be used straight away on their platform for open source projects
- fancy UI
- a bit slower than querying locally, probably because of shared environment or some sort of restrictions
- once the project has been imported, the database can be exported




# Create a database 🛠️🖍️

```
vulnado on  master [?]
[I] → codeql database create qldb --language=java -c "mvn clean install"
Initializing database at /Users/sinderella/clones/vulnado/qldb.
Running command [mvn, clean, install] in /Users/sinderella/clones/vulnado.
[2020-07-18 13:45:22] [build] [INFO] Scanning for projects...
[2020-07-18 13:45:22] [build] [INFO]
[2020-07-18 13:45:22] [build] [INFO] ------------------------< com.scalesec:vulnado >------------------------
[2020-07-18 13:45:22] [build] [INFO] Building vulnado 0.0.1-SNAPSHOT
[2020-07-18 13:45:22] [build] [INFO] --------------------------------[ jar ]---------------------------------
[2020-07-18 13:45:22] [build] [INFO]
[2020-07-18 13:45:22] [build] [INFO] --- maven-clean-plugin:3.1.0:clean (default-clean) @ vulnado ---
[2020-07-18 13:45:22] [build] [INFO] Deleting /Users/sinderella/clones/vulnado/target
[2020-07-18 13:45:22] [build] [INFO]
[2020-07-18 13:45:22] [build] [INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ vulnado ---
[2020-07-18 13:45:23] [build] [INFO] Using 'UTF-8' encoding to copy filtered resources.
[2020-07-18 13:45:23] [build] [INFO] Copying 1 resource
[2020-07-18 13:45:23] [build] [INFO] Copying 0 resource
[2020-07-18 13:45:23] [build] [INFO]
[2020-07-18 13:45:23] [build] [INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ vulnado ---
[2020-07-18 13:45:23] [build] [INFO] Changes detected - recompiling the module!
[2020-07-18 13:45:23] [build] [INFO] Compiling 10 source files to /Users/sinderella/clones/vulnado/target/classes
```

```
vulnado on  master [?] took 3s
[I] → l qldb
drwxr-xr-x sinderella staff 256 B  Sat Jul 18 13:28:34 2020   ./
drwxr-xr-x sinderella staff 736 B  Sat Jul 18 13:51:26 2020   ../
.rw-r--r-- sinderella staff 104 B  Sun Jul 12 14:35:56 2020   codeql-database.yml
drwxr-xr-x sinderella staff 160 B  Sun Jul 12 14:35:56 2020   db-java/
drwxr-xr-x sinderella staff 192 B  Sat Jul 18 13:28:52 2020   log/
drwxr-xr-x sinderella staff  96 B  Sat Jul 18 13:29:04 2020   results/
.rw------- sinderella staff 9.4 KB Sun Jul 12 14:35:56 2020   src.zip
drwxr-xr-x sinderella staff 160 B  Sun Jul 12 14:35:51 2020   working/
```

# Running the built-in queries 🧰

## Issues ⚠️

| Issue | Type | Location | Line Number |
|---|---|---|---|
| Query built without neutralizing special characters | error | /src/main/java/com/scalesec/vulnado/User.java | 49, 40, 49, 44 |
| Use of a broken or risky cryptographic algorithm | warning | /src/main/java/com/scalesec/vulnado/Postgres.java | 67, 32, 67, 63 |
| Executing a command with a relative path | warning | /src/main/java/com/scalesec/vulnado/Cowsay.java | 11, 28, 11, 33 |

## Run `codeql database analyze` to run queries in a batch

```
vulnado on  master [?]
[I] → codeql database analyze qldb ~/codeql-home/codeql-repo/java/ql/src/codeql-suites/java-security-extended.qls --format=csv --output=analysed_db.csv
Running queries.
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-209/StackTraceExposure.ql.
[1/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-209/StackTraceExposure.ql (707ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-807/TaintedPermissionsCheck.ql.
[2/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-807/TaintedPermissionsCheck.ql (329ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-601/UrlRedirect.ql.
[3/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-601/UrlRedirect.ql (282ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-113/NettyResponseSplitting.ql.
[4/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-113/NettyResponseSplitting.ql (164ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-113/ResponseSplitting.ql.
[5/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-113/ResponseSplitting.ql (275ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-732/ReadingFromWorldWritableFile.ql.
[6/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-732/ReadingFromWorldWritableFile.ql (151ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-335/PredictableSeed.ql.
[7/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-335/PredictableSeed.ql (158ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-614/InsecureCookie.ql.
[8/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-614/InsecureCookie.ql (158ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-089/SqlTainted.ql.
[9/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-089/SqlTainted.ql (237ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-089/SqlUnescaped.ql.
[10/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-089/SqlUnescaped.ql (204ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-090/LdapInjection.ql.
[11/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-090/LdapInjection.ql (229ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-312/CleartextStorageCookie.ql.
[12/41] Compiled /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-312/CleartextStorageCookie.ql (176ms).
Compiling query plan for /Users/sinderella/codeql-home/codeql-repo/java/ql/src/Security/CWE/CWE-134/ExternallyControlledFormatString.ql.
```

# List all the endpoints

- let's observe how the application declare endpoints
  - it should be within controller files
- what are the characteristics or the definitions?
  - find the patterns and confirm your results

```
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import org.springframework.beans.factory.annotation.*;
import org.springframework.boot.autoconfigure.*;
import java.util.List;
import java.io.Serializable

@RestController
@EnableAutoConfiguration
public class CommentsController {
  @Value("${app.secret}")
  private String secret;

  @CrossOrigin(origins = "*")
  @RequestMapping(value = "/comments", method = RequestMethod.GET, produces =
"application/json")
  List<Comment> comments(@RequestHeader(value="x-auth-token") String token) {
    User.assertAuth(secret, token);
    return Comment.fetch_all();
  }
  [...]
}
```

```
import java

from Annotation ann, AnnotationType anntp, Method m, Class c
where
  c.hasChildElement(m) and
  ann = m.getAnAnnotation() and
  anntp = ann.getType() and
  anntp.hasQualifiedName("org.springframework.web.bind.annotation", "RequestMapping")
select c, m, ann.getValue("method"), ann.getValue("value"), ann.getValue("produces")
```

# List all public endpoint

```
  @CrossOrigin(origins = "*")
  @RequestMapping(value = "/comments", method = RequestMethod.GET, produces =
"application/json")
  List<Comment> comments(@RequestHeader(value="x-auth-token") String token) {
    User.assertAuth(secret, token);
    return Comment.fetch_all();
  }

  @CrossOrigin(origins = "*")
  @RequestMapping(value = "/comments", method = RequestMethod.POST, produces =
"application/json", consumes = "application/json")
  Comment createComment(@RequestHeader(value="x-auth-token") String token,
@RequestBody CommentRequest input) {
    return Comment.create(input.username, input.body);
  }
```

```
import java

predicate getEndpoints(Annotation ann, AnnotationType anntp, Method m, Class c) {
  c.hasChildElement(m) and
  ann = m.getAnAnnotation() and
  anntp = ann.getType() and
  anntp.hasQualifiedName("org.springframework.web.bind.annotation", "RequestMapping")
}

predicate getAuthCheckMethod(Method m) {
  m.hasName("assertAuth") and
  exists(Class c | c.hasQualifiedName("com.scalesec.vulnado", "User") and c.contains(m))
}

predicate allowList(Expr expr) { expr.toString().splitAt("\"", 1) in ["/login"] }

from Annotation ann_endpoint, AnnotationType anntp_endpoint, Method m, Method auth_m, Class c
where
  getEndpoints(ann_endpoint, anntp_endpoint, m, c) and
  getAuthCheckMethod(auth_m) and
  not allowList(ann_endpoint.getValue("value")) and
  not m.calls(auth_m)
select c, m, ann_endpoint.getValue("method"), ann_endpoint.getValue("value")
```

# List all the user input

```
  @CrossOrigin(origins = "*")
  @RequestMapping(value = "/comments", method = RequestMethod.POST, produces =
"application/json", consumes = "application/json")
  Comment createComment(@RequestHeader(value="x-auth-token") String token,
@RequestBody CommentRequest input) {
    return Comment.create(input.username, input.body);
  }

  @CrossOrigin(origins = "*")
  @RequestMapping(value = "/comments/{id}", method = RequestMethod.DELETE,
produces = "application/json")
  Boolean deleteComment(@RequestHeader(value="x-auth-token") String token,
@PathVariable("id") String id) {
    return Comment.delete(id);
  }
```

```
import java

predicate getUserInput(Parameter p, Annotation ann, AnnotationType anntp, Method m) {
  m.hasChildElement(p) and
  p.getAnAnnotation() = ann and
  ann.getType() = anntp and
  anntp.getName() in ["RequestHeader", "RequestBody", "RequestParam", "PathVariable"] and
  not anntp.hasName("Nullable")
}

predicate getEndpoints(Annotation ann, AnnotationType anntp, Method m, Class c) {
  c.hasChildElement(m) and
  ann = m.getAnAnnotation() and
  anntp = ann.getType() and
  anntp.hasQualifiedName("org.springframework.web.bind.annotation", "RequestMapping")
}

from
  Annotation ann_endpoint, AnnotationType anntp_endpoint, Annotation ann_user,
  AnnotationType anntp_user, Method m, Class c, Parameter p
where getEndpoints(ann_endpoint, anntp_endpoint, m, c) and getUserInput(p, ann_user, anntp_user, m)
select c, m, ann_endpoint.getValue("method"), ann_endpoint.getValue("value"),
  ann_endpoint.getValue("produces"), p
```

# User Input Taint Tracking

## we could

- list all the endpoints
  - including public ones, with allow list even
- list all the user input
  - including URL query, HTTP body, URL path

## Now, we would like to see which input ends up in an unsafe method

# I am sold! How do I get started?

## RTFM

https://help.semmle.com/QL/learn-ql/

### References

https://help.semmle.com/QL/ql-handbook/index.html

### Built-in queries

https://help.semmle.com/wiki/display/JAVA/Java+queries
https://github.com/github/codeql/tree/master/java/ql/src/Security/CWE

## CTF challenges with real bugs

https://securitylab.github.com/ctf

## Real bugs advisories

https://blog.semmle.com/

## Or just use the aforementioned example and play with it

# Thank you 2600 staff members and the community 🙏