

## ACCIDENT with CodeQL

Benjamin Smid

*Cedarville University*, bsmid@cedarville.edu

Nathan Johnson

*Cedarville University*, nathanjohnson@cedarville.edu

Christopher Bellanti

*Cedarville University*, cbellanti@cedarville.edu

Caleb Collins

*Cedarville University*, cmcollins@cedarville.edu

Follow this and additional works at: [https://digitalcommons.cedarville.edu/rs\\_symposium](https://digitalcommons.cedarville.edu/rs_symposium)

Smid, Benjamin; Johnson, Nathan; Bellanti, Christopher; and Collins, Caleb, "ACCIDENT with CodeQL" (2023). *Scholars Symposium*. 8.

[https://digitalcommons.cedarville.edu/rs\\_symposium/2023/poster\\_presentations/8](https://digitalcommons.cedarville.edu/rs_symposium/2023/poster_presentations/8)

This Poster is brought to you for free and open access by DigitalCommons@Cedarville, a service of the Centennial Library. It has been accepted for inclusion in Scholars Symposium by an authorized administrator of DigitalCommons@Cedarville. For more information, please contact [digitalcommons@cedarville.edu](mailto:digitalcommons@cedarville.edu).



# ACCIDENT

## Automated Cryptographic Misuse Detection in JavaScript Code



Faculty Advisor: *Dr. Seth Hamman*

Students: *Chris Bellanti, Caleb Collins, Nate Johnson, Ben Smid*

Sponsor: *RIVERSIDE RESEARCH – Dr. Mike Clark, Quinn Hirt*

### Code Analysis

Code analysis is the process of examining source code to identify potential issues and vulnerabilities, helping catch errors early in the development process and improving code quality.



### Static / Dynamic Analysis

Static code analysis examines source code for issues without running it. Dynamic analysis looks at code behavior during execution and observes its interactions with other system components and the outcomes it produces.



### CodeQL

CodeQL is a declarative, object-oriented query language used to find vulnerabilities in code. It is part of the GitHub security suite and allows developers to write custom queries to analyze code and find potential issues.

### Our Work

- ✓ **Password Hashing Algorithms** – Improved coverage for detecting passwords hashed and stored with insufficient computational effort using the CryptoJS library.
- ✓ **Certificate Chain Validation** – Added checks for disabled certificate chain validation.
- ✓ **Unsafe Block Cipher Mode** – Improved granularity of checks to account for unsafe block cipher modes like ECB.

**Password Hashing Algorithms** – Continuing to improve coverage for other libraries.

**Insecure HTTP Parse** – Writing new query to detect use of unsafe HTTP parser.

### Abstract

Cryptography is an important tool in the security of our software systems. However, mistakes are often made by developers who do not implement cryptography correctly in their projects. As JavaScript becomes more popular as a language for full-stack development, vulnerabilities in JavaScript due to misuses of the cryptographic APIs and incorrect practices have increased as well. Our project focuses on the use of GitHub's default code scanning tool, CodeQL. We are contributing to GitHub's CodeQL repository with improvements that broaden the scope of vulnerabilities that its queries detect. Since CodeQL is widely used by developers and companies who use GitHub to host their projects, our contributions will have an immediate and direct impact on a large community of programmers and help them find cryptographic errors in their code bases.

### CodeQL Snippet

```
private class InstantiatedAlgorithm extends DataFlow::CallNode {
    private string algorithmName;

    InstantiatedAlgorithm() {
        // var crypto = require("crypto-js");
        // var hash = crypto.algo.<algorithmName>.create();
        this =
            API::moduleImport("crypto-js")
                .getMember("algo")
                .getMember(algorithmName)
                .getMember("create")
                .getACall() and
        not isStrongPasswordHashingAlgorithm(algorithmName)
    }

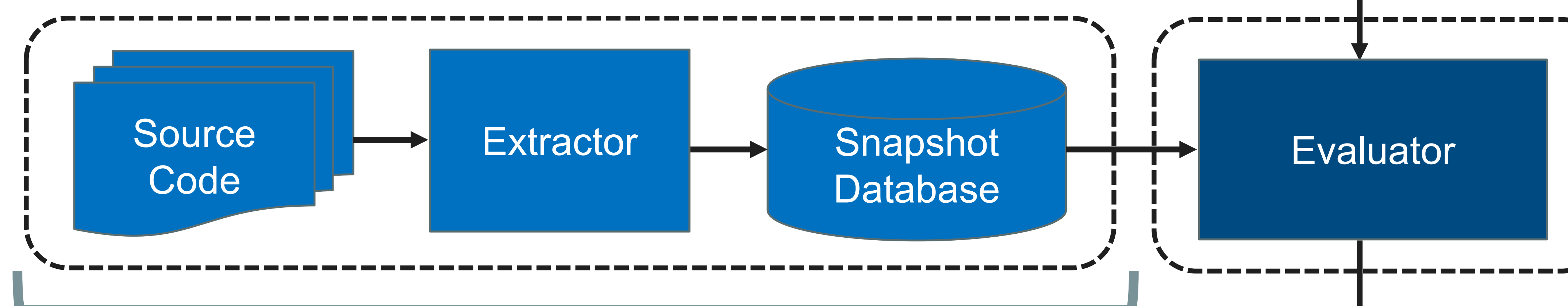
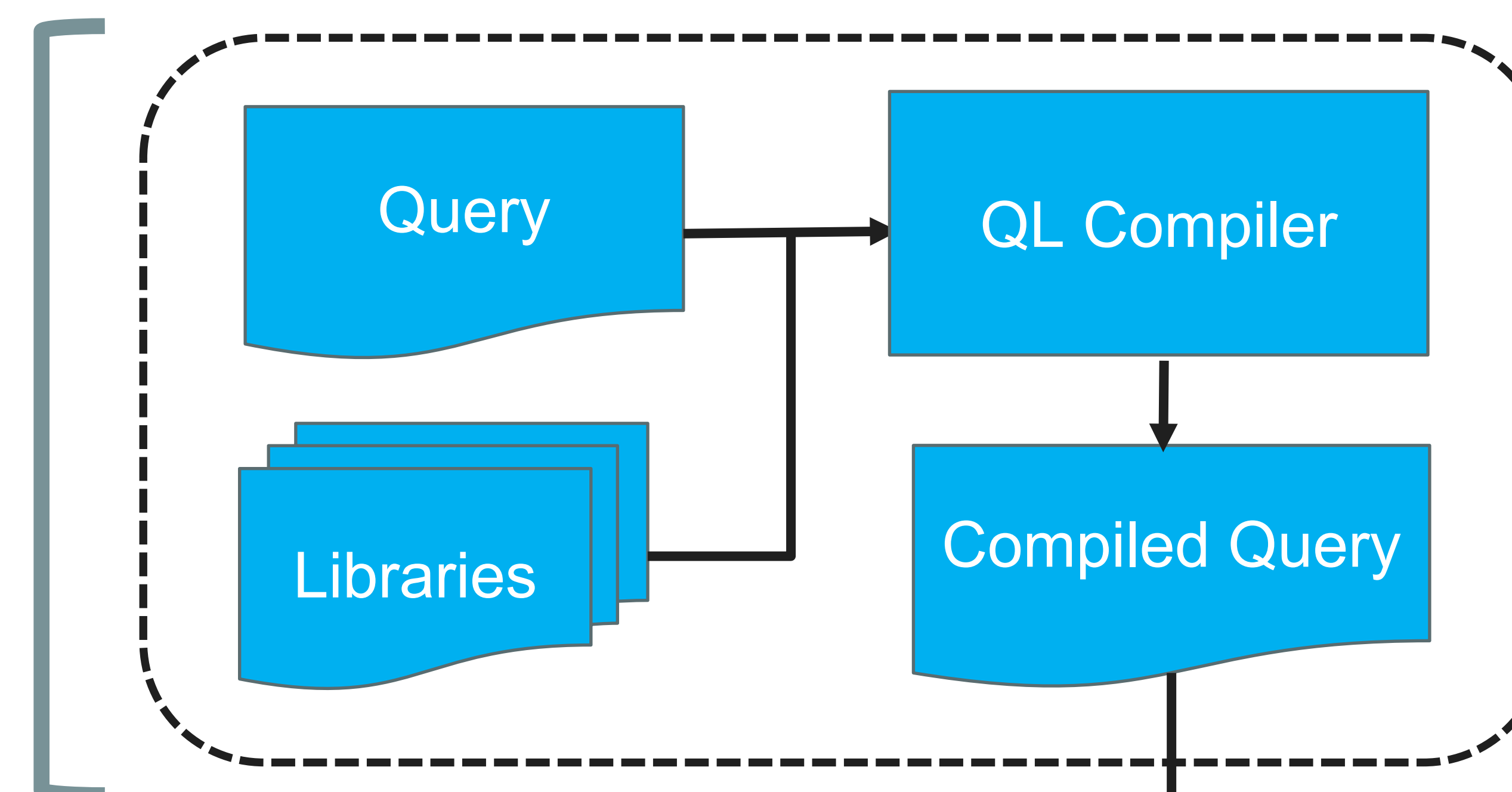
    private API::CallNode getUpdatedApplication(
        API::Node input,
        InstantiatedAlgorithm instantiation) {

        // hash.update(<sensitiveString>);
        result = instantiation.getAMemberCall("update") and
        input = result.getParameter(0)
    }

    // Query
    from Configuration cfg, DataFlow::PathNode source,
        DataFlow::PathNode sink
    where cfg.hasFlowPath(source, sink)
    select sink.getNode(), source, sink,
        "Password from $@ is hashed insecurely.",
        source.getNode(), source.getNode().(Source).describe()
}
```



**Query compilation** translates CodeQL queries into a format that can be executed efficiently by the query engine.



**Database creation** calls for CodeQL to turn the provided source files into a database containing queryable, object-oriented data, including ASTs, DFGs, and CFGs.



### CodeQL Workflow

**Query execution** involves executing the compiled queries against the database and returning variables that meet conditions specified in the queries.

**Result interpretation** highlights errors in the source code and gives explanations and suggestions.