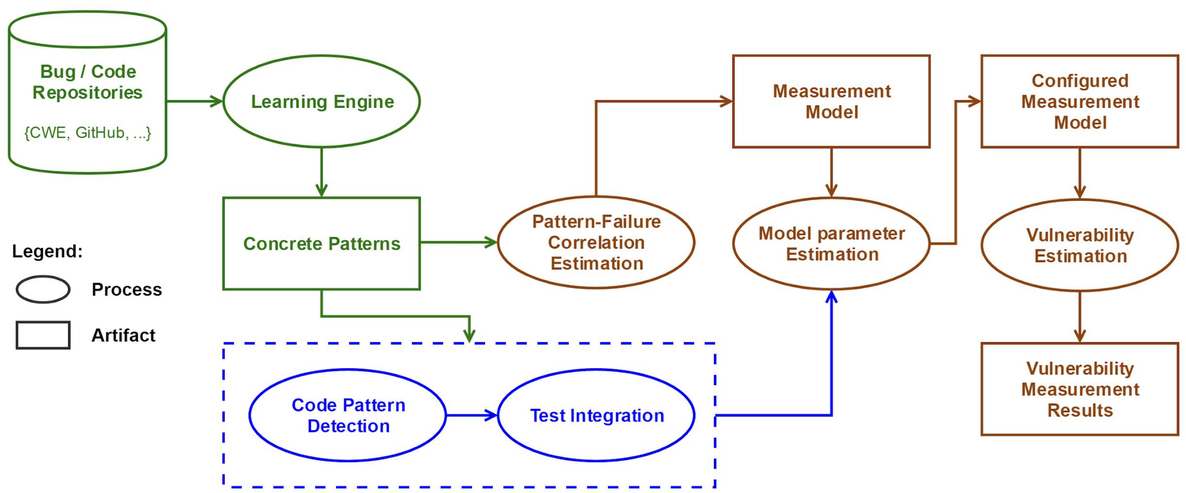
Toward A Code Pattern Based Vulnerability Measurement Model

John Heaps, Rocky Slavin, Xiaoyin Wang

Goal

- We propose a framework to detect access control bugs based on code pattern detection
- Existing bug detection approaches for access control are process-based and suffer from many limitations
- Our empirical analysis-based framework will mine and generate bug patterns, detect those patterns in code, and calculate a vulnerability measure
- Our framework will determine the severity of vulnerability caused by bugs and allow stakeholders to make informed decisions about software



Learning Engine

- Bug definitions and categories are collected from the Common Weakness Enumeration (CWE)
- Code-level bug examples are collected from Github

Improper Authorization Example Bug Code:

```
public ResultSet runEmployeeQuery(Connection conn, String name){
     PreparedStatement stmt = conn.prepareStatement("SELECT * " +
          "FROM employees WHERE name = ?");
     stmt.setString(1, name);
     ResultSet ts = stmt.executeQuery()
     return rs;
}
...

// "canQueryEmployee()" returns true if current user is authorized to
// query the employees table.
if(AuthCheck.canQueryEmployee()){
     ResultSet employeeRecord = runEmployeeQuery(dbConn, employeeName);
}
```

Pattern Detection

- SpotBugs is utilized to perform pattern detection using the repository of patterns from the Learning Engine.
- To estimate vulnerability, bug patterns are linked to abstract quality aspects:
  - **Control Integrity**: how likely the software may incorrectly interact with its users.
  - **Data Integrity**: how likely the software may provide incorrect output.
  - **Data Confidentiality**: how likely the software may release data to entities not authorized to receive it.
  - **Data Availability**: how likely the software may not be able to provide data that should be in storage.

Improper Authorization Example Abstract Qualities:

```
Control Integrity: 0
Data Integrity: 0
Data Confidentiality: 1
Data Availability: 0
```

0 denotes that the quality is unaffected by the bug
1 denotes that the quality is affected by the bug

Measurement Model

- Estimates the vulnerability of a piece of software based on the detected instances of code patterns.
- The following formula generates a vulnerability value in the range [0, 1]; *Detected* is the set of found bug pattern instances, *Risk* denotes the risk value of a given bug *b*, and *R* is a constant which represents the average risk sum per software project.

$$Vulnerability = 1 - \frac{R}{R + \sum_{Detected} Risk(b)}$$

- *Risk* is determined by calculating the *Impact* of a bug and that bug's *Susceptibility*; *Susceptibility* indicates how likely the bug will be triggered at run time and is estimated using testing.

$$Risk = Impact * Susceptibility$$

- *Impact* is calculated by summing the weighted aspects identified in Pattern Detection.

$$Impact = A * Integrity_{Control} + B * Integrity_{Data} + C * Confidentiality_{Data} + D * Availability_{Data}$$

Improper Authorization Example Bug Pattern:

```
public void sawOpcode(int seen) {
     if ("AuthCheck".equals(classConstant) &&
               seen == INVOKESTATIC &&
               "canQueryEmployee".equals(nameConstant) && "()Z".equals(sigConstant)) {
          seenGuardClauseAt = PC;
          return;
     }
     if (seen == IFEQ && (PC >= seenGuardClauseAt + 3 && PC < seenGuardClauseAt + 7)) {
          logBlockStart = branchFallThrough;
          logBlockEnd = branchTarget;
     }
     if (seen == INVOKEVIRTUAL && "runEmployeeQuery".equals(nameConstant)) {
          if (PC < logBlockStart || PC >= logBlockEnd) {
               bugReporter.reportBug(
                    new BugInstance("IMPROPER_AUTHORIZATION", HIGH_PRIORITY)
                    .addClassAndMethod(this).addSourceLine(this));
          }
     }
}
```

Future Work

- Bug patterns are currently produced manually, which is slow and tedious; we plan to investigate machine learning applications to help automatically generate bug patterns.
- SpotBugs is Java specific and requires built software projects to perform detection; we plan to implement other tools to overcome these limitation.
- The current test coverage integration is preliminary, so we plan to further develop it to include automatic test generation and execution of test cases for each software feature.
- We plan to integrate positive code pattern detection to estimate the mitigation of risks in software projects.

We would like to thank the MSI STEM Research & Development Consortium (MSRDC, Award #D01_W911SR14-2-0001-0012), the National Science Foundation (NSF, Award #1736209), and CREST Center for Security and Privacy Enhanced Cloud Computing (C-SPECC, Award #1736209) for their contributions to this project.