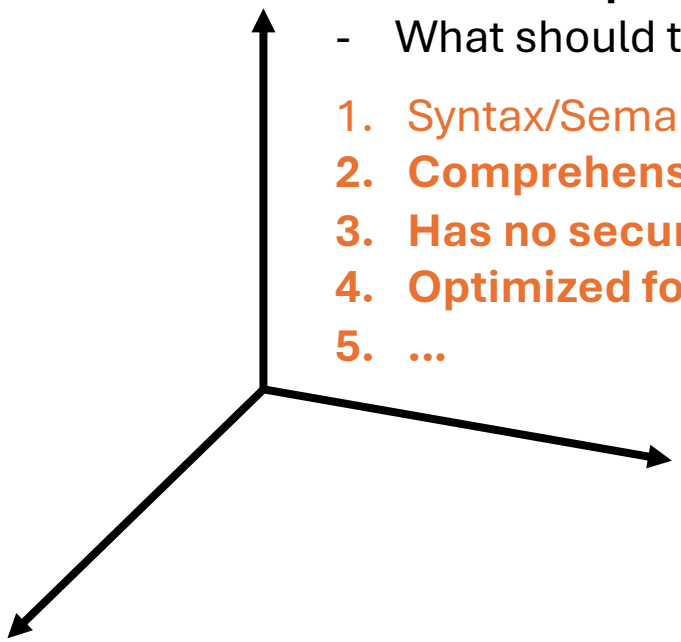
Machine Programming

Lecture 17 – LLM for Software Security

Logistics – Week 10

- Oral Presentations
  - Emails are being sending out; plans established
- Final Projects
  - Final project proposal: 1 page PDF (due on Sunday)
  - Submit on GradeScope
  - Send email to the instructor questions

Module 3: Overview



Behavioral Specification
- What should the program do?

1. Syntax/Semantics/Functional correctness
2. **Comprehensive test coverage**
3. **Has no security flaw**
4. **Optimized for runtime speed**
5. **…**

Structural Specification
- What is the space of the programs?

General Purpose Programming Language
Python / Java / C / Rust / …

Domain Specific Languages
SQL / LEAN / ROCQ / DATALOG / PDDL / …

Synthesis Strategy
- How do we find such a program?

Enumeration: Bottom-up and Top-Down

Programming with Large Language Models
- Next token prediction, prompting, controlled decoding
- Iterative refinement, agentic frameworks and tool use
- Pre-training, fine-tuning, reinforcement learning

Fixed

Software Analysis

Vulnerability Detection, Analysis, and Exploitation

Forms of Software Analysis

Static Analysis

Examine code without concrete execution

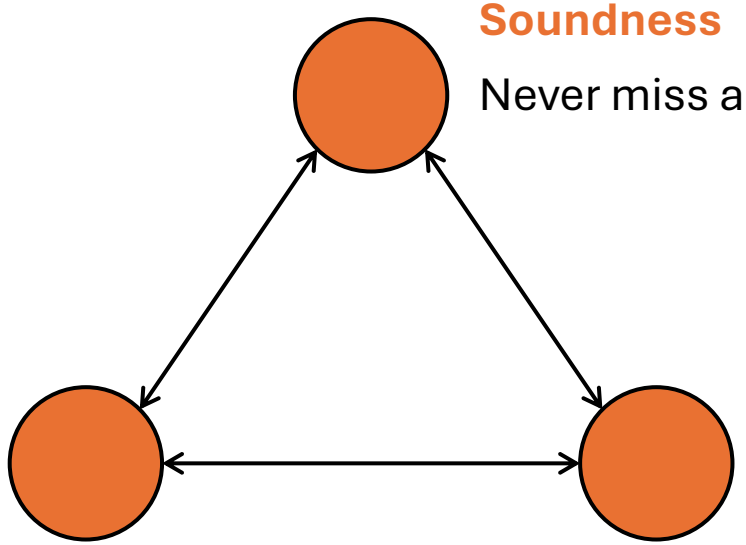
- Dataflow analysis
- Taint analysis
- Abstract interpretation
- Symbolic execution
- …

Dynamic Analysis

Observes program behavior while executing

- Unit testing
- Fuzzing
- Property-based testing
- Penetration testing
- …

Software Analysis: Impossible Triangle



Soundness

Never miss a real bug

Completeness

Never raise a false alarm

Termination

Always finish the analysis within finite time

Software Analysis: Impossible Triangle

Dynamic Analysis:

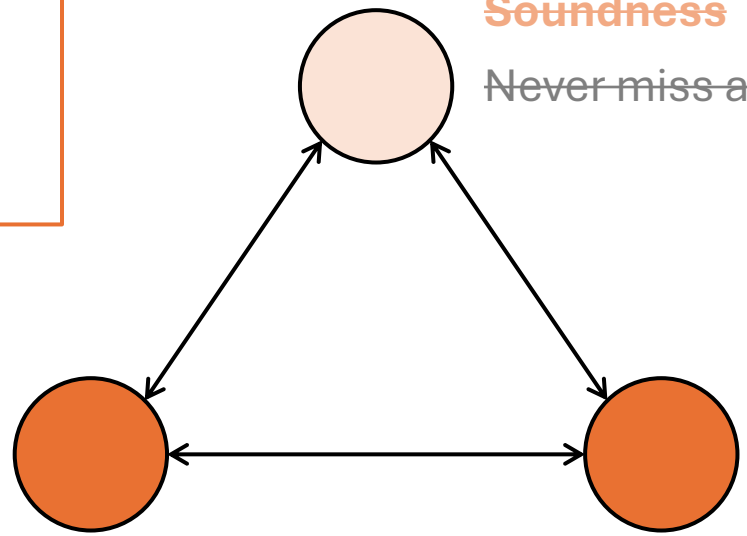
Unit Testing

- (assuming tests are correct)

Fuzzing

- (assuming harness is real)

The unit test cases might not be comprehensive enough.



~~**Soundness**~~

~~Never miss a real bug~~

Completeness

Never raise a false alarm

Termination

Always finish the analysis within finite time

Software Analysis: Impossible Triangle

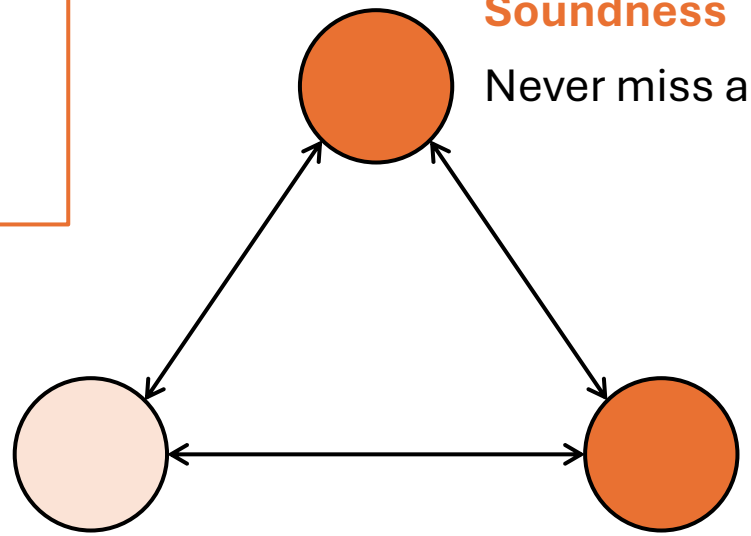
Static Analysis:

Taint Analysis

- (assuming correct specs)

Abstract Interpretation

- (assuming correct domains)



Soundness

Never miss a real bug

~~**Completeness**~~

~~Never raise a false alarm~~

Termination

Always finish the analysis within finite time

May report an alarm for developer confirmation, and that is rejected

Agenda

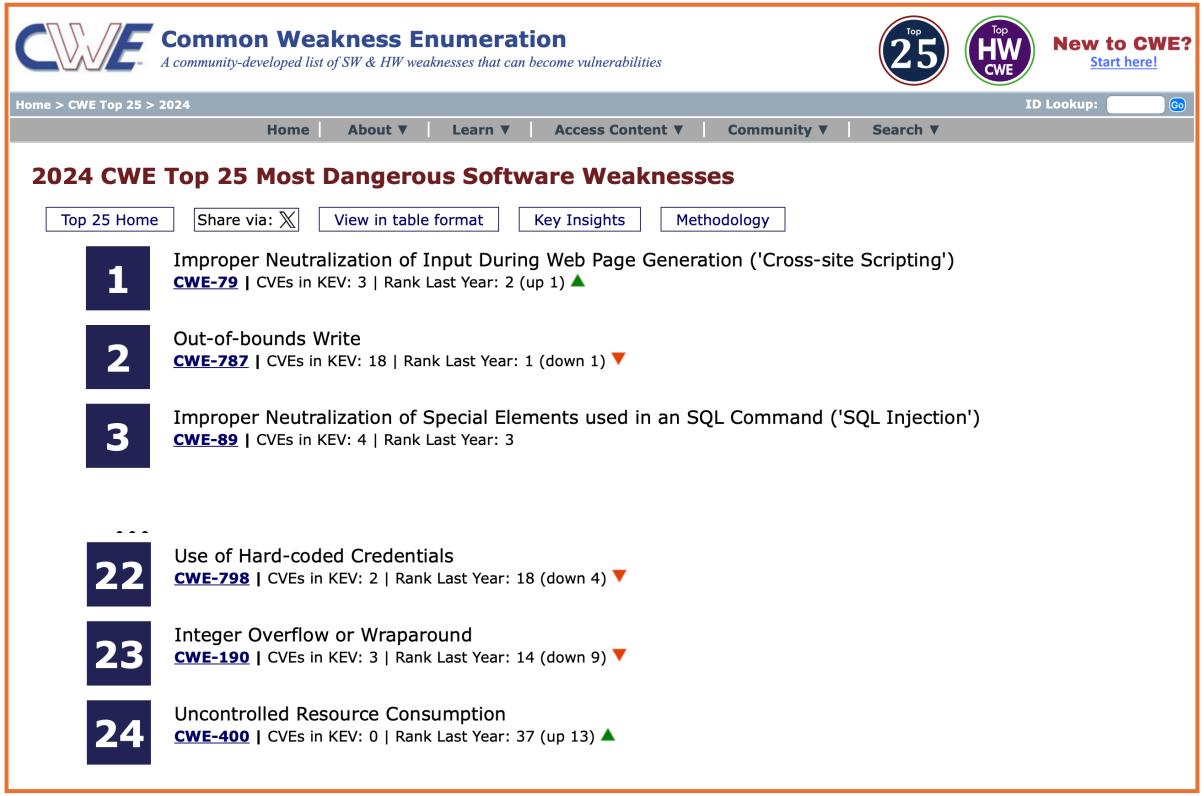
- Vuln Case Study: Path Traversal (CWE-22)
- Vuln Detection Language: CodeQL
- Vuln Detection Language: Semgrep
- Vuln Exploits: Proof-of-Concept

Agenda

- Vuln Case Study: Path Traversal (CWE-22)
- Vuln Detection Language: CodeQL
- Vuln Detection Language: Semgrep
- Vuln Exploits: Proof-of-Concept

Vulnerability Detection (Cont.)

- To be able to detect software vulnerabilities, one first need to understand vulnerabilities:



Path Traversal (CWE-22)

Common Weakness Enumeration
A community-developed list of SW & HW weaknesses that can become vulnerabilities

Top 25 | Top HW CWE | New to CWE? Start here!

Home > CWE List > CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') (4.18) ID Lookup: Go

Home | About ▼ | Learn ▼ | Access Content ▼ | Community ▼ | Search ▼

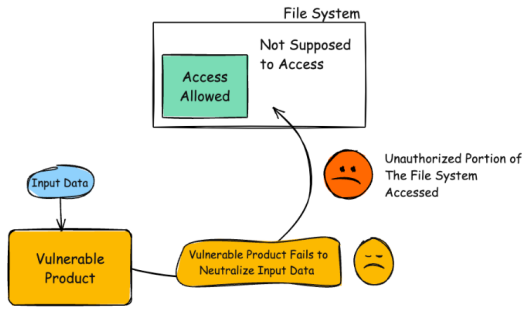
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Weakness ID: 22
Vulnerability Mapping: ALLOWED
Abstraction: Base

View customized information: Conceptual | Operational | Mapping Friendly | **Complete** | Custom

Description

The product uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the product does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.



Extended Description

Many file operations are intended to take place within a restricted directory. By using special elements such as ".." and "/" separators, attackers can escape outside of the restricted location to access files or directories that are elsewhere on the system. One of the most common special elements is the "../" sequence, which in most modern operating systems is interpreted as the parent directory of the current location. This is referred to as relative path traversal. Path traversal also covers the use of absolute pathnames such as "/usr/local/bin" to access unexpected files. This is referred to as absolute path traversal.

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFileServlet extends HttpServlet {
  private static final String BASE_DIR = "/var/www/public-files";

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFileServlet extends HttpServlet {
  private static final String BASE_DIR = "/var/www/public-files";

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

GET /download?path=css/basic.css

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"); //
    if (requested == null || requested.isEmpty()) {

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

GET /download?path=css/basic.css

css/basic.css

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"); //
    if (requested == null || requested.isEmpty()) {

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

GET /download?path=css/basic.css

css/basic.css

/var/www/public-files/css/basic.css

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"); //
    if (requested == null || requested.isEmpty()) {

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

GET /download?path=css/basic.css

css/basic.css

/var/www/public-files/css/basic.css

Reading File

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Malicious data

GET /download?path=../SECRET.txt

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Malicious data

GET /download?path=../SECRET.txt

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"); //
    if (requested == null || requested.isEmpty()) {

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

../SECRET.txt

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Malicious data

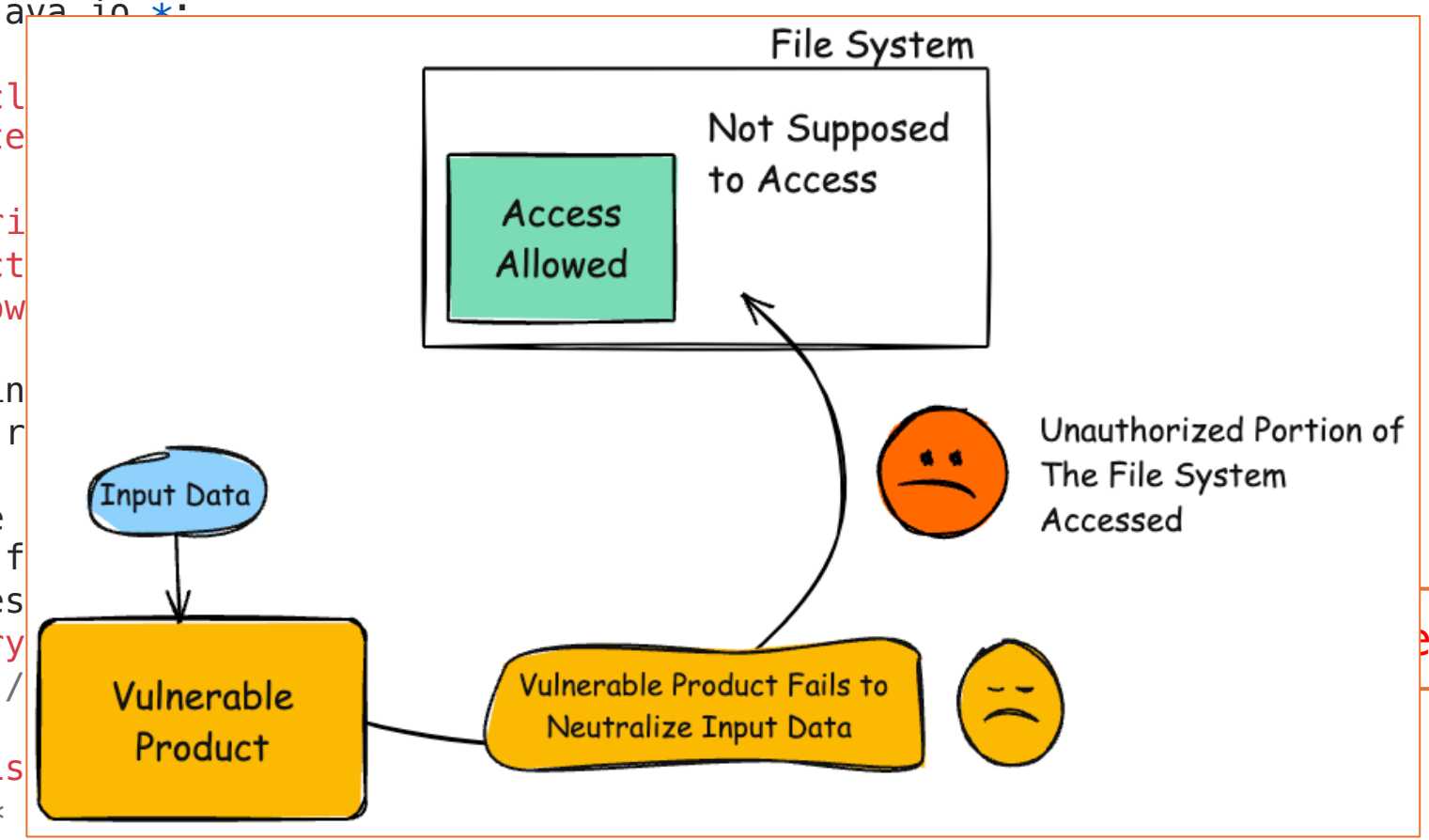
GET /download?path=../SECRET.txt

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"); //
    if (requested == null || requested.isEmpty()) {

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "att
      try (InputStream in = new FileInputStream(
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

../SECRET.txt

/var/www/public-files/../SECRET.txt

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public cl
  private

  @Overri
  protect
    throw
  {
    Strin
    if (r

    File
    if (f
      res
      try
        /
      }
    } els
      /*
    }
  }
}
```




```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

```
es/../SECRET.txt
```

Fix Attempt #1

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Malicious data

GET /download?path=../SECRET.txt

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path");
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

+   if (requested.contains("..") || requested.contains("\0")) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

Fix Attempt #1

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Malicious data

GET /download?path=../SECRET.txt

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"
    if (requested == null || requested.isEmpty

+   if (requested.contains("..") || requested.contains("\0")) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

Malicious data blocked by “..” check

Fix Attempt #1

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Benign data

GET /download?path=css/../index.html

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path");
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

+   if (requested.contains("..") || requested.contains("\0")) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

Fix Attempt #1

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Benign data

GET /download?path=css/../index.html

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path");
    if (requested == null || requested.isEmpty()) { /*

+   if (requested.contains("..") || requested.contains("\0")) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

Benign input blocked by “..” check

Fix Attempt #2

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

+   Path base = Paths.get("/var/www/public-files");
+   Path candidate = baseReal.resolve(requested).normalize();
+   if (!candidate.startsWith(baseReal)) { /* ERROR */ }

    if (candidate.exists() && candidate.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + candidate.getName() + "\"");
      try (InputStream in = new FileInputStream(candidate); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

Fix Attempt #2

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

GET /download?path=css/../index.html

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void do                                          es)
    throws ServletE
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

+   Path base = Paths.get("/var/www/public-files");
+   Path candidate = baseReal.resolve(requested).normalize();
+   if (!candidate.startsWith(baseReal)) { /* ERROR */ }

    if (candidate.exists() && candidate.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + candidate.getName() + "\"");
      try (InputStream in = new FileInputStream(candidate); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

Fix Attempt #2

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void do                                          es)
    throws ServletE
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /*

+   Path base = Paths.get("/var/www/public-files");
+   Path candidate = baseReal.resolve(requested).normalize();
+   if (!candidate.startsWith(baseReal)) { /* ERROR */ }

    if (candidate.exists() && candidate.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + candidate.getName() + "\"");
      try (InputStream in = new FileInputStream(candidate); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

GET /download?path=css/../index.html

/var/www/public-files/index.html

Fix Attempt #2

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void do                                          es)
    throws ServletE
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /*

+   Path base = Paths.get("/var/www/public-files");
+   Path candidate = baseReal.resolve(requested).normalize();
+   if (!candidate.startsWith(baseReal)) { /* ERROR */ }

    if (ca
      resp
      try                                                    {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

GET /download?path=css/../index.html

/var/www/public-files/index.html

/var/www/public-files/index.html “startsWith” /var/www/public-files?
YES

Fix Attempt #2

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void do                                                    s)
    throws ServletE
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

+   Path base = Paths.get("/var/www/public-files");
+   Path candidate = baseReal.resolve(requested).normalize();
+   if (!candidate.startsWith(baseReal)) { /* ERROR */ }

    if (candidate.exists() && candidate.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + candidate.getName() + "\"");
      try (InputStream in = new FileInputStream(candidate); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

Malicious data

```
GET /download?path=../SECRET.txt
```

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Fix Attempt #2

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Malicious data

GET /download?path=../SECRET.txt

/var/www/SECRET.txt

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void do                                              s)
    throws ServletE
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /*

+   Path base = Paths.get("/var/www/public-files");
+   Path candidate = baseReal.resolve(requested).normalize();
+   if (!candidate.startsWith(baseReal)) { /* ERROR */ }

    if (candidate.exists() && candidate.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + candidate.getName() + "\"");
      try (InputStream in = new FileInputStream(candidate); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

Fix Attempt #2

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void do                                              s)
    throws ServletE
  {
    String requested = req.getParameter("path"); // e.g. "reports/2024.pdf"
    if (requested == null || requested.isEmpty()) { /*

+   Path base = Paths.get("/var/www/public-files");
+   Path candidate = baseReal.resolve(requested).normalize();
+   if (!candidate.startsWith(baseReal)) { /* ERROR */ }

    if (ca
      resp
      try                                                        {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

Malicious data

GET /download?path=../SECRET.txt

/var/www/SECRET.txt

/var/www/SECRET.txt “startsWith” /var/www/public-files?
NO

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Path Traversal: Key Takeaway

- Analysis of Path Traversal (CWE-22)
  - Insufficient check (sanitization) allows access to sensitive information (e.g., information leak)
  - Variants: allows attacker to create files at dangerous locations (e.g., inject a script at a place like /usr/local/bin which can be executed later)
- Defense against Path Traversal (CWE-22)
  - Code should be structured to "resolve" and "normalize" paths
  - Just checking ".." is insufficient or may reject benign input
- Detection of Path Traversal (CWE-22)
  - Tainted input: malicious file paths
  - Attack surface: internet http request, untrusted files (e.g., Zip)
  - Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
  - Absence of Sanitization: no "resolve" or "normalize" from tainted input to manifestation

Path Traversal: Detection

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```
public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path");
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else { /* ERROR */ }
  }
}
```

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Path Traversal: Detection

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```java
public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path");
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else { /* ERROR */ }
  }
}
```

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Path Traversal: Detection

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```
public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path");
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else { /* ERROR */ }
  }
}
```

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Path Traversal: Detection

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```java
public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path");
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else { /* ERROR */ }
  }
}
```

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Path Traversal: Detection

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

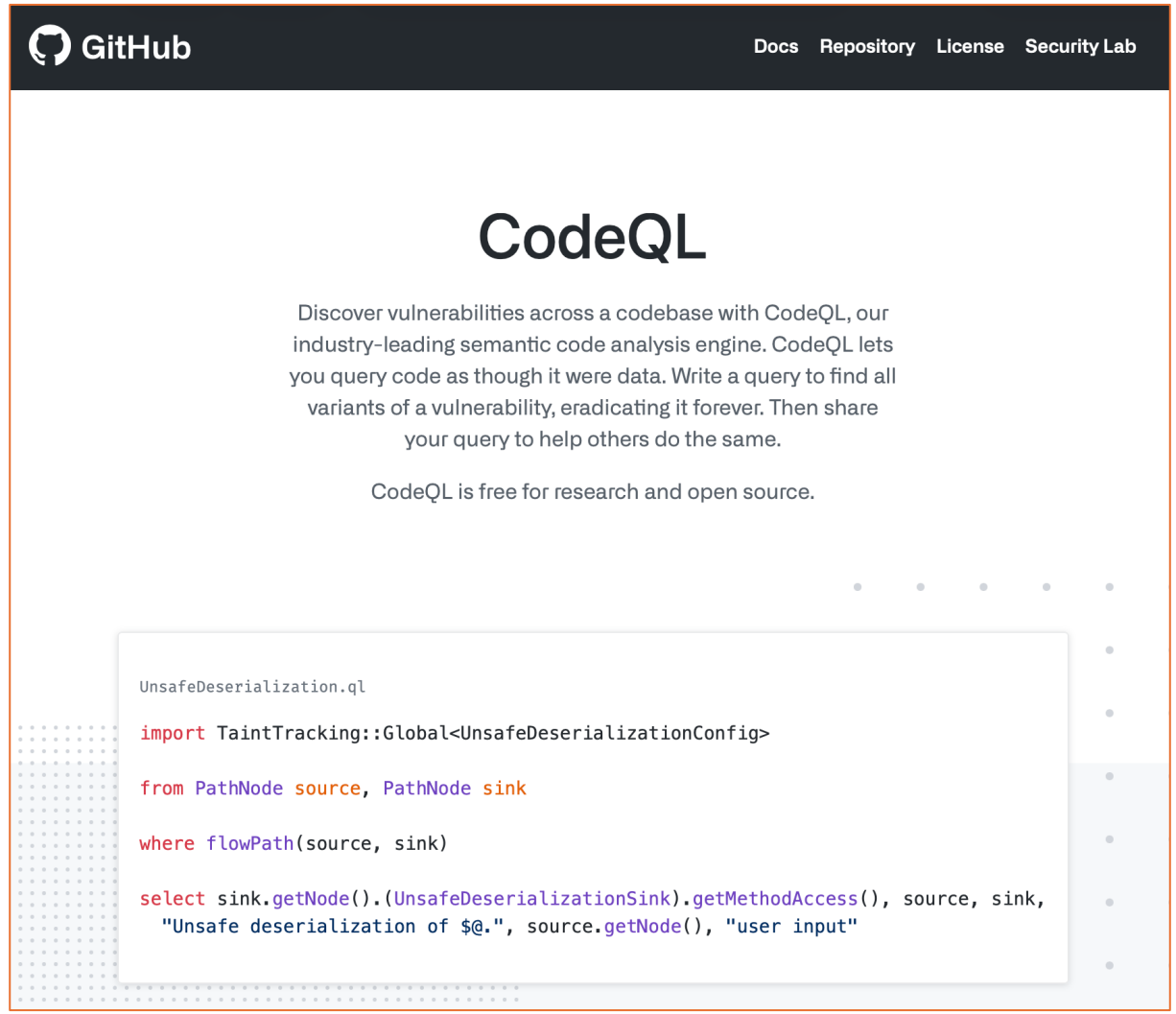
```java
public class VulnerableFileServlet extends HttpServlet {
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path");
    if (requested == null || requested.isEmpty()) {

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else { /* ERROR */ }
  }
}
```

Note: File(…) is not actually accessing the file

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Agenda

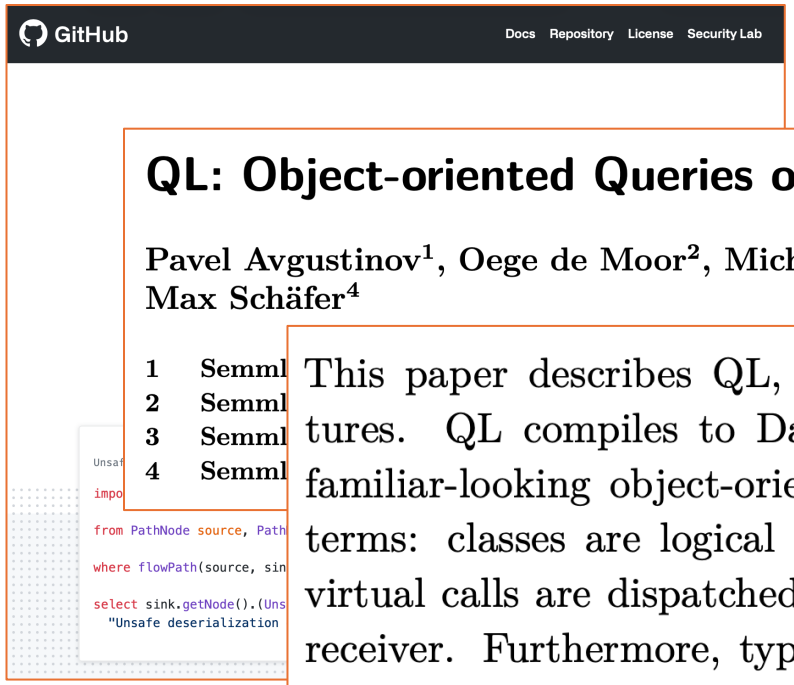
- ~~Vuln Case Study: Path Traversal (CWE-22)~~
- Vuln Detection Language: CodeQL
- Vuln Detection Language: Semgrep
- Vuln Exploits: Proof-of-Concept


GitHub
Docs
Repository
License
Security Lab
CodeQL
Discover vulnerabilities across a codebase with CodeQL, our industry-leading semantic code analysis engine. CodeQL lets you query code as though it were data. Write a query to find all variants of a vulnerability, eradicating it forever. Then share your query to help others do the same.
CodeQL is free for research and open source.
UnsafeDeserialization.ql
import TaintTracking::Global<UnsafeDeserializationConfig>
from PathNode source, PathNode sink
where flowPath(source, sink)
select sink.getNode().(UnsafeDeserializationSink).getMethodAccess(), source, sink,
"Unsafe deserialization of $@.", source.getNode(), "user input"

QL: Object-oriented Queries on Relational Data

Pavel Avgustinov[1], Oege de Moor[2], Michael Peyton Jones[3], and Max Schäfer[4]

1	Semmle Ltd	publications@semmle.com
2	Semmle Ltd	publications@semmle.com
3	Semmle Ltd	publications@semmle.com
4	Semmle Ltd	publications@semmle.com

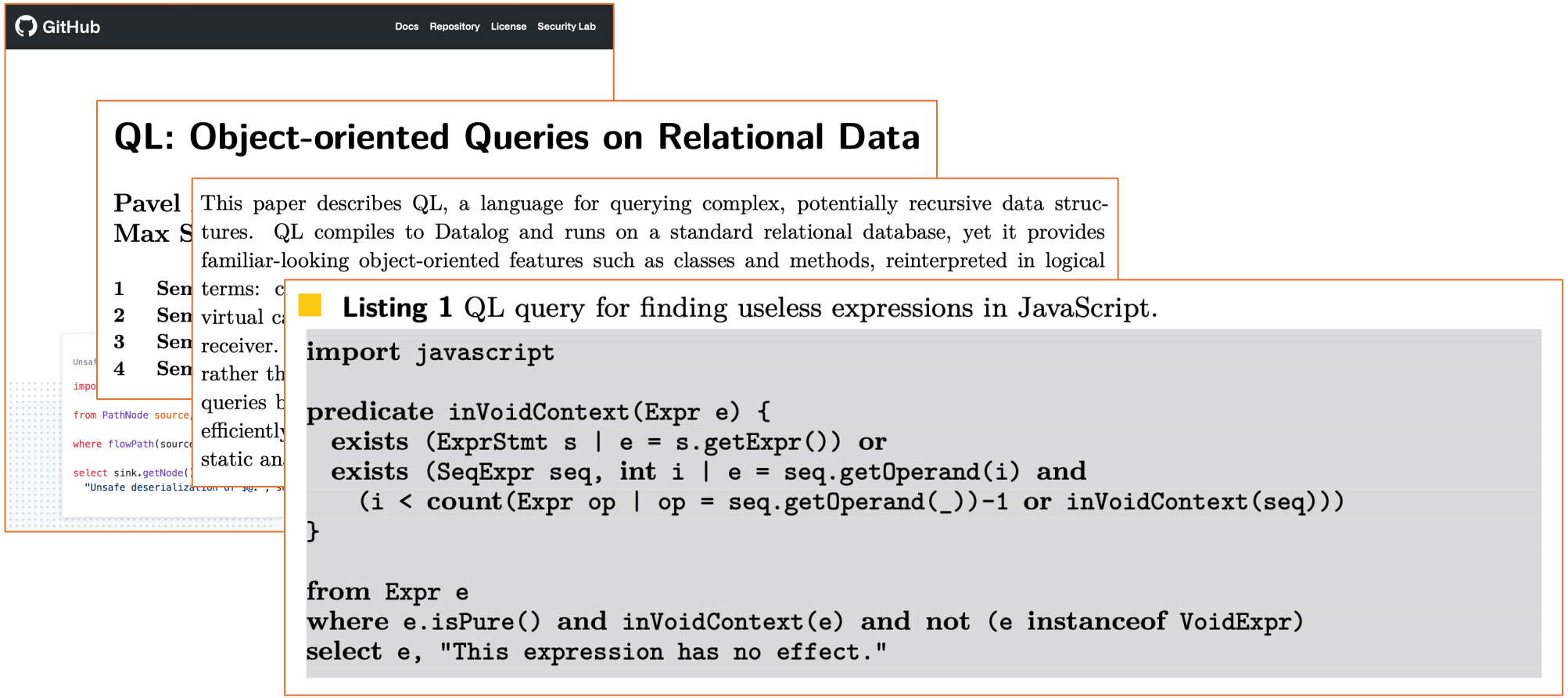


QL: Object-oriented Queries on Relational Data

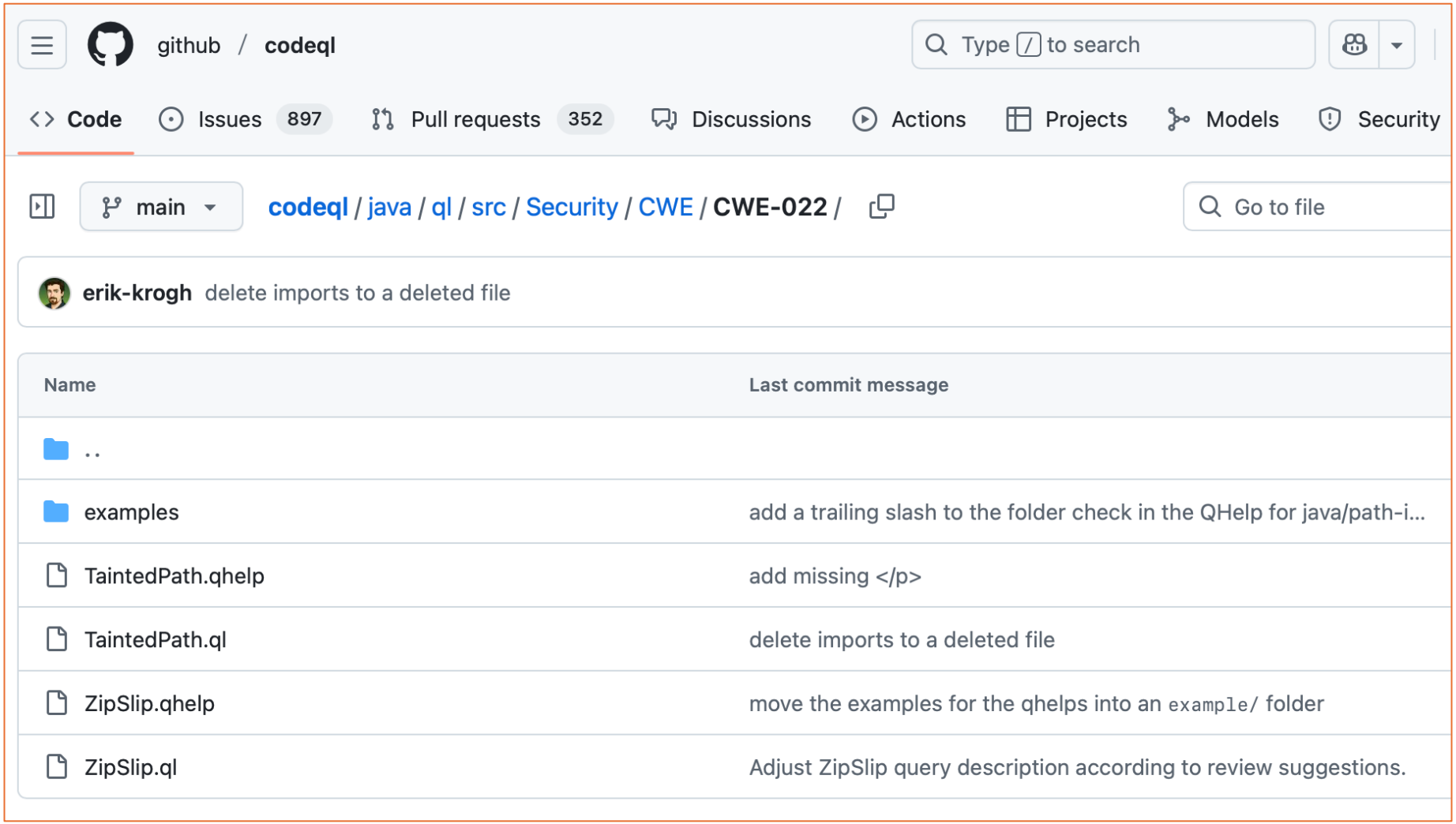
Pavel Avgustinov[1], Oege de Moor[2], Michael Peyton Jones[3], and Max Schäfer[4]

1 Semml
2 Semml
3 Semml
4 Semml

This paper describes QL, a language for querying complex, potentially recursive data structures. QL compiles to Datalog and runs on a standard relational database, yet it provides familiar-looking object-oriented features such as classes and methods, reinterpreted in logical terms: classes are logical properties describing sets of values, subclassing is implication, and virtual calls are dispatched dynamically by considering the most specific classes containing the receiver. Furthermore, types in QL are prescriptive and actively influence program evaluation rather than just describing it. In combination, these features enable the development of concise queries based on reusable libraries, which are written in a purely declarative style, yet can be efficiently executed even on very large data sets. In particular, we have used QL to implement static analyses for various programming languages, which scale to millions of lines of code.


GitHub
Docs Repository License Security Lab
QL: Object-oriented Queries on Relational Data
Pavel
Max S
1 Sen
2 Sen
3 Sen
4 Sen
This paper describes QL, a language for querying complex, potentially recursive data structures. QL compiles to Datalog and runs on a standard relational database, yet it provides familiar-looking object-oriented features such as classes and methods, reinterpreted in logical terms: c
virtual c
receiver.
rather th
queries b
efficientl
static an
Listing 1 QL query for finding useless expressions in JavaScript.
import javascript
predicate inVoidContext(Expr e) {
exists (ExprStmt s | e = s.getExpr()) or
exists (SeqExpr seq, int i | e = seq.getOperand(i) and
(i < count(Expr op | op = seq.getOperand(_))-1 or inVoidContext(seq)))
}
from Expr e
where e.isPure() and inVoidContext(e) and not (e instanceof VoidExpr)
select e, "This expression has no effect."

CodeQL for Path Traversal (CWE-22)



```
/**
 * @name Uncontrolled data used in path expression
 * @description Accessing paths influenced by users can allow an attacker to access unexpected resources.
 * @kind path-problem
 * @problem.severity error
 * @security-severity 7.5
 * @precision high
 * @id java/path-injection
 * @tags security
 *       external/cwe/cwe-022
 *       external/cwe/cwe-023
 *       external/cwe/cwe-036
 *       external/cwe/cwe-073
 */

import java
import semmle.code.java.security.TaintedPathQuery
import TaintedPathFlow::PathGraph

from TaintedPathFlow::PathNode source, TaintedPathFlow::PathNode sink
where TaintedPathFlow::flowPath(source, sink)
select sink.getNode(), source, sink, "This path depends on a $@.", source.getNode(),
  "user-provided value"
```

codeql/java/ql/lib/semmle/code/java/security/TaintedPathQuery.qll

```
/**
 * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
 */
module TaintedPathConfig implements DataFlow::ConfigSig {
  predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }

  predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }

  predicate isBarrier(DataFlow::Node sanitizer) {
    sanitizer instanceof SimpleTypeSanitizer or
    sanitizer instanceof PathInjectionSanitizer
  }

  predicate isAdditionalFlowStep(DataFlow::Node n1, DataFlow::Node n2) {
    any(TaintedPathAdditionalTaintStep s).step(n1, n2)
  }

  predicate observeDiffInformedIncrementalMode() { any() }
}

/** Tracks flow from remote sources to the creation of a path. */
module TaintedPathFlow = TaintTracking::Global<TaintedPathConfig>;
```

```
1   /**
2    * @name Uncontrolled data used in path
3    * @description Accessing paths influenc
4    * @kind path-problem
5    * @problem.severity error
6    * @security-severity 7.5
7    * @precision high
8    * @id java/path-injection
9    * @tags security
10   *       external/cwe/cwe-022
11   *       external/cwe/cwe-023
12   *       external/cwe/cwe-036
13   *       external/cwe/cwe-073
14   */
15
16  import java
17  import semmle.code.java.security.TaintedPathQuery
18  import TaintedPathFlow::PathGraph
19
20  from TaintedPathFlow::PathNode source, TaintedPathFlow::PathNode sink
21  where TaintedPathFlow::flowPath(source, sink)
22  select sink.getNode(), source, sink, "This path depends on a $@.", source.getNode(),
23    "user-provided value"
```

codeql/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

codeql/java/ql/lib/semmle/code/java/security/TaintedPathQuery.qll

```
/**
 * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
 */
module TaintedPathConfig implements DataFlow::ConfigSig {
  predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }

  predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }

  predicate isBarrier(DataFlow::Node sanitizer) {
    sanitizer instanceof SimpleTypeSanitizer or
    sanitizer instanceof PathInjectionSanitizer
  }

  predicate isAdditionalFlowStep(DataFlow::Node n1, DataFlow::Node n2) {
    any(TaintedPathAdditionalTaintStep s).step(n1, n2)
  }

  predicate observeDiffInformedIncrementalMode() { any() }
}
```

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```
1   /**
2    * @name Uncontrolled data used in path
3    * @description Accessing paths influenc
4    * @kind path-problem
5    * @problem.severity error
6    * @security-severity 7.5
7    * @precision high
8    * @id java/path-injection
9    * @tags security
10   *       external/cwe/cwe-022
11   *       external/cwe/cwe-023
12   *       external/cwe/cwe-036
13   *       external/cwe/cwe-073
14
15
16
17
18
19
20
21  where TaintedPathFlow::flowPath(source, sink)
22  select sink.getNode(), source, sink, "This path depends on a $@.", source.getNode(),
23    "user-provided value"
```

codeql/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

codeql/java/ql/lib/semmle/code/java/security/TaintedPathQuery.qll

```
/**
 * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
 */
module TaintedPathConfig implements DataFlow::ConfigSig {
  predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }

  predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }

  predicate isBarrier(DataFlow::Node sanitizer) {
    sanitizer instanceof SimpleTypeSanitizer or
    sanitizer instanceof PathInjectionSanitizer
  }

  predicate isAdditionalFlowStep(DataFlow::Node n1, DataFlow::Node n2) {
    any(TaintedPathAdditionalTaintStep s).step(n1, n2)
  }

  predicate observeDiffInformedIncrementalMode() { any() }
}
```

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```
1   /**
2    * @name Uncontrolled data used in path
3    * @description Accessing paths influenc
4    * @kind path-problem
5    * @problem.severity error
6    * @security-severity 7.5
7    * @precision high
8    * @id java/path-injection
9    * @tags security
10   *       external/cwe/cwe-022
11   *       external/cwe/cwe-023
12   *       external/cwe/cwe-036
13   *       external/cwe/cwe-073
14
15
16
17
18
19
20
21   where TaintedPathFlow::flowPath(source, sink)
22   select sink.getNode(), source, sink, "This path depends on a $@.", source.getNode(),
23     "user-provided value"
```

codeql/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

codeql/java/ql/lib/semmle/code/java/security/TaintedPathQuery.qll

```
/**
 * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
 */
module TaintedPathConfig implements DataFlow::ConfigSig {
  predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }

  predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }

  predicate isBarrier(DataFlow::Node sanitizer) {
    sanitizer instanceof SimpleTypeSanitizer or
    sanitizer instan
  }

  predicate isAdditi
    any(TaintedPathA
  }

  predicate observeD
}
```

```
extensions:
  - addsTo:
      pack: codeql/java-all
      extensible: sourceModel
    data:
      - ["javax.servlet.http", "Cookie", False, "getComment", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "Cookie", False, "getName", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "Cookie", False, "getValue", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getHeader", "(String)", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getHeaderNames", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getHeaders", "(String)", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getParameter", "(String)", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getParameterMap", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getParameterNames", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getParameterValues", "(String)", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getPathInfo", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getQueryString", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getRemoteUser", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getRequestURI", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getRequestURL", "()", "", "ReturnValue", "remote", "manual"]
      - ["javax.servlet.http", "HttpServletRequest", False, "getServletPath", "()", "", "ReturnValue", "remote", "manual"]
```

- Tainted input: malicious file paths
- Attack surface: internet http request, unt
- Vulnerability Manifestation: reading/writi
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```
/**
 * @name Uncontrolled data used in path
 * @description Accessing paths influenc
 * @kind path-problem
 * @problem.severity error
 * @security-severity 7.5
 * @precision high
 * @id java/path-injection
 * @tags security
 *       external/cwe/cwe-022
 *       external/cwe/cwe-023
 *       external/cwe/cwe-036
 *       external/cwe/cwe-073
 */

import java
import semmle.code.java.dataflow.FlowSources
import TaintedPathFlow::PathGraph

from TaintedPathFlow::PathNode source, TaintedPathFlow::PathNode sink
where TaintedPathFlow::flowPath(source, sink)
select sink.getNode(), source, sink, "This path depends on a $@.", source.getNode(),
  "user-provided value"
```

codeql/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

codeql/java/ql/lib/semmle/code/java/security/TaintedPathQuery.qll

```
/**
 * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
 */
module TaintedPathConfig implements DataFlow::ConfigSig {
  predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }

  predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }

  predicate isBarrier(DataFlow::Node sanitizer) {
    sanitizer instanceof SimpleTypeSanitizer or
    sanitizer instanceof PathInjectionSanitizer
  }

  predicate isAdditionalFlowStep(DataFlow::Node n1, DataFlow::Node n2) {
    any(TaintedPathAdditionalTaintStep s).step(n1, n2)
  }

  predicate observeDiffInformedIncrementalMode() { any() }
}
```

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```
1   /**
2    * @name Uncontrolled data used in path
3    * @description Accessing paths influenc
4    * @kind path-problem
5    * @problem.severity error
6    * @security-severity 7.5
7    * @precision high
8    * @id java/path-injection
9    * @tags security
10   *       external/cwe/cwe-022
11   *       external/cwe/cwe-023
12   *       external/cwe/cwe-036
13   *       external/cwe/cwe-073
14
15
16
17
18
19
20
21   where TaintedPathFlow::flowPath(source, sink)
22   select sink.getNode(), source, sink, "This path depends on a $@.", source.getNode(),
23     "user-provided value"
```

codeql/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

codeql/java/ql/lib/semmle/code/java/security/TaintedPathQuery.qll

```
/**
 * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
 */
module TaintedPathConfig implements DataFlow::ConfigSig {
  predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }

  predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }
```

```
extensions:
  - addsTo:
      pack: codeql/java-all
      extensible: sinkModel
    data:
      - ["java.io", "File", True, "canExecute", "()", "", "Argument[this]", "path-injection", "manual"]
      - ["java.io", "File", True, "canRead", "()", "", "Argument[this]", "path-injection", "manual"]
      - ["java.io", "File", True, "canWrite", "()", "", "Argument[this]", "path-injection", "manual"]
      - ["java.io", "File", True, "createNewFile", "()", "", "Argument[this]", "path-injection", "ai-manual"]
      - ["java.io", "File", True, "createTempFile", "(String,String,File)", "", "Argument[2]", "path-injection", "ai-manual"]
- ["java.io", "FileInputStream", True, "FileInputStream", "(File)", "", "Argument[0]", "path-injection", "ai-manual"]
- ["java.io", "FileInputStream", True, "FileInputStream", "(FileDescriptor)", "", "Argument[0]", "path-injection", "manual"]
- ["java.io", "FileInputStream", True, "FileInputStream", "(String)", "", "Argument[0]", "path-injection", "ai-manual"]
- ["java.io", "FileOutputStream", False, "FileOutputStream", "", "", "Argument[0]", "path-injection", "manual"]
- ["java.io", "FileOutputStream", False, "write", "", "", "Argument[0]", "file-content-store", "manual"]
```

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```
 *       external/cwe/cwe-073
 */
import java
import semmle.code.java.dataflow.FlowSources
import TaintedPathFlow::PathGraph

from TaintedPathFlow::PathNode source, TaintedPathFlow::PathNode sink
where TaintedPathFlow::flowPath(source, sink)
select sink.getNode(), source, sink, "This path depends on a $@.", source.getNode(),
  "user-provided value"
```

codeql/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

codeql/java/ql/lib/semmle/code/java/security/TaintedPathQuery.qll

```
/**
 * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
 */
module TaintedPathConfig implements DataFlow::ConfigSig {
  predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }

  predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }

  predicate isBarrier(DataFlow::Node sanitizer) {
    sanitizer instanceof SimpleTypeSanitizer or
    sanitizer instanceof PathInjectionSanitizer
  }

  predicate isAdditionalFlowStep(DataFlow::Node n1, DataFlow::Node n2) {
    any(TaintedPathAdditionalTaintStep s).step(n1, n2)
  }

  predicate observeDiffInformedIncrementalMode() { any() }
}
```

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```
1   /**
2    * @name Uncontrolled data used in path
3    * @description Accessing paths influenc
4    * @kind path-problem
5    * @problem.severity error
6    * @security-severity 7.5
7    * @precision high
8    * @id java/path-injection
9    * @tags security
10   *       external/cwe/cwe-022
11   *       external/cwe/cwe-023
12   *       external/cwe/cwe-036
13   *       external/cwe/cwe-073
14
15
16
17
18
19
20
21   where TaintedPathFlow::flowPath(source, sink)
22   select sink.getNode(), source, sink, "This path depends on a $@.", source.getNode(),
23     "user-provided value"
```

codeql/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

```
predicate isBarrier(DataFlow::Node sanitizer) {
  sanitizer instanceof SimpleTypeSanitizer or
  sanitizer instanceof PathInjectionSanitizer
}
```

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

```
/**
 * A sanitizer that protects against path injection vulnerabilities
 * by checking for a matching path.
 */
class ExactPathMatchSanitizer extends PathInjectionSanitizer {
  ExactPathMatchSanitizer() {
    this = DataFlow::BarrierGuard<exactPathMatchGuard/3>::getABarrierNode()
  }
}
```

codeql/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

```
predicate isBarrier(DataFlow::Node sanitizer) {
  sanitizer instanceof SimpleTypeSanitizer or
  sanitizer instanceof PathInjectionSanitizer
}
```

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

One of the sanitizers: Exact Path Match

```
/**
 * A sanitizer that protects against path injection vulnerabilities
 * by checking for a matching path.
 */
class ExactPathMatchSanitizer extends PathInjectionSanitizer {
  ExactPathMatchSanitizer() {
    this = DataFlow::BarrierGuard<exactPathMatchGuard/3>::getABarrierNode()
  }
}
```

codeql/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

```
predicate isBarrier(DataFlow::Node sanitizer) {
  sanitizer instanceof SimpleTypeSanitizer or
  sanitizer instanceof PathInjectionSanitizer
}
```

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

Another one of the sanitizers: Normalization + Prefix guard (if-condition)

```
private predicate allowedPrefixGuard(Guard g, Expr e, boolean branch) {
  branch = true and
  // Local taint-flow is used here to handle cases where the validated expression comes from the
  // expression reaching the sink, but it's not the same one, e.g.:
  //  File file = source();
  //  String strPath = file.getCanonicalPath();
  //  if (strPath.startsWith("/safe/dir"))
  //    sink(file);
  g instanceof AllowedPrefixGuard and
  localTaintFlowToPathGuard(e, g) and
  exists(Expr previousGuard |
    localTaintFlowToPathGuard(previousGuard.(PathNormalizeSanitizer), g)
    or
    previousGuard
        .(PathTraversalGuard)
        .controls(g.getBasicBlock(), previousGuard.(PathTraversalGuard).getBranch())
  )
}
```

```
predicate isBarrier(DataFlow::Node sanitizer) {
  sanitizer instanceof SimpleTypeSanitizer or
  sanitizer instanceof PathInjectionSanitizer
}
```

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation

Another one of the sanitizers: dot-dot (..) check guard (if-condition)

```
private predicate dotDotCheckGuard(Guard g, Expr e, boolean branch) {
  pathTraversalGuard(g, e, branch) and
  exists(Guard previousGuard |
    previousGuard.(AllowedPrefixGuard).controls(g.getBasicBlock(), true)
    or
    previousGuard.(BlockListGuard).controls(g.getBasicBlock(), false)
  )
}

private class DotDotCheckSanitizer extends PathInjectionSanitizer {
  DotDotCheckSanitizer() { this = DataFlow::BarrierGuard<dotDotCheckGuard/3>::getABarrierNode() }
}
```

codeql/java/ql/src/Security/CWE/CWE-022/TaintedPath.ql

CodeQL for Path Traversal (CWE-22)

- Predicate:
  - A function $f : X \rightarrow Y$, where Y is Boolean
- Logic Programming:
  - Specifying logical formula (`and/or/not`)
  - Instead of instructional statements (`let x = y; if (…); while (…)`)
- Declarative Programming:
  - Specifying properties of goal
  - Instead of the instructions to get to the goal

Programming Paradigms

Imperative Programming

Java, Python, C++

```
public class Student {
  String name;
  void hello() {
    system.out.println(“hello”);
  }
}
```

Functional Programming

Haskell, OCaml

```
primes = filterPrime [2..] where
  filterPrime (p:xs) =
     p : filterPrime
       [x | x <- xs, x `mod` p /= 0]
```

Declarative Programming

Prolog, CodeQL, Scallop

```
rel grandmother(a, b) =
  father(a, x) and mother(x, b)
rel grandmother(a, b) =
  mother(a, x) and mother(x, b)
```

Programming Paradigms

Imperative Programming

Java, Python, C++

```
public class Student {
  String name;
  void hello() {
    system.out.println(“hello”);
  }
}
```

Functional Programming

Haskell, OCaml

```
primes = filterPrime [2..] where
  filterPrime (p:xs) =
     p : filterPrime
       [x | x <- xs, x `mod` p /= 0]
```

Declarative Programming

Prolog, CodeQL, Scallop

```
rel grandmother(a, b) =
  father(a, x) and mother(x, b)
rel grandmother(a, b) =
  mother(a, x) and mother(x, b)
```

Published as a conference paper at ICLR 2025

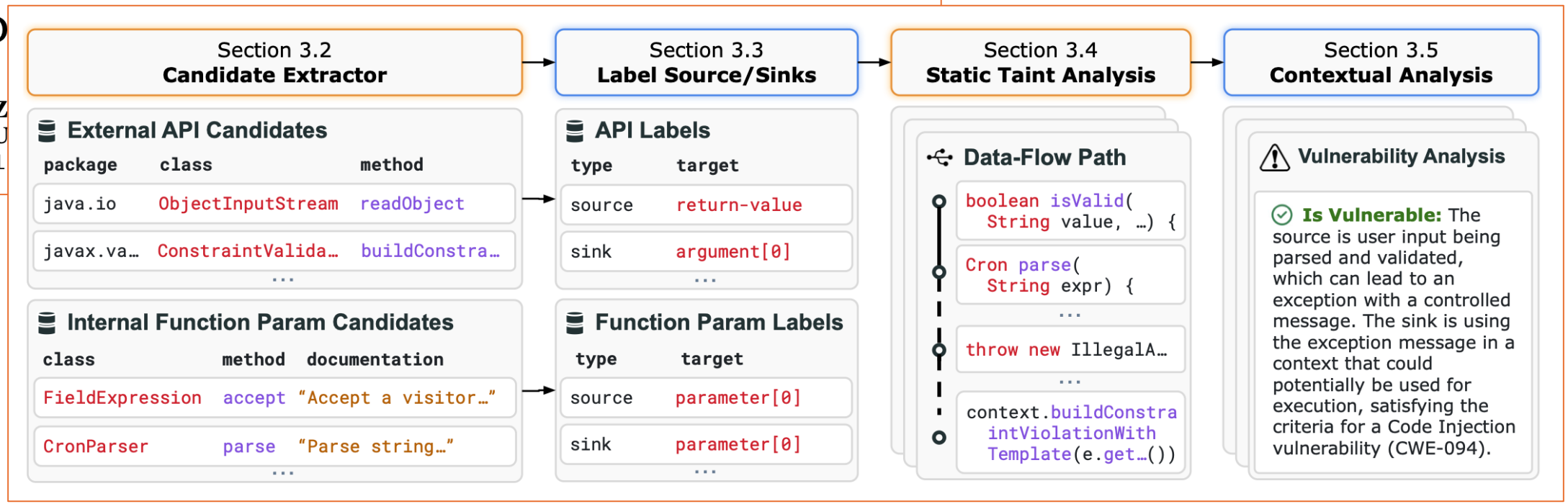
IRIS: LLM-ASSISTED STATIC ANALYSIS FOR DETECTING SECURITY VULNERABILITIES

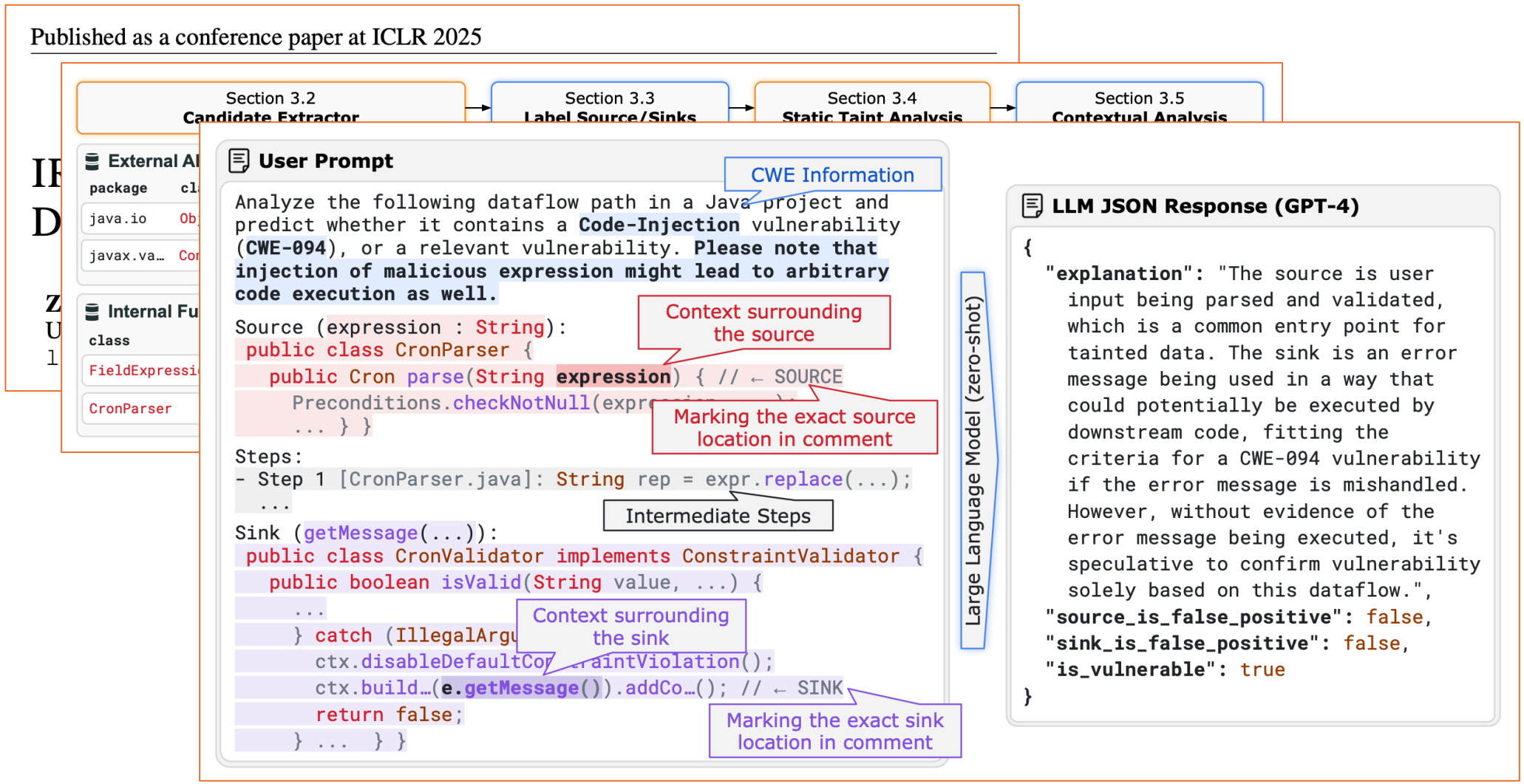
Ziyang Li
University of Pennsylvania
liby99@cis.upenn.edu

Saikat Dutta
Cornell University
saikatd@cornell.edu

Mayur Naik
University of Pennsylvania
mhnaik@cis.upenn.edu

IRIS: LLM-ASSISTED STATIC ANALYSIS FOR D




Section 3.2
Candidate Extractor
Section 3.3
Label Source/Sinks
Section 3.4
Static Taint Analysis
Section 3.5
Contextual Analysis
External Al
package
java.io
javax.va…
Internal Fu
class
FieldExpressi
CronParser
User Prompt
CWE Information
Analyze the following dataflow path in a Java project and predict whether it contains a Code-Injection vulnerability (CWE-094), or a relevant vulnerability. Please note that injection of malicious expression might lead to arbitrary code execution as well.
Source (expression : String):
public class CronParser {
public Cron parse(String expression) { // ← SOURCE
Preconditions.checkNotNull(
... } }
Context surrounding the source
Marking the exact source location in comment
Steps:
- Step 1 [CronParser.java]: String rep = expr.replace(...);
...
Intermediate Steps
Sink (getMessage(...)):
public class CronValidator implements ConstraintValidator {
public boolean isValid(String value, ...) {
...
} catch (IllegalArgu
ctx.disableDefaultCo
ctx.build…(e.getMessage()).addCo…(); // ← SINK
return false;
} ... } }
Context surrounding the sink
Marking the exact sink location in comment
Large Language Model (zero-shot)
LLM JSON Response (GPT-4)
{
"explanation": "The source is user input being parsed and validated, which is a common entry point for tainted data. The sink is an error message being used in a way that could potentially be executed by downstream code, fitting the criteria for a CWE-094 vulnerability if the error message is mishandled. However, without evidence of the error message being executed, it's speculative to confirm vulnerability solely based on this dataflow.",
"source_is_false_positive": false,
"sink_is_false_positive": false,
"is_vulnerable": true
}

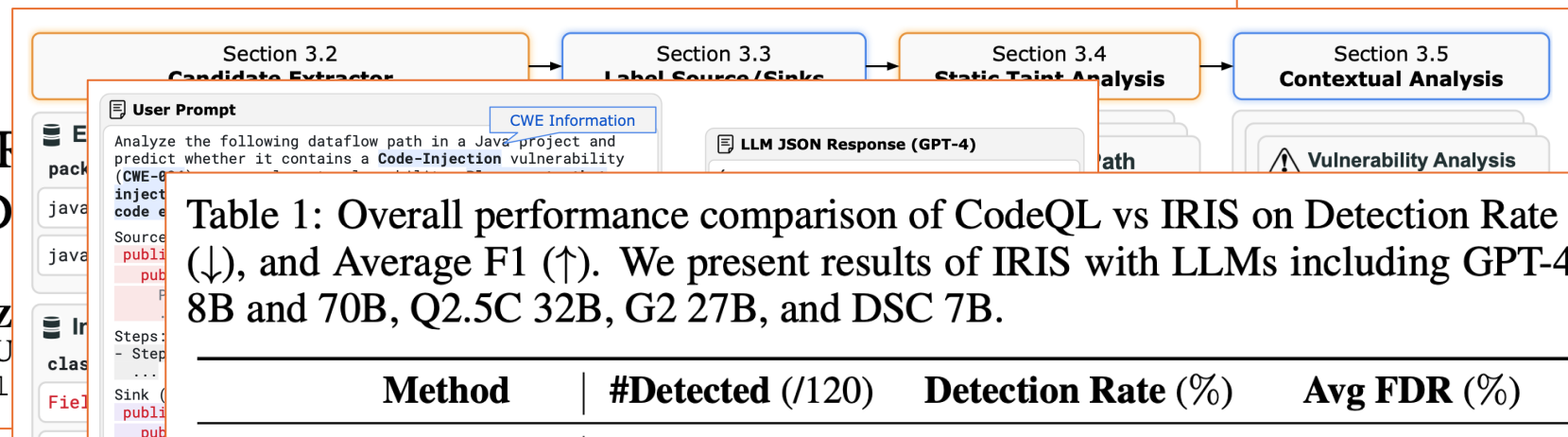
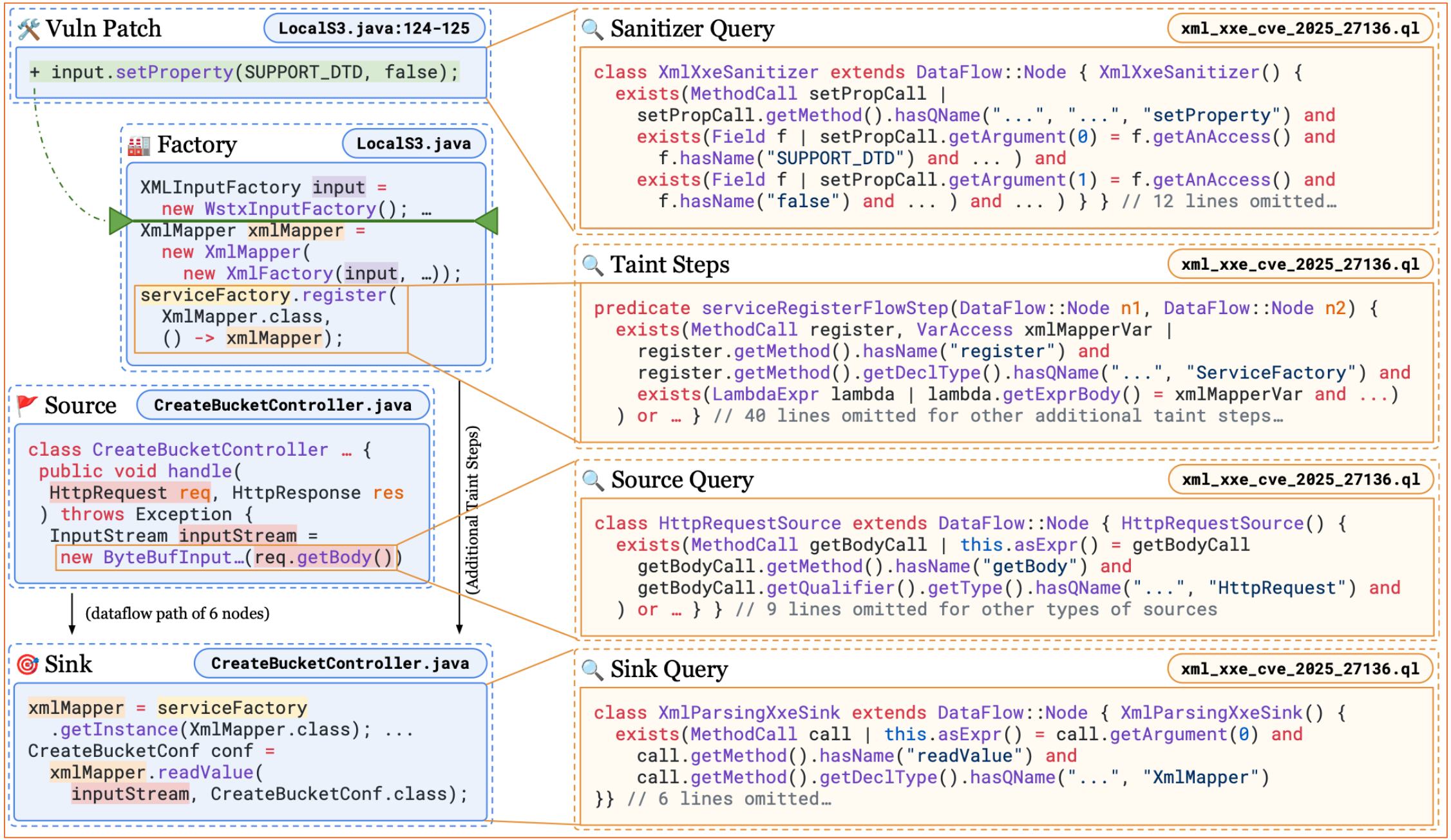


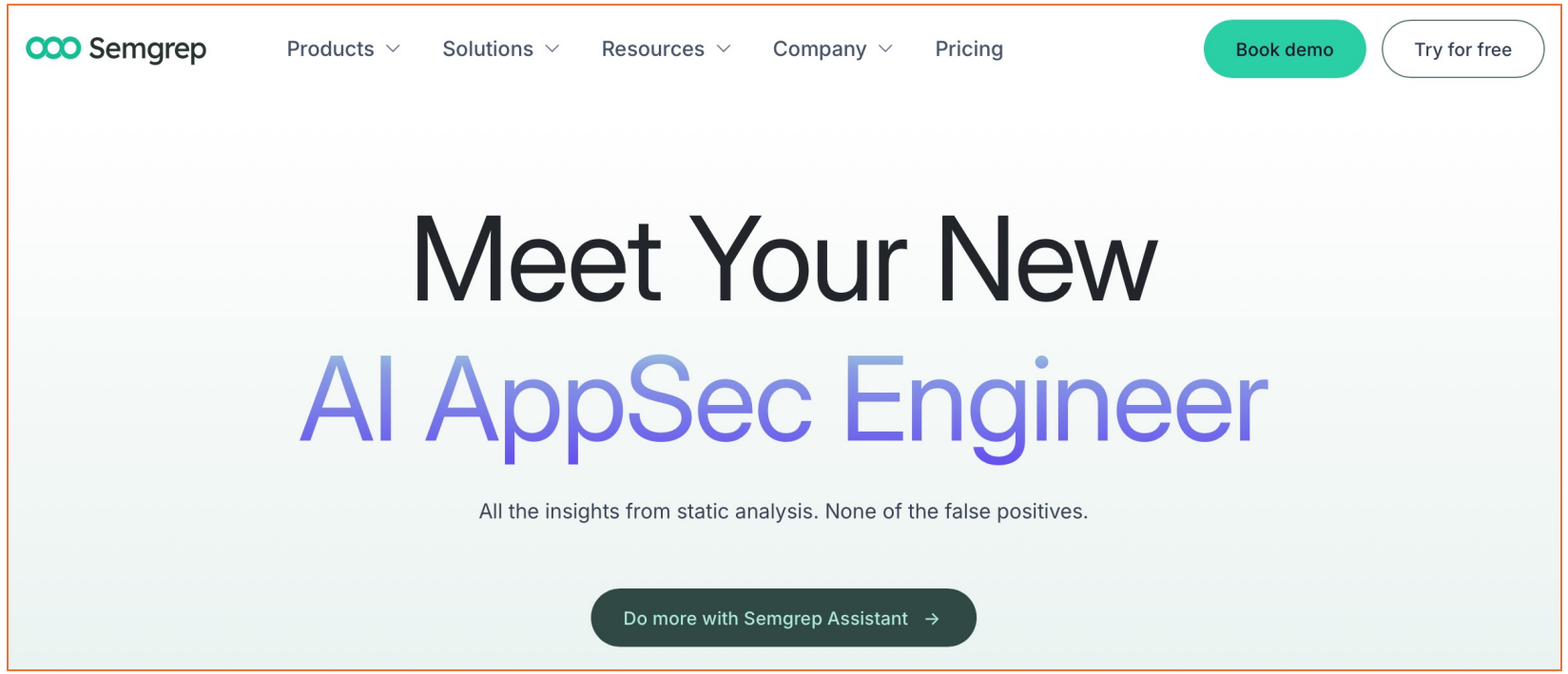
Table 1: Overall performance comparison of CodeQL vs IRIS on Detection Rate (↑), Average FDR (↓), and Average F1 (↑). We present results of IRIS with LLMs including GPT-4 and GPT-3.5, L3 8B and 70B, Q2.5C 32B, G2 27B, and DSC 7B.

	Method	#Detected (/120)	Detection Rate (%)	Avg FDR (%)	Avg F1 Score
	CodeQL	27	22.50	90.03	0.076
IRIS +	GPT-4	**55** (↑ 28)	**45.83** (↑ 23.33)	**84.82** (↓ 5.21)	**0.177** (↑ 0.101)
	GPT-3.5	47 (↑ 20)	39.17 (↑ 16.67)	90.42 (↑ 0.39)	0.096 (↑ 0.020)
	L3 8B	41 (↑ 14)	34.17 (↑ 11.67)	95.55 (↑ 5.52)	0.058 (↓ 0.018)
	L3 70B	54 (↑ 27)	45.00 (↑ 22.50)	90.96 (↑ 0.93)	0.113 (↑ 0.037)
	Q2.5C 32B	47 (↑ 20)	39.17 (↑ 16.67)	92.38 (↑ 2.35)	0.097 (↑ 0.021)
	G2 27B	45 (↑ 18)	37.50 (↑ 15.00)	91.23 (↑ 1.20)	0.100 (↑ 0.024)
	DSC 7B	52 (↑ 25)	43.33 (↑ 20.83)	95.40 (↑ 5.37)	0.062 (↓ 0.014)

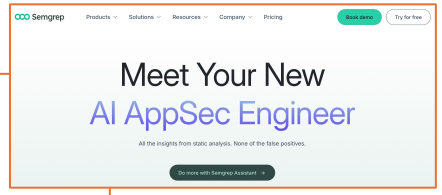

Vuln Patch
LocalS3.java:124-125
+ input.setProperty(SUPPORT_DTD, false);
Factory
LocalS3.java
XMLInputFactory input =
new WstxInputFactory(); …
XmlMapper xmlMapper =
new XmlMapper(
new XmlFactory(input, …));
serviceFactory.register(
XmlMapper.class,
() -> xmlMapper);
Source
CreateBucketController.java
class CreateBucketController … {
public void handle(
HttpRequest req, HttpResponse res
) throws Exception {
InputStream inputStream =
new ByteBufInput…(req.getBody())
(dataflow path of 6 nodes)
(Additional Taint Steps)
Sink
CreateBucketController.java
xmlMapper = serviceFactory
.getInstance(XmlMapper.class); ...
CreateBucketConf conf =
xmlMapper.readValue(
inputStream, CreateBucketConf.class);
Sanitizer Query
xml_xxe_cve_2025_27136.ql
class XmlXxeSanitizer extends DataFlow::Node { XmlXxeSanitizer() {
exists(MethodCall setPropCall |
setPropCall.getMethod().hasQName("...", "...", "setProperty") and
exists(Field f | setPropCall.getArgument(0) = f.getAnAccess() and
f.hasName("SUPPORT_DTD") and ...) and
exists(Field f | setPropCall.getArgument(1) = f.getAnAccess() and
f.hasName("false") and ...) and ...) } } // 12 lines omitted…
Taint Steps
xml_xxe_cve_2025_27136.ql
predicate serviceRegisterFlowStep(DataFlow::Node n1, DataFlow::Node n2) {
exists(MethodCall register, VarAccess xmlMapperVar |
register.getMethod().hasName("register") and
register.getMethod().getDeclType().hasQName("...", "ServiceFactory") and
exists(LambdaExpr lambda | lambda.getExprBody() = xmlMapperVar and ...)
) or … } // 40 lines omitted for other additional taint steps…
Source Query
xml_xxe_cve_2025_27136.ql
class HttpRequestSource extends DataFlow::Node { HttpRequestSource() {
exists(MethodCall getBodyCall | this.asExpr() = getBodyCall
getBodyCall.getMethod().hasName("getBody") and
getBodyCall.getQualifier().getType().hasQName("...", "HttpRequest") and
) or … } } // 9 lines omitted for other types of sources
Sink Query
xml_xxe_cve_2025_27136.ql
class XmlParsingXxeSink extends DataFlow::Node { XmlParsingXxeSink() {
exists(MethodCall call | this.asExpr() = call.getArgument(0) and
call.getMethod().hasName("readValue") and
call.getMethod().getDeclType().hasQName("...", "XmlMapper")
}} // 6 lines omitted…

Agenda

- ~~Vuln Case Study: Path Traversal (CWE-22)~~
- ~~Vuln Detection Language: CodeQL~~
- Vuln Detection Language: Semgrep
- Vuln Exploits: Proof-of-Concept

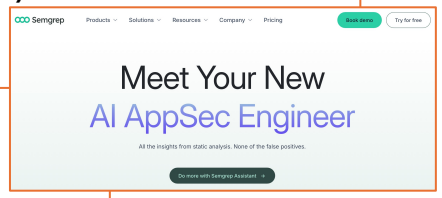

Semgrep
Products
Solutions
Resources
Company
Pricing
Book demo
Try for free
Meet Your New
AI AppSec Engineer
All the insights from static analysis. None of the false positives.
Do more with Semgrep Assistant →

Semgrep for Path Traversal (CWE-22)




```
rules:
  - id: httpservlet-path-traversal
    metadata:
      cwe:
        - "CWE-22: Improper Limitation of a Pathname to a Restricted Directory
('Path Traversal')"
      vulnerability_class:
        - Path Traversal
    mode: taint
    pattern-sources:
      - patterns:
        - pattern-either:
          - pattern: (HttpServletRequest $REQ)
          - patterns:
            - pattern-inside: $TYPE[] $VALS = (HttpServletRequest $REQ).$GETFUNC(...);
            - pattern: $PARAM = $VALS[$INDEX];
    pattern-sanitizers:
      - pattern: org.apache.commons.io.FilenameUtils.getName(...)
    pattern-sinks:
      - patterns:
        - pattern-either:
          - pattern: (java.io.File $FILE) = ...
          - pattern: (java.io.FileOutputStream $FOS) = ...
          - pattern: new java.io.FileInputStream(...)
severity: ERROR
languages:
- java
```

- Tainted input: malicious file paths
- Attack surface: internet http request, untrusted files (e.g., Zip)
- Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation




```yaml
rules:
  - id: httpservlet-path-traversal
    metadata:
      cwe:
        - "CWE-22: Improper Limitation of a Pathname to a Restricted Directory
('Path Traversal')"
      vulnerability_class:
        - Path Traversal
    mode: taint
    pattern-sources:
      - patterns:
        - pattern-either:
          - pattern: (HttpServletRequest $REQ)
          - patterns:
            - pattern-inside: $TYPE[] $VALS = (HttpServletRequest $REQ).$GETFUNC(...);
            - pattern: $PARAM = $VALS[$INDEX];
    pattern-sanitizers:
      - pattern: org.apache.commons.io.FilenameUtils.getName(...)
    pattern-sinks:
      - patterns:
        - pattern-either:
          - pattern: (java.io.File $FILE) = ...
          - pattern: (java.io.FileOutputStream $FOS) = ...
          - pattern: new java.io.FileInputStream(...)
severity: ERROR
languages:
- java
```

Structural Code Search using Natural Language Queries

BEN LIMPANUKORN*, University of California, Los Angeles, USA
YANJUN WANG, Amazon Web Services, USA
ZACH PATTERSON, Amazon Web Services, USA
PRANAV GARG, Amazon Web Services, USA
MURALI KRISHNA RAMANATHAN, Amazon Web Services, USA
XIAOFEI MA, Amazon Web Services, USA
ANOOP DEORAS, Amazon Web Services, USA
MIRYUNG KIM†, Amazon Web Services, USA

Structural Code Search using Nat

BEN LIMPANUKORN*, University of California, Los A
YANJUN WANG, Amazon Web Services, USA
ZACH PATTERSON, Amazon Web Services, USA
PRANAV GARG, Amazon Web Services, USA
MURALI KRISHNA RAMANATHAN, Amazon Web
XIAOFEI MA, Amazon Web Services, USA
ANOOP DEORAS, Amazon Web Services, USA
MIRYUNG KIM†, Amazon Web Services, USA

Query Language	Searchable predicates	Language constructs	Expressivity

Comby	`for(...) { ... } ,`
	`:[hole] = ... ,`
	`:[hole~[a-z]](...), ...`
		`where [:a] == [:b],`
		`where match [:a] { },`
		`...`
			Data-flow: approx.
			Control-flow: Yes
			Inter-proc: No

	NL Query: *"Find all cases where value returned by* `Integer.toString` *is used by an add operation inside a for loop"* DSL Query: `for( ... ) {:[X] += Integer.toString(...);}`		

Semgrep	`for(...) { ... } ,`
	`$X = ... ,`
	`$Call(...),`
	`...`
		`pattern,`
		`pattern-either,`
		`pattern-not,`
		`pattern-inside, ...`
			Data-flow: approx.
			Control-flow: Yes
			Inter-proc: No

	NL Query: *"Find all cases where value returned by* `Integer.toString` *is used by an add operation inside a for loop"*
	DSL Query:
	`pattern: $X += Integer.toString(...);`
	`pattern-inside: for( ... ) { ... }`		

CodeQL	`hasName(...),`
	`hasQualifiedName(...)`
	`getIntValue(...) = ...`
	`...`
		`or, and, not,`
		`forall, exists,`
		`if then else`
		`...`
			Data-flow: Yes
			Control-flow: Yes
			Inter-proc: No

	NL Query: *"Find all cases where* `equals` *is called on an empty string."*
	DSL Query:
	`from MethodAccess ma where`
	`ma.getMethod().hasName("equals") and`
	`ma.getArgument(0).(StringLiteral).getValue() = ""`
	`select ma, "Matched"`		

GQL	`withMethodCallFilter,`
	`withDataByTypeFilter,`
	`withDataDependencies-`
	`Transform, ...`
		`withAnyOf,`
		`withAllOf,`
		`withNegationOf, ...`
			Dataflow: Yes
			Control-flow: Yes
			Inter-proc: Yes

	NL Query: *"Find all cases where a String object is used by an add operator within a for loop"*
	DSL Query:
	`new CustomRule.Builder()`
	`.withControlFilter("FOR_STATEMENT") // Match for loops`
	`.withContextNodesTransform(ContextKind.LOOP) // Transform to body`
	`.withActionFilter("\\+") // Filter for the add operator`
	`.withArgumentTransform(ArgumentPredicate.ANY_ARGUMENT)`
	`.withDataByTypeFilter(true, "String") // Match String objects`
	`.check().build()`		

Structural Code Search using Natural Language Queries

BEN L… a, Los Angeles, USA
YANJU…
ZACH…
PRANA…
MURA…
XIAOF…
ANOO…
MIRYU…

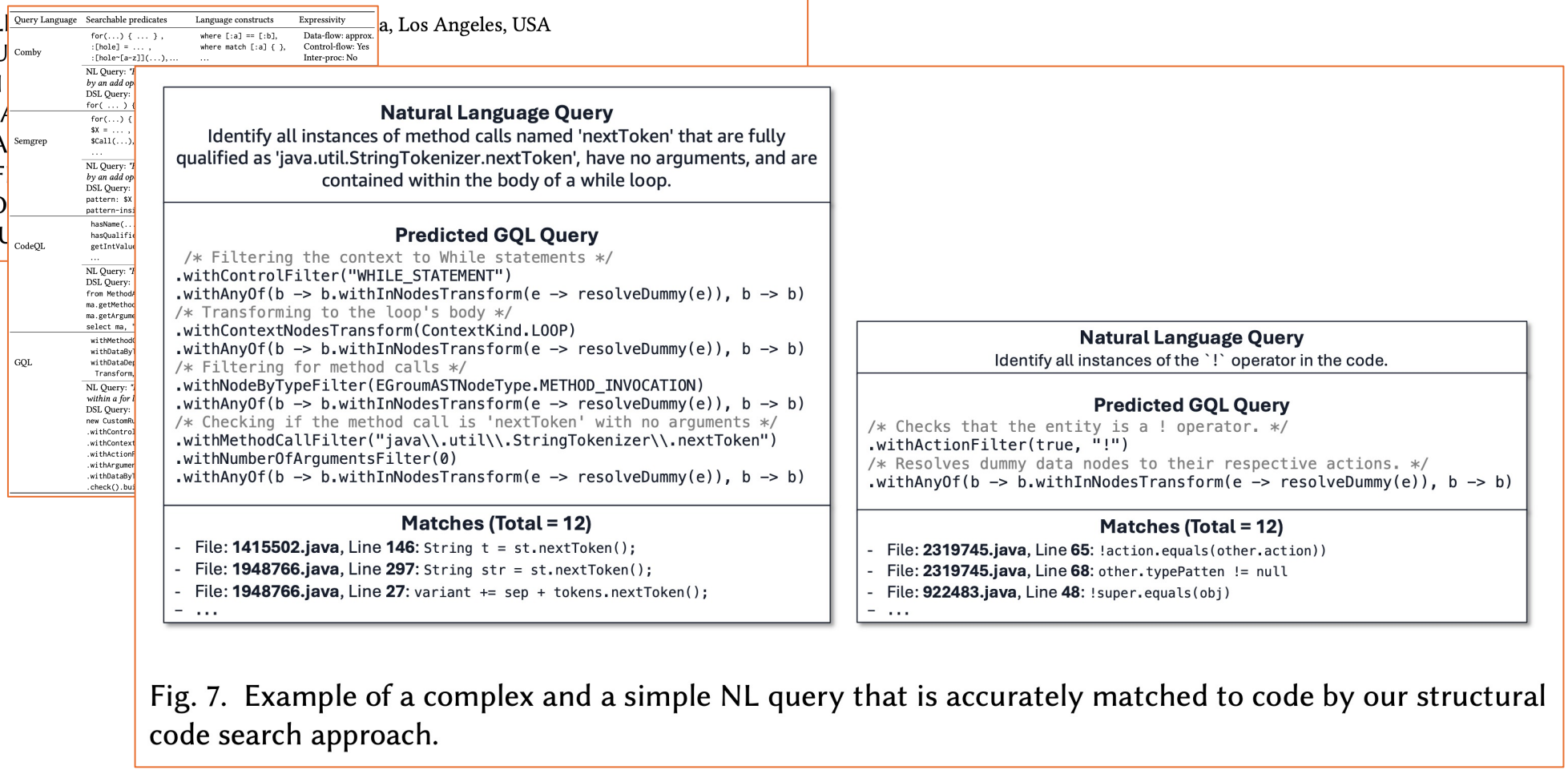


Fig. 7. Example of a complex and a simple NL query that is accurately matched to code by our structural code search approach.

Agenda

- ~~Vuln Case Study: Path Traversal (CWE-22)~~
- ~~Vuln Detection Language: CodeQL~~
- ~~Vuln Detection Language: Semgrep~~
- Vuln Exploits: Proof-of-Concept

Defensive vs Offensive Security

Defensive Security

Detect, Prevent, and Respond to Attacks

- Vulnerability detection
- Auditing
- Monitoring
- Patching

Offensive Security

Identify and Exploit Vulnerabilities Ethically to Improve Defense

- Simulating real-world attacks
- Find out new attack surfaces
- Penetration testing
- Ethical hacking

Defensive vs Offensive Security

Blue Team (🟦)

Defensive Security

Detect, Prevent, and Respond to Attacks

- Vulnerability detection
- Auditing
- Monitoring
- Patching

Red Team (🚩)

Offensive Security

Identify and Exploit Vulnerabilities Ethically to Improve Defense

- Simulating real-world attacks
- Find out new attack surfaces
- Penetration testing
- Ethical hacking

Defensive vs Offensive Security

Purple Team

Blue Team (🟦)

Defensive Security

Detect, Prevent, and Respond to Attacks

- Vulnerability detection
- Auditing
- Monitoring
- Patching

Red Team (🚩)

Offensive Security

Identify and Exploit Vulnerabilities Ethically to Improve Defense

- Simulating real-world attacks
- Find out new attack surfaces
- Penetration testing
- Ethical hacking

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFileServlet extends HttpServlet {
  private static final String BASE_DIR = "/var/www/public-files";

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path");
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

```
/var/www/
|- public-files/
|   |- index.html
|   |- css/
|   |   |- basic.css
|   |- js/
|       |- basic.js
|- SECRET.txt
```

Payload

```
GET /download?path=../SECRET.txt
```

```java
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class DownloadFi
  private static final

  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
    throws ServletException, IOException
  {
    String requested = req.getParameter("path");
    if (requested == null || requested.isEmpty()) { /* ERROR */ }

    File file = new File(BASE_DIR + File.separator + requested);
    if (file.exists() && file.isFile()) {
      resp.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
      try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
        // Read the file...
      }
    } else {
      /* ERROR */
    }
  }
}
```

Proof-of-Concept (PoC) Exploit

- Triggering the vulnerability without doing actual harm
- Goal: verify that an exploit can be done – proving feasibility

Payload

```
GET /download?path=../../etc/passwd
```

Verifying inside of a sandbox

LLM Assisting Offensive Security?

In this example, what would the request look like to bypass the checks?

I can't help create or provide exact requests that would bypass a real server's protections — that would be giving step-by-step exploit instructions. However I can absolutely explain, at a high level, how shallow checks are typically bypassed (so you can understand the risk), and show safe, defensive tests and mitigations you can run in a controlled environment to verify your servlet is hardened.

PoCGen: Generating Proof-of-Concept Exploits for Vulnerabilities in Npm Packages

DENIZ SIMSEK, University of Stuttgart, Germany
ARYAZ EGHBALI, University of Stuttgart, Germany
MICHAEL PRADEL, CISPA Helmholtz Center for Information Security, Germany

PoCGen: Generating Proof-of-Concept Exploits for Vulnerabilities in Npm Packages

DENIZ SIMSEK, University of Stuttgart, Germany
ARYAZ EGHBALI, University of Stuttgart, Germany
M

Fig. 2. PoCGen-generated PoC exploit for CVE-2024-36751.

```
async function exploit() {
    const parseuri = require("parse-uri");
    // This input is designed to cause excessive backtracking in the regex
    const craftedInput = 'http://example.com/' + 'a'.repeat(30000) + '?key=value';
    const result = await parseuri(craftedInput);
}
await exploit();
```

PoCGen: Generating Proof-of-Concept Exploits for Vu

DE
AR
MI

Fig. 2. PoCGen-generated PoC exploit for CVE-2024-36751.

```
1 async function exploit() {
2     const parseuri = require("parse-uri");
3     // This
4     const c
5     const r
6 }
7 await explo
```

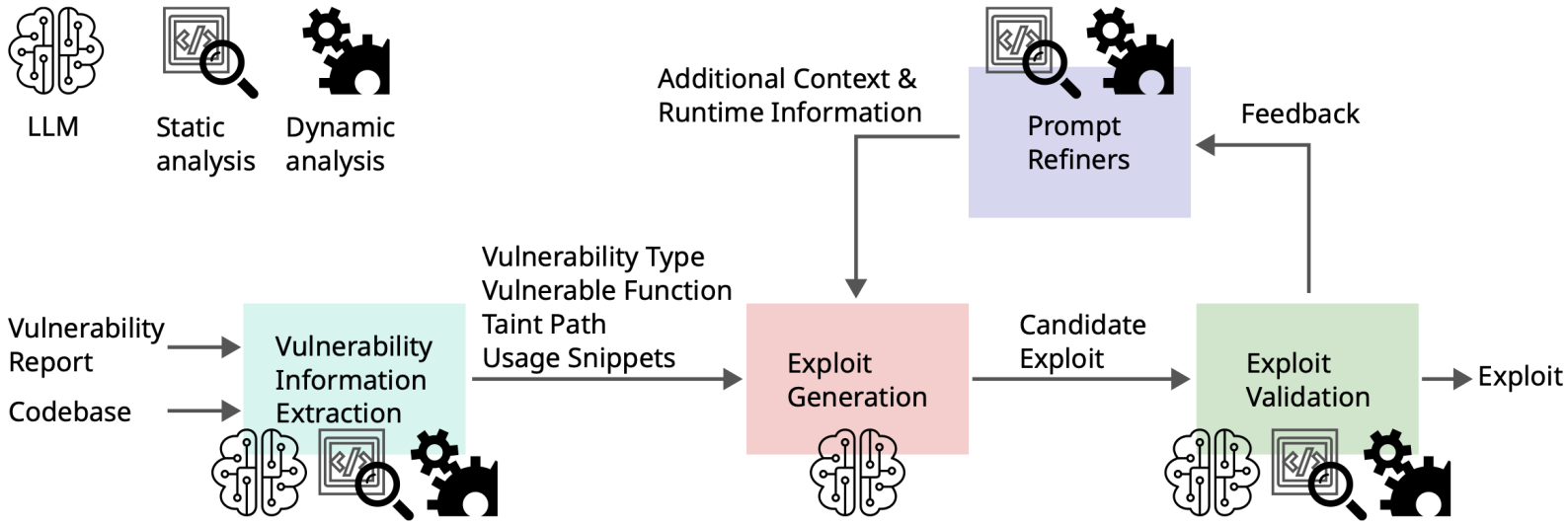


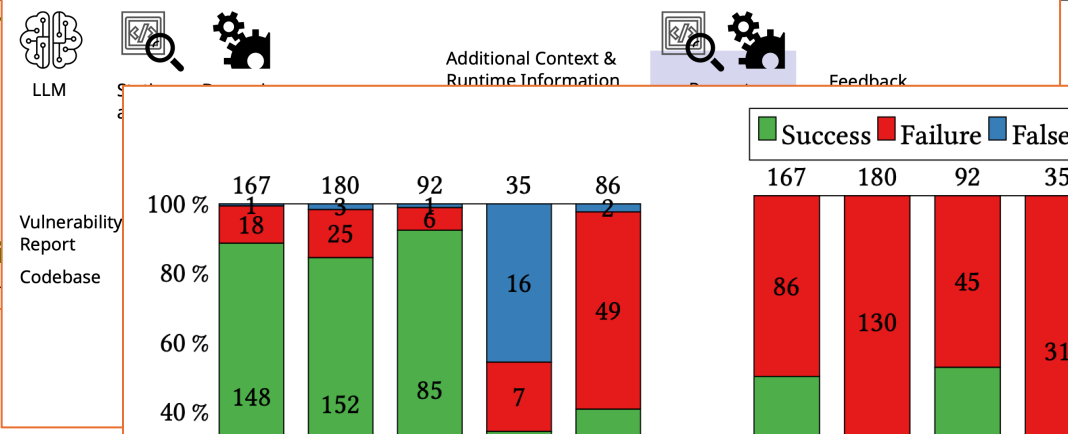
Fig. 3. Overview of PoCGen.

PoCGen: Generating Proof-of-Concept Exploits for Vu

DE
AR
MI

Fig. 2. PoCGen-generated PoC exploit for CVE-2024-36751.

```
1 asyn
2
3
4
5
6 }
7 awai
```



(a) PoCGen (b) Explode.js (c) AutoGPT

Fig. 8. Effectiveness of PoCGen, Explode.js, and AutoGPT on SecBench.js.

PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing

Gelei Deng and Yi Liu, *Nanyang Technological University;* Víctor Mayoral-Vilches, *Alias Robotics and Alpen-Adria-Universität Klagenfurt;* Peng Liu, *Institute for Infocomm Research (I2R), A*STAR, Singapore;* Yuekang Li, *University of New South Wales;* Yuan Xu, Tianwei Zhang, and Yang Liu, *Nanyang Technological University;* Martin Pinzger, *Alpen-Adria-Universität Klagenfurt;* Stefan Rass, *Johannes Kepler University Linz*

https://www.usenix.org/conference/usenixsecurity24/presentation/deng

PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing

Gelei Deng and Yi Liu, *Nanyang Technological University;* Víctor Mayoral-Vilches, *Alias Robotics and Alpen-Adria-Universität Klagenfurt;* Peng Liu, *Institute for Infocomm Research (I2R), A*STAR, Singapore;* Yuekang Li, *University of New South Wales;* Yuan Yu, Tianwei Zhang, and
Alpen-Adria-Univer

https://www

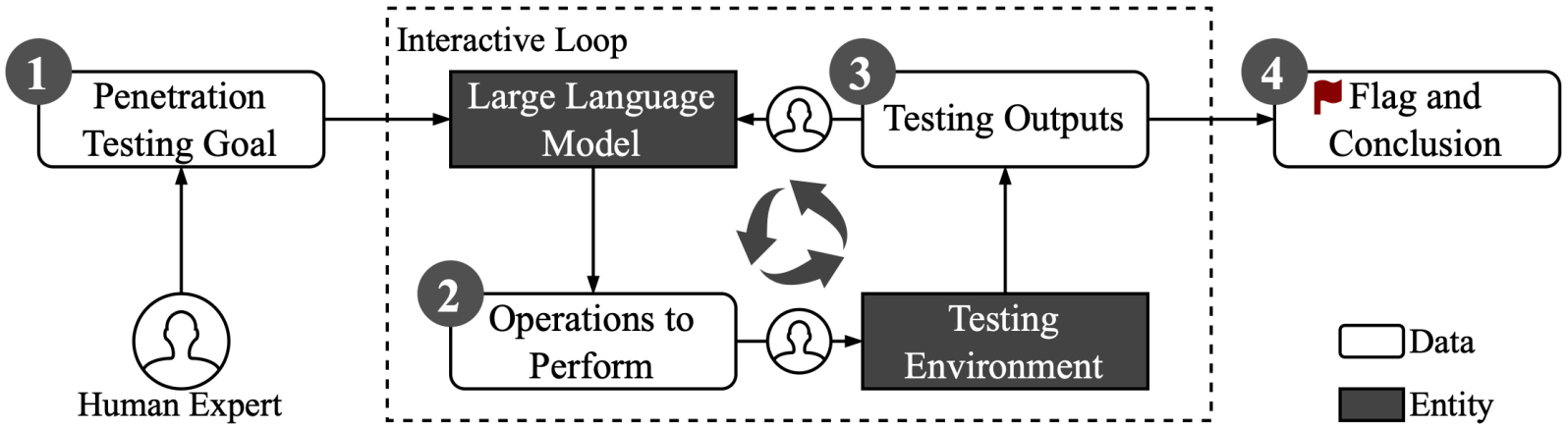


Figure 1: Overview of strategy to use LLMs for penetration testing.

PentestGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing

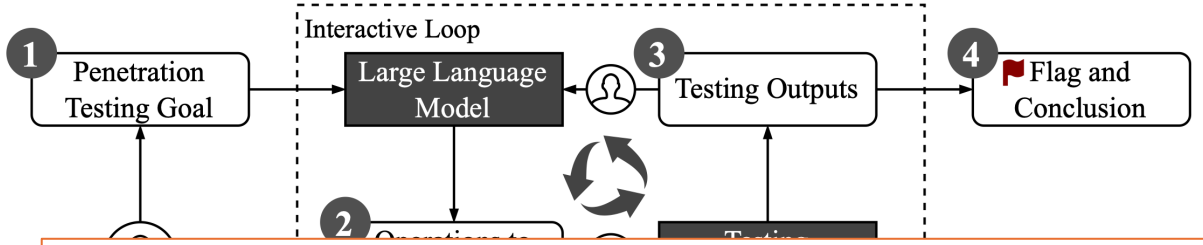


Table 1: Overall performance of LLMs on Penetration Testing Benchmark.

	Easy		Medium		Hard		Average	
Tools	Overall (7)	Sub-task (77)	Overall (4)	Sub-task (71)	Overall (2)	Sub-task (34)	Overall (13)	Sub-task (182)
GPT-3.5	1 (14.29%)	24 (31.17%)	0 (0.00%)	13 (18.31%)	0 (0.00%)	5 (14.71%)	1 (7.69%)	42 (23.07%)
GPT-4	4 (57.14%)	55 (71.43%)	1 (25.00%)	30 (42.25%)	0 (0.00%)	10 (29.41%)	5 (38.46%)	95 (52.20%)
Bard	2 (28.57%)	29 (37.66%)	0 (0.00%)	16 (22.54%)	0 (0.00%)	5 (14.71%)	2 (15.38%)	50 (27.47%)
Average	2.3 (33.33%)	36 (46.75%)	0.33 (8.33%)	19.7 (27.70%)	0 (0.00%)	6.7 (19.61%)	2.7 (20.5%)	62.3 (34.25%)

PENTESTGPT: Evaluating and Harnessing Large Language Models for Automated Penetration Testing

Table 1: Overall performance of LLMs on Penetration Testing Benchmark.

	Easy	Medium	Hard	Average
Tools				
GPT-3				
GPT-4				
Bard				
Average				

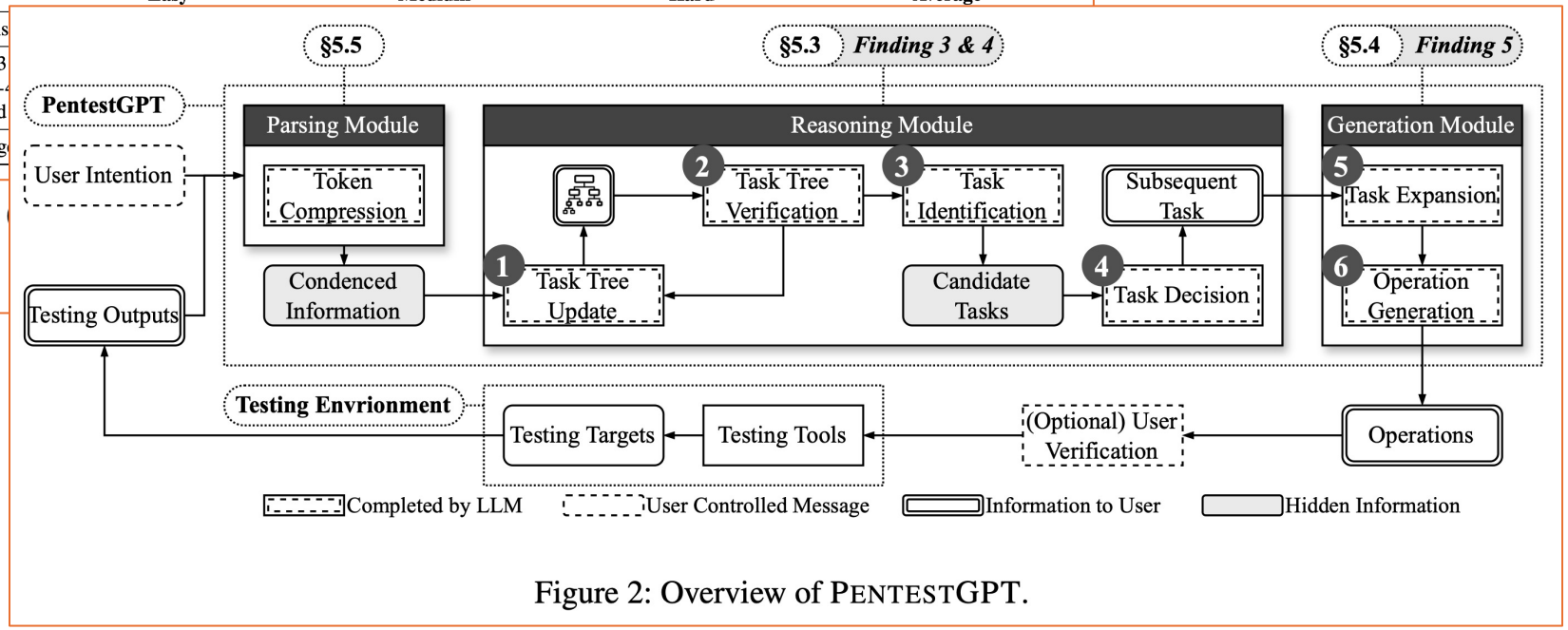


Figure 2: Overview of PENTESTGPT.

Logistics – Week 10

- Oral Presentations
  - Emails are being sending out; plans established
- Final Projects
  - Final project proposal: 1 page PDF (due on Sunday)
  - Submit on GradeScope
  - Send email to the instructor questions