

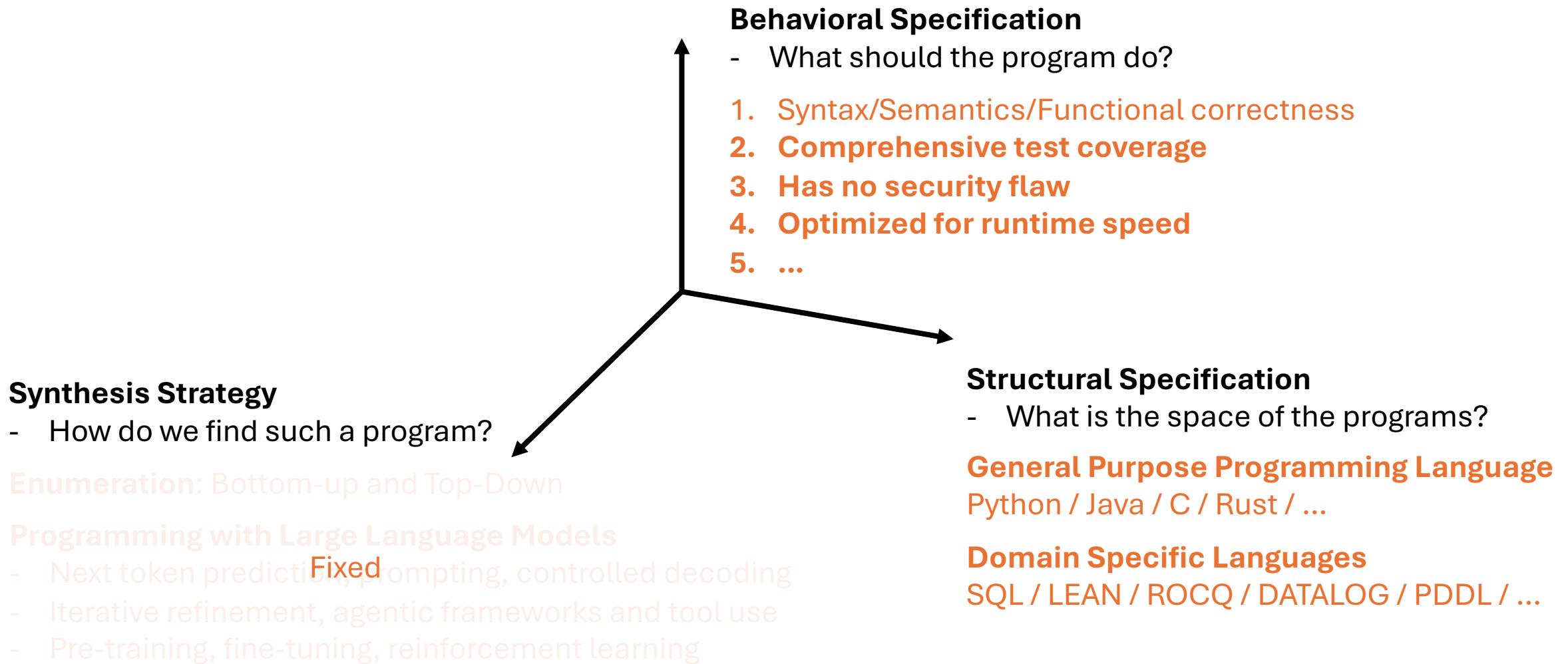
Machine Programming

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



Software Analysis

Vulnerability Detection, Analysis, and Exploitation

Forms of Software Analysis

Static Analysis

Examine code without concrete execution

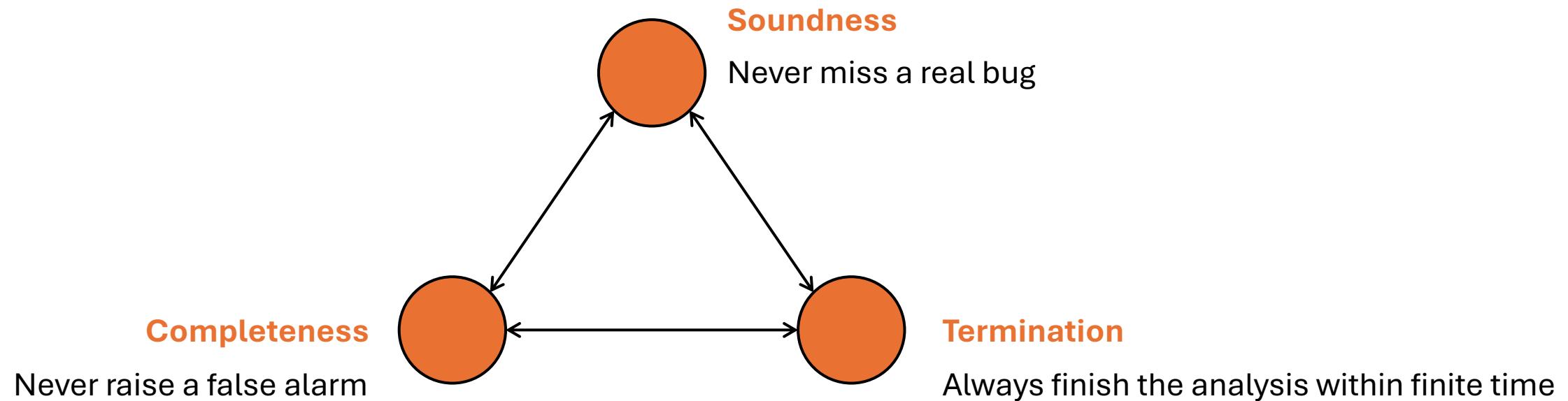
- Dataflow analysis
- Taint analysis
- Abstract interpretation
- Symbolic execution
- ...

Dynamic Analysis

Observes program behavior while executing

- Unit testing
- Fuzzing
- Property-based testing
- Penetration testing
- ...

Software Analysis: Impossible Triangle



Software Analysis: Impossible Triangle

Dynamic Analysis:

Unit Testing

- (assuming tests are correct)

Fuzzing

- (assuming harness is real)

Completeness

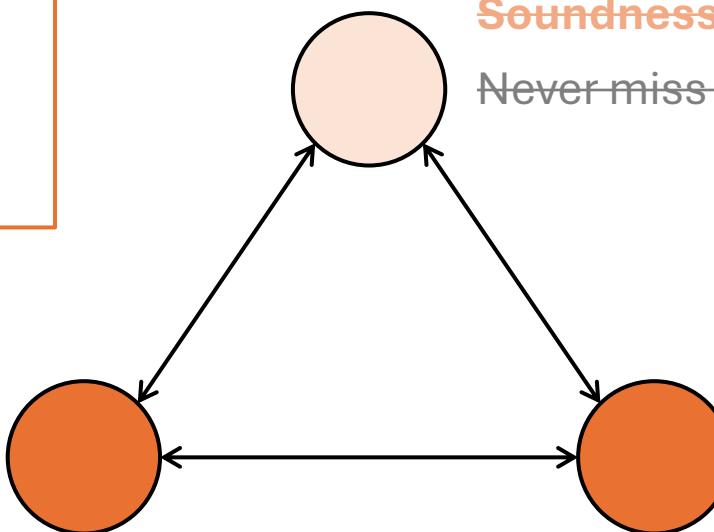
Never raise a false alarm

Soundness

Never miss a real bug

Termination

Always finish the analysis within finite time



Software Analysis: Impossible Triangle

Static Analysis:

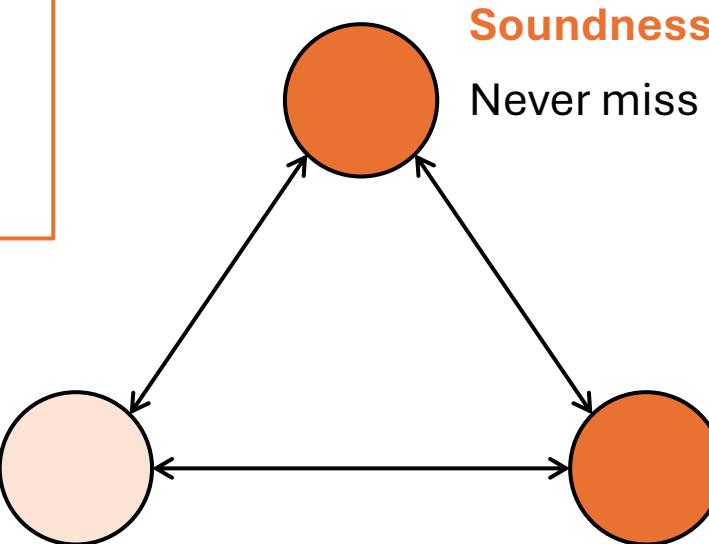
Taint Analysis

- (assuming correct specs)

Abstract Interpretation

- (assuming correct domains)

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:

The screenshot shows the '2024 CWE Top 25 Most Dangerous Software Weaknesses' page. The top navigation bar includes links for Home, About, Learn, Access Content, Community, and Search. On the right, there are links for 'Top 25' (blue circle), 'Top HW CWE' (green circle), and 'New to CWE? Start here!' (red text). A search bar with an ID lookup field and a 'Go' button is also present.

The main content area displays a list of 25 weaknesses, each with a rank, name, and details. The weaknesses listed are:

- 1 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[CWE-79](#) | CVEs in KEV: 3 | Rank Last Year: 2 (up 1) ▲
- 2 Out-of-bounds Write
[CWE-787](#) | CVEs in KEV: 18 | Rank Last Year: 1 (down 1) ▼
- 3 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[CWE-89](#) | CVEs in KEV: 4 | Rank Last Year: 3
- 4 Use of Hard-coded Credentials
[CWE-798](#) | CVEs in KEV: 2 | Rank Last Year: 18 (down 4) ▼
- 5 Integer Overflow or Wraparound
[CWE-190](#) | CVEs in KEV: 3 | Rank Last Year: 14 (down 9) ▼
- 6 Uncontrolled Resource Consumption
[CWE-400](#) | CVEs in KEV: 0 | Rank Last Year: 37 (up 13) ▲

Path Traversal (CWE-22)

CWE 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 ▾

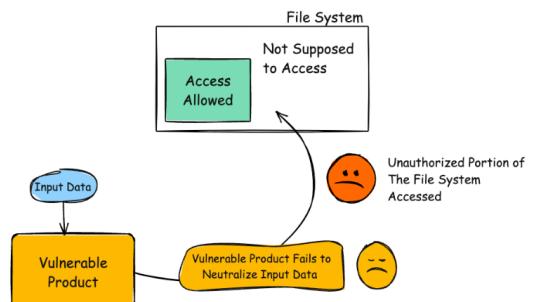
CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

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.

```
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 */
        }
    }
}
```

```
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
```

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

public class DownloadFile {
    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
```

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

public class DownloadFile {
    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()) {
            res.setHeader("Content-Type", "text/css");
            res.getWriter().println("css/basic.css");
            return;
        }

        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 DownloadFile {
    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()) {
            res.getWriter().println("Path is empty or null");
            return;
        }

        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 DownloadFile {
    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()) {
            res.getWriter().println("Path is required!");
            return;
        }

        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

css/basic.css

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

Reading File

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

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

    public static void main(String[] args) {
        try {
            String path = "SECRET.txt";
            File file = new File(BASE_DIR + path);
            if (!file.exists() || !file.isFile()) {
                System.out.println("File not found: " + path);
                return;
            }
            resp.setHeader("Content-Disposition", "attachment; filename=" + file.getName());
            try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
                // Read the file...
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

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

```
/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 DownloadFile {
    private static final String BASE_DIR = "/var/www/public-files/";

    public static void main(String[] args) {
        try {
            String path = "SECRET.txt";
            String url = "GET /download?path=.." + path;
            URL urlObject = new URL(url);
            HttpURLConnection connection = (HttpURLConnection) urlObject.openConnection();
            connection.setRequestMethod("GET");
            connection.connect();

            int responseCode = connection.getResponseCode();
            if (responseCode == 200) {
                BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
                String inputLine;
                while ((inputLine = in.readLine()) != null) {
                    System.out.println(inputLine);
                }
                in.close();
            } else {
                System.out.println("HTTP error code: " + responseCode);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Malicious data

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

```
/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 DownloadFile {
    private static final String BASE_DIR = "/var/www/public-files/";

    public static void main(String[] args) {
        try {
            String path = "SECRET.txt";
            File file = new File(BASE_DIR + path);
            if (file.exists() && file.isFile()) {
                System.out.println("File found: " + file.getAbsolutePath());
            } else {
                System.out.println("File not found: " + file.getAbsolutePath());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
        String requested = req.getParameter("path");
        if (requested == null || requested.isEmpty()) {
            res.getWriter().println("No path specified");
            return;
        }

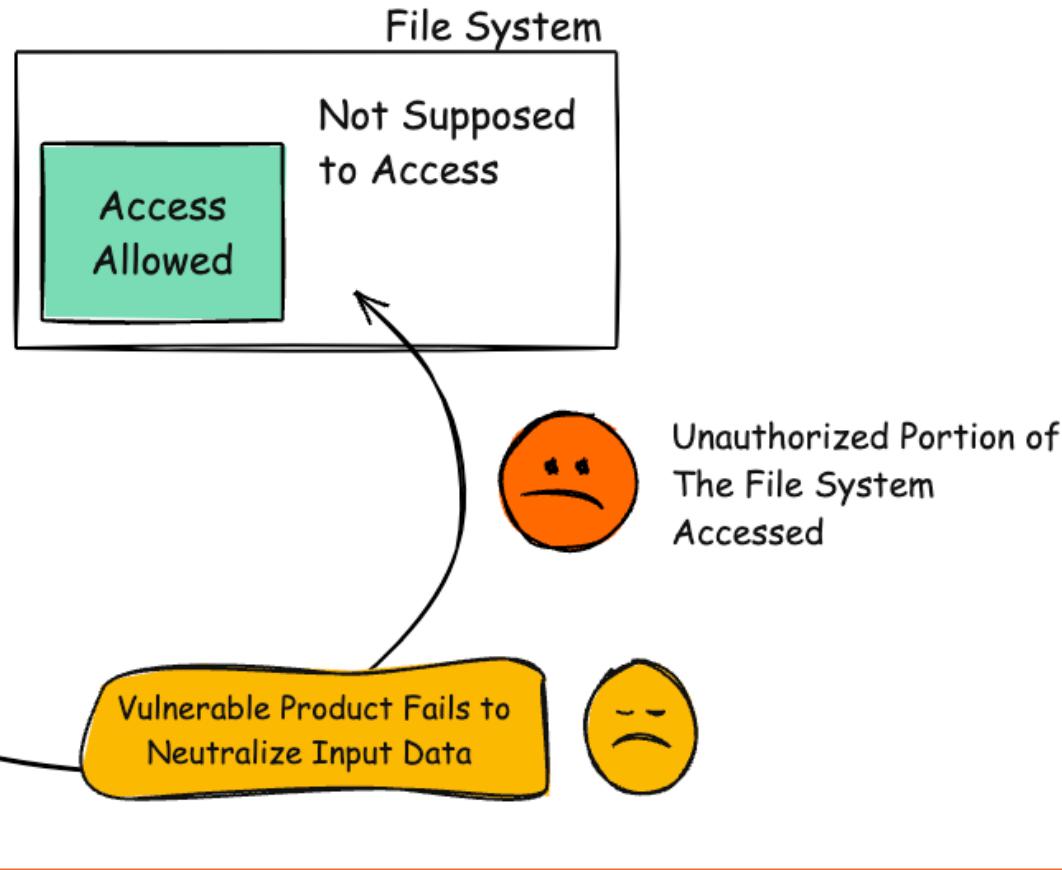
        File file = new File(BASE_DIR + File.separator + requested);
        if (file.exists() && file.isFile()) {
            resp.setHeader("Content-Disposition", "attachment; filename=" + file.getName());
            try (InputStream in = new FileInputStream(file)) {
                // Read the file...
            }
        } 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 FileServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String fileName = request.getParameter("file");
        File file = new File(fileName);
        if (file.exists()) {
            response.setContentType("application/octet-stream");
            response.setHeader("Content-Disposition", "attachment; filename=" + file.getName());
            try {
                FileInputStream fis = new FileInputStream(file);
                byte[] buffer = new byte[1024];
                int length;
                while ((length = fis.read(buffer)) > 0) {
                    response.getOutputStream().write(buffer, 0, length);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else {
            response.getWriter().println("File not found");
        }
    }
}
```



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

res.../SECRET.txt {

Fix Attempt #1

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

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

    public static void main(String[] args) {
        try {
            GET /download?path=../SECRET.txt
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @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 */
        }
    }
}
```

Malicious data

The diagram shows a directory structure under /var/www/public-files/. It contains index.html, basic.css, basic.js, and SECRET.txt. A red box highlights the SECRET.txt file.

Fix Attempt #1

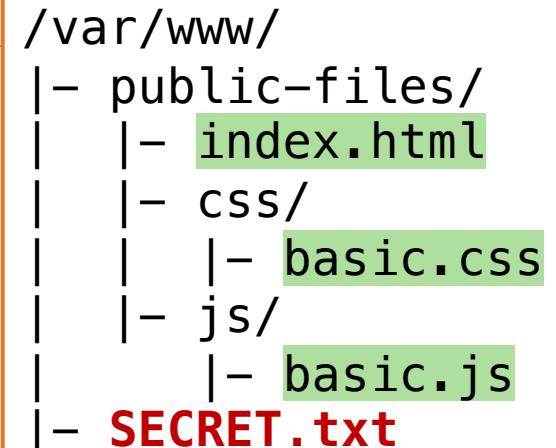
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

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

    public static void main(String[] args) {
        try {
            GET /download?path=../SECRET.txt
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

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

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

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

    public static void main(String[] args) {
        try {
            String path = args[0];
            File file = new File(BASE_DIR + path);
            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 */
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Benign data

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

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

Fix Attempt #1

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

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

    public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        String requested = req.getParameter("path");
        if (requested == null || requested.isEmpty()) { /* Benign input blocked by ".." check */
            if (requested.contains("..") || requested.contains("\0")) { /* ERROR */ }
            res.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");
            try (InputStream in = new FileInputStream(file); OutputStream out = resp.getOutputStream()) {
                // Read the file...
            }
        } else {
            /* ERROR */
        }
    }
}
```

The diagram illustrates the directory structure and the benign input attempt:

- The directory structure is shown as `/var/www/public-files/`. It contains `index.html`, `css/basic.css`, `js/basic.js`, and a file named `SECRET.txt`.
- A green box highlights the URL `GET /download?path=css/..../index.html`, which is labeled `Benign data`.
- An orange callout box points from the URL to the code line `if (requested == null || requested.isEmpty())`, with the text `Benign input blocked by ".." check`.

Fix Attempt #2

```
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 */
        }
    }
}
```

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

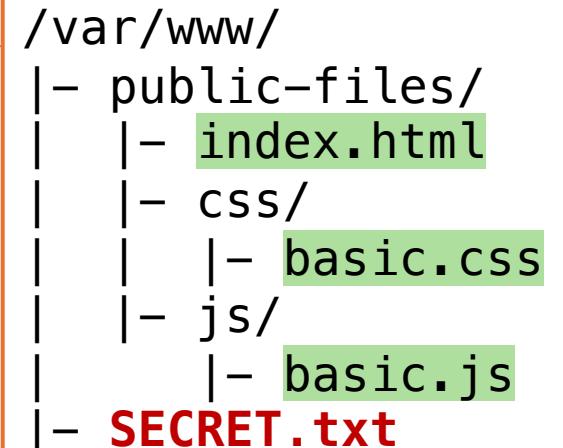
Fix Attempt #2

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

public class VulnerableFileServlet extends HttpServlet {
    @Override
    protected void doGET(HttpServletRequest req, HttpServletResponse resp)
        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

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

public class VulnerableFileServlet extends HttpServlet {
    @Override
    protected void doGET(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        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 */
            }
        }
    }
}
```

The diagram illustrates the directory structure and the flow of the fix attempt:

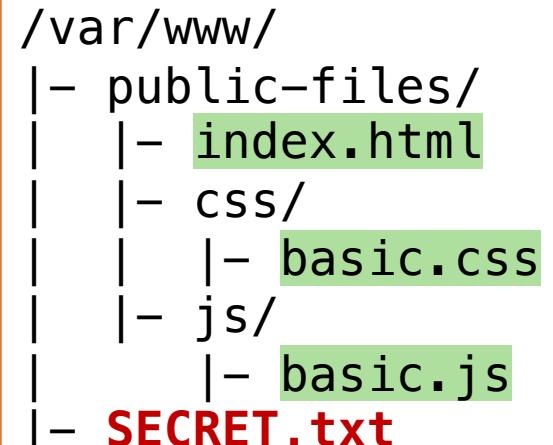
- Directory Structure:** The root directory is `/var/www/`, which contains a `public-files/` folder. Inside `public-files/`, there is an `index.html` file and a `SECRET.txt` file. The `public-files/` folder also contains `css/` and `js/` subfolders, each containing a `basic.css` and `basic.js` file respectively.
- Request Flow:** A user sends a `GET /download?path=css/../index.html` request to the `VulnerableFileServlet`.
- Path Resolution:** The servlet extracts the `requested` parameter value (`"css/../index.html"`). It then constructs a `Path` object starting from the `base` (`/var/www/public-files`) and resolving the `requested` path relative to it. The resulting `candidate` path is `/var/www/public-files/index.html`.
- Validation:** The code checks if the `candidate` path starts with the `baseReal` path (`/var/www/public-files`). Since it does not, it triggers an `ERROR`.
- Output:** The code then checks if the `candidate` path exists and is a file. If so, it sets the `Content-Disposition` header to attach the file and reads it into the response stream.

Fix Attempt #2

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

public class VulnerableFileServlet extends HttpServlet {
    @Override
    protected void doGET(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        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.startsWith(baseReal)) {
            resp.setContentType("text/html");
            try {
                // Read the file...
            }
        } else {
            /* ERROR */
        }
    }
}
```



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

YES

Fix Attempt #2

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

public class VulnerableFileServlet extends HttpServlet {    Malicious data
    @Override
    protected void doGET(HttpServletRequest req, HttpServletResponse res) throws ServletException {
        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 */
        }
    }
}
```



The diagram shows a directory structure under `/var/www/public-files`. The structure is as follows:

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

Fix Attempt #2

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

public class VulnerableFileServlet extends HttpServlet {
    @Override
    protected void doGET(HttpServletRequest req, HttpServletResponse resp) 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 */
        }
    }
}
```

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

/var/www/SECRET.txt



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

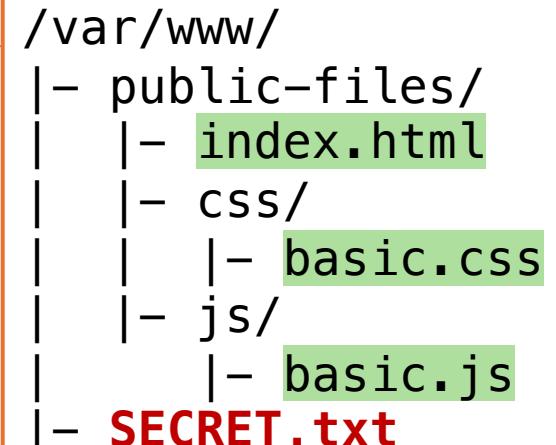
Fix Attempt #2

```
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()) { /* Malicious data */ }

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

        if (candidate.toFile().exists()) {
            resp.setContentType("text/plain");
            try (OutputStream os = res.getOutputStream()) {
                // Read the file...
            }
        } else {
            /* ERROR */
        }
    }
}
```



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

/var/www/SECRET.txt

NO

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 */ }  
    }  
}
```



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 */ }  
    }  
}
```



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 */ }  
    }  
}
```



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 */ }  
    }  
}
```



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()) {  
  
            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



Agenda

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



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"
```

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<UnsafeDeserialization>
from PathNode source, PathNode sink
where flowPath(source, sink)
select sink.getNode().(UnsafeDeserialization of $@, source)
    "Unsafe deserialization of $@.", source
```

QL: Object-oriented Queries on Relational Data

Pavel Avgustinov¹, Oege de Moor², Michael Peyton Jones³, and Max Schäfer⁴

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 Avgustinov¹, Oege de Moor², Michael Peyton Jones³, and Max Schäfer⁴

1 Semml
2 Semml
3 Semml
4 Semml

```
Unsafe
import
from PathNode source, Path
where flowPath(source, sin
select sink.getNode().(Uns
"Unsafe deserialization
```

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.

QL: Object-oriented Queries on Relational Data

Pavel
Max S

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

1 Sen terms: c
2 Sen virtual c
3 Sen receiver.
4 Sen rather th
queries b
efficiently
static an
Unsafe deserialization or seq., s

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)

The screenshot shows a GitHub repository page for the 'codeql' repository. The repository path is 'github / codeql'. The main navigation bar includes 'Code' (selected), 'Issues' (897), 'Pull requests' (352), 'Discussions', 'Actions', 'Projects', 'Models', and 'Security'. The current branch is 'main'. The breadcrumb navigation shows 'codeql / java / ql / src / Security / CWE / CWE-022 /'. A search bar at the top right contains the placeholder 'Type / to search'. A dropdown menu icon is also present.

A commit by 'erik-krogh' is highlighted with the message 'delete imports to a deleted file'.

Name	Last commit message
..	
examples	add a trailing slash to the folder check in the QHelp for java/path-i...
TaintedPath.qhelp	add missing </p>
TaintedPath.ql	delete imports to a deleted file
ZipSlip.qhelp	move the examples for the qhelps into an example/ folder
ZipSlip.ql	Adjust ZipSlip query description according to review suggestions.

```
1  /**
2   * @name Uncontrolled data used in path expression
3   * @description Accessing paths influenced by users can allow an attacker to access unexpected resources.
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/lib/semmle/code/java/security/TaintedPathQuery.ql

```
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"
```

```
/**
 * 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>;
```

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

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

```
1  /**
2   * @name Uncontrolled data used in path
3   * @description Accessing paths influencing
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   * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
17   */
18
19 module TaintedPathConfig implements DataFlow::ConfigSig {
20     predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }
21
22     predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }
23
24     predicate isBarrier(DataFlow::Node sanitizer) {
25         sanitizer instanceof SimpleTypeSanitizer or
26         sanitizer instanceof PathInjectionSanitizer
27     }
28
29     predicate isAdditionalFlowStep(DataFlow::Node n1, DataFlow::Node n2) {
30         any(TaintedPathAdditionalTaintStep s).step(n1, n2)
31     }
32
33     predicate observeDiffInformedIncrementalMode() { any() }
34 }
```

- **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 influenced by uncontrolled data
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  • Tainted input: malicious file paths
16  • Attack surface: internet http request, untrusted files (e.g., Zip)
17  • Vulnerability Manifestation: reading/writing files (e.g., InputStream, OutputStream, mkdir)
18  • Absence of Sanitization: no “resolve” or “normalize” from tainted input to manifestation
19
20
21 where TaintedPathFlow:::flowPath(source, sink)
22 select sink.getNode(), source, sink, "This path depends on a $0.", source.getNode(),
23     "user-provided value"

```

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

```
1  /**
2   * @name Uncontrolled data used in path
3   * @description Accessing paths influenced by uncontrolled data
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   * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
17
18  module TaintedPathConfig implements DataFlow::ConfigSig {
19
20    predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }
21
22    predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }
23
24    predicate isBarrier(DataFlow::Node sanitizer) {
25      sanitizer instanceof SimpleTypeSanitizer or
26      sanitizer instanceof StringSanitizer
27    }
28
29    predicate isAdditionalSink(DataFlow::Node sink) {
30      any(TaintedPathAdditionalSinkExtension e) {
31        e.addsTo(sink)
32      }
33    }
34
35    predicate observed(DataFlow::Node node) {
36      any(TaintedPathObservedExtension e) {
37        e.observes(node)
38      }
39    }
40
41    extensions:
42      - addsTo:
43          pack: codeql/java-all
44          extensible: sourceModel
45
46      data:
47        - ["javax.servlet.http", "Cookie", False, "getComment", "()", "", "ReturnValue", "remote", "manual"]
48        - ["javax.servlet.http", "Cookie", False, "getName", "()", "", "ReturnValue", "remote", "manual"]
49        - ["javax.servlet.http", "Cookie", False, "getValue", "()", "", "ReturnValue", "remote", "manual"]
50        - ["javax.servlet.http", "HttpServletRequest", False, "getHeader", "(String)", "", "ReturnValue", "remote", "manual"]
51        - ["javax.servlet.http", "HttpServletRequest", False, "getHeaderNames", "()", "", "ReturnValue", "remote", "manual"]
52        - ["javax.servlet.http", "HttpServletRequest", False, "getHeaders", "(String)", "", "ReturnValue", "remote", "manual"]
53        - ["javax.servlet.http", "HttpServletRequest", False, "getParameter", "(String)", "", "ReturnValue", "remote", "manual"]
54        - ["javax.servlet.http", "HttpServletRequest", False, "getParameterMap", "()", "", "ReturnValue", "remote", "manual"]
55        - ["javax.servlet.http", "HttpServletRequest", False, "getParameterNames", "()", "", "ReturnValue", "remote", "manual"]
56        - ["javax.servlet.http", "HttpServletRequest", False, "getParameterValues", "(String)", "", "ReturnValue", "remote", "manual"]
57        - ["javax.servlet.http", "HttpServletRequest", False, "getPathInfo", "()", "", "ReturnValue", "remote", "manual"]
58        - ["javax.servlet.http", "HttpServletRequest", False, "getQueryString", "()", "", "ReturnValue", "remote", "manual"]
59        - ["javax.servlet.http", "HttpServletRequest", False, "getRemoteUser", "()", "", "ReturnValue", "remote", "manual"]
60        - ["javax.servlet.http", "HttpServletRequest", False, "getRequestURI", "()", "", "ReturnValue", "remote", "manual"]
61        - ["javax.servlet.http", "HttpServletRequest", False, "getRequestURL", "()", "", "ReturnValue", "remote", "manual"]
62        - ["javax.servlet.http", "HttpServletRequest", False, "getServletPath", "()", "", "ReturnValue", "remote", "manual"]
```

- **Tainted input:** malicious file paths
- **Attack surface:** internet http request, untrusted user input
- **Vulnerability Manifestation:** reading/writing to disk, database, network, memory, etc.
- **Absence of Sanitization:** no “resolve” or “normalize” from tainted input to manifestation

```

1  /**
2   * @name Uncontrolled data used in path
3   * @description Accessing paths influenced by uncontrolled data
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

```

```

/** 
 * 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

```

21 where TaintedPathFlow::flowPath(source, sink)
22 select sink.getNode(), source, sink, "This path depends on a $0.", source.getNode(),
23     "user-provided value"

```

```

1  /**
2   * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
3   */
4
5 module TaintedPathConfig implements DataFlow::ConfigSig {
6     predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }
7
8     predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }
9
10
11 extensions:
12 - addsTo:
13   pack: codeql/java-all
14   extensible: sinkModel
15
16 data:
17 - ["java.io", "File", True, "canExecute", "()", "", "Argument[this]", "path-injection", "manual"]
18 - ["java.io", "File", True, "canRead", "()", "", "Argument[this]", "path-injection", "manual"]
19 - ["java.io", "File", True, "canWrite", "()", "", "Argument[this]", "path-injection", "manual"]
20 - ["java.io", "File", True, "createNewFile", "()", "", "Argument[this]", "path-injection", "ai-manual"]
21 - ["java.io", "File", True, "createTempFile", "(String,String,File)", "", "Argument[2]", "path-injection", "ai-manual"]
22
23 - ["java.io", "FileInputStream", True, "FileInputStream", "(File)", "", "Argument[0]", "path-injection", "ai-manual"]
24 - ["java.io", "FileInputStream", True, "FileInputStream", "(FileDescriptor)", "", "Argument[0]", "path-injection", "manual"]
25 - ["java.io", "FileInputStream", True, "FileInputStream", "(String)", "", "Argument[0]", "path-injection", "ai-manual"]
26 - ["java.io", "FileOutputStream", False, "FileOutputStream", "", "", "Argument[0]", "path-injection", "manual"]
27 - ["java.io", "FileOutputStream", False, "write", "", "", "Argument[0]", "file-content-store", "manual"]

```

13 * external/cwe/cwe-073

- **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

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

```

1  /**
2   * @name Uncontrolled data used in path
3   * @description Accessing paths influenced by uncontrolled data
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   * A taint-tracking configuration for tracking flow from remote sources to the creation of a path.
17   */
18
19 module TaintedPathConfig implements DataFlow::ConfigSig {
20     predicate isSource(DataFlow::Node source) { source instanceof ActiveThreatModelSource }
21
22     predicate isSink(DataFlow::Node sink) { sink instanceof TaintedPathSink }
23
24     predicate isBarrier(DataFlow::Node sanitizer) {
25         sanitizer instanceof SimpleTypeSanitizer or
26         sanitizer instanceof PathInjectionSanitizer
27     }
28
29     predicate isAdditionalFlowStep(DataFlow::Node n1, DataFlow::Node n2) {
30         any(TaintedPathAdditionalTaintStep s).step(n1, n2)
31     }
32
33     predicate observeDiffInformedIncrementalMode() { any() }
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
15
```

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

```
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

```
5     * @problem.severity error  
6     * @security-severity 7.5  
7     * @precision high  
8     * @id java/path-injection  
9     * @tag  
10    *  
11    *  
12    *  
13    *  
14    */  
15  
16    import  
17    import  
18    import  
19    from T  
20    where T  
21    select T  
22    }  
23    "user-provided value"  
  
    /** Tracks flow from remote sources to the creation of a path. */  
    module TaintedPathFlow = TaintTracking::Global<TaintedPathConfig>;  
  
    class ExactPathMatchSanitizer extends PathInjectionSanitizer {  
        ExactPathMatchSanitizer() {  
            this = DataFlow::BarrierGuard<exactPathMatchGuard/3>::getABarrierNode()  
        }  
    }
```

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

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

```
5   * @problem.severity error
6   * @security-severity 7.5
7   * @precision high
8   * @id ...
9   * @tag ...
10  *
11  */
12  */
13  */
14  */
15
16 import ...
17 import ...
18 import ...
19
20 from T...
21 where ...
22 select ...
23 "user-provided value"
```

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

One of the sanitizers: Exact Path Match

```
1 /**
2  * A sanitizer that protects against path injection vulnerabilities
3  * by checking for a matching path.
4  */
5
6 class ExactPathMatchSanitizer extends PathInjectionSanitizer {
7     ExactPathMatchSanitizer() {
8         this = DataFlow::BarrierGuard<exactPathMatchGuard/3>::getABarrierNode()
9     }
10 }
```

```

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

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

```

5   * @problem.severity error
6   * @security-severity 7.5
7   * @precision high
8   * @id ...
9   * @taint ...
10  */
11  /**
12   * Tracks flow from remote sources to the creation of a path.
13  */
14  module TaintedPathFlow = TaintTracking::Global<TaintedPathConfig>;

```

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

```

15
16  import ...
17  import ...
18  import ...
19  exists(Expr previousGuard |
20    localTaintFlowToPathGuard(e, g) and
21    previousGuard.(PathNormalizeSanitizer).getBasicBlock() == g.getBasicBlock()
22  )
23
24  .controls(g.getBasicBlock(), previousGuard.(PathTraverser).getBranch())
25
26  }

```

codeql/java/

```

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

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

```

5   * @problem.severity error
6   * @security-severity 7.5
7   * @precision high
8   * @id java-path-injection
9   * @tags security

```

```

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

```

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

```

10  *
11  * @problem.severity error
12  * @security-severity 7.5
13  * @precision high
14  * @id java-path-injection
15  * @tags security
16  *
17  import java.util.List;
18  import java.util.ArrayList;
19  import java.util.stream.Collectors;
20  from TaintedPathFlow
21  where TaintedPathConfig config = DataFlow::getGlobalConfig()
22  select config.getGuardFactory()
23  "use GuardFactory"

```

```

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 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

Z
U
1

Section 3.2 Candidate Extractor

External API Candidates

package	class	method
java.io	ObjectInputStream	readObject
javax.v...	ConstraintValida...	buildConstra...
		...

Internal Function Param Candidates

class	method	documentation
FieldExpression	accept	"Accept a visitor..."
CronParser	parse	"Parse string..."
		...

Section 3.3 Label Source/Sinks

API Labels

type	target
source	return-value
sink	argument[0]
	...

Function Param Labels

type	target
source	parameter[0]
sink	parameter[0]
	...

Section 3.4 Static Taint Analysis

Data-Flow Path

```

graph TD
    A(( )) --> B(( ))
    B --> C(( ))
    C --> D(( ))
    D --> E(( ))
    E --> F(( ))
    F --> G(( ))
    G --> H(( ))
    H --> I(( ))
    I --> J(( ))
    J --> K(( ))
    K --> L(( ))
    L --> M(( ))
    M --> N(( ))
    N --> O(( ))
    O --> P(( ))
    P --> Q(( ))
    Q --> R(( ))
    R --> S(( ))
    S --> T(( ))
    T --> U(( ))
    U --> V(( ))
    V --> W(( ))
    W --> X(( ))
    X --> Y(( ))
    Y --> Z(( ))
    Z --> A
  
```

boolean isValid(
 String value, ...)
 {
 Cron parse(
 String expr) {
 ...
 throw new IllegalA...
 ...
 context.buildConstra...
 intViolationWith
 Template(e.get...())

Section 3.5 Contextual Analysis

Vulnerability Analysis

Is Vulnerable: The source is user input being parsed and validated, which can lead to an exception with a controlled message. The sink is using the exception message in a context that could potentially be used for execution, satisfying the criteria for a Code Injection vulnerability (CWE-094).

IDE

ZU1

Section 3.2
Candidate Extractor

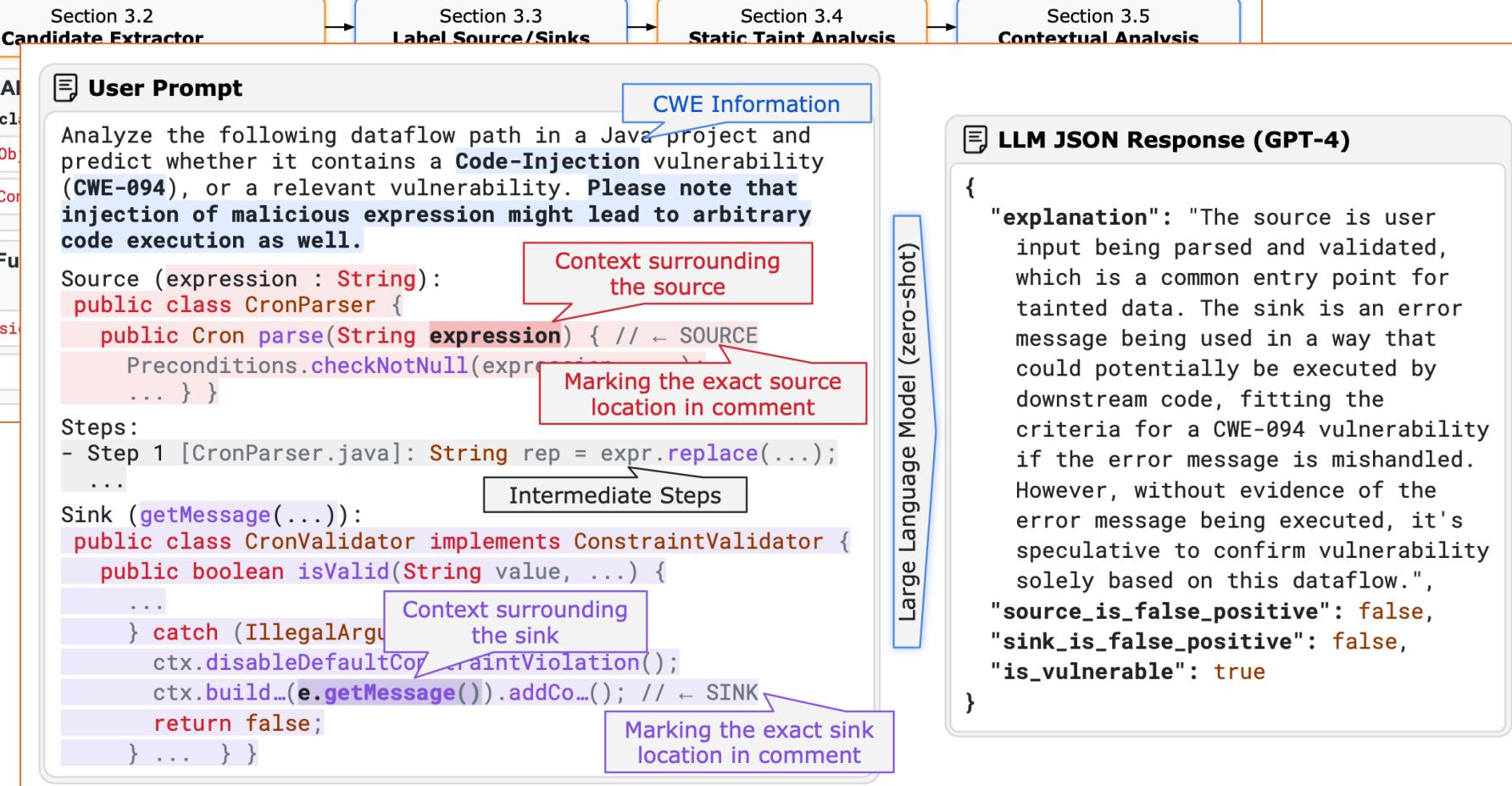
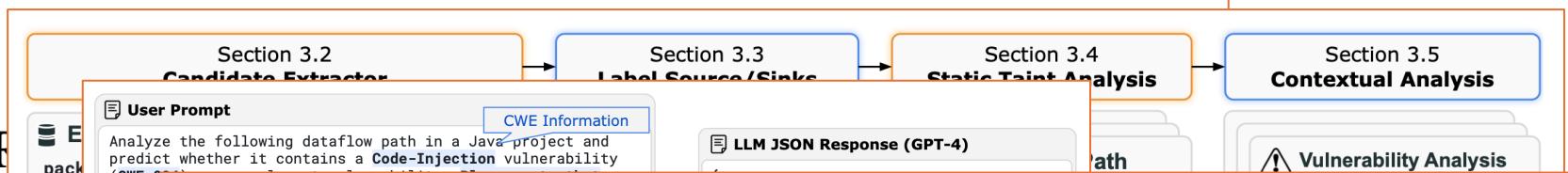
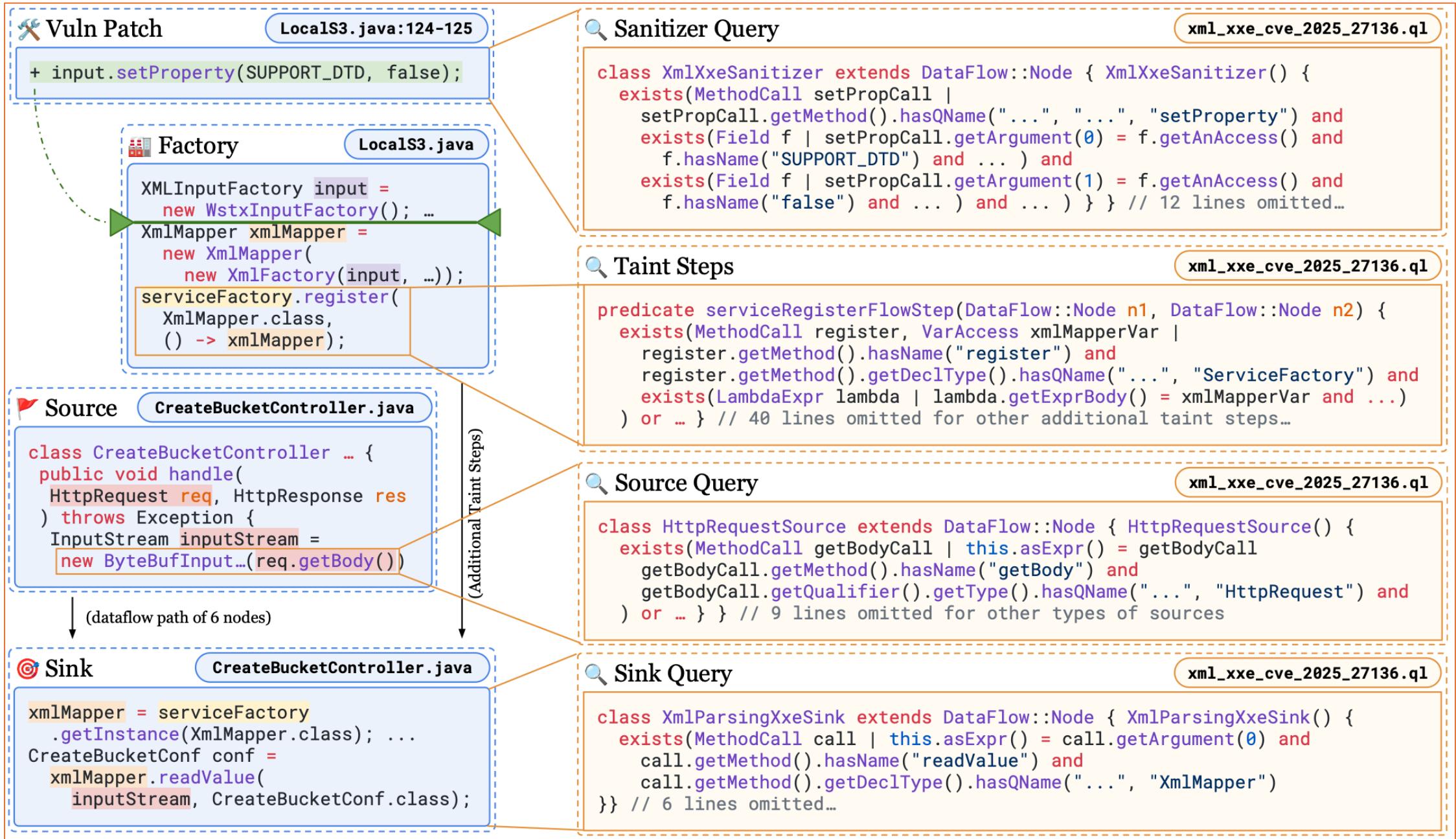


Table 1: Overall performance comparison of CodeQL vs IRIS on Detection Rate (\uparrow), Average FDR (\downarrow), and Average F1 (\uparrow). We present results of IRIS with LLMs including GPT-4 and GPT-3.5, L3 8B and 70B, Q2.5C 32B, G2 27B, and DSC 7B.

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





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: http servlet-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
```

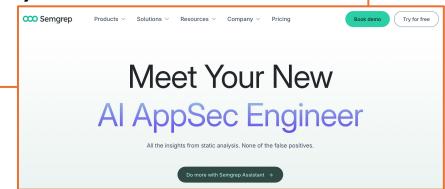


- **Tainted input**: malicious file paths
- **Attack surface**: internet http request, untrusted files (e.g., Zip)
- **Vulnerability Manifestation**: reading/writing files (e.g., InputStream, OutputStream, mkdir)
- **Absence of Sanitization**: no “resolve” or “normalize” from tainted input to manifestation

```

rules:
  - id: http servlet-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 Natural Language

BEN LIMPANUKORN*, University of California, Los Angeles
 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

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: <i>"Find all cases where value returned by Integer.toString is used by an add operation inside a for loop"</i>		
	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: <i>"Find all cases where value returned by Integer.toString is used by an add operation inside a for loop"</i>		
	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: <i>"Find all cases where equals is called on an empty string."</i>		
	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: <i>"Find all cases where a String object is used by an add operator within a for loop"</i>		
	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 LI
YANJU
ZACH
PRANA
MURA
XIAOF
ANOO
MIRYU

Query Language	Searchable predicates	Language constructs	Expressivity
Comby	for(...){...}, :hole = ..., :hole[=z](...),...	where [:a] == [:b], where match [:a] { ... }, ...	Data-flow approx. Control-flow: Yes Inter-proc: No

a, Los Angeles, USA

Comby	NL Query: "I by an add op DSL Query: for(...) { for(...){...} \$X = ...; \$Call(...); ... }
Semgrep	NL Query: "I by an add op DSL Query: pattern: \$X pattern-ins: hasName(...) hasQualifie getIntValue ... NL Query: "I DSL Query: from Method ma.getMethod ma.getArgume select ma. withMethod withDataBy withDataDep Transform NL Query: "I within a for i DSL Query: new CustomR .withContro .withActio .withArgume .withDataBy .check().bu
CodeQL	/* Filtering the context to While statements */ .withControlFilter("WHILE_STATEMENT") .withAnyOf(b -> b.withInNodesTransform(e -> resolveDummy(e)), b -> b) /* Transforming to the loop's body */ .withContextNodesTransform(ContextKind.LOOP) .withAnyOf(b -> b.withInNodesTransform(e -> resolveDummy(e)), b -> b) /* Filtering for method calls */ .withNodeByTypeFilter(EGroumASTNodeType.METHOD_INVOCATION) .withAnyOf(b -> b.withInNodesTransform(e -> resolveDummy(e)), b -> b) /* Checking if the method call is 'nextToken' with no arguments */ .withMethodCallFilter("java\\.util\\.StringTokenizer\\.nextToken") .withNumberOfArgumentsFilter(0) .withAnyOf(b -> b.withInNodesTransform(e -> resolveDummy(e)), b -> b)
GQL	

Natural Language Query

Identify all instances of method calls named 'nextToken' that are fully qualified as 'java.util.StringTokenizer.nextToken', have no arguments, and are contained within the body of a while loop.

Predicted GQL Query

```
/* Filtering the context to While statements */  
.withControlFilter("WHILE_STATEMENT")  
.withAnyOf(b -> b.withInNodesTransform(e -> resolveDummy(e)), b -> b)  
/* Transforming to the loop's body */  
.withContextNodesTransform(ContextKind.LOOP)  
.withAnyOf(b -> b.withInNodesTransform(e -> resolveDummy(e)), b -> b)  
/* Filtering for method calls */  
.withNodeByTypeFilter(EGroumASTNodeType.METHOD_INVOCATION)  
.withAnyOf(b -> b.withInNodesTransform(e -> resolveDummy(e)), b -> b)  
/* Checking if the method call is 'nextToken' with no arguments */  
.withMethodCallFilter("java\\.util\\.StringTokenizer\\.nextToken")  
.withNumberOfArgumentsFilter(0)  
.withAnyOf(b -> b.withInNodesTransform(e -> resolveDummy(e)), b -> b)
```

Matches (Total = 12)

- File: **1415502.java**, Line 146: String t = st.nextToken();
- File: **1948766.java**, Line 297: String str = st.nextToken();
- File: **1948766.java**, Line 27: variant += sep + tokens.nextToken();
- ...

Natural Language Query

Identify all instances of the `!` operator in the code.

Predicted GQL Query

```
/* Checks that the entity is a ! operator. */  
.withActionFilter(true, "!")  
/* Resolves dummy data nodes to their respective actions. */  
.withAnyOf(b -> b.withInNodesTransform(e -> resolveDummy(e)), b -> b)
```

Matches (Total = 12)

- File: **2319745.java**, Line 65: !action.equals(other.action))
- File: **2319745.java**, Line 68: other.typePatten != null
- File: **922483.java**, Line 48: !super.equals(obj)
- ...

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

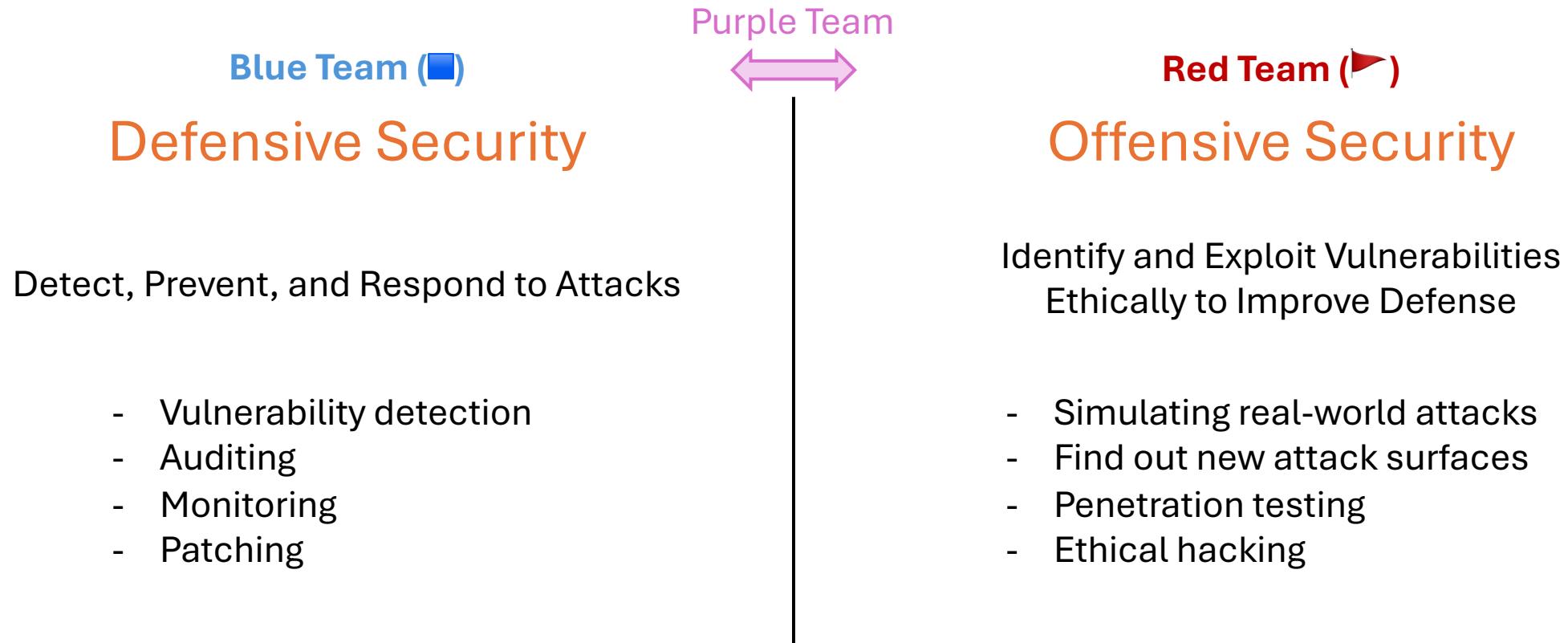
Red Team (🚩)

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



```
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
```

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

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

    public static void main(String[] args) {
        String payload = "GET /download?path=../SECRET.txt";
        System.out.println(payload);
    }

    @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 */
        }
    }
}
```

Payload

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

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

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.

PoC_{GEN}: 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

PoC_{GEN}: Generating Proof-of-Concept Exploits for Vulnerabilities in Npm Packages

DENIZ SIMSEK, University of Stuttgart, Germany

ARYA Z FGHRAII, University of Stuttgart, Germany

M

Fig. 2. PoC_{GEN}-generated PoC exploit for CVE-2024-36751.

```
1 async function exploit() {  
2     const parseuri = require("parse-uri");  
3     // This input is designed to cause excessive backtracking in the regex  
4     const craftedInput = 'http://example.com/' + 'a'.repeat(30000) + '?key=value';  
5     const result = await parseuri(craftedInput);  
6 }  
7 await exploit();
```

PoC_{GEN}: Generating Proof-of-Concept Exploits for Vulnerabilities

Fig. 2. PoC_{GEN}-generated PoC exploit for CVE-2024-36751.

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

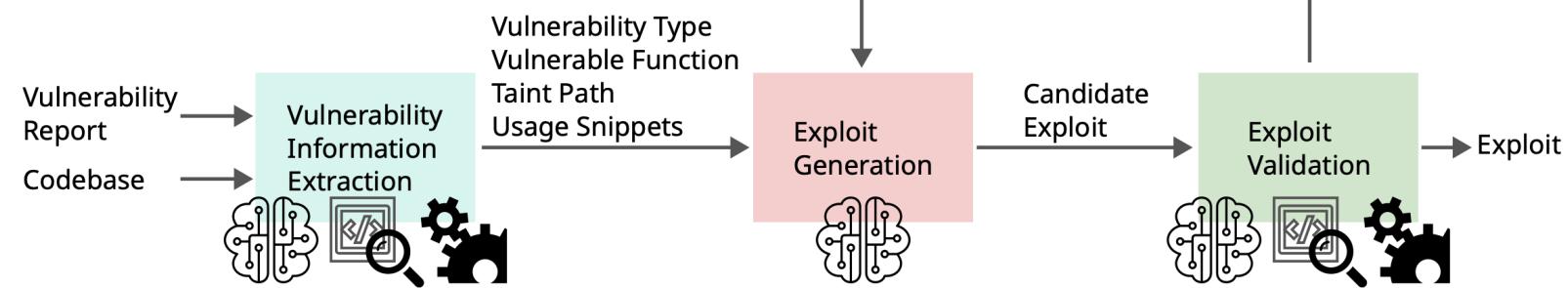
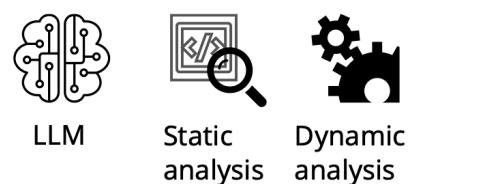


Fig. 3. Overview of PoC_{GEN}.

PoC_{GEN}: Generating Proof-of-Concept Exploits for

Vu

DE
AR
MI

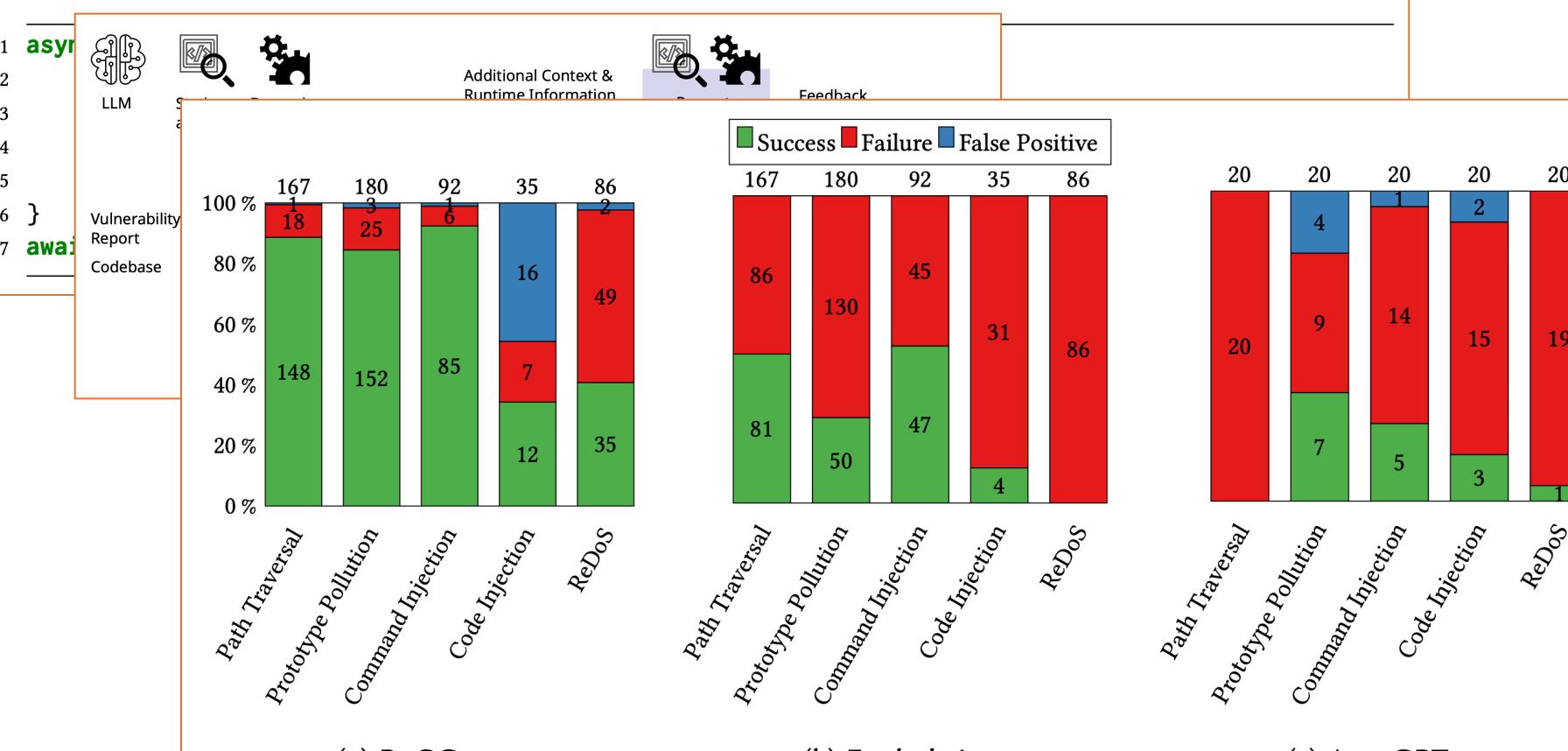


Fig. 8. Effectiveness of PoC_{GEN}, Explode.js, and AutoGPT on SecBench.js.

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*; Yuankang Li, *University of New South Wales*; Yuan Xu, Tianwei Zhang, and Alpen-Adria-Universität Klagenfurt

<https://www>

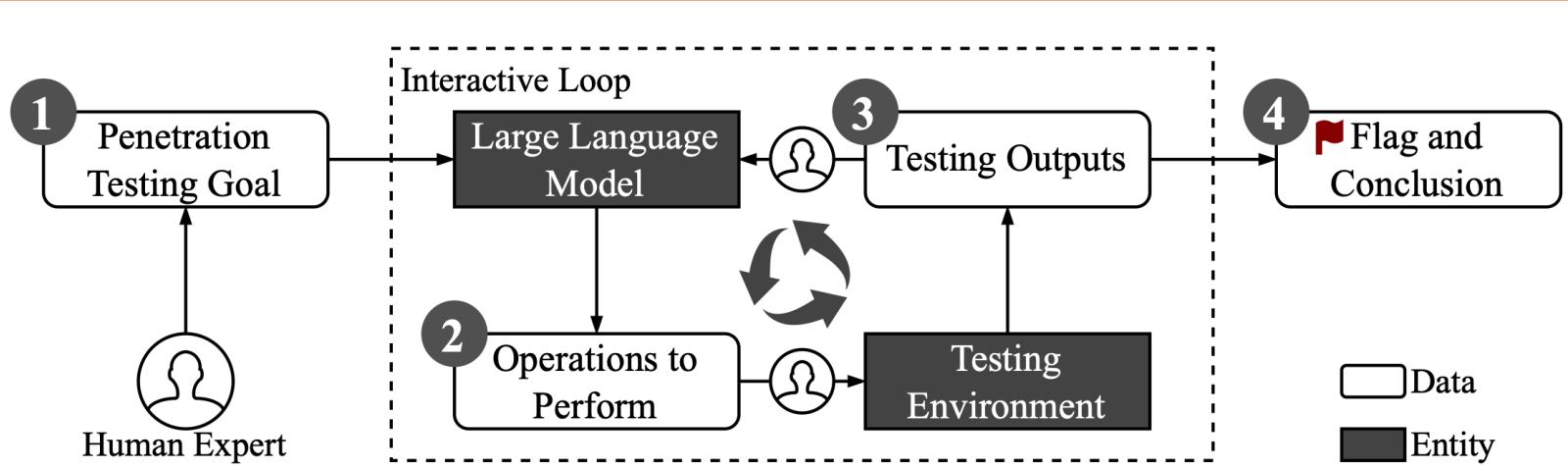


Figure 1: Overview of strategy to use LLMs for penetration testing.

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

Gele
Alias R
Research
Tian
Alp

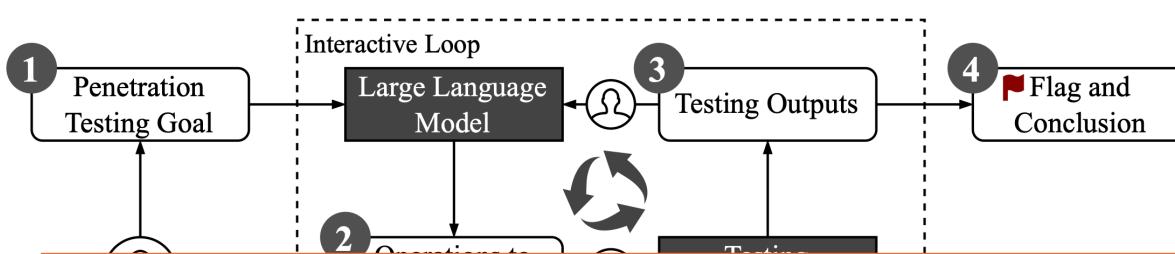


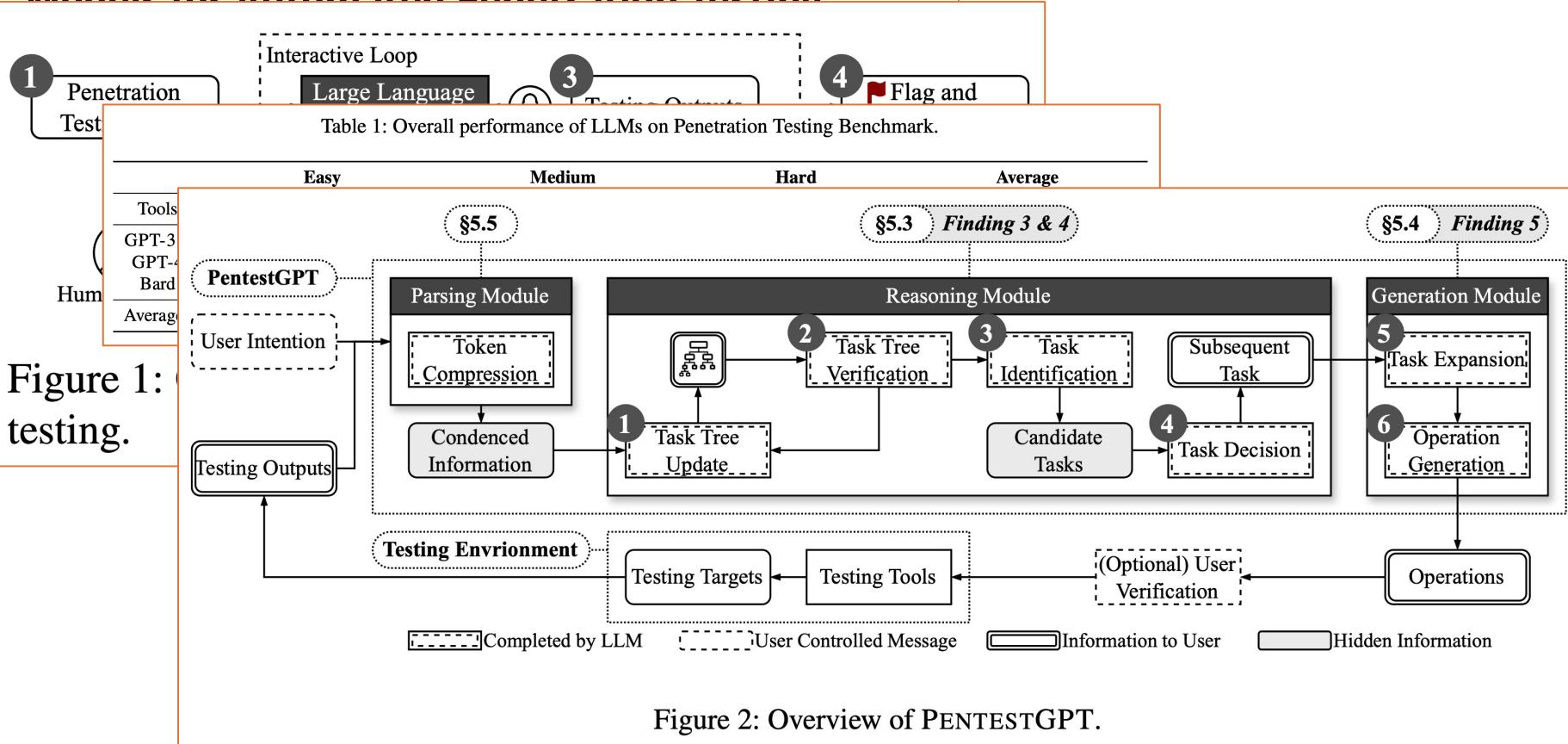
Table 1: Overall performance of LLMs on Penetration Testing Benchmark.

Tools	Easy		Medium		Hard		Average	
	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

Gel
Alias R
Research
Tian
Alp

Figure 1:
testing.



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