Interactive Static Analysis for Early Detection of Software Vulnerabilities

Bill Chu, Jun Zhu

Department of Software and Information Systems

University of North Carolina at Charlotte

billchu, jzhu16@uncc.edu

With contributions from Jing Xie and Heather Lipford

Overview

- Software vulnerabilities is a major contributor to information security problems
- Education and training
  - Gap between conceptual understanding of secure programming and its practice
- Static and dynamic analysis tools
  - Reactive
  - Analysis is performed after application is developed, which renders higher cost to fix the identified issues
  - Used by security professionals, who are not familiar with application details.

Our Approach – Interactive Static Analysis

- A mix-initiative developer-oriented paradigm for interacting with developers to aid in the detection and prevention of vulnerabilities
- Developer in the "security loop"
- Advantages
  - Proactive support for developers with vulnerability detection and prevention
  - Leverage developer' knowledge about application context and logic to drive customized security analysis
  - Identify and fix vulnerabilities at the development phase, which saves a lot of money and time cost

Interactive Static Analysis

- **Interactive data flow analysis:** a programmer would be warned and assisted with properly handling of untrusted data
- **Interactive control flow analysis:** a programmer would be warned and assisted with properly enforcing of security policies, e.g. access control policies

Agenda for this talk

- Quick overview of ASIDE (part of ASIDE talk at APPSEC 2011)
- New results
- Current research

Prototype

- Eclipse plug-in for Java, named **ASIDE**
- Integrate static analysis into the Integrated Development Environment (IDE)
- Design rationales
  - Recognition instead of recalling, a key HCI design principle
  - Minimize extra burden on developer
  - Customize security analysis by soliciting application knowledge from developer (e.g. business logic, application context)
  - Support best secure software development practice, e.g. using trusted library OWASP ESAPI

ASIDE Demo

- ASIDE stands for Application Security in Integrated Development Environment
- Based on Eclipse Java Development Tooling (JDT)

- Two major features
  - Interactive data flow analysis
  - Interactive control flow analysis

Interactive Data Flow Analysis

- Policy-driven
- Take advantage of existing popular data flow analysis algorithms

- Remind and assist developer to fix each taint source through automatic validation/encoding code refactoring
- Instant re-analysis on its related data flow once developer fixes a tainted source

Interactive Control Flow Analysis

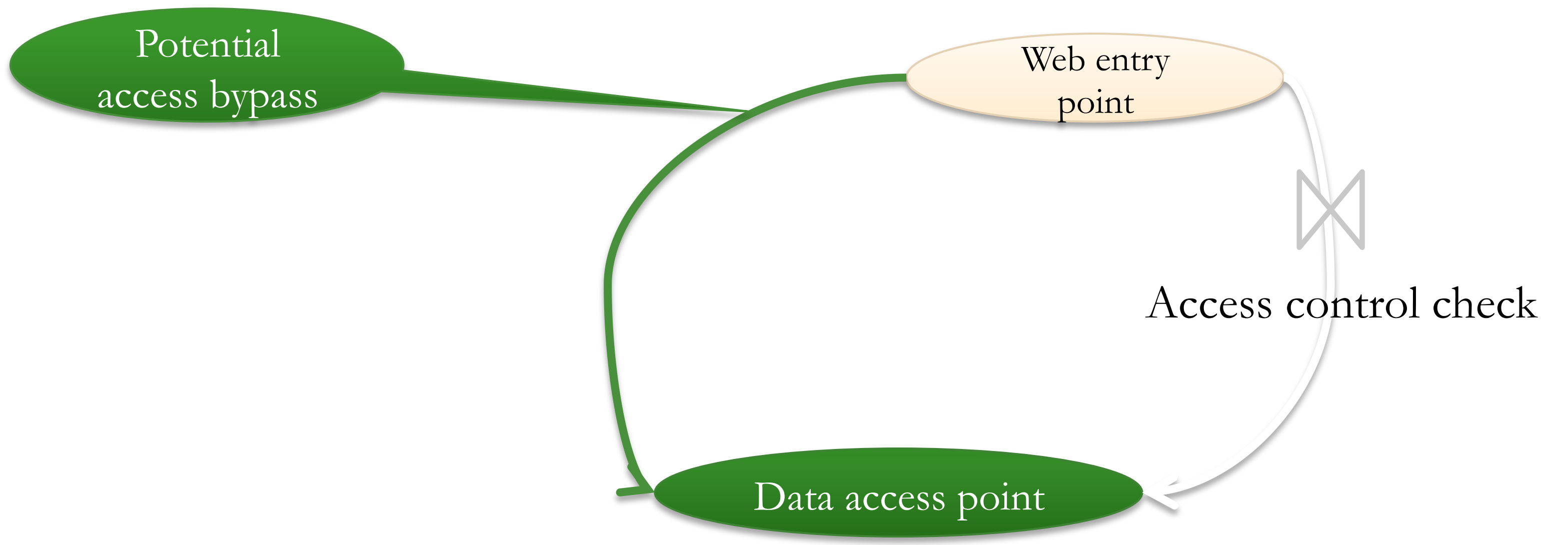
- Identify critical/sensitive operations code
  - Database access functions
  - Identify "use case"/transaction level routines that lead to accessing protected data (e.g. a statement within a Servlet/Action for Java web applications)
- Solicit control logic checks from the developer via point-and-click annotations

Point-and-click Annotation

- Point-and-click fashion to minimize extra burden on developers
- What is a valid annotation?
  - A set of logic tests, or assertion (e.g. Spring Security)
  - On an execution path from web entry to data access point
- Leveraged to drive customized security analysis
- Could be kept for code review

Security Analysis Based on Annotation

- Unchecked access path
  - There might be an execution path from web entry to data access point without access control check



Evaluation on Interactive Data Flow Analysis

- Target Project: Apache Roller 3.0.0
  - 65K+ lines of code
  - Full featured blog server (1.8M+ hits on Google for "powered by Apache Roller")
- Comparison base: Fortify SCA based code review
- This is joint work with John Melton (OWASP AppSensor)

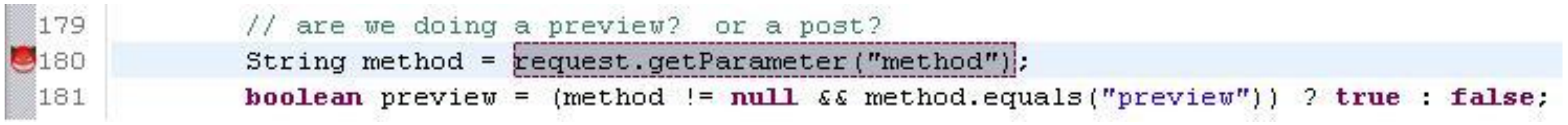
Jing Xie, Bill Chu, Heather Lipford, John Melton "IDE Support for Web Application Security" in 27th *Annual Conference on Application of Computer Security*, 2011 (Acceptance rate: 18%)

Validation/Encoding of Untrusted Data

- 922 Fortify issues caused by 143 taint sources
  - Primitive data type (e.g. *java.lang.String*)
  - Composite data type (e.g. *java.util.Map*)
  - Variables require output encoding always result from untrusted data
- ASIDE identified 131 of 143 (92%) taint sources
- Taint source of composite data type is 41
- 12 issues not detected by ASIDE
  - JSP (not yet implemented in prototype)
  - Framework binding
    - Delayed binding (implementing the Dependency Injection design pattern)

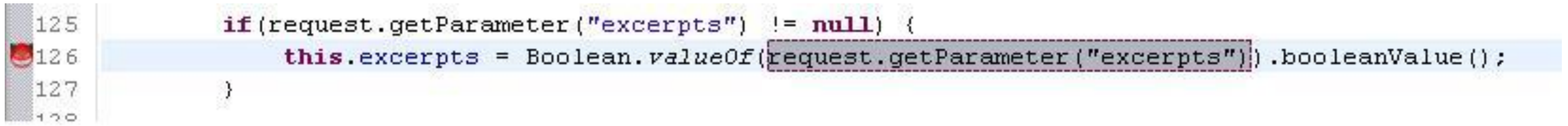
False Positives of ASIDE

- ASIDE reported 118 **more** taint sources of primitive data types
  - Potentially exploitable (94), validate to practice defensive security
  - False positive (24)



```
        // are we doing a preview?  or a post?
        String method = request.getParameter("method");
        boolean preview = (method != null && method.equals("preview")) ? true : false;
```

False Positive Example 1: Untrusted input is used for logic test



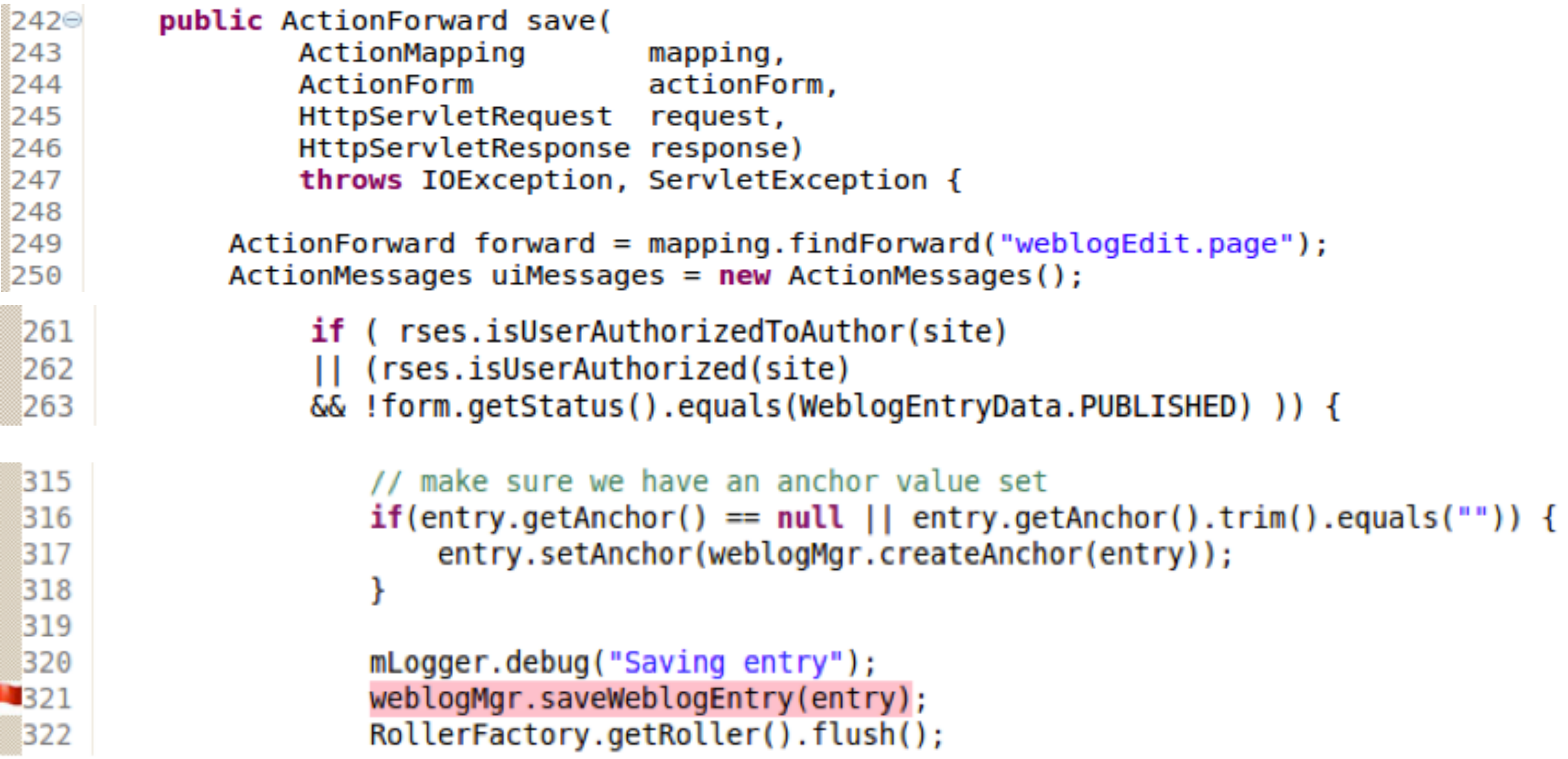
```
        if(request.getParameter("excerpts") != null) {
            this.excerpts = Boolean.valueOf(request.getParameter("excerpts")).booleanValue();
        }
```

False Positive Example 2: Untrusted input is parsed into harmless Boolean value

Evaluation on Interactive Control Flow Analysis

- Detected zero-day vulnerabilities on Apache Roller 5.0
- Seven Cross-Site Request Forgery vulnerabilities found in Apache Roller 5.0 (CVE-2012-2380)
- All earlier versions are also vulnerable
- Verified by penetration testing
- According to our report, Apache Roller team has released Apache Roller 5.0.1 with security fix (http://rollerweblogger.org/project/entry/roller_501_security_fix)

CSRF in Creating Weblog Entry



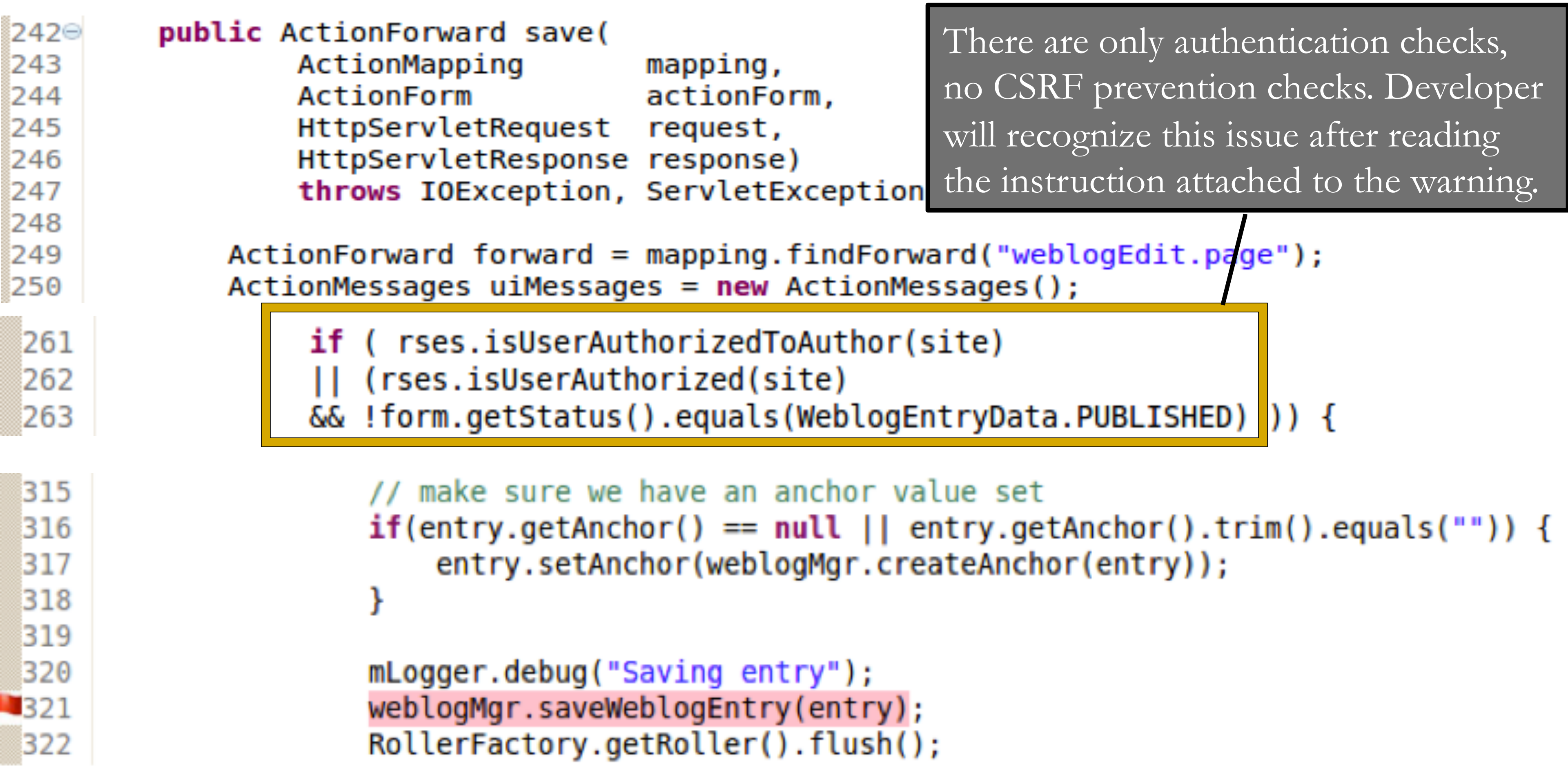
```
    public ActionForward save(
            ActionMapping       mapping,
            ActionForm          actionForm,
            HttpServletRequest  request,
            HttpServletResponse response)
            throws IOException, ServletException {

        ActionForward forward = mapping.findForward("weblogEdit.page");
        ActionMessages uiMessages = new ActionMessages();
```

```
            if ( rses.isUserAuthorizedToAuthor(site)
            || (rses.isUserAuthorized(site)
            && !form.getStatus().equals(WeblogEntryData.PUBLISHED) )) {
```

```
                // make sure we have an anchor value set
                if(entry.getAnchor() == null || entry.getAnchor().trim().equals("")) {
                    entry.setAnchor(weblogMgr.createAnchor(entry));
                }

                mLogger.debug("Saving entry");
                weblogMgr.saveWeblogEntry(entry);
                RollerFactory.getRoller().flush();
```

Developer is warned and instructed to find and annotate the control logic for the database update operation

CSRF in Creating Weblog Entry



```
    public ActionForward save(
            ActionMapping       mapping,
            ActionForm          actionForm,
            HttpServletRequest  request,
            HttpServletResponse response)
            throws IOException, ServletException

        ActionForward forward = mapping.findForward("weblogEdit.page");
        ActionMessages uiMessages = new ActionMessages();

            if ( rses.isUserAuthorizedToAuthor(site)
            || (rses.isUserAuthorized(site)
            && !form.getStatus().equals(WeblogEntryData.PUBLISHED) )) {

                // make sure we have an anchor value set
                if(entry.getAnchor() == null || entry.getAnchor().trim().equals("")) {
                    entry.setAnchor(weblogMgr.createAnchor(entry));
                }

                mLogger.debug("Saving entry");
                weblogMgr.saveWeblogEntry(entry);
                RollerFactory.getRoller().flush();
```

There are only authentication checks, no CSRF prevention checks. Developer will recognize this issue after reading the instruction attached to the warning.

Developer is warned and instructed to find and annotate the control logic for the database update operation

ASIDE for Education

- Teaching secure programming in universities is challenging
  - Most faculty, like developers, focus on subject matters instead of code security
  - Most faculty have no experience with secure programming
  - Secure programming needs to be reinforced throughout the program
- How about using ASIDE?
  - Most universities use Java and Eclipse for many of their assignments
  - ASIDE can be configured as an aid to teach students about best secure programming practices
    - Educating students about software vulnerabilities and secure programming
    - Grading aids to professors
  - ASIDE for education demo

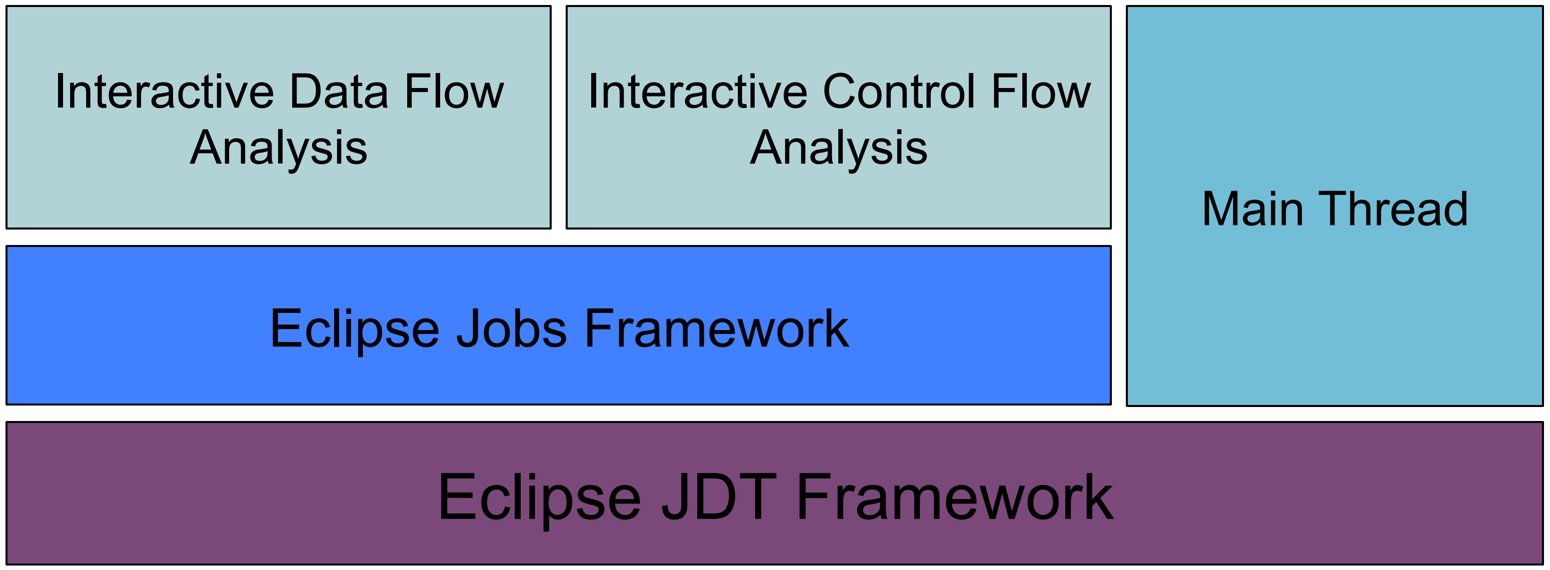
Preliminary Results

- 20 graduate students with no secure programming training
  - Worked on their programming assignment in the lab using Eclipse/Java for 3 hours
  - Students were tested about their secure programming knowledge (a set of true/false questions)
- 10.3 increased in average scores after using ASIDE
  - Average pre-test score: 53.03
  - Average post-test score: 63.33
- The differences between pre- and post test scores are statistically significant
- All students clicked on warnings, read secure programming material, and used ASIDE functions to improve code security
- None of the students used prepared SQL statements to start with
  - After being warned by ASIDE, ~ 25% of the students switched to prepared statements, even though doing so will not increase their grades

Zhu, J., Lipford, H. R., Chu, B. "Interactive Support for Secure Programming Education" SIGCSE 2013

Current Research on Interactive Static Analysis Algorithm Development

- Three threads
  - Main thread
  - Interactive data flow analysis
  - Interactive control flow analysis


Interactive Data Flow Analysis
Interactive Control Flow Analysis
Main Thread
Eclipse Jobs Framework
Eclipse JDT Framework

ASIDE Main Thread

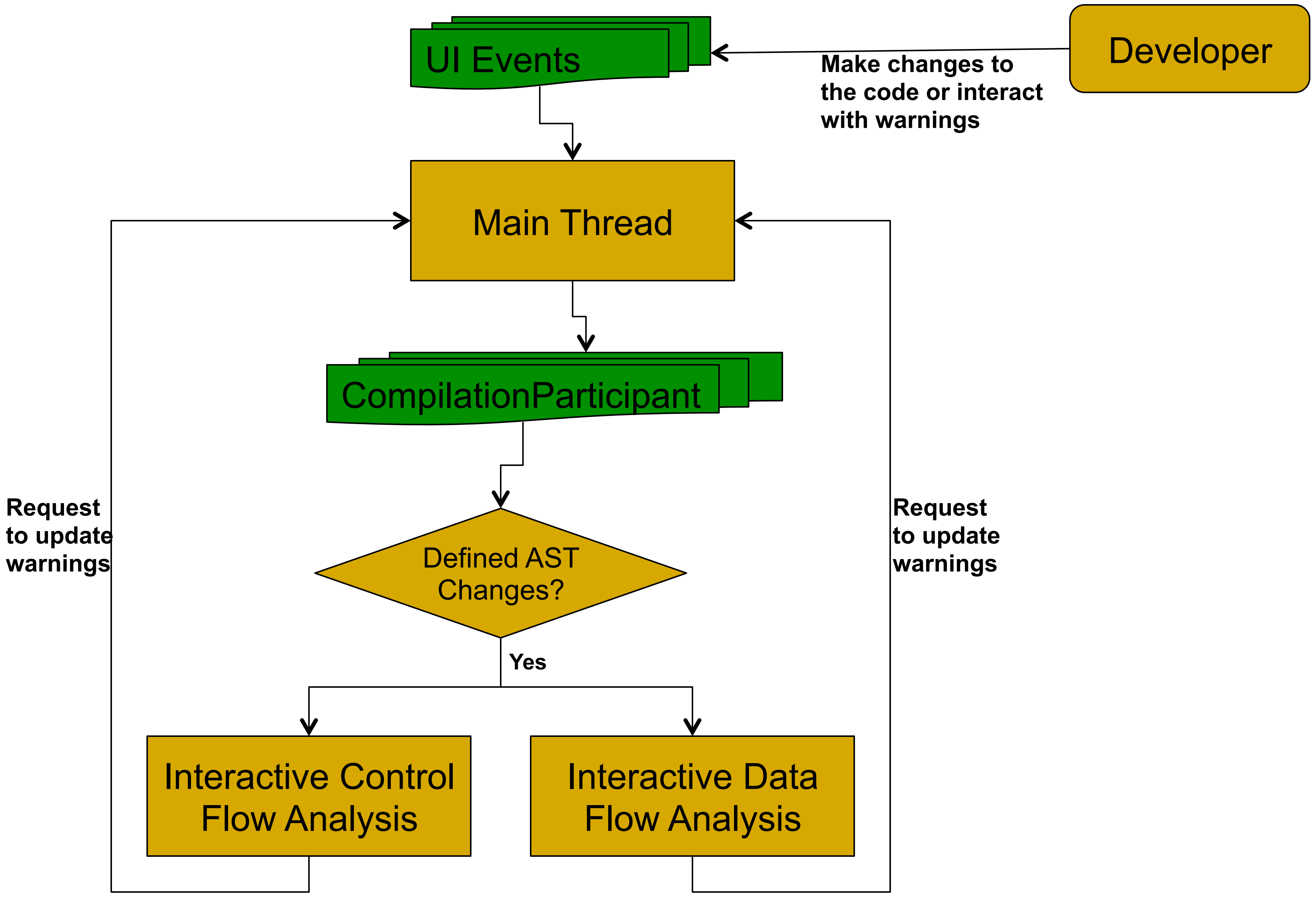
- Main thread (UI thread) rationales
  - Creates the event loop for the user interface (UI)
  - The only thread that is allowed to interact with the UI
  - All events in the user interface will be executed one after another

- To make UI responsive, analysis work are put in two other separate threads (jobs)

ASIDE Analysis Jobs

- Interactive data flow analysis
- Interactive control flow analysis

- Background running
- Could be scheduled and synchronized with the main thread


UI Events
Developer
Make changes to the code or interact with warnings
Main Thread
CompilationParticipant
Defined AST Changes?
Yes
Interactive Control Flow Analysis
Interactive Data Flow Analysis
Request to update warnings
Request to update warnings

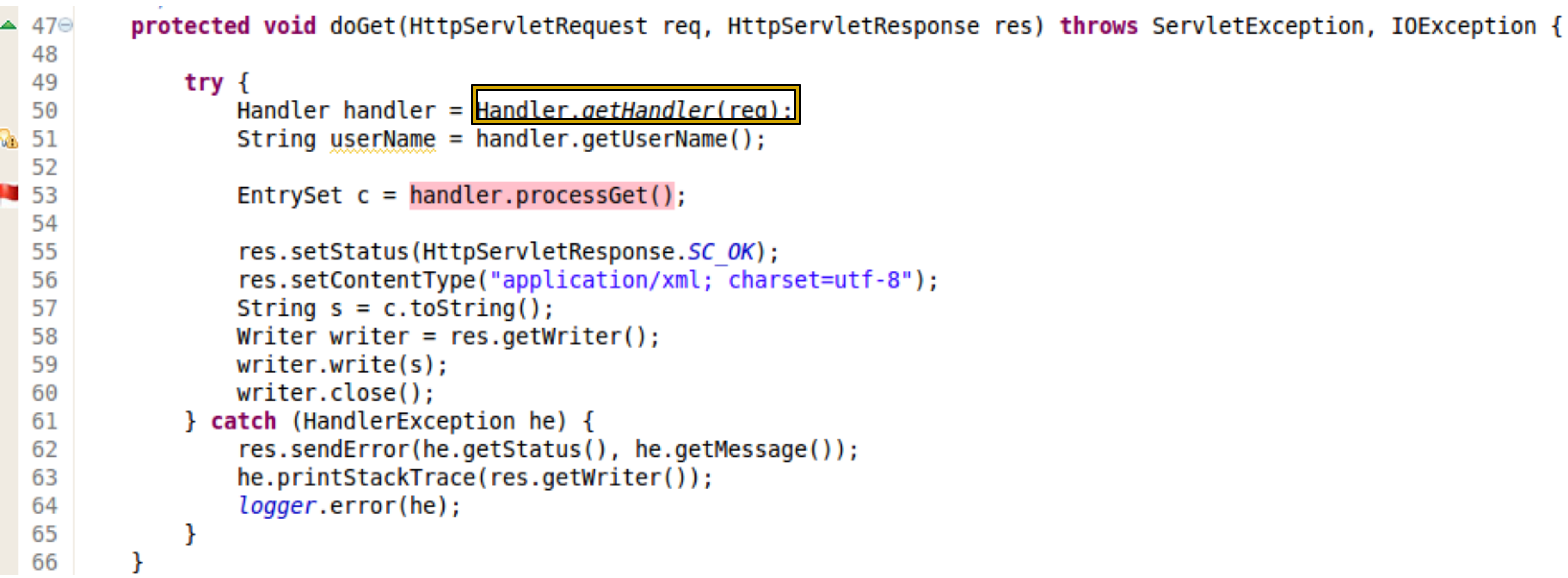
Path Coverage Analysis

- Interactive control flow analysis are designed to detect code vulnerability resulting from failure to check for security policy (invariants)
- Once annotations are provided by developers, **path coverage analysis** can be performed to detect additional vulnerabilities

Jing Xie, Bill Chu, Heather Lipford, John Melton "IDE Support for Web Application Security" in 27th *Annual Conference on Application of Computer Security*, 2011 (Acceptance rate: 18%)

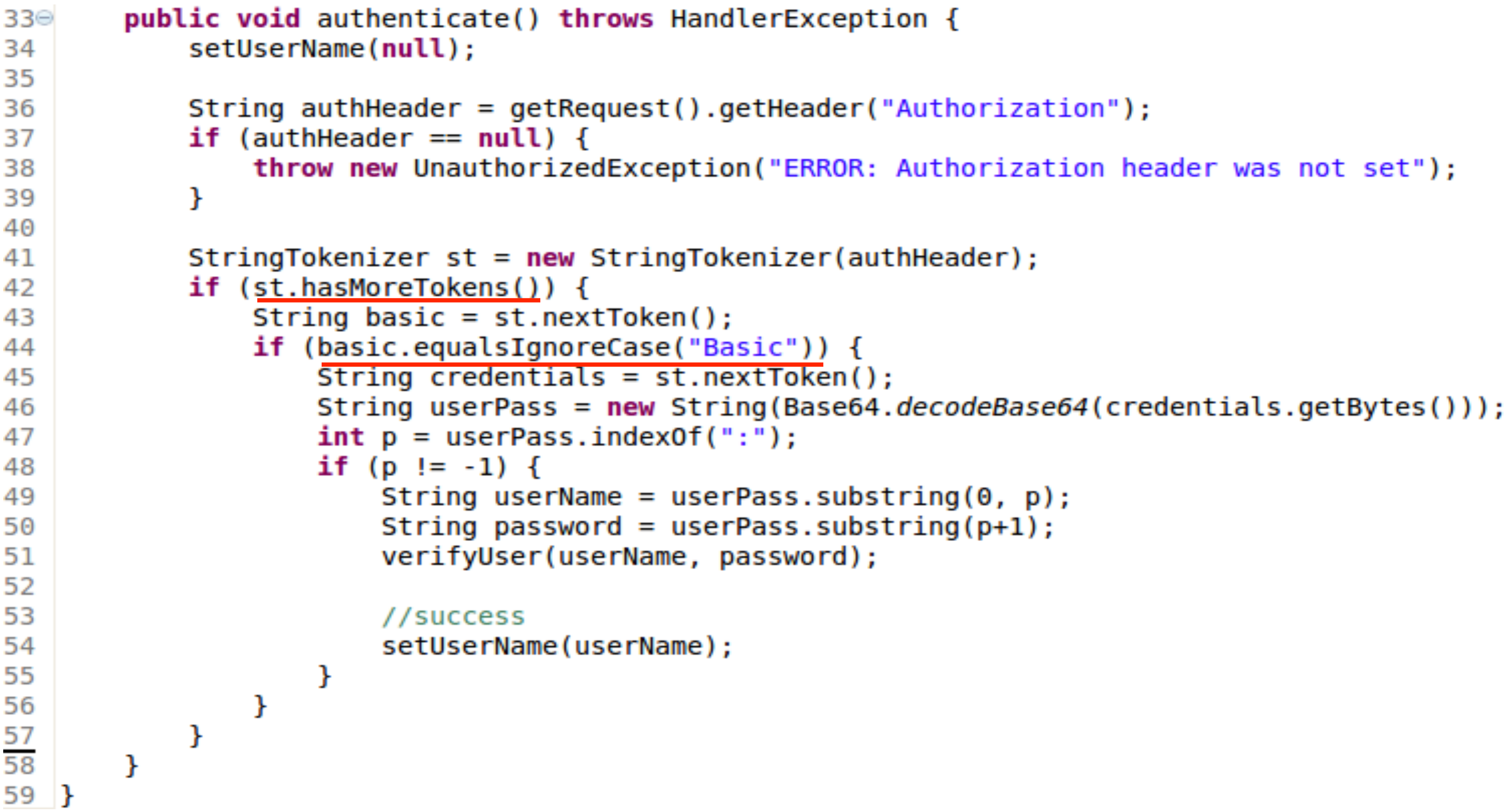
Example of Path Coverage Analysis

- **roller.weblogger.webservices.adminprotocol. BasicAuthenticator** is vulnerable to authentication bypass.

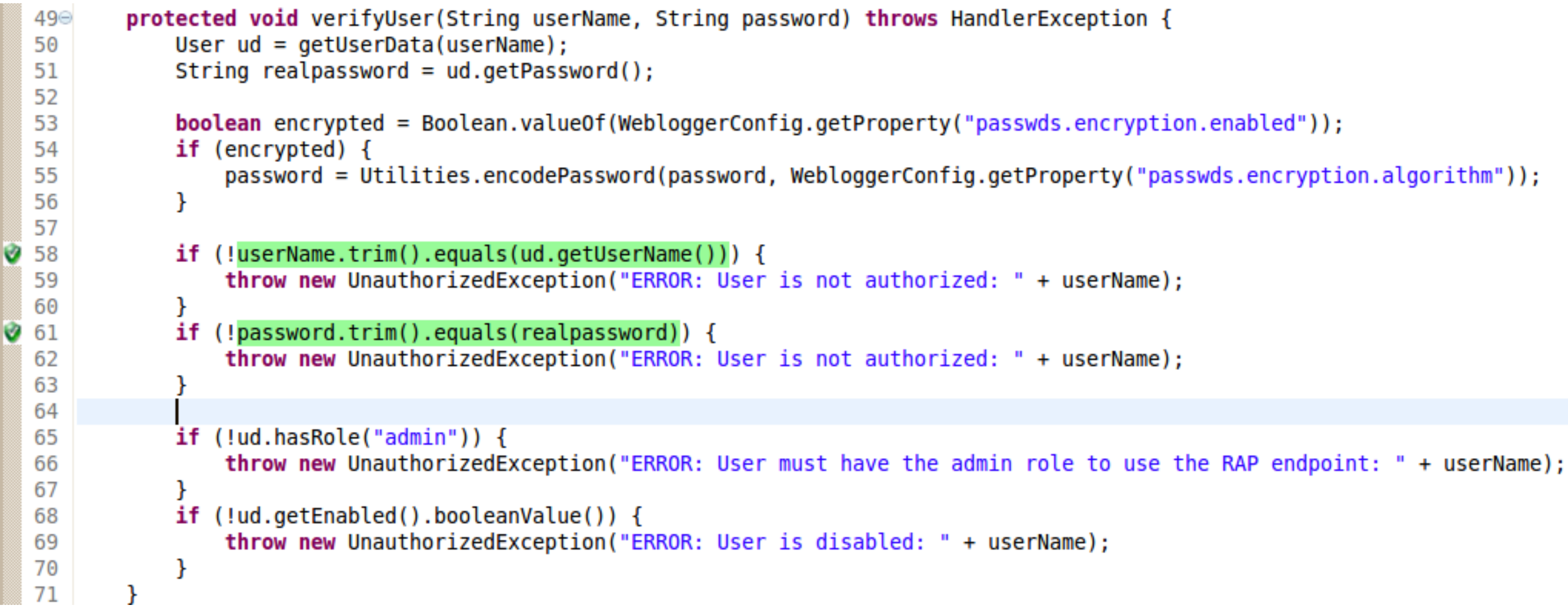


```
47    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
48
49        try {
50            Handler handler = Handler.getHandler(req);
51            String userName = handler.getUserName();
52
53            EntrySet c = handler.processGet();
54
55            res.setStatus(HttpServletResponse.SC_OK);
56            res.setContentType("application/xml; charset=utf-8");
57            String s = c.toString();
58            Writer writer = res.getWriter();
59            writer.write(s);
60            writer.close();
61        } catch (HandlerException he) {
62            res.sendError(he.getStatus(), he.getMessage());
63            he.printStackTrace(res.getWriter());
64            logger.error(he);
65        }
66    }
```

ASIDE raises question at
Line **53**



```java
    public void authenticate() throws HandlerException {
        setUserName(null);

        String authHeader = getRequest().getHeader("Authorization");
        if (authHeader == null) {
            throw new UnauthorizedException("ERROR: Authorization header was not set");
        }

        StringTokenizer st = new StringTokenizer(authHeader);
        if (st.hasMoreTokens()) {
            String basic = st.nextToken();
            if (basic.equalsIgnoreCase("Basic")) {
                String credentials = st.nextToken();
                String userPass = new String(Base64.decodeBase64(credentials.getBytes()));
                int p = userPass.indexOf(":");
                if (p != -1) {
                    String userName = userPass.substring(0, p);
                    String password = userPass.substring(p+1);
                    verifyUser(userName, password);

                    //success
                    setUserName(userName);
                }
            }
        }
    }
}
```

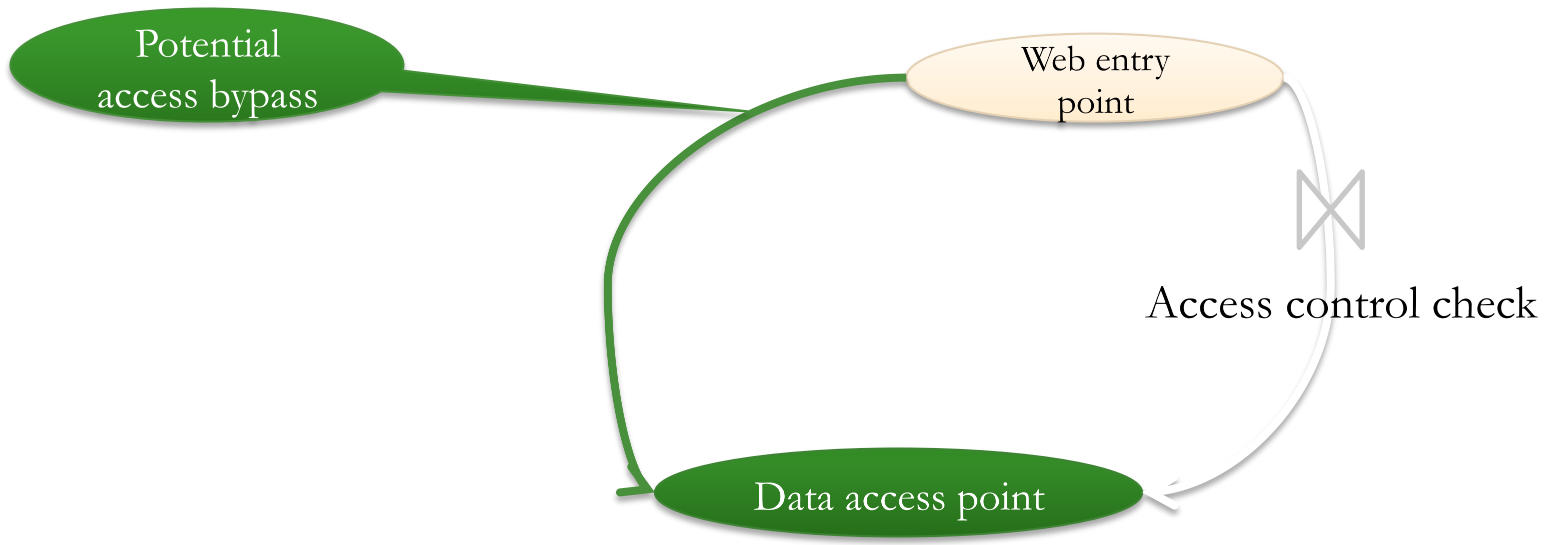


```
    protected void verifyUser(String userName, String password) throws HandlerException {
        User ud = getUserData(userName);
        String realpassword = ud.getPassword();

        boolean encrypted = Boolean.valueOf(WebloggerConfig.getProperty("passwds.encryption.enabled"));
        if (encrypted) {
            password = Utilities.encodePassword(password, WebloggerConfig.getProperty("passwds.encryption.algorithm"));
        }

        if (!userName.trim().equals(ud.getUserName())) {
            throw new UnauthorizedException("ERROR: User is not authorized: " + userName);
        }
        if (!password.trim().equals(realpassword)) {
            throw new UnauthorizedException("ERROR: User is not authorized: " + userName);
        }

        if (!ud.hasRole("admin")) {
            throw new UnauthorizedException("ERROR: User must have the admin role to use the RAP endpoint: " + userName);
        }
        if (!ud.getEnabled().booleanValue()) {
            throw new UnauthorizedException("ERROR: User is disabled: " + userName);
        }
    }
```

Developer annotates control checks for the data access point

Security Analysis Based on Annotation

- Unchecked access path
  - There might be an execution path from web entry to data access point without access control check



Applicability to OWASP Top 10

Type of Analysis	OWASP Top 10
Interactive Data Flow Analysis	Injection(1), XSS (2), Unvalidated Redirects and Forwards (10)
Interactive Control Flow Analysis	Broken Authentication (3), Insecure Direct Object Reference (4), CSRF (5), Unvalidated Redirects and Forwards (10)

Conclusions

- Interactive Static Analysis could discover common web application vulnerabilities
- Interactive Static Analysis could significantly reduce the effort of finding and fixing vulnerabilities.
- Interactive Static Analysis could find vulnerabilities not found easily by current static analysis tools
- Interactive Static Analysis could assist professional developers/ students write more secure code

Future Work

- Completely implement interactive control flow analysis, and evaluate it against an active open source project
- Perform user study to evaluate developers' behavior and reaction to ASIDE interactive control flow analysis
- Perform user study with ASIDE deployed in real-world workspace of developers

Thank You!

- Acknowledgement
  - National Science Foundation funding
  - Fortify education license
  - Jing Xie's implementation of the first version of ASIDE
- Your input
- https://www.owasp.org/index.php/OWASP_ASIDE_Project#tab=Main

UNC CHARLOTTE