Secure Programming Support in IDE

Bill Chu, Jing Xie
Department of Software and Information Systems
University of North Carolina at Charlotte
billchu, jxie2@uncc.edu

UNC CHARLOTTE

Overview

- Software vulnerabilities is a major contributor to information security problems
- Education and training is critical
- But even the most experienced developers make mistakes
- Tool support
  - Static and dynamic analysis tools are reactive
  - More effort is needed to proactively support programmers avoid making mistakes in the first place
  - Include developers in the "security loop"

UNC CHARLOTTE

Causes of developer errors

- Don Knuth's case study on TEX

  Knuth documented 867 errors over a period of 10 years. 368 errors were implementation errors, the rest are requirements / design errors. **Mistake of omission** is the largest contributor of implementation errors.

  "*Here I did not remember to do everything I had intended, when I actually got around to writing a particular part of the code. .. This seems to be one of my favorite mistakes: I often forget the most obvious things*"

  Knuth, D. The errors of TeX – *Software: Practice and Experience* 19(7) 1989

UNC CHARLOTTE

Causes of developer errors

- Disconnect between conceptual understanding of secure programming and its practice
- Our interviews of professional programmers indicates a clear pattern of programmers having a solid conceptual understanding of security but do not consistently apply them in practice
  - Reliance on framework and /or process
  - Input validation: functional/business vs. security

Jing Xie, Heather Lipford, Bill Chu "Why programmers make security errors?" *IEEE Conference on Visual Language and Human Centered Computing , 2011*

UNC CHARLOTTE

Our Approach

- Many common software vulnerabilities are caused by the **mistake of omission**, e.g.
  - Failure to perform input validation/output filtering
  - Failure to check security invariants before performing critical actions
    - CSRF
    - Broken Access Control
- **Interactively** identify common secure programming issues using reliable heuristics
- Enable developers to **select appropriate actions** while they are in the process of composing the program
  - Interactive code refactoring
  - Interactive code annotation
- **ASIDE (Application Security in IDE plugin for Java and Eclipse) prototype**

UNC CHARLOTTE

ASIDE Design Rationales

- Recognition instead of recalling, a key HCI design principle
- Take full advantage of developer's application knowledge (e.g. business logic, application context)
- Support best secure software development practice
  - Using trusted library (e.g. OWASP ESAPI)
  - Statistics collection
- Policy driven (adapted to other development environment)

UNC CHARLOTTE

ASIDE Demo

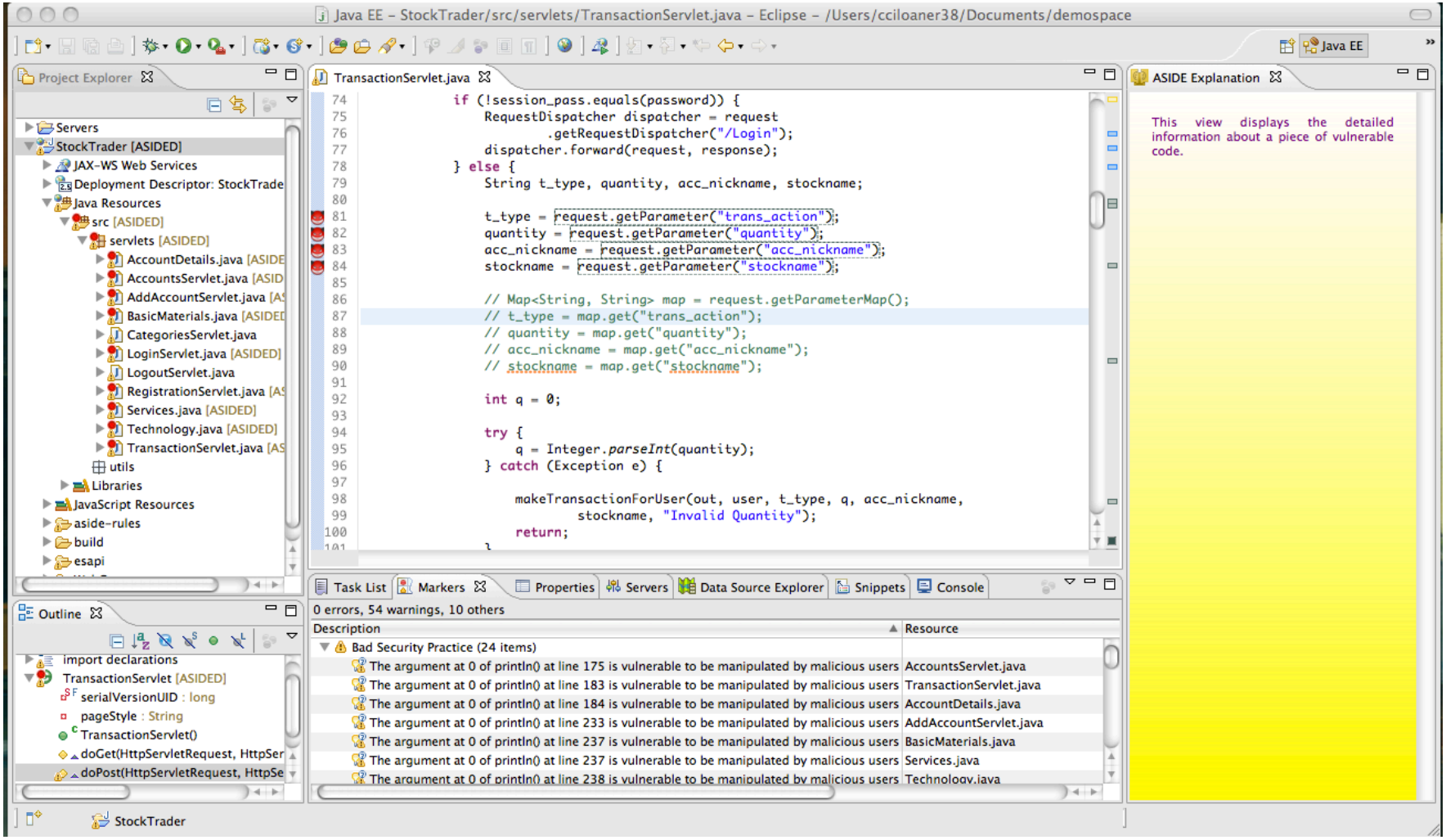
- ASIDE stands for **A**pplication **S**ecurity in **I**ntegrated **D**evelopment **E**nvironment.
- Based on Eclipse Java Development Tooling (JDT).
- Two major features:
  - **Code refactoring (implemented)**
  - Code annotation

Jing Xie, Bill Chu, Heather Lipford "Interactive Support for Secure Software Development" in *International Symposium on Engineering Secure Software and Systems,* 2011.

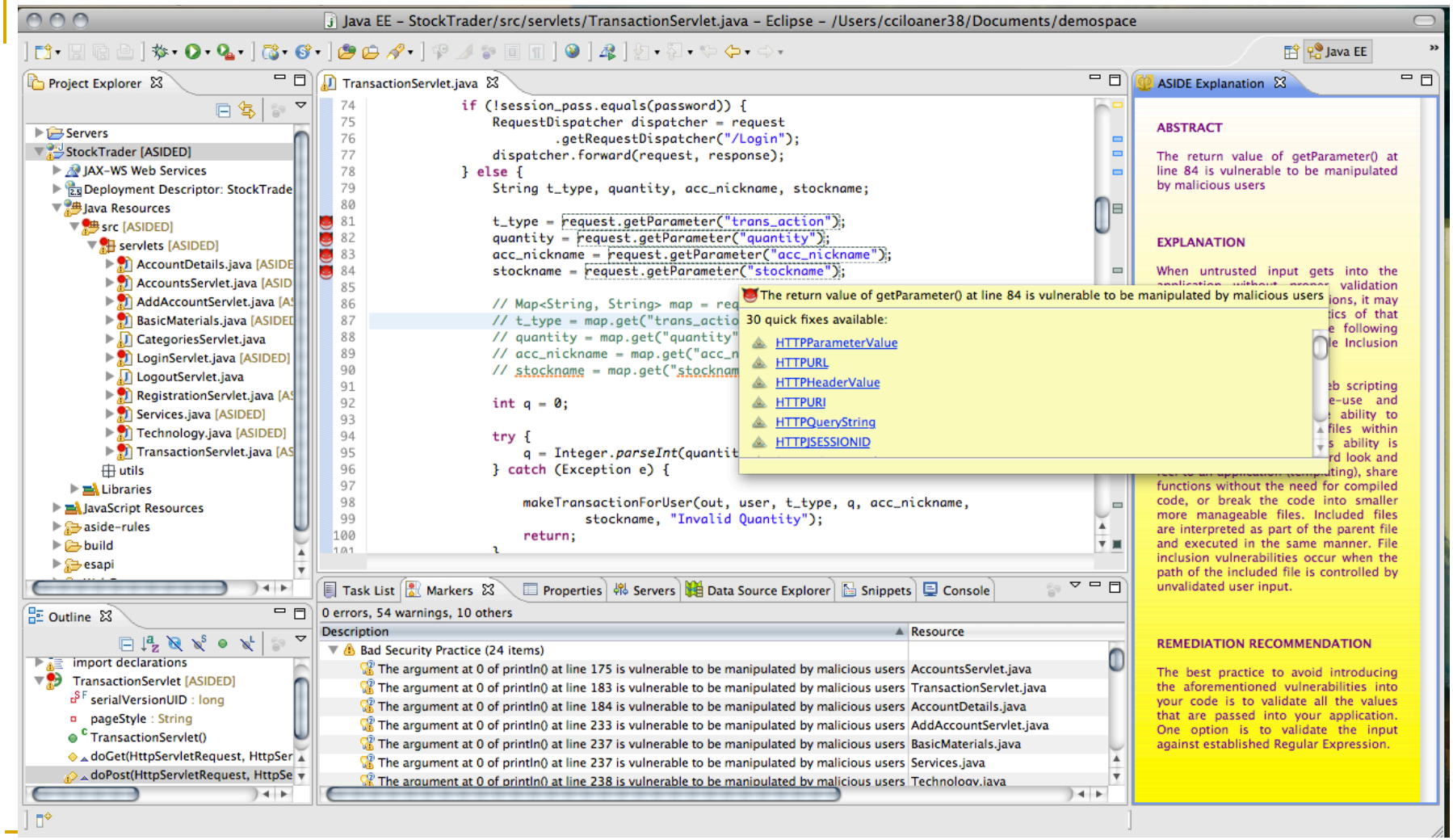
UNC CHARLOTTE

Example: interactive code refactoring for input validation

- Identifying untrusted input requiring validation



UNC CHARLOTTE



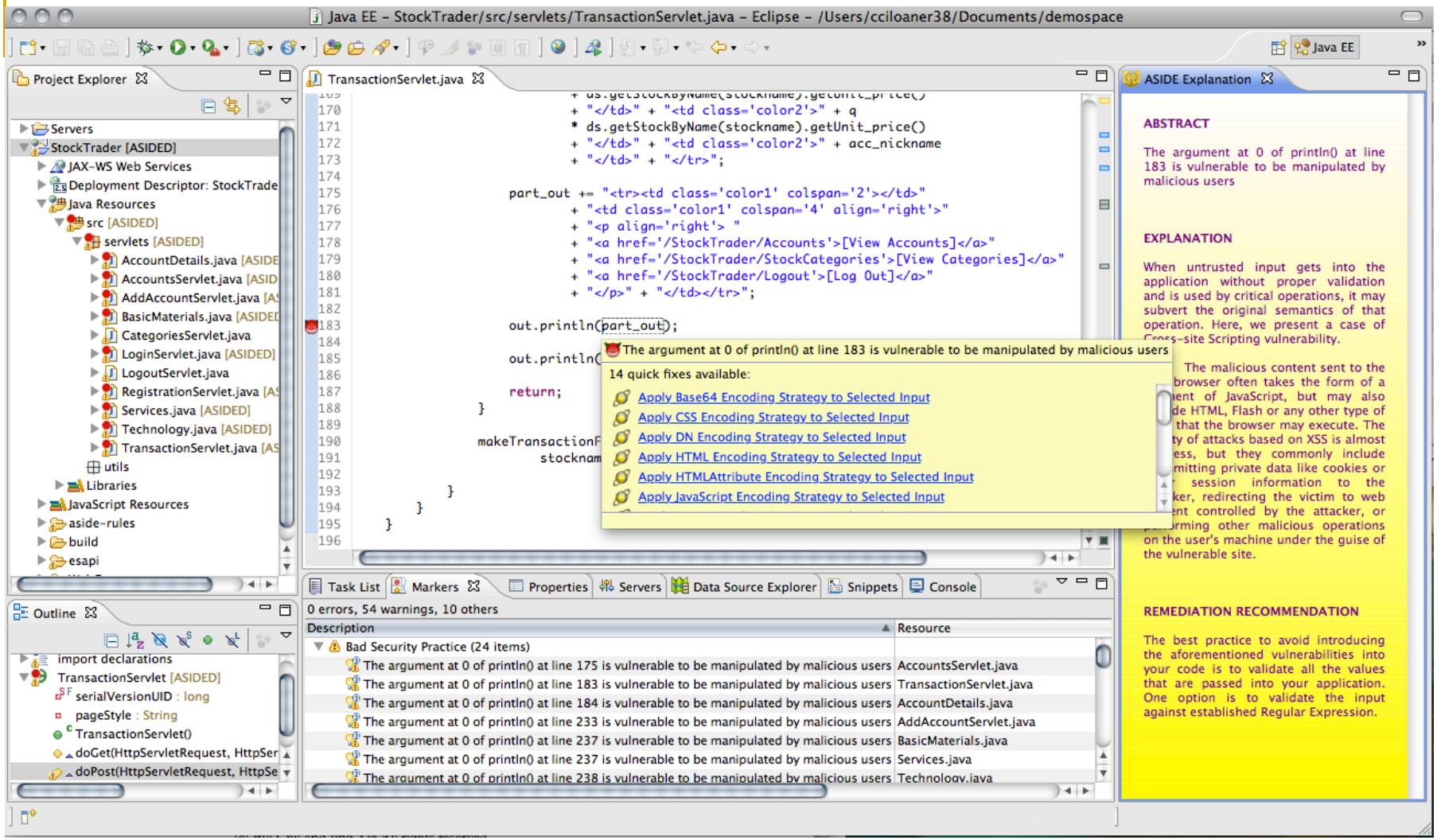
UNC CHARLOTTE

```
27          String username = req.getParameter("username");
```

```
30          String username = req.getParameter("username");
31          // NOTE: Input Validation code generated by ASIDE
32          try {
33              ESAPI.validator().getValidInput(
34                      "replace ME with validation context", username,
35                      "SafeString", 200, false);
36          } catch (ValidationException e) {
37          } catch (IntrusionException e) {
38          }
```

Utilize reputable input validation library, e.g. OWASP ESAPI Validator.

UNC CHARLOTTE



UNC CHARLOTTE

Code refactoring strategies for input validation

Input validation strategy	Advantage(s)	Disadvantage(s)
Right before critical operations (e.g. inserting into database).	Developer knows for sure the type of input. (e.g. first name, password, credit card number, SSN, and etc.)	**Redundant validation:** a variable used in multiple places. **Failure to validate:** difficult to foresee all critical operations.
As soon as an untrusted input is read into a variable.	Has developer's attention. Make sure all untrusted inputs are validated.	Can lead to **false positive**. Does not work well with **dependency injection design pattern**.

UNC CHARLOTTE

ASIDE implementation

- ASIDE can support either strategy
- We evaluated a version of ASIDE using the second strategy and discuss some of our evaluation results

UNC CHARLOTTE

Additional input validation features

- **Semantic validation**
  - E.g. once the input is identified as file path, further restrict to a particular file subtree
  - Bounds of integers
- For untrusted input of **composite type** (e.g. *getParameterMap()*)
  - Perform flow analysis
  - Request for validation as soon as an primitive type (e.g. *java.lang.String*) of data is extracted

UNC CHARLOTTE

Code Refactoring Evaluation

- Target Project: Apache Roller 3.0.0
  - 65K+ lines of code
  - Full featured blog server (1.8M+ hits on google for "powered by Apatche Roller)
- Comparison base: Fortify SCA based code review

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

UNC CHARLOTTE

Industry Best Practice Security Audit

- Performed by John Melton, a member of SSG at a large financial service company, core committer of OWASP AppSensor
- Fortify SCA reported **3,416** issues
- John manually audited each issue
- John determined **1,655** issues, as he would have done according to industry best practice
- Software of **average** quality according to John
- Would take **2.5** days based on standard workload estimate metrics

UNC CHARLOTTE

Details of Manual Audit

- Whether appropriate input validation/encoding has been performed
- Validate Fortify's environmental assumptions (e.g. for log forging, whether logging mechanism has not been wrapped)
- Validate Fortify's trust boundary assumptions
- Scrutinize input validation and encoding routines (e.g. black-list filtering)
- Filter out false positives in DOS warnings

UNC CHARLOTTE

Secure Code Review Results

	Critical	High	Medium	Low
Fortify Vulnerability Categories	8	18	2	52
Raw Issues	164	653	13	2,597
Exploitable in Roller 3.0.0	37	397	0	1,221

922 of **1,655** findings are related to lack of proper validation/encoding

UNC CHARLOTTE

Validation /filter of untrusted data

Severity	Category Name	Number of Issues
Critical	Cross-Site Scripting: Persistent	2
	Cross-Site Scripting: Reflected	2
	Path Manipulation	19
	SQL Injection	11
High	Cross-Site Scripting: Persistent	31
	Denial of Service	4
	Header Manipulation	52
	Log Forging	252
	Path Manipulation	6
Low	Cross-Site Scripting: Poor Validation	6
	Log Forging (debug)	531
	SQL Injection	3
	Trust Boundary Violation	3
Total		922

Table 2. Detail results from security auditing against Roller using Fortify SCA.

UNC CHARLOTTE

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)

UNC CHARLOTTE

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)

UNC CHARLOTTE

Defensive Security

- A taint request URL is directly passed into an *InvalidRequestException* constructor.

```
100             } else {
101                 throw new InvalidRequestException("bad path info, "+
102                         request.getRequestURL());
103             }
```

UNC CHARLOTTE

False positives

- We regard 24 reported taint sources as real false positives, where inputs are used in ways that do not lead to any recognized security vulnerabilities.

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

Figure 5. Untrusted input is used for logic test

```
125         if(request.getParameter("excerpts") != null) {
126             this.excerpts = Boolean.valueOf(request.getParameter("excerpts")).booleanValue();
127         }
```

Figure 6. Untrusted input is parsed into harmless Boolean value

UNC CHARLOTTE

Summary of benefits of code refactoring

- Address Validation/Encoding issues at the time of development
  - Requires no specialized training
  - Capturing application context
  - Saving time to fix vulnerabilities that might be found later in security code audit
  - Saving efforts in fixing vulnerabilities (e.g. 143 taint sources vs. 922 issues)
- Save security code audit efforts
  - Significant reduction of workload (e.g. 922 out of 3,416)

UNC CHARLOTTE

Practical implications for using ASIDE

- Compliment to static analysis
  - e.g. Generating "cleansing rules" after validation/ encoding to reduce number of issues raised
- A "light version of static analysis"
  - e.g. handling validation/encoding issues

UNC CHARLOTTE

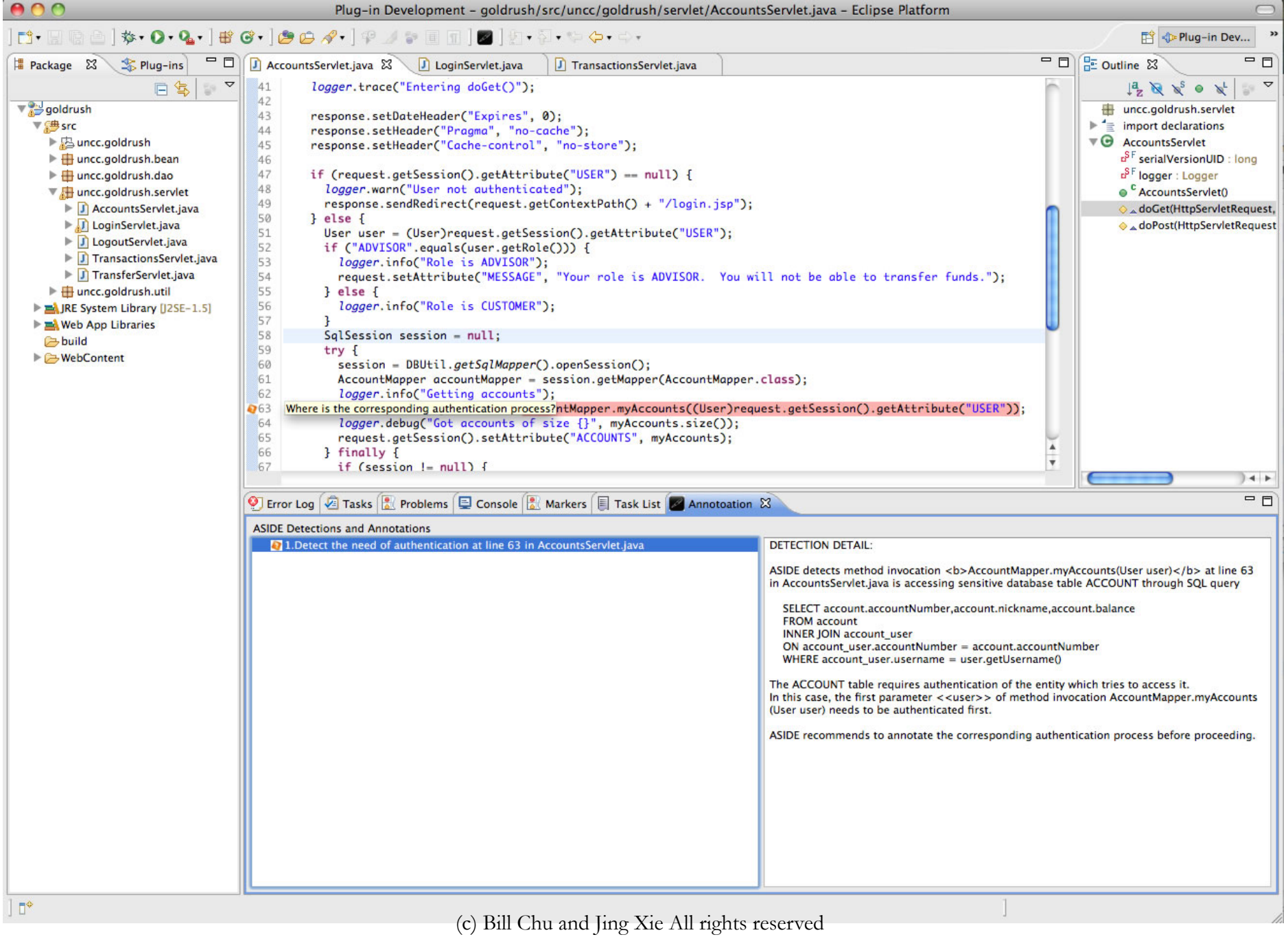
Interactive Code Annotation

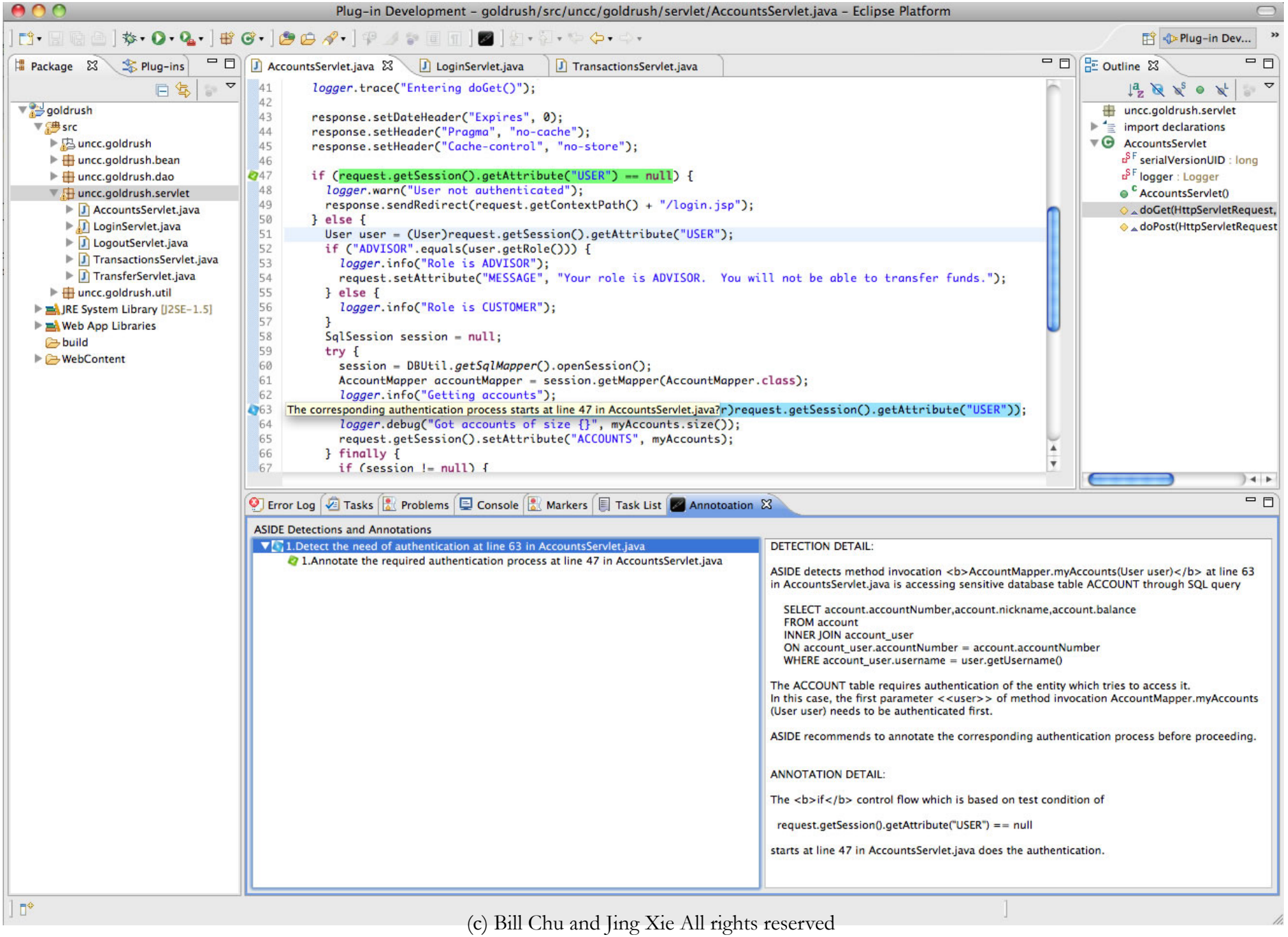
- Remind developers important program constructs for secure coding
  - Prevent vulnerable code from being written
- Annotate key application logic for
  - Source code review
  - Advanced analysis
- Different from traditional code annotation
  - Annotate security relationship between different parts of the system
  - Point an click

UNC CHARLOTTE

Interactive Code Annotation Example

- Database tables
  - **user**(username,role,surname,givenName)
  - **account**(accountNumber,nickname,balance)
  - **transaction**(id,accountNumber,date,payee,amount)
  - **account_user**(accountNumber,username)
- All tables are protected by SSG





UNC CHARLOTTE

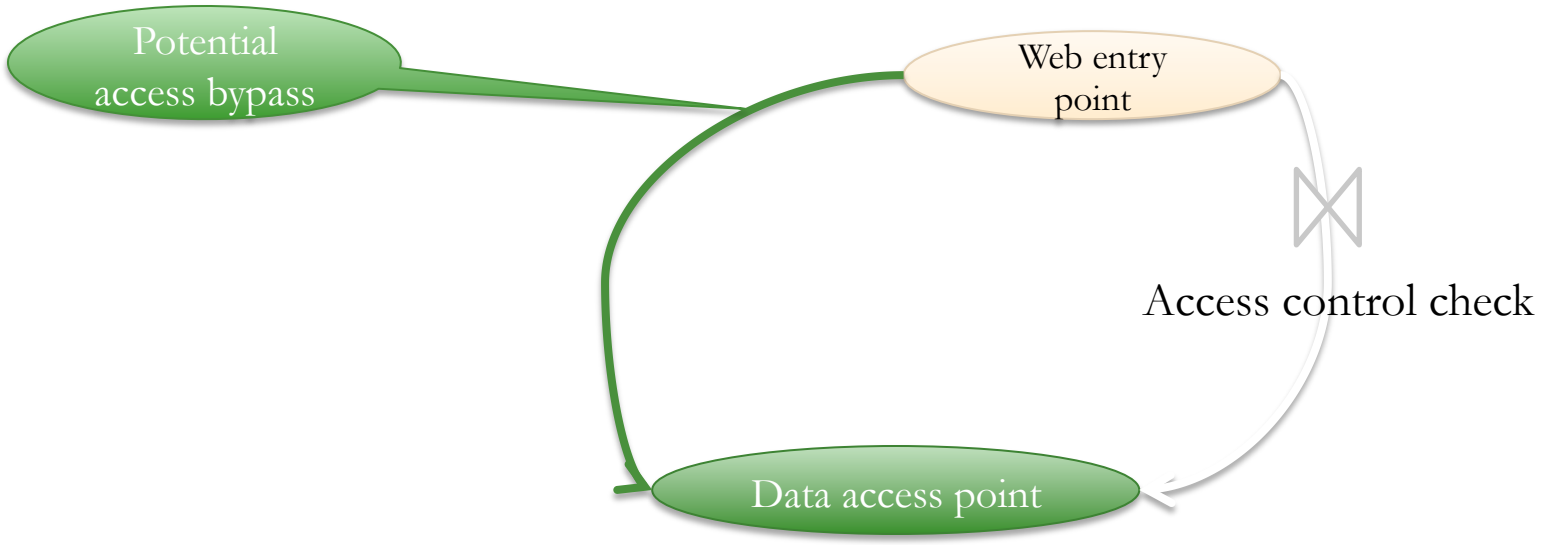
Context for annotation

- **Where to raise question?**
  - Identifying database access functions (e.g. SQL statements), may be too low level
  - Access routines may be shared in different application threads
  - Identify "use case"/transaction level routines that lead to accessing protected data
  - E.g. a statement within a Servlet/Action for Java web applications
- **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

UNC CHARLOTTE

Advanced analysis based on annotation

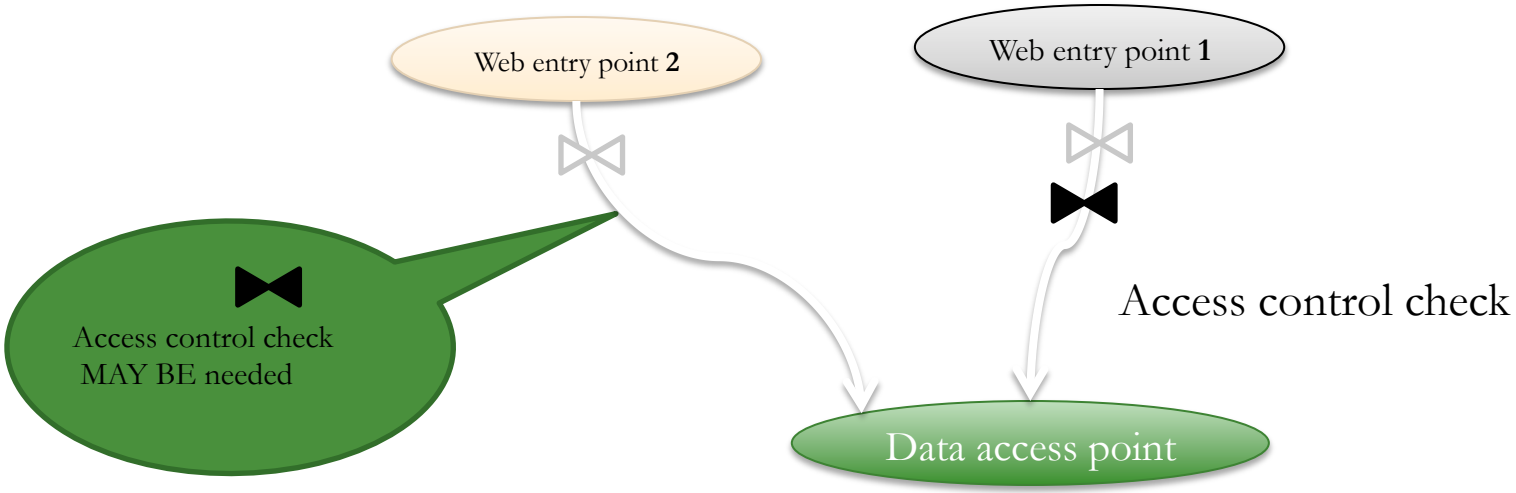
- Unchecked access path
  - There might be an execution path from web entry to data access point without access control check



UNC CHARLOTTE

Advanced analysis based on annotation –cont (Triangulation)

- Missing access control check
  - Suppose there are two "use cases" that invoke the same access function
  - They have different access control checks



UNC CHARLOTTE

Case Study

- Open source project
  - Apache **Roller** (Java): blog server software
  - **Moodle** (PHP): course management system (CMS)
- Statistics (bug track & security reports)

	Fixed issues with detailed information	Code Refactoring	Code Annotation
Roller	6	3	1
Moodle	14	1	2

UNC CHARLOTTE

Improper/Insufficient Input Validation

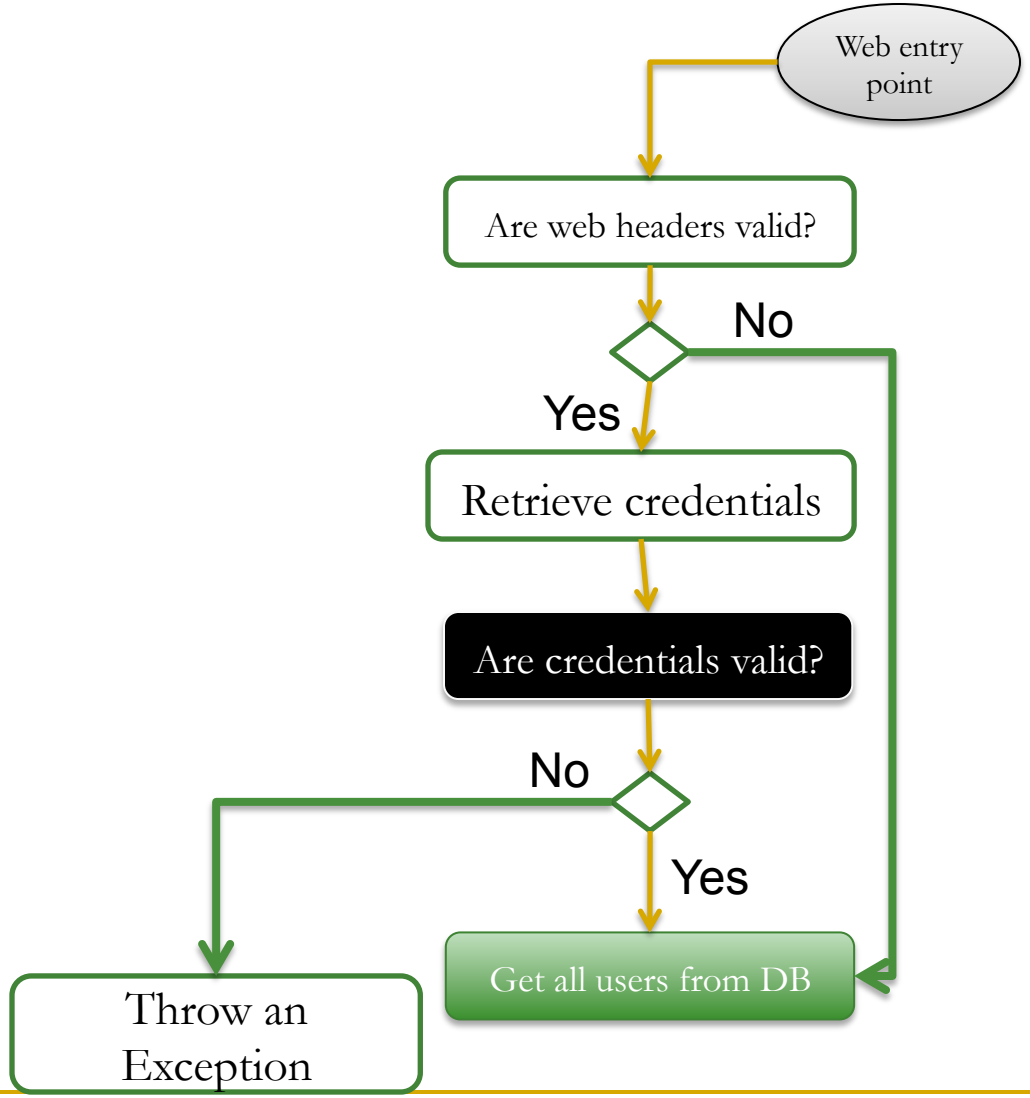
- 3/7 cases are vulnerabilities caused by insufficient input validation.
- All these cases can be handled by ASIDE as described above

UNC CHARLOTTE

Broken Access Control

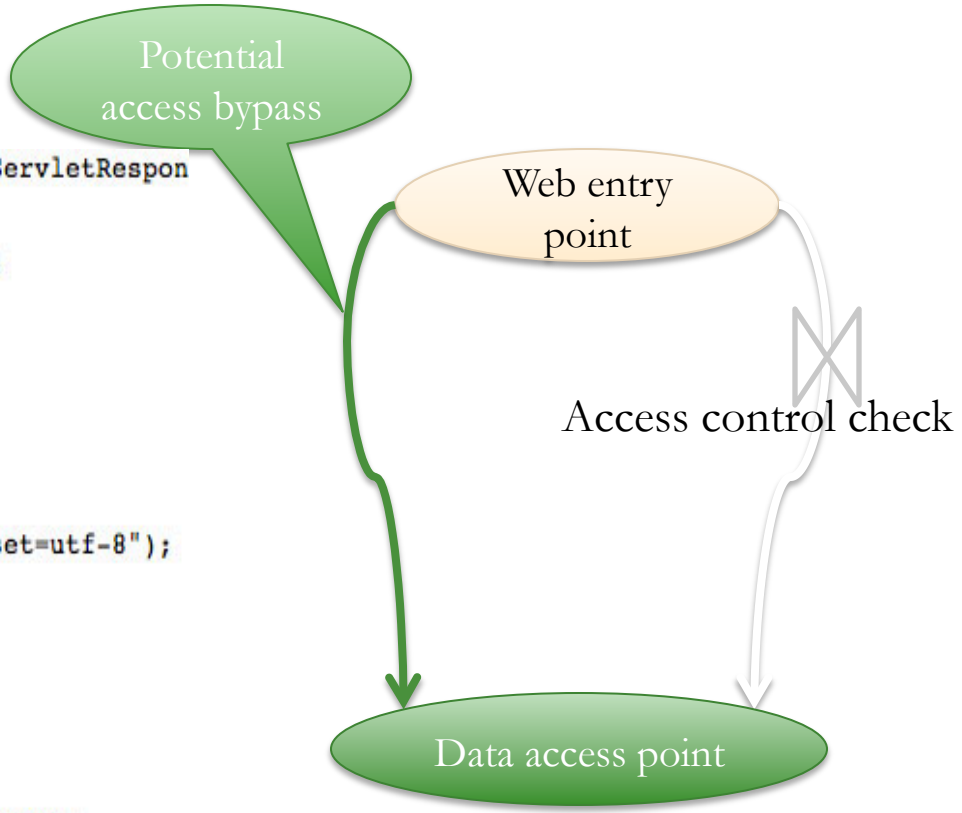
- ROL-1701 (https://issues.apache.org/jira/browse/ROL-1701)
- **roller.weblogger.webservices.adminprotocol.**

  BasicAuthenticator is vulnerable to authentication bypass. If **invalid headers** are passed to it, an invalid user can gain access to protected resources.



UNC CHARLOTTE

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



ASIDE raises question at Line **52**

```
45      /**
46       * This method should be called by extensions of this class within their
47       * implementation of authenticate().
48       */
49      protected void verifyUser(String userName, String password) throws HandlerException {
50          User ud = getUserData(userName);
51          String realpassword = ud.getPassword();
52
53          boolean encrypted = Boolean.valueOf(WebloggerConfig.getProperty("passwds.encryption.enabled"));
54          if (encrypted) {
55              password = Utilities.encodePassword(password, WebloggerConfig.getProperty("passwds.encryption.algorithm"));
56          }
57
58          if (!userName.trim().equals(ud.getUserName())) {
59              throw new UnauthorizedException("ERROR: User is not authorized: " + userName);
60          }
61          if (!password.trim().equals(realpassword)) {
62              throw new UnauthorizedException("ERROR: User is not authorized: " + userName);
63          }
64
65          if (!ud.hasRole("admin")) {
66              throw new UnauthorizedException("ERROR: User must have the admin role to use the RAP endpoint: " + userName);
67          }
68          if (!ud.getEnabled().booleanValue()) {
69              throw new UnauthorizedException("ERROR: User is disabled: " + userName);
70          }
71      }
```

UNC CHARLOTTE

Cross-site Request Forgery

- 2/7 are CSRF vulnerabilities
- MSA-08-0013 & MSA-09-0008
- Moodle has developed a pattern to prevent CSRF
- But it was missed in at least these two cases by developers

UNC CHARLOTTE

Change an existing user's profile

Client form submission

#	Line 69
69	
70	/// If data submitted, then process and store.
71	
72	if ($usernew = data_submitted()) {
73	
74	if (($USER->id <> $usernew->id) && !isadmin()) {
75	print_error('onlyeditown');

Update database content

```
$userold = get_record('user','id',$usernew->id);
if (update_record("user", $usernew)) {
   if (function_exists("auth_user_update")){
      // pass a true $userold here
      if (!auth_user_update($userold, $usernew)) {
         // auth update failed, rollback for moodle
         update_record("user", $userold);
         error('Failed to update user data on external auth: '.$user->auth.
                  '. See the server logs for more details.');
      }
   };
```

UNC CHARLOTTE

ASIDE solution

- **Heuristic:** Whenever a form submission/web request contains operation to update (add, delete, modify) database entries, the form submission needs to be checked for CSRF.
- Raise question at **Line 72**

UNC CHARLOTTE

Change an existing user's profile

Client form submission

```
#       Line 264
264
265         $replycount = forum_count_replies($post);
266
267         if (!empty($confirm)) {    // User has confirmed the delete
268
269             if ($post->totalscore) {
270                 notice(get_string("couldnotdeleteratings", "forum"),
```

```
if (! $post->parent) {  // post is a discussion topic as well, so delete discussion
    if ($forum->type == 'single') {
        notice("Sorry, but you are not allowed to delete that discussion!",
                forum_go_back_to("discuss.php?d=$post->discussion"));
    }
    forum_delete_discussion($discussion, false, $course, $cm, $forum);

    add_to_log($discussion->course, "forum", "delete discussion",
            "view.php?id=$cm->id", "$forum->id", $cm->id);

    redirect("view.php?f=$discussion->forum");

} else if (forum_delete_post($post, has_capability('mod/forum:deleteanypost', $modcontext),
    $course, $cm, $forum)) {
```

Update(delete) database content

Triangulation

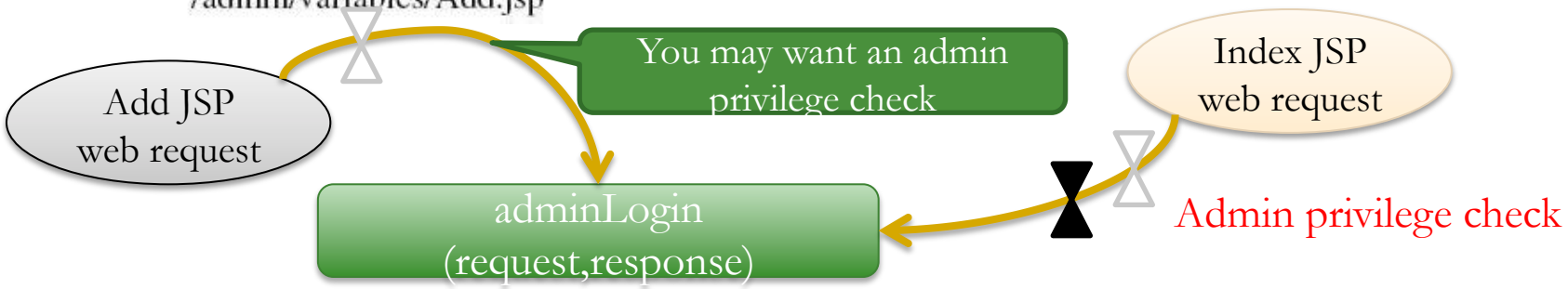
-Open source project JspCart

```
1  public void _jspService(HttpServletRequest req,
2                          HttpServletResponse res) {
3
4    User user = (User) session.getAttribute("User");
5    if(user==null) {
6      User.adminLogin(request,response);
7      return;
8    }
9    ...
10   if(request.getMethod().equalsIgnoreCase("post")) {
11     result = website.variables.
12              insert(new Variable(req));
13   }
14 }
```

/admin/variables/Add.jsp

```
1  public void _jspService(HttpServletRequest req,
2                          HttpServletResponse res) {
3
4    User user = (User) session.getAttribute("User");
5    if(user==null || (!user.isAdmin())) {
6      User.adminLogin(request,response);
7      return;
8    }
9    ...
10   out.println("<a href=\"admin/variables/\
11            Add.jsp\">Add New</a>");
12 }
```

/admin/variables/index.jsp



Viktoria Felmetsger, Ludovico Cavedon, Christopher Kruegel, and Giovanni Vigna. 2010.
Toward automated detection of logic vulnerabilities in web applications.
In *Proceedings of the 19th USENIX conference on Security* (USENIX Security'10).

UNC CHARLOTTE

Applicability to secure coding errors

Secure programming practices	**CWE/SANS top 25 Dangerous Programming Errors**
Interactive code refactoring	XSS(1), SQL injection (2), Untrusted input (6), Path traversal (7), Dangerous file types (8), OS command injection (9), Improper control of file name (14), URL redirection (23).
Interactive code annotation	Buffer copy without checking size of input (3), CSRF (4), Improper access control (5), Buffer Access with incorrect length value (12), Missing authentication (19), Download code without integrity check (20), Incorrect permission assigned for critical resource (21), Race condition (25).

UNC CHARLOTTE

Summary of benefits

technique / audience	Code refactoring	Code annotation
Students	Shape awareness, reminder of secure coding best practices, aid in grading	Shape awareness, reminder of secure coding best practices, aid in grading
Professional developers	Reminder of secure coding best practices, take care of "grunt work"	Reminder of secure coding best practice, advanced analysis
Enterprise	Encourage secure coding, policies, practice and standards, collect SSDLC statistics	Collect SSDLC statistics, aid in code review

UNC CHARLOTTE

What do programmers think about Security?

- Conducted two user studies
  - 15 graduate students in web application development class
  - 10 professional Java developers
- Works well for students
  - All used ASIDE functions even though it is not required
  - Most of them felt it was very helpful
- Mixed reaction from developers
  - Developers are much more focused on functions, they are used to have warnings not being addressed
  - They need more contextual explanation before they accept code generated by ASIDE
  - **Security savvy developers tend to reject the necessity of secure programming for code that does not impose immediate vulnerability threat**

UNC CHARLOTTE

Conclusions

- Introduced two mechanisms to support secure programming in IDE (interactive code refactoring and annotation)
- ASIDE's approach can be an effective addition to best practice SDLC
  - Preventing vulnerable code
  - Improve efficiency of static analysis
- ASIDE appears to be effective as an education tool in universities (NSF funded project to study the effect of ASIDE in CS1, CS2, and Web programming courses at three universities)
- Improvements are needed to make it usable by professional developers

UNC CHARLOTTE

Future work

- UI design, especially for Annotation
- Support Web frameworks (Struts I and II, Spring MVC, etc.)
- Make ASIDE appeal to professional developers
- Study the effect of ASIDE in university curriculum

UNC CHARLOTTE

Thank You

- Acknowledgement
  - National Science Foundation funding
  - Fortify education license

- Your input
- https://www.owasp.org/index.php/OWASP_ASIDE_Project#tab=Main