Everything you know about Injection Attack is Wrong

Pravir Chandra

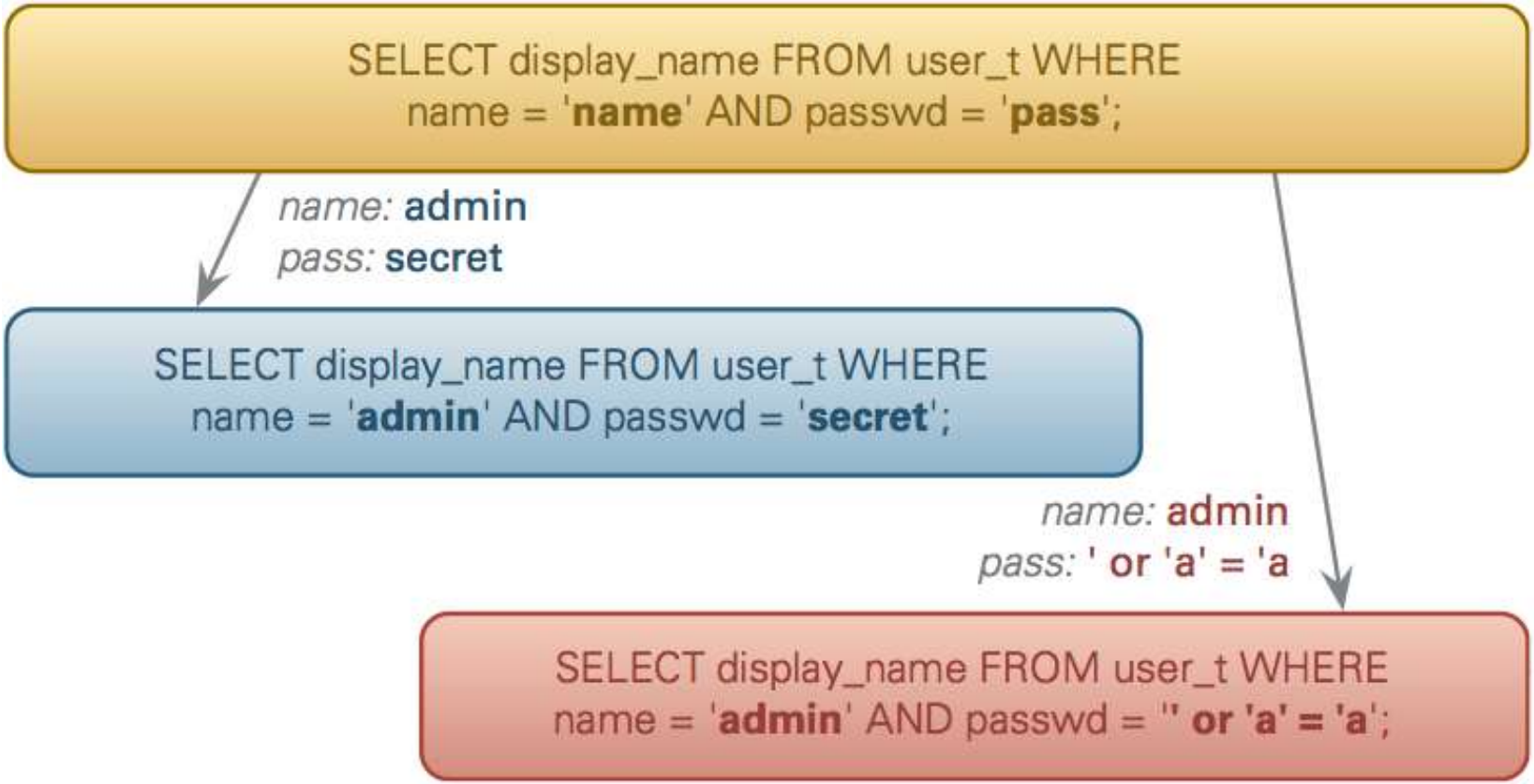
Bloomberg

chandra@owasp.org

SQL Injection

SQL Injection

```
01. public boolean login(String username, String password) {
02.   Statement stmt = this.conn.CreateStatement();
03.   String sql = "SELECT display_name FROM user_t " +
04.                "WHERE name = \'" + username + "\' " +
05.                "AND passwd = \'" + password + "\'";
06.   ResultSet r = stmt.executeQuery(sql);
07.   return r.first();
08. }
```


SELECT display_name FROM user_t WHERE
name = '**name**' AND passwd = '**pass**';
name: **admin**
pass: **secret**
SELECT display_name FROM user_t WHERE
name = '**admin**' AND passwd = '**secret**';
name: **admin**
pass: **' or 'a' = 'a**
SELECT display_name FROM user_t WHERE
name = '**admin**' AND passwd = '**' or 'a' = 'a**';

Prepared Statement

```
01. public boolean login(String username, String password) {
02.   String sql = "SELECT display_name FROM user_t " +
03.                "WHERE name = ? AND passwd = ?";
04.   PreparedStatement pst = this.conn.PrepareStatement(sql);
05.   pst.setString(1, username);
06.   pst.setString(2, password);
07.   ResultSet r = pst.executeQuery();
08.   return r.first();
09. }
```

Now what?

```
01. public ResultSet fetchRecords(String dbname, int id) {
02.   String sql = "SELECT * FROM " + dbname + ".record_t "
03.                "WHERE id = ?";
04.   PreparedStatement pst = this.conn.PrepareStatement(sql);
05.   pst.setInt(1, id);
06.   return pst.executeQuery();
07. }
```

Input Validation!!!

Input Validation???

~~Input Validation~~

Cross Site Scripting

XSS – HTML

```
01. <div name="welcome">
02.   Welcome, <%= request.getParameter("display_name") %>!
03. </div>
```

XSS – Attribute

```
01. <p bgcolor="<%= request.getParameter('color') %>">
02.   Your payment has been received. Thank you!
03. </p>
```

XSS – CSS

```
01. <style type="text/css">
02. .themed a {
03.   background-color: <%= request.getParameter("color") %>;
04. }
05. </style>
```

XSS – Javascript

```
01. <script type="text/javascript">
02.   function goBack() {
03.     var prev = "<%= request.getParameter('previous') %>";
04.     document.location = "http://mysite.com/" + prev;
05.   }
06. </script>
```

XSS – URL

```
01. <a href="/login?name=<%= request.getParameter('name') %>">
02.   Sign In!
03. </a>
```

So what’s the real problem here?

LDAP Injection

```
01. public NamingEnumeration getEmployees(String mgrName) {
02.   DirContext ctx = new InitialDirContext(env);
03.   // retrieve all of the employees who report to manager
04.   String filter = "(manager=" + mgrName + ")";
05.   NamingEnumeration employees;
06.   employees = ctx.search("ou=People,dc=example,dc=com",
07.                          filter);
08.   return employees;
09. }
```

Malicious Input: foo (| (objectclass=*))

XPath Injection

```
01. public Object xpathLookupById(String acctID) {
02.   String query = "/accounts/account[acctID='" +
03.                  acctID + "']/email/text()";
04.   DocumentBuilderFactory domFactory;
05.   domFactory = DocumentBuilderFactory.newInstance();
06.   domFactory.setNamespaceAware(true);
07.   DocumentBuilder build = domFactory.newDocumentBuilder();
08.   Document doc = build.parse("accounts.xml");
09.   XPath xpath = XPathFactory.newInstance().newXPath();
10.   XPathExpression expr = xpath.compile(query);
11.   return expr.evaluate(doc, XPathConstants.NODESET);
12. }
```

Malicious Input: ’ or 1=1 or ‘’=‘

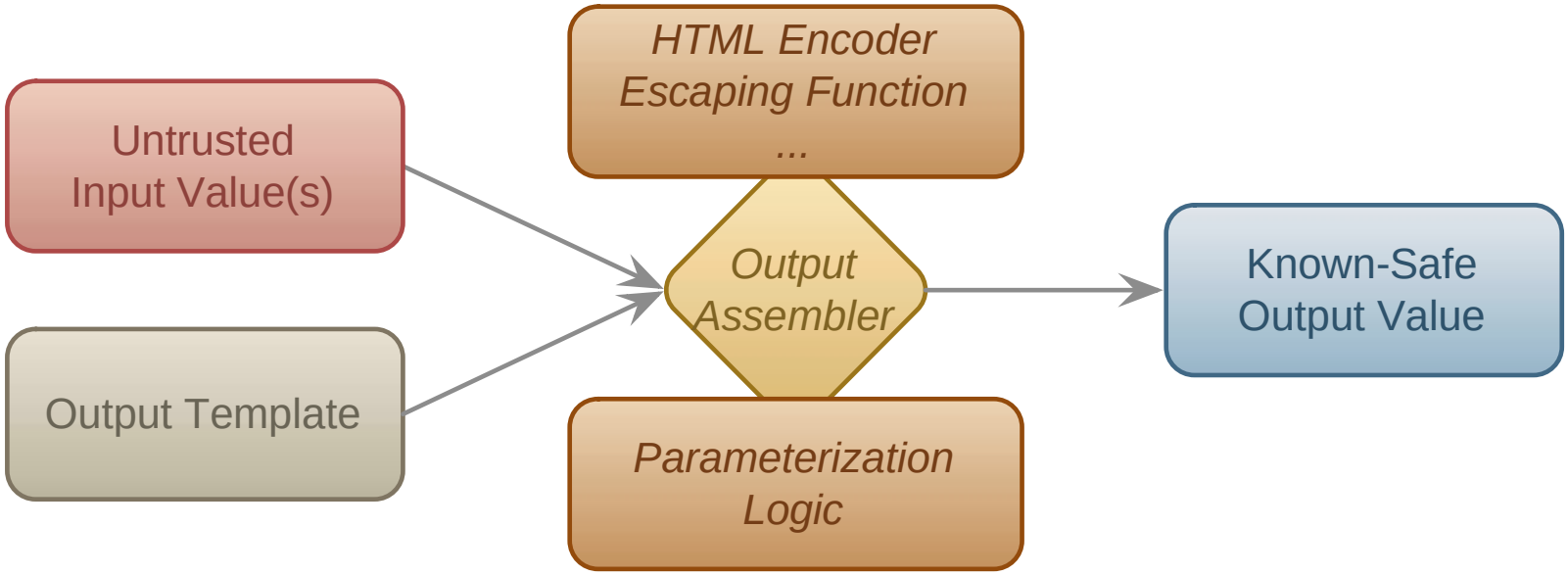
Log Injection

```
01. String val = request.getParameter("val");
02. try {
03.   int value = Integer.parseInt(val);
04. }
05. catch (NumberFormatException) {
06.   log.info("Failed to parse val = " + val);
07. }
```

Malicious Input: abc\nUser “admin” logged in successfully

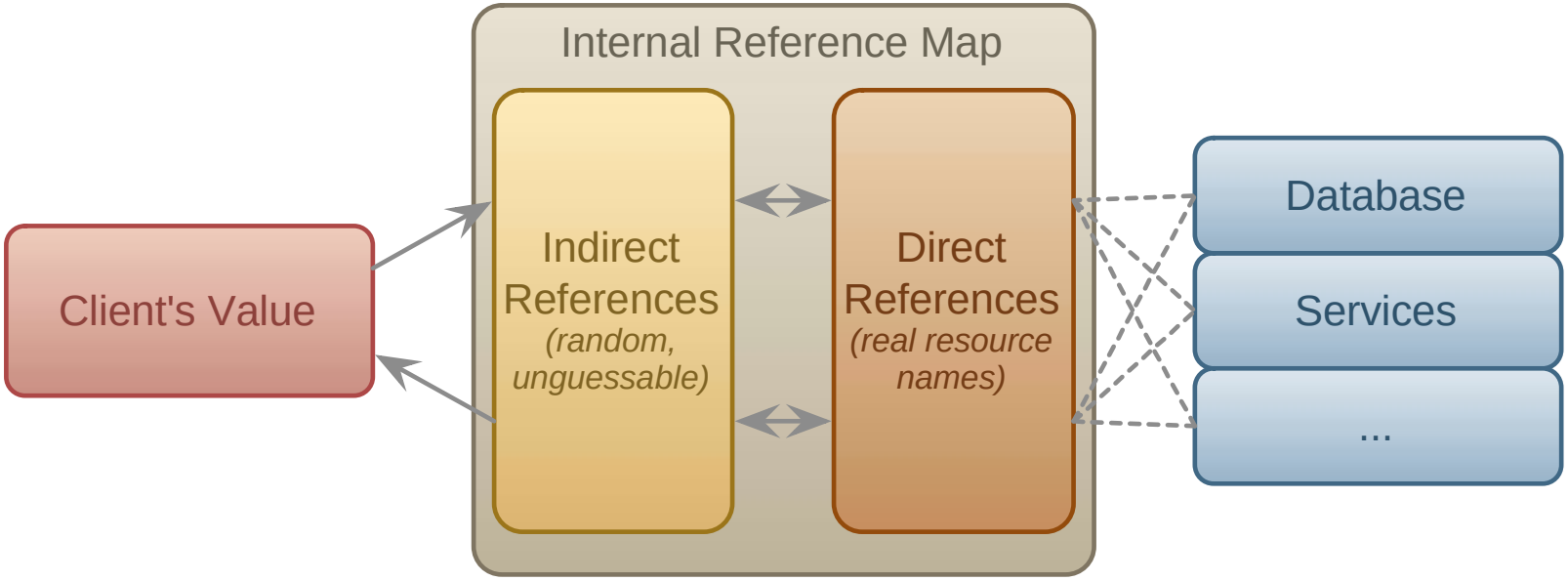
So how do we prevent it?

Protect output contexts by design



Where the API isn't given by your platforms/libraries, ***BUILD IT***!

Expose “control” resources indirectly



Thanks for your time!

Pravir Chandra
chandra@owasp.org