

Proposing SQL Statement Coverage Metrics

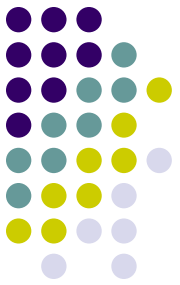
Stuart A Jaskowiak

CSC 682

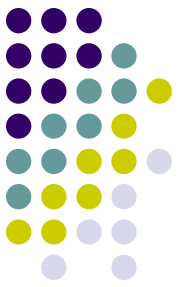
Fall 2008

About me

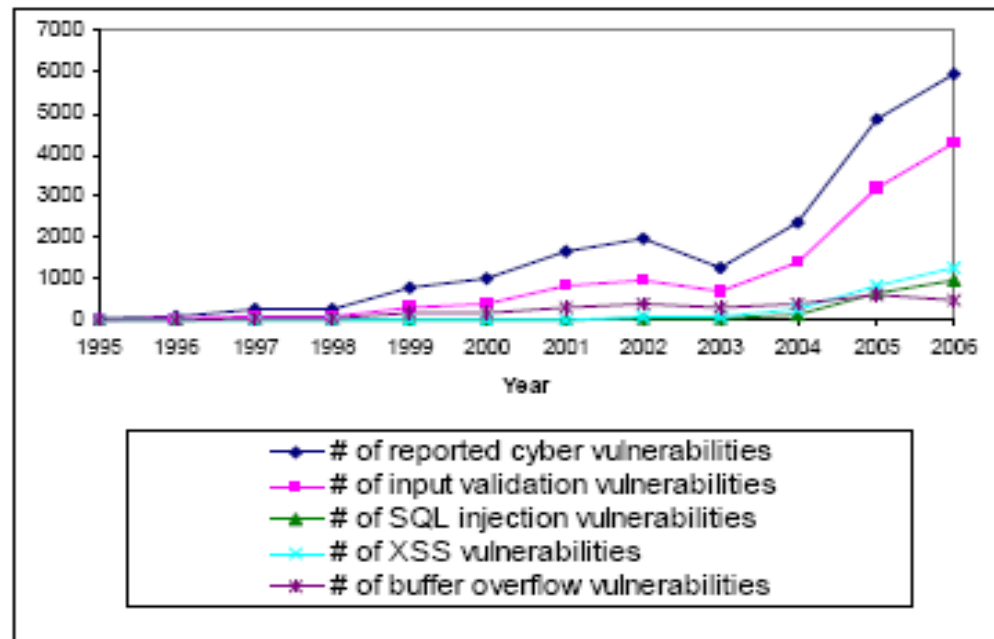
- Trip three through NKU
- 20+ years IT experience
 - Developer
 - Novell Administrator
 - Unix guy since 1994



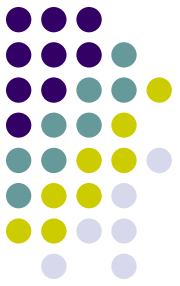
Why do we need metrics?



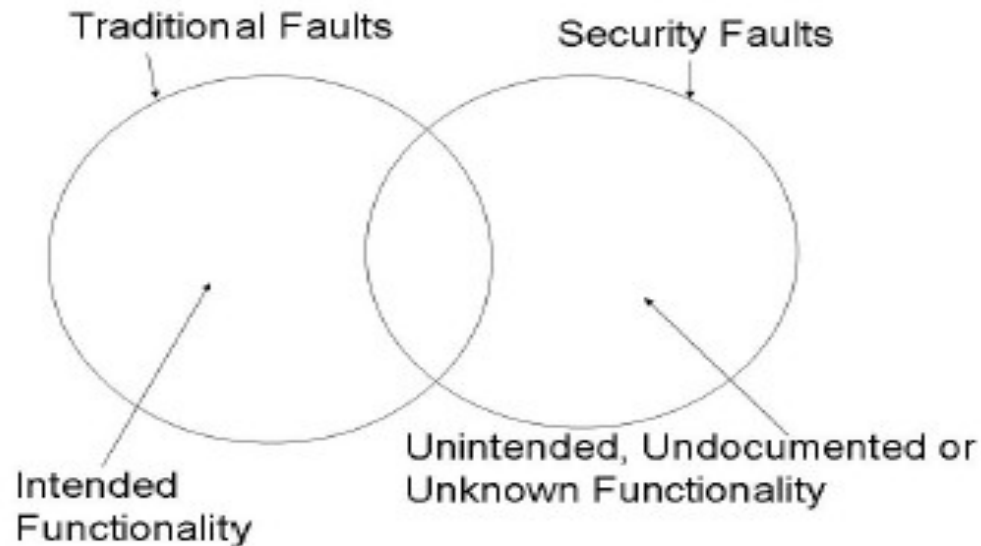
- Authors assumption no testing for issue is the reason for the rising number of SQL injection and XSS issues:



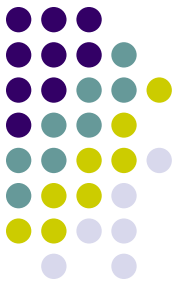
Lack of Input Sanitization



- Developers don't do user data sanitization testing
- Why?
 - Author's contention – traditional testing leaves out this area:



What do we need to test?



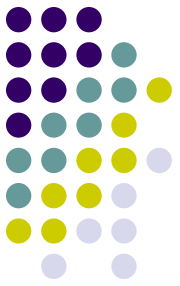
- Target Statements

- `java.sql.Statement.executeQuery(String sql)`
- `$result = mysql_query("select * from users where username = '$username' AND password = '$password'");`

- Input Variables

- **Any** variable on the server-side that gets assigned user supplied content

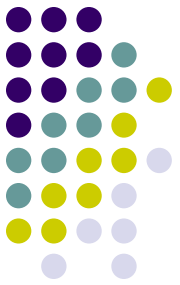
The Metrics



- Target Coverage
 - $\text{ServerSideStatement} = \text{Test}(t) / t * 100$
 - t is number of SQL statements in code
- Input Variable Coverage
 - $\text{InputVariableCoverage} = \text{Test}(f) / f * 100$
 - f is number of input variables in at least 1 test

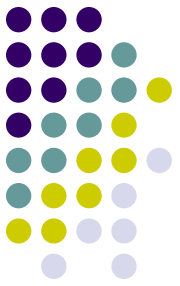
We're shooting for 100% coverage on both

Issues



- Does 100% input variable coverage and 100% target statement coverage mean it's a good test?
 - 100 % input variables is probably a good one to see since these variables typically allow user supplied data into the system

Where to add Metrics



- Paper's case study used hand written code inserted around SQL statements
 - Tedious, complicate and prone to error
- Better to encourage Junit extensions with these metrics in mind

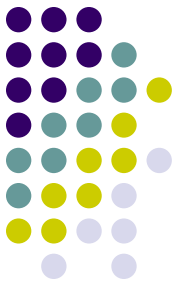
Other Issues



- Case study code JDBC specific
- Test case code contained well-written unit tests
- Issues with bulk commits:

```
public void updateDiscretionaryAccess(List<DiagnosisBean> updateDiagnoses)
{
    java.sql.Connection conn = factory.getConnection();
    java.sql.PreparedStatement ps = conn.prepareStatement("UPDATE OVDiagnosis SET
    DiscretionaryAccess=? WHERE ID=?");
    for (DiagnosisBean d : updateDiagnoses) {
        ps.setBoolean(1, d.isDiscretionaryAccess());
        ps.setLong(2, d.getOvDiagnosisID());
        ps.addBatch();
    }
    SQLMarker.mark(1, 2);
    ps.executeBatch();
}
```

Final Thoughts



- Definitely a novel concept
- Is some error prone testing better than no testing?
- Can it be expanded to other frameworks?

Questions?

