Static and Runtime Solutions for Web Application Vulnerabilities

Benjamin Livshits, Stanford University

Web Application Vulnerabilities on the Rise

Compared to several years ago vulnerabilities like SQL injections and cross-site scripting attacks dominate the charts.

Griffin Application Security Project

http://suif.stanford.edu/~livshits/work/griffin/

We propose a hybrid static/runtime solution to Web application vulnerabilities. Our focus is on Java J2EE applications.

Goes after the most prominent vulnerability types:
- SQL injections
- Cross-site scripting
- Path traversal
- HTTP splitting
- etc.

An extensible definition language PQL is used for specifying vulnerabilities.

Static Error Detection

Analyze applications as they are being developed

Advantages:
- Finds vulnerabilities early in development cycle
- Sounds, so finds all vuln. of a particular type
- Can run after every build ensuring continuous security


```java
query simpleSQLInection
returns
  object String param, derived;
uses
  object HttpServletRequest req;
  object Connection con;
  object StringBuffer temp;
matches {
  param = req.getParameter(_);
  temp.append(param);
  derived = temp.toString();
  con.executeQuery(derived);
}
```

Static analysis is based on a state-of-the-art fully context-sensitive pointer analysis with extensions.

Many practical issues needed to be addressed:
- Handle containers without a loss of precision
- Construct the application call graph in the presence of reflective constructs of Java (see "Reflection Analysis for Java", Livshits, Whaley, and Lam, Nov. 2005)

Result summary:
- Analyzed 9 large open-source Web applications in Java
- Thousands of users combined
- 29 vulnerabilities found, most confirmed and fixed

Runtime Prevention & Recovery

Protect existing applications

Advantages:
- Prevents vulnerabilities from doing harm
- Safe mode for Web application execution
- Can quarantine suspicious actions, application continues to run
- No false positives


Runtime analysis works by instrumenting an existing application to look for matches of a specified pattern. A recovery policy can be specified also

Some issues to address:
- Overhead can be high (usually 35-55%)
- Have a static optimization technique that brings the overhead down to several percent

Result summary:
- Detected and prevented exploits in all our experiments
- Unoptimized overhead: 57% average
- Optimized overhead: 14% average
- Static privation removes 82-99% of instr. points