About me

- Security Researcher @ Fortify Software
  - Focus on new techniques to find vulnerabilities (static and dynamic)
  - Find new ways to protect WebApps
- Contributor to BSIMM Europe
- Conference Speaker (academic and industry)
- History in Code Obfuscation (& Binary Rewriting)
Setup

- Introduction to static analysis
- Demo:
  - Scanning a sample application
  - Going through issues
  - Fine tuning the analysis (custom rules)
Success is foreseeing failure.
– Henry Petroski
## Security approach these days

<table>
<thead>
<tr>
<th>Try Harder</th>
<th>Fix It Later</th>
<th>Test Your Way Out</th>
</tr>
</thead>
</table>
| • Our people are smart and work hard.  
  • Just tell them to stop making mistakes. | • Code as usual.  
  • Build a better firewall (app firewall, intrusion detection, etc.) | • Do a penetration test on the final version.  
  • Scramble to patch findings. |

---

<table>
<thead>
<tr>
<th>Fix It Later</th>
<th>Test Your Way Out</th>
</tr>
</thead>
</table>
| • More walls don’t help when the software is meant to communicate.  
  • Security team can’t keep up. | • Pen testing is good for demonstrating the problem.  
  • Doesn’t work for the same reason you can’t test quality in. |

---

• Not everyone is going to be a security expert.  
• Getting security right requires feedback.
Security in the Development Lifecycle

- Plan
- Build
- Test
  - Firewalls
  - Intrusion Detection
  - Penetration Testing
- Field
Security in the Development Lifecycle

Plan
- Risk Assessment
- Code Review
- Security Testing

Build

Test

Field

Effective security from non-experts
Security in the Development Lifecycle
This Talk: Analysis during the Development Lifecycle

- **Plan**
- **Build**
  - Static Analysis
- **Test**
  - Dynamic Analysis
- **Field**
  - Runtime Protection
Security in the Development Lifecycle

Plan

Build

Static Analysis

Test

Dynamic Analysis

Field

Runtime Protection
Static Analysis: Defined

- Analyze code without executing it
- Consider many more possibilities than you could execute with conventional testing
- Doesn’t know what your code is supposed to do
- Must be told what to look for
Static Analysis: The Tool
Under the Hood of a Static Analysis Tool

Source Code → Build Model → Perform Analysis → Present Results

Security Knowledge
Code Example: SQL Injection

```java
user = request.getParameter("p_user");

try {
    sql = "SELECT * FROM users WHERE id='" + user + "'";
    stmt.executeQuery(sql);
}
```

Sources of taint:
- Class: ServletRequest, Function: getParameter

PassThrough:
- Class: String

Sinks:
- Class: Statement, Function: executeQuery

1. Source Code
2. Model
3. Security Knowledge
user = request.getParameter("p_user");
try {
    sql = "SELECT * FROM users " + 
          "WHERE id='" + user + "'";
    stmt.executeQuery(sql);
}

Sources of taint:
- Class: ServletRequest, Function: getParameter

PassThrough:
- Class: String

Sinks
- Class: Statement, Function: executeQuery

1. Source Code
2. Model
3. Security Knowledge
4. Perform Analysis
5. Present Results
Critical Attributes

- Language support
  - Understands the relevant languages/dialects
- Analysis algorithms
  - Uses the right techniques to find and prioritize issues
- Capacity
  - Able to gulp down millions of lines of code
- Rule set
  - Modeling rules, security properties
- Results management
  - Allow human to review results
  - Prioritization of issues
  - Control over what to report
Only Two Ways to Go Wrong

- False positives (false issues reported)
  - Incomplete/inaccurate model
  - Missing rules
  - Conservative analysis

- False negatives (real issues not reported)
  - Incomplete/inaccurate model
  - Missing rules
  - Forgiving analysis

The tool that cried “wolf!”

Missing a detail can kill.

Developer

Auditor
Two Ways to Use the Static Analysis Tool

1. Analyze completed programs
   - Large number of results
   - Most people have to start here
   - Good motivator

1. Analyze as you write code
   - Run as part of build
   - Nightly/weekly/milestone
   - Fix as you go
Adopting a Static Analysis Tool

1) Some culture change required
   - More than just another tool
   - Often carries the banner for software security program
   - Pitfall: the tool doesn’t solve the problem by itself

2) Define the playing field
   - Choose specific objectives
   - Build a gate

3) Teach up front
   - Software security education is paramount
   - Tool training is helpful too
4) Start small
   - Do a pilot rollout to a friendly dev group
   - Build on your success

5) Go for the throat
   - Tools detect lots of stuff. **Turn most of it off.**
   - Focus on easy-to-understand, highly relevant problems.

6) Appoint a champion
   - Make sure there is a point person on the dev team
   - Choose a developer who knows a little about everything
Adopting a Static Analysis Tool

7) Measure the outcome
   - Keep track of tool findings
   - Keep track of outcome (issues fixed)

8) Make it your own
   - Investigate customization
   - Map tool against internal security standards.
   - Best case scenario is cyclic:
     - The tool reinforces coding guidelines
     - Coding guidelines are written with automated checking in mind

9) The first time around is the worst
   - Budget 2x typical cycle cost
   - Typical numbers: 10% of time for security, 20% for the first time
Challenges of Static Analysis

1. Completed programs
   - Are not written with security in mind
   - Contain multiple paradigms and technologies
   - Exemplify varying developer skill and techniques

2. Which causes static analysis to produce
   - Large numbers of issues
   - Widely varying issues
   - Issues that are difficult to triage
Security in the Development Lifecycle

Plan

Build

Test

Field

Static Analysis

Dynamic Analysis

Runtime Protection
Team Sizes at Microsoft

![Graph showing the growth of Windows Dev vs. Test team size from 1992 to 2006.]

From *The Build Master: Microsoft’s Software Configuration Management Best Practices* (Maraia 2005)
QA people lack security understanding (and we will not force them to have that!)

Good:
- Have good test coverage
- Time and resources
Why Fault Injection Fails

- Bad input derail the program
- Cannot mutate function tests and retain coverage

Result:
- Bad test coverage
- Missed Vulnerabilities
Example: SQL Injection

```java
...
user = request.getParameter("p_user");
TaintUtil.setTaint(user, 1);
try {
    sql = "SELECT * FROM users " +
          "WHERE id='" + user + "'";
    TaintUtil.setTaint(sql, user.getTaint());
    TaintUtil.checkTaint(sql);
    stmt.executeQuery(sql);
}
...
```
Framework

- Instrument the program
  1. Methods that introduce input
     - HttpServletRequest.getParameter()
     - PreparedStatement.executeQuery()
     - ...
  2. Methods to check for taint
     - Statement.executeQuery()
     - JspWriter.print()
     - ...

- Mechanism to track Taint
  - Modify the java.lang.String class
  - Modify StringBuilder en StringBuffer
Security in the Development Lifecycle

Plan
Build
Test
Field

Static Analysis
Dynamic Analysis
Runtime Protection
Protecting Programs at Runtime

- If you can find bugs: fix them!
- Additional layer of protection
- More context than external systems:
  - Flexible response: log, block, etc
  - Low performance overhead is a must
  - Potential to detect misuse in addition to bugs
Security in the Development Lifecycle

So the 360 view of the program during the development cycle
Summary

- Mistakes happen. Plan for them!
- Security is now part of the SDLC
- Tools bring security expertise
- Tools make code review efficient
- They are not an out-of-the box solution
Thanks!

Matias Madou
mmadou@fortify.com