Static Analysis Techniques for Testing Application Security

OWASP San Antonio
January 31st, 2008
Dan Cornell – dan@denimgroup.com
Agenda

• What is Application Security?
• What is Static Analysis?
  – Static versus Dynamic
  – Overview
• Different Approaches
• Examples of Static Analysis Tools
  – FindBugs (Java)
  – PMD (Java)
  – FxCop (.NET)
  – XSSDetect (.NET)
• Process Implications
• Questions
What is Application Security?

- Ensuring that applications behave as expected under the entire range of possible inputs
- Really a subset of software correctness/QA – however…
- More typically focused on what an application is NOT supposed to do rather than what it IS supposed to do
What is Static Analysis?

- Analyzing software artifacts in order to gain information about the software
  - *Source code*
  - *Binaries*
  - *Configuration files*
- Analyzing software “at rest”
- Also called “white box testing” and “source code review”

- PLEASE NOTE: Unless otherwise discussed, Static Analysis will refer to Static Analysis being performed by an automated tool
Dynamic Analysis

- Examining running software to see how it behaves under different stimuli
  - Analyzing request and response patterns
  - Checking remotely-detectable configuration settings
Which to Use?

• Static Analysis
  – Advantages
  – Disadvantages

• Dynamic Analysis
  – Advantages
  – Disadvantages

• Actually Making a Decision
Static Analysis Advantages

- Have access to the actual instructions the software will be executing
  - No need to guess or interpret behavior
  - Full access to all of the software’s possible behaviors
Static Analysis Disadvantages

- Require access to source code or at least binary code
  - Typically need access to enough software artifacts to execute a build
- Typically require proficiency running software builds
- Will not find issues related to operational deployment environments
Dynamic Analysis Advantages

- Only requires a running system to perform a test
- No requirement to have access to source code or binary code
- No need to understand how to write software or execute builds
  - Tools tend to be more “fire and forget”
- Tests a specific, operational deployment
  - Can find infrastructure, configuration and patch errors that Static Analysis tools will miss
Dynamic Analysis Disadvantages

• Limited scope of what can be found
  – Application must be footprinted to find the test area
  – That can cause areas to be missed
  – You can only test what you have found

• No access to actual instructions being executed
  – Tool is exercising the application
  – Pattern matching on requests and responses
Actually Making a Decision

- No access to source or binaries?  **Dynamic**

- Not a software developer, don’t understand software builds?  **Dynamic**

- Performing a “pen test” or other test of an operational environment?  **Dynamic**

- None of the previous problems?  **Static**

- Really want to do the job right?  **Both (and then some…)**
Actually Making a Decision

- In our experience:
  - **Information Security** practitioners are more comfortable with the Dynamic Analysis tools
    - *Analog to scanners such as Nessus or ISS*
  - **Software Development** practitioners are comfortable with both Static and Dynamic Analysis tools, but can get the most value out of Static Analysis tools
    - *More complete view of the software*
    - *Integration with IDEs is a plus*
- Understand that there are things that tools can find, and things tools can’t find. **Running a tool doesn’t make you “secure”**
Overview

- General Approach
- Source or Binary?
General Approach
Source or Binary?

- Having access to source typically provides more information to the analysis tool than only having access to the binaries
- Some environments (Java, .NET) are easy to decompile
  - “Source” is thus easily recoverable
  - Most .NET tools actually use binaries and disassemble them into IL
Different Approaches

- Increasing the scope of analysis increases the capability of the tool to find potential errors.
- As scope increases, tools must either effectively prioritize analysis options or risk having excessive runtimes.
Scope and Capability

Scope of Analysis versus Capability of Tool
Line Focus

• Like using “grep” to identify banned or suspect function calls
• This was the approach taken by early tools
• Good way to make a quick pass for potential vulnerabilities
  – Good for targeting manual review
• Challenging to use on large codebases
• The more “signatures” that are included, the higher the noise to signal ratio will be
  – Just looking for specific functions
Line Focus Example

- Rule: `gets()` is BAD

- Input:
  ```c
  my_str = gets();
  ```

- Result: Flag this line for review

- Pretty basic, but better than nothing
Line Focus: C/C++

- Known “bad” APIs:
  - `strcpy()`
  - `gets()`
  - `scanf()`
  - `sprintf()`
Line Focus: Java

- SQL injection
  - [Connection].createStatement()
- XSS
  - <%= 
- More general parameter tampering:
  - [HttpServletRequest].getParameter()
  - [HttpServletRequest].getParameterValue()
  - [HttpServletRequest].getCookies()
  - [HttpServletRequest].getHeader()
Line Focus: .NET

- SQL Injection:
  - `SqlCommand`

- XSS
  - `<%=` 

- More general parameter tampering
  - `Request` 
  - `Request.Cookies` 
  - `Request.Headers`
Two (Crappy) Scripts I Wrote

• dotnetcheck.sh and javacheck.sh
• Implement the checks I mentioned above
Function and Module Focus

- At this point the tool needs to be acting as a compiler
  - *Parse into tokens, determine lexical structure*
- This allows for much more sophisticated analysis
  - *State machines*
  - *Control flow*
  - *Data flow*
Function and Module Focus
Example

• Rule: Memory should only be freed once

• Input:
  void f()
  {
    my_mem = malloc(256);
    free(my_mem);
    free(my_mem);
  }

• Result:
  – my_mem is marked as allocated
  – my_mem is marked as freed
  – Flag the second call to free(my_mem) as an issue
Program and System Focus

- Expanding the scope of inquiry allow tools to find more and more subtle flaws
- Also helps avoid false positives
Dataflow and Taint Tracking

• Track dataflows through the system
  – Sources and Sinks

• Attach taint flags to inputs
  – Web parameters and cookies
  – Data read from files
  – Environment variables
  – Data read from databases
  – Data read from web services

• What type of taint?
  – From the network
  – From a configuration setting
  – From a database
  – And so on

• Identify “cleaning” functions
Program and System Focus

Example

• Rule:
  – User-supplied data should never be included in a SQL query without being properly escaped
Program and System Focus
Example (continued)

- **Input:**

```java
public void doGet(HttpServletRequest req, HttpServletResponse resp) {
    String user = req.getParameter("username");
    logStuff(user, "my_page");
    // Render out HTML...
}

private logStuff(String user, String location) {
    Connection con = getConnection();
    Statement stmt = con.createStatement();
    String sql
        = "INSERT INTO log (user, location) VALUES (" + user + ", ", " + location + ");
    stmt.executeUpdate(sql);
}
```
Program and System Focus Example (continued)

- Result:
  - Input from getParameter() call is marks user variable as tainted (Source)
  - Flow of data is traced into the logStuff() method
  - sql variable is also marked as tainted when it is concatenated with username parameter
  - executeUpdate() is marked as a security issue because it received tainted data (Sink)
Examples of Static Analysis Tools

- FindBugs (Java)
- PMD (Java)
- FxCop (.NET)
- XSSDetect (.NET)
FindBugs (Java)

- Java-based static analysis tool
- LGPL-licensed
- Originally developed by Dr. Bill Pugh from the University of Maryland
- Intended to find correctness issues, also identifies some security issues

findbugs.sourceforge.net
PMD (Java)

- Java-based static analysis tool
- BSD-licensed
- Lead developers are David Dixon-Peugh and Tom Copeland
- Intended to find correctness and complexity issues, also finds some security issues

pmd.sourceforge.net
FxCop (.NET)

- Microsoft-provided tool for .NET static analysis
- Freely available
- Enforces coding standards (variable naming, etc)
- Similar to FindBugs in its security capabilities

www.gotdotnet.com/Team/FxCop/
**XSSDetect (.NET)**

- Microsoft-provided tool for .NET static analysis
- Freely available (BETA!)
- Performs data flow analysis to identify Cross Site Scripting (XSS) defects


- Based on the Microsoft Research Phoenix framework
  - *For software analysis and optimization*
  - research.microsoft.com/phoenix/
Limitations

• Static Analysis tools are a **starting point** for code review.  **Not a complete solution.**
• Static Analysis tools (like all automated tools) do not understand what your application is supposed to do
  – *Out of the box rules are for general classes of security defects*
  – *Applications can still have issues with authorization and other trust issues*
  – *Only cover 50% of security defects (Dr. Gary McGraw)*
• False positives can be time consuming to address
• Solutions?
  – *Custom rules can help to add some application specific context*
Process Implications

- Static Analysis tools can provide tremendous benefits
- It is easier to start a new project using a tool than to impose one on an existing system
- I have found that using a Static Analysis tool while developing helps to improve my coding skills
  - Immediate feedback when mistakes are made
  - Learn more about language and platform internals
Process Implications: Questions

• Who is going to run the tool?
• When is the tool going to be run?
• What will be done with the results?

• Until you can answer these questions, you should not assume that a Static Analysis tool will help you improve security
Additional Resources

- Book: Secure Programming With Static Analysis (Brian Chess and Jacob West)
  - blogs.msdn.com/fxcop/
- Website: FindBugs publications page
  - findbugs.sourceforge.net/publications.html
- Various commercial vendors…
Questions

Dan Cornell

dan@denimgroup.com

(210) 572-4400

Website: www.denimgroup.com
Blog: denimgroup.typepad.com