SAMATE Project Update

Paul E. Black
Romain Gaucher
National Institute of Standards and Technology
http://samate.nist.gov/
paul.black@nist.gov
romain.gaucher@nist.gov

28 January 200

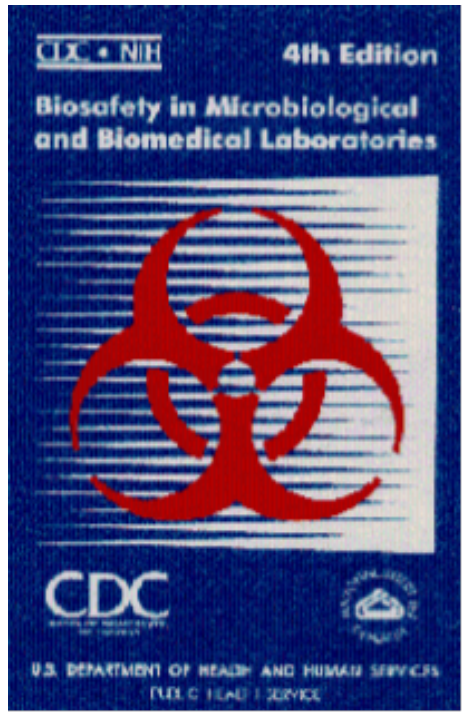
National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Overview

- **Software Assurance Metrics And Tool Evaluation (SAMATE) project is sponsored in part by DHS**
- **Current areas of concentration**
  - **Web application scanners**
  - **Source code security analyzers**
  - **Tool effectiveness studies**
- **New areas**
  - ***Binary analyzers***
  - ***Static analyzer tool exposition (SATE)***
  - ***Software labels***
  - ***Malware research protocols***
- **Web site http://samate.nist.gov/**

Researching Risky Software

- **Many people research malware, but there are no widely accepted protocols.**
- **Biological research has defined levels with associated practices, safety equipment, and facilities.**
- **Some approaches are**
  - **Weakened programs (auxotrophs)**
  - **Programs that ALERT**
  - **Outgoing firewalls**
  - **Isolated networks**



Software Facts Label

- **Software Facts should:**
  - **Voluntary**
  - **Absolutely simple to produce**
  - **Have a standard format for other claims**
- **What could be easily supplied?**
  - **Source available? Yes/No/Escrowed**
  - **Is default installation secure?**
  - **Accessed: network, disk, ...**
  - **What configuration files? (registry, ...)**
  - **Certificates (eg, "No Severe weaknesses found by CodeChecker ver. 3.2")**
- **Cautions**
  - **A label can give false confidence.**
  - **A label shut out better software.**
  - **Labeling diverts effort from real improvements.**

Software Facts

Name InvadingAlienOS
Version 1996.7.04
Expected number of users 15

Modules 5 483 Modules from libraries 4 102

	% Vulnerability
Cross Site Scripting 22	65%
Reflected 12	55%
Stored 10	55%
SQL Injection 2	10%
Buffer overflow 5	95%
Total Security Mechanisms 284	100%
Authentication 15	5%
Access control 3	1%
Input validation 230	81%
Encryption 3	1%
AES 256 bits, Triple DES	

Report security flaws to: ciwnmcyi@mothership.milkyway

Total Code 3.1415×10^9 function points	100%
C 1.1×10^9 function points	35%
Ratfor 2.0415×10^9 function points	65%
Test Material 2.718×10^6 bytes	100%
Data 2.69×10^6 bytes	99%
Executables 27.18×10^3 bytes	1%
Documentation 12 058 pages	100%
Tutorial 3 971 pages	33%
Reference 6 233 pages	52%
Design & Specification 1 854 pages	15%

Libraries: Sun Java 1.5 runtime, Sun J2EE 1.2.2, Jakarta log4j 1.5, Jakarta Commons 2.1, Jakarta Struts 2.0, Harold XOM 1.1rc4, Hunter JDOMv1

Compiled with gcc (GCC) 3.3.1

Stripped of all symbols and relocation information.

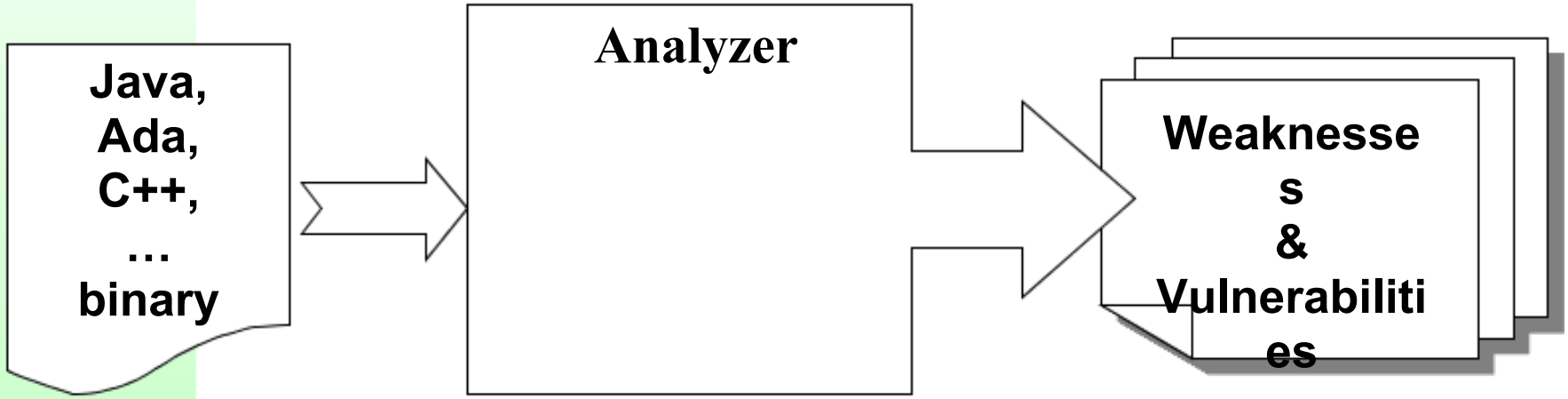
NIST National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

Paul E. Black

SATE

- **Static Analysis Tool Exposition**
  - http://samate.nist.gov/index.php/SATE
- **Goals:**
  - Enable research based on large test sets
  - Encourage improvement of tools
  - Speed adoption of tools
- **Protocol**
  1. Choose test set of programs (in C & Java)
  2. Tool makers run tools on programs
  3. Organizers develop a "master list" and analyze results
  4. All report their experience at June 2008 workshop
- **Currently choosing test set**
  - ***do you have any to share?***

Source Code Security Analyzers



- C++ test suite done
  - Covers 21 weaknesses
  - 38 for weakness, 38 for false positives, 14 for weakness suppression
- prototype Test Case Generator demonstrated
- 12 formalized CWE definitions

Web Application Security Scanners

Problems and Solutions for evaluating the tools

Romain Gaucher

January 31st 2oo8

NIST / SAMATE

Outline

1. Web Applications | Security | Scanners
2. NIST & Web Apps Security Scanner
3. Why it's hard to test a web apps scanner
4. Our solution: 3 Components based diagnostic
5. Conclusions

Web Application

security oriented characterization

A collection of technologies, server-side or client-side. But roughly, a webapp is:

- HTTP communication between client/server
- HTML/XML as main content
- JavaScript, CSS, Flash and other client-side technologies to increase the application usability
- HTTP is stateless, so Cookies/Sessions/POST/GET are used to maintain the state artificially

Web Application Security

A Web Application Security Scanner

- Black-box remote tool: access the website like a user
- Find security flaws by attacking the web site
- The tool performs 3 operations:
  1. Crawl/Extract/Understand information
  2. Attack the website
  3. Probe/Report the successful attacks

NIST & Web Apps Security Scanner

The role of NIST is to help the user to choose a tool they need.

- Specification released
- Simple test cases in the SRD
- Test plan in progress...

Our first try: the test suites

Test suites:

- Web application with seeded vulnerabilities
- Configurable defense mechanisms on each vulnerability

Experiment results & conclusions:

- simple defense mechanisms are giving hard time to the tools
- they failed at finding some vulnerabilities and we didn't know why
- Our test application is synthetic...

Challenge for a scanner

- Authentication mechanisms

With CAPTCHA? Ajax login forms? Client-Side certificate?

- Client-Side technologies

JavaScript, Flash, Aero, Silverlight, ActiveX,...

- Application logic

http://foo.com/index.php?template=accounts_page
http://foo.com/index.php?template=profile_page

- Session handling: URL/GET/POST/COOKIES

http://foo.com/09a9djs9/admin
http://foo.com/?sid=09a9djs9
http://foo.com cookie: sid=09a9djs9

Attacking is also a problem...

Example of different XSS attacks

Simple attack, can inject HTML:
"><script>alert(42)</script>
">
In HTML tag if single quotes & HTML entities escaped:
' onmouseover='alert(42)' foo='
' style=expression(alert(42)) foo='
Filter evasion using character encodings:
¼script¾alert(¢XSS¢)¼/script¾
+ADw-SCRIPT+AD4-alert('XSS');+ADw-/SCRIPT+AD4-

3 Components to diagnose the problems

Our prime interest is to understand why a tool didn't find a particular vulnerability

The solution: inserting sensors during the assessment

- Seeded vulnerabilities
- Looking at the attack surface coverage
- Probing the type of attacks (with granularity)

Seeded vulnerabilities with levels of defense

A level of defense is a combination of orthogonal defense mechanisms

- Each vulnerability in our test suites have different levels of defenses (XSS Level 0, SQL Injection Level 2, ...)
- For testing, we know where the vulnerabilities are

Attack Surface Coverage

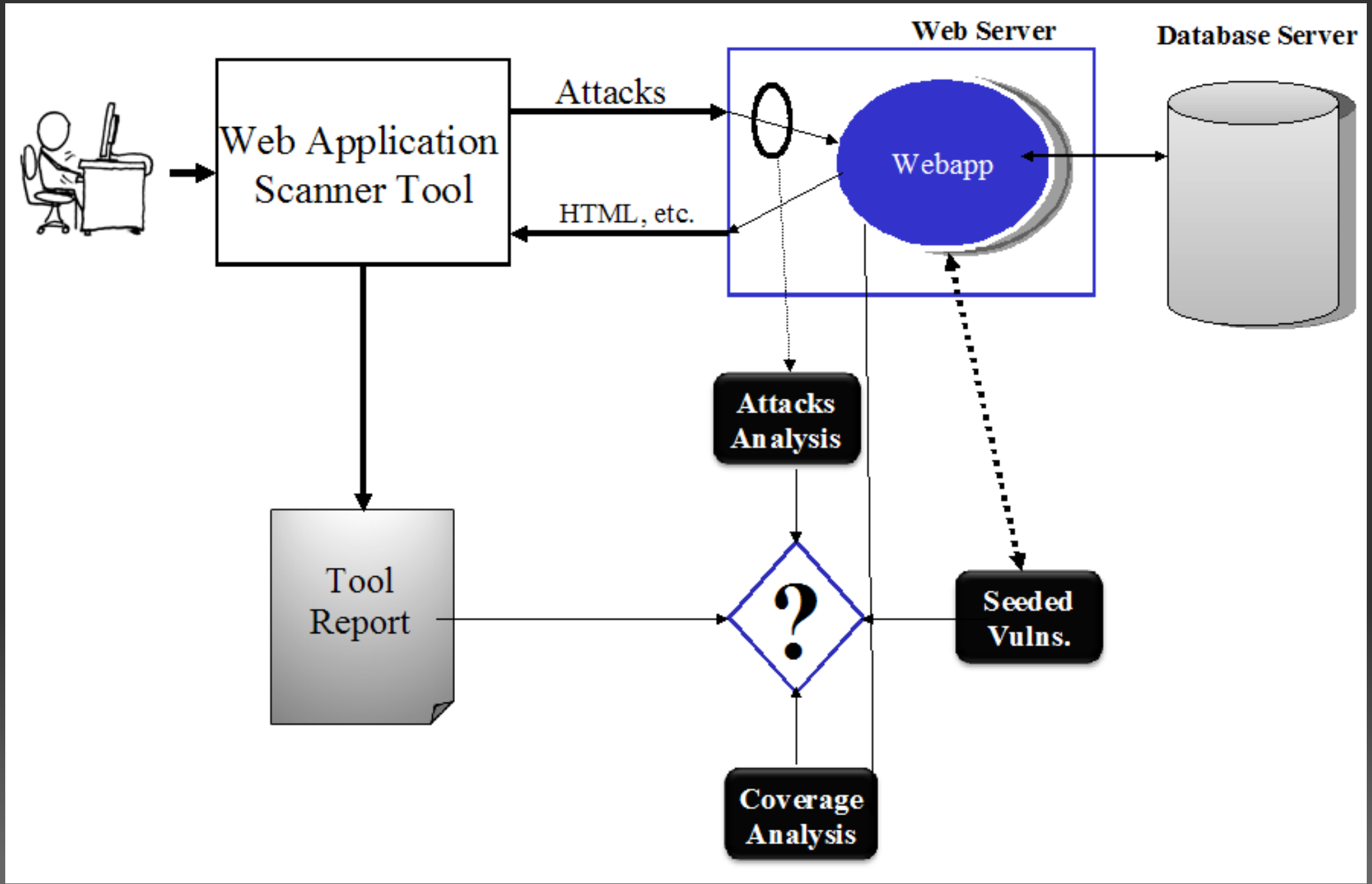
In our context, Attack Surface is all the places where a problem is most likely to occur

- Track the scanner in pages, scripts and functions
- Check that the scanner generates errors from the test application
- Look at sequences and paths

Probing the type of attacks

- Check the type of attacks the scanner did:
  - Did the tool check for Cross-Site Scripting, SQL Injection, Remote File Inclusion,...?
- What are the granularities of the attacks:
  - Did the tool attack with sophisticated attacks? ex: did it try encodings?

Combining the data while testing



Conclusions

By instrumenting our test application we can answer the question why the tools didn't find some vulnerabilities

Future work:

- Coming up with metrics
- Tool profiling: Is this tool better for a particular type of vulnerabilities? Will this fit with my website profile?

Contacts

- SAMATE website, http://samate.nist.gov

- Project Leader: Dr. Paul E. Black
- Project Team Members:
- Elizabeth Fong, Romain Gaucher, Michael Kass, Michael Koo, Vadim Okun, Will Guthrie,