VIBRANCE

A new technology security tool for Java applications

Contact:
Alessandro Coglio at Kestrel Institute
vibrance@kestrel.edu
650-493-6871

More information, including a demo video, at http://vibrance.kestrel.edu

August 2014

Technical Overview

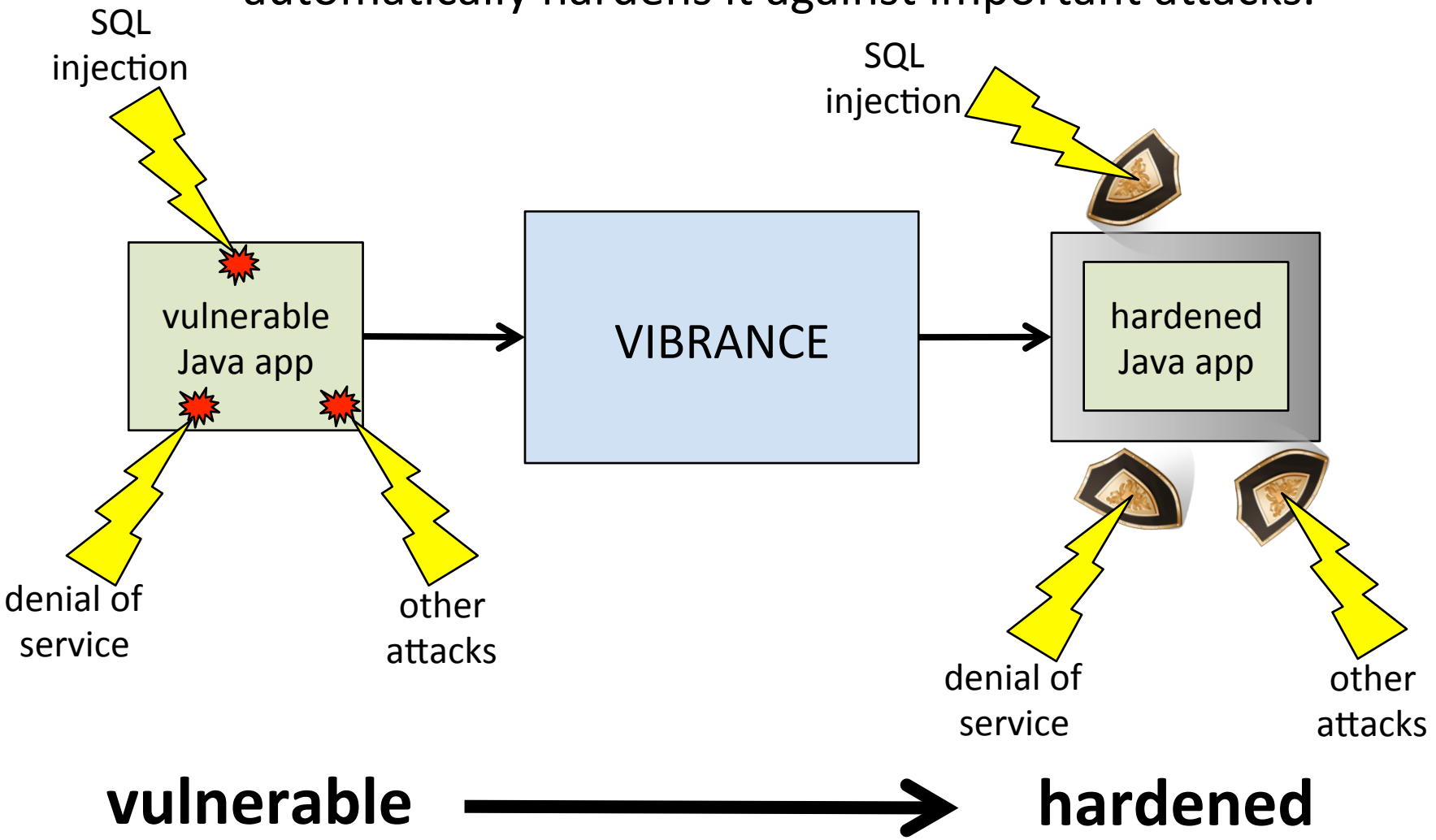
- VIBRANCE automatically hardens Java bytecode to detect and automatically remediate or block attacks.
- No source code is needed, so VIBRANCE can be used on 3rd-party code to protect servers.
- VIBRANCE can be deployed independently of other security tools, adding an additional level of security to existing systems.

VIBRANCE project background

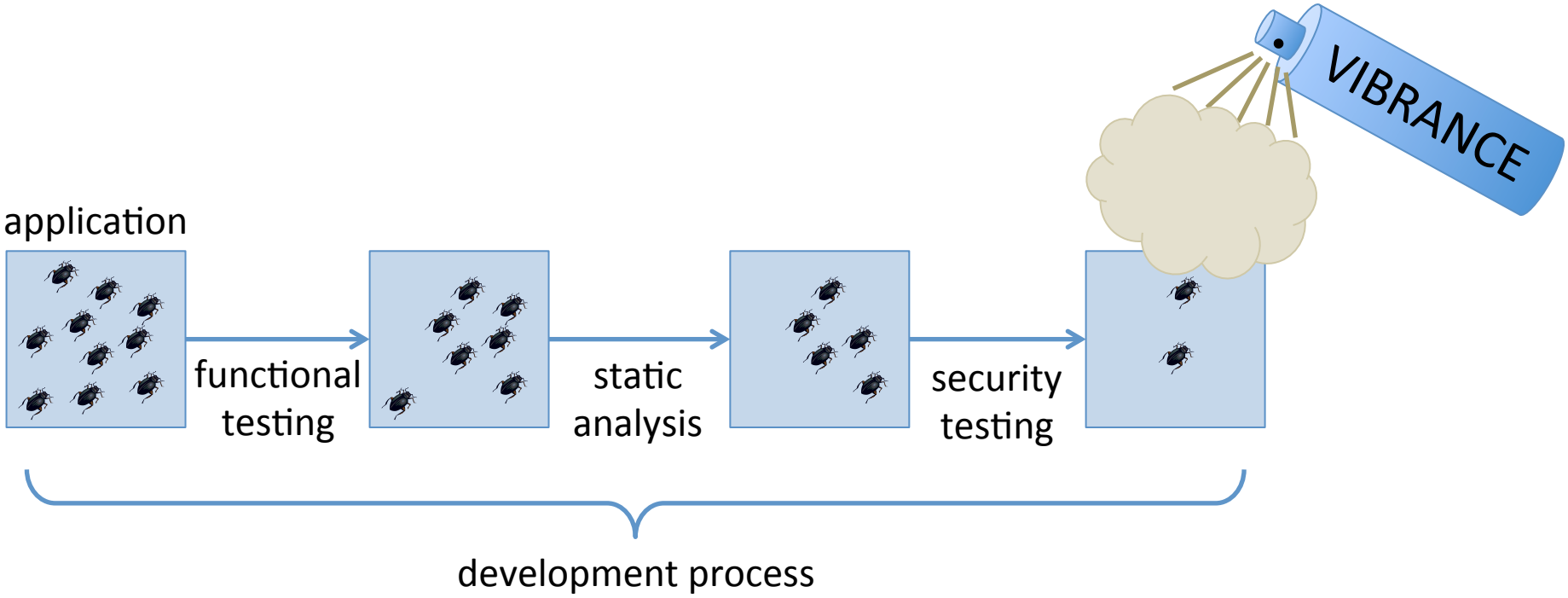
- Government funded software security research
- Significant research program: many researchers; many innovations
- At this time, VIBRANCE hardens against injection vulnerabilities; work on other vulnerabilities is ongoing.

The VIBRANCE Concept

The VIBRANCE tool starts with a vulnerable Java app and automatically hardens it against important attacks.



VIBRANCE addresses the security bugs that “slip through” the development process.



VIBRANCE technology

- Advanced static analysis: sound abstract interpretation identifies safe code
- Advanced dynamic analysis: taint is tracked down to the individual character level
- Advanced hardening technology:
  - detection
  - remediation or confinement
  - reporting

VIBRANCE protects Java applications from the following **important weaknesses**, which lead to **data theft/loss** and **denial of service**:

- injection
  - SQL – #1 of CWE/SANS Top 25
  - OS command – #2 of CWE/SANS Top 25
  - LDAP, XPath, XQuery
- tainted data
  - unrestricted file upload – #9 of CWE/SANS Top 25
  - file path traversal – #13 of CWE/SANS Top 25
  - loop bound – Apache Tomcat CVE 2014-0050
  - server crash
  - …
- number handling (e.g. integer overflow)
- error handling (e.g. uncaught exception)
- resource handling
- concurrency handling

VIBRANCE can also help Testing

- In addition to making the code more secure, VIBRANCE generates “forensic” information when attacks are caught and blocked.
- This forensic knowledge can then be applied to the development process, because this knowledge can be useful to discover vulnerabilities in the original (i.e., not VIBRANCE-hardened) applications, enabling a developer to fix them.
- We can extend VIBRANCE to produce more information at run time---even for benign inputs, not just attack inputs. This additional information can provide insight into how data flows inside the application.

How Well Does VIBRANCE Do?

- In an independent test and evaluation, VIBRANCE caught most vulnerabilities with a very small number of false positives.
  - False positives may be application-dependent, since they depend on policy. VIBRANCE allows fine-tuning policies by editing a configuration file. Given an application of interest, we believe that VIBRANCE's configuration can be fine-tuned to eliminate all false positives for that application.
- VIBRANCE scales to large applications (tested up to 500 kLOC so far).

Example

Blocked password attack

```
21:36:55 10/03/2013
WARN: CWE-89: check sqlQuoteChk: directive qstring detected one or more protected quotes
Altered line to: "SELECT id FROM employees WHERE name='Abigail' AND password=''' OR 1=1 -- ';"

Executing: java/sql/Statement.executeQuery(Ljava/lang/String;)
????????????????????????????????????????????????????????????????
......................................_______.................._.__.___.__...
SELECT id FROM employees WHERE name='Abigail' AND password='' OR 1=1 -- ';
Action: replace
  pac.config.Notify.getApplicationStackTrace  line:295
  pac.config.Notify.appendToLogFile  line:345
  pac.config.Notify.appendToLogFile  line:365
  pac.config.Notify.notifyAndRespond  line:286
  pac.config.Notify.run_checks  line:574
  pac.web.StoneSouplet.verifyPassword222  line:255
  pac.web.StoneSouplet.doGet222  line:292
  javax.servlet.http.HttpServlet.service222  line:621
  javax.servlet.http.HttpServlet.service222  line:728
  org.apache.catalina.core.ApplicationFilterChain.internalDoFilter222  line:305
  org.apache.catalina.core.ApplicationFilterChain.doFilter222  line:210
```

Blocked password attack

MITRE “Common Weakness Enumeration”
“89” is SQL Injection

New, safer SQL
constructed by VIBRANCE at runtime

```
21:36:55 10/03/2013
WARN: CWE-89: check sqlQuoteChk: directive qstring detected one or more protected quotes
Altered line to: "SELECT id FROM employees WHERE name='Abigail' AND password=''' OR 1=1 -- ';"

Executing: java/sql/Statement.executeQuery(Ljava/lang/String;)
????????????????????????????????????????????????????????????????
......................................_______.................._.__.___.__...
SELECT id FROM employees WHERE name='Abigail' AND password='' OR 1=1 -- ';
Action: replace
  pac.config.Notify.getApplicationStackTrace  line:295
  pac.config.Notify.appendToLogFile  line:345
  pac.config.Notify.appendToLogFile  line:365
  pac.config.Notify.notifyAndRespond  line:286
  pac.config.Notify.run_checks  line:574
  pac.web.StoneSouplet.verifyPassword222  line:255
  pac.web.StoneSouplet.doGet222  line:292
  javax.servlet.http.HttpServlet.service222  line:621
  javax.servlet.http.HttpServlet.service222  line:728
  org.apache.catalina.core.ApplicationFilterChain.internalDoFilter222  line:305
  org.apache.catalina.core.ApplicationFilterChain.doFilter222  line:210
```

underbars show
tainted regions

Original SQL line

Java Source for Password Query Webapp

Note: this toy example is for illustrative purposes only. Vibrance is capable of protecting much more complicated code that is hard to analyze statically.

```
public String verifyPasswordQuery(final String TEST_NAME, final String TEST_PASSWORD) {
    return String.format(
            "SELECT id FROM %s WHERE name='%s' AND password='%s';",
            TEST_TABLE_NAME, TEST_NAME, TEST_PASSWORD);
}

public String verifyPassword(final String TEST_NAME,final String TEST_PASSWORD)
throws SQLException {
    String resultString = null;
    Statement stmt = null;
    ResultSet rs = null;

    stmt = connection.createStatement();
    String query = verifyPasswordQuery(TEST_NAME, TEST_PASSWORD);
    rs = stmt.executeQuery(query);
    if (rs.next()) {
        resultString = rs.getString(1);
    }

    stmt.close();
    return resultString;
}
```

Additional Details

Where does VIBRANCE fit into the development cycle?

After functional testing, the developer uses VIBRANCE to harden the compiled code.

Then functional testing is repeated, to fine-tune the security policies so that VIBRANCE does not affect code functionality.

When the code is in production, VIBRANCE will automatically detect and prevent attacks against a variety of vulnerabilities.

When an attack is detected/prevented, VIBRANCE reports information on the attack. This report can be used like a bug report, but under most circumstances the application can remain in production until the regular development cycle permanently fixes the vulnerability.

How can VIBRANCE benefit your organization?

- improve code security and reliability
- save time and money
- give more confidence in vendor-supplied code

Today's application developer has a wealth of tools and procedures available, but for a variety of reasons, vulnerabilities can still arise in production code. VIBRANCE can catch some of those vulnerabilities at runtime, increasing security and reliability.

Developer time is a limited and expensive resource. When VIBRANCE is seen to block an attack, you can fix the problem during your regular development cycle rather than dropping everything to rush out an urgent fix.

VIBRANCE can also be used to harden code received from vendors and contractors, reducing the risk of putting that code into production.

How can VIBRANCE benefit your organization?

Dynamic Analysis improves the results of Static analysis tools.

Commercial static analysis tools typically have settings that let the developer see more or fewer potential vulnerabilities. If the developer tells a tool to show everything, they often get so many False Positives that it is impractical to review them all. Common responses are:

- ask the tool to show only high-confidence potential vulnerabilities
- similarly, ignore reported low-ranked potential vulnerabilities
- find classes of seemingly-similar vulnerabilities, and if a sample of a class are all false positives, assume all instances of the class are FP

All of these responses can and do result in missed vulnerabilities.

Solution: If the code has been hardened with VIBRANCE, it will block attacks on vulnerabilities missed by the static analysis process.

How can VIBRANCE benefit your organization?

Problem: Static analyzers are not perfect.

Even if the developer runs several static analyzers, and perfectly examines all the potential vulnerabilities, some real vulnerabilities can be missed. This is because all commercial static analyzers take shortcuts of one kind or another. Programs often have complex control flow or difficult-to-analyze operations, so analyzers have to prune the search in order to perform acceptably. An example is a regular expression used to construct a query.

Solution: VIBRANCE hardening can protect an application from vulnerabilities that static analyzers can not find.

How can VIBRANCE benefit your organization?

Problem: People are not perfect.

A mistake might not be caught by the regular development process, by unit testing, peer review, static analysis, or dynamic testing.

A vulnerability reported by static analysis or caught by functional testing might be fixed incorrectly, or an exclusion rule might accidentally be made too broad.

Solution: VIBRANCE hardening can catch attacks on vulnerabilities that may slip through the development process.

How can VIBRANCE benefit your organization?

Problem: Dynamic testing cannot test every input combination.

Even fuzzers can fail to find bad combinations of inputs. The limits to functional testing are a major reason why developers do static analysis.

Solution: VIBRANCE hardened code catches attacks on low-frequency paths and unlikely inputs just as easily as on high-frequency paths or common inputs.

How can VIBRANCE benefit your organization?

Problem: Receiving code from a vendor that is supposed to be production-ready, but not really knowing how well they practiced good software engineering.

Vendors and contractors will claim to have done everything right, but if a developer is integrating the code, the developer will have some responsibility for it. But the developer's budget for static analysis and testing might not be sufficient for the full treatment. Worse, the developer might not be getting source code, just binary or jar files.

Solution: VIBRANCE hardening can reduce the developer's worry about vendor-supplied code, including compiled code.

How can VIBRANCE benefit your organization?

Problem: Other runtime protection systems have holes and are limited to specific attack vectors.

There are a number of dynamic security tools available, but they have a less deep understanding of the application, and hence suffer from holes (false negatives) and false positives. They also only block attacks over a particular channel.

For example, a WAF (Web Application Firewall) such as ModSecurity can block many attack patterns that use HTTP. However, they are difficult to set up correctly and they can be prone to false positives. Also, they cannot handle taint that comes from files or databases, or attacks that are constructed in the application from inputs that do not match the attack patterns.

Solution: VIBRANCE hardened code can block attacks that other runtime protection systems miss.

VIBRANCE

A new technology security tool for Java applications

Contact:
Alessandro Coglio at Kestrel Institute
vibrance@kestrel.edu
650-493-6871

More information, including a demo video, at http://vibrance.kestrel.edu

August 2014