Put a “Firewall” in Your JVM
Securing Java Applications

Prateep Bandharangshi

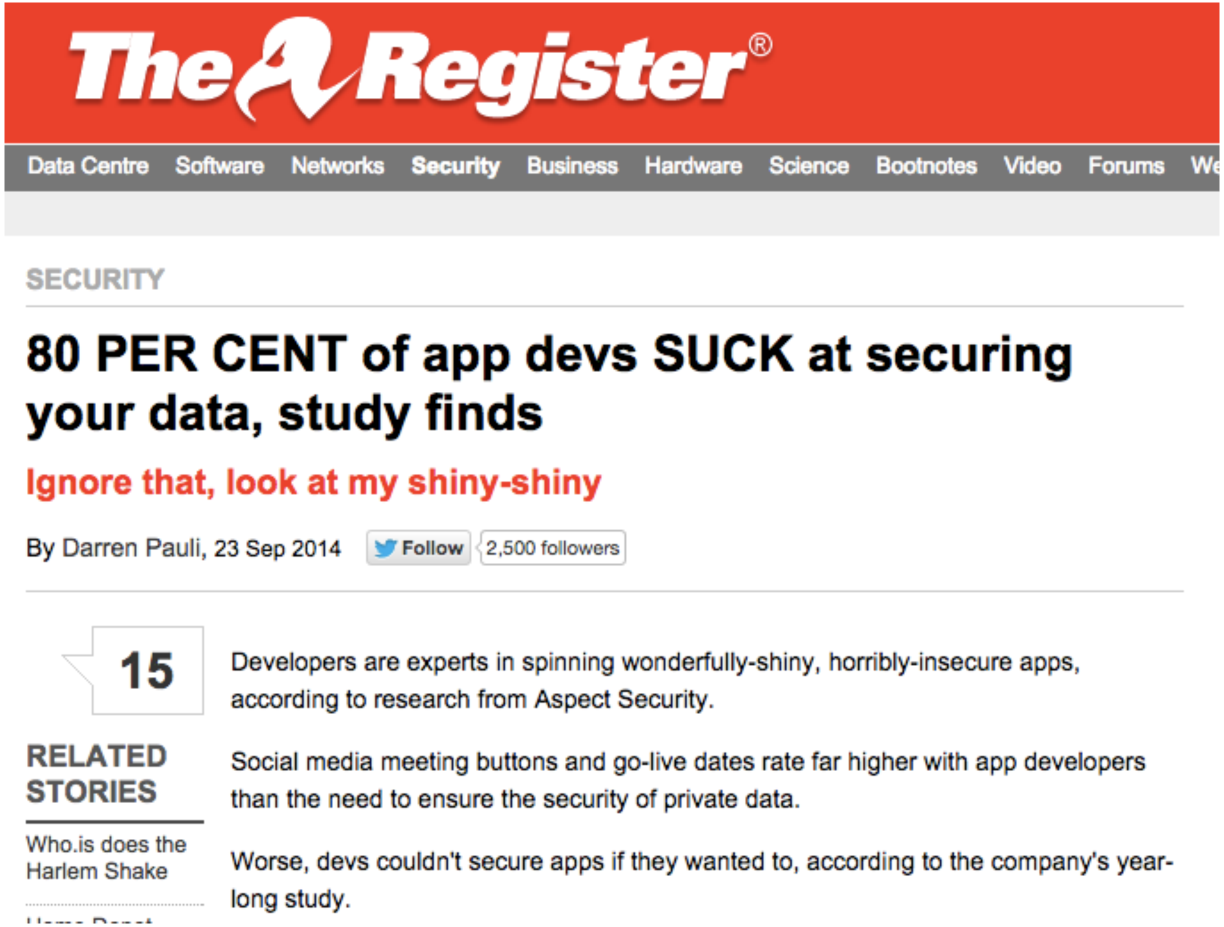
Waratek Director of Client Security Solutions

@prateep

Hussein Badakhchani

Deutsche Bank Ag London Vice President

@husseinb



SECURITY

80 PER CENT of app devs SUCK at securing your data, study finds

Ignore that, look at my shiny-shiny

By Darren Pauli, 23 Sep 2014

Developers are experts in spinning wonderfully-shiny, horribly-insecure apps, according to research from Aspect Security.

Social media meeting buttons and go-live dates rate far higher with app developers than the need to ensure the security of private data.

Worse, devs couldn't secure apps if they wanted to, according to the company's year-long study.

RELATED STORIES

Who.is does the Harlem Shake

Several former Home Depot employees said they were not surprised the company had been hacked. They said that over the years, when they sought new software and training, managers came back with the same response:

"We sell hammers"

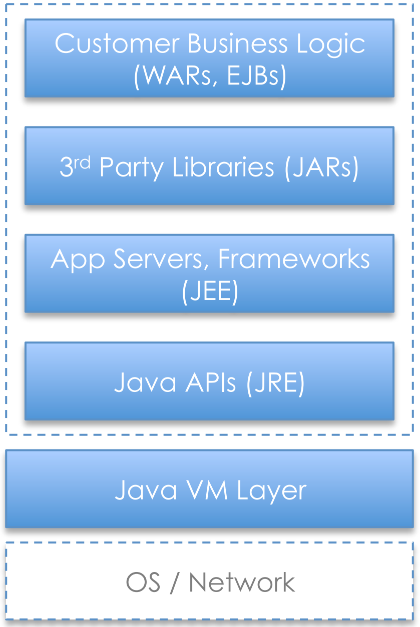
- New York Times, September 19, 2014

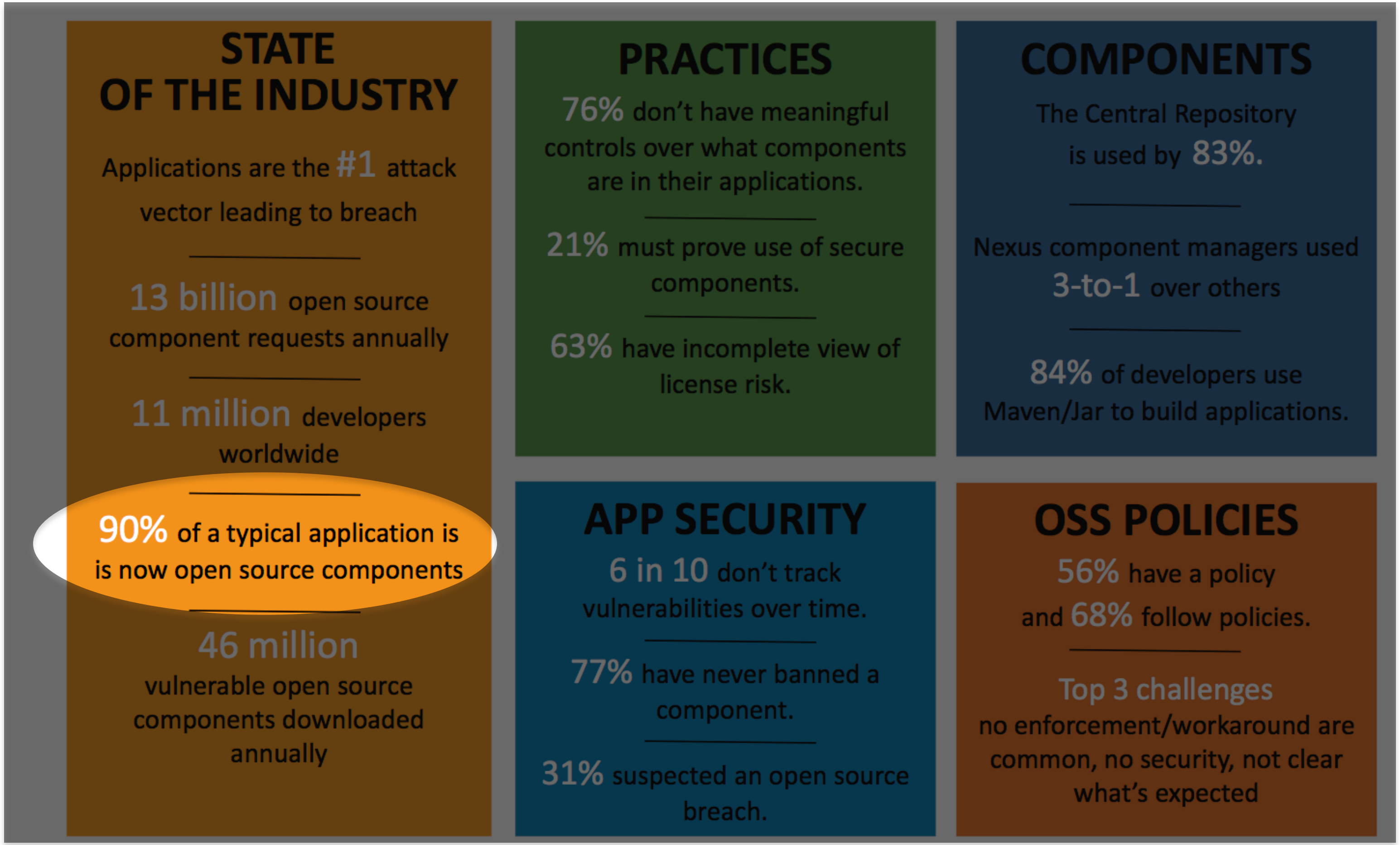
“There are over 1,000 different types of software weaknesses, most of which are impossible for developers to avoid without training and development support”

- State of Developer Application Security Knowledge
- Aspect Security, September 2014

“More Secure Application Coding” Cannot Solve the Problem

- Relatively little application code is written and controlled by the enterprise
  - 3rd party libraries heavily used and often open sourced
  - Application servers and frameworks (JBOSS, Websphere, Weblogic, Tomcat, etc.)
  - Java APIs
  - Purchased software packages
- None of these are written and maintained by the enterprise, creating substantial exposure


Customer Business Logic (WARs, EJBs)
3rd Party Libraries (JARs)
App Servers, Frameworks (JEE)
Java APIs (JRE)
Java VM Layer
OS / Network



STATE OF THE INDUSTRY

Applications are the #1 attack vector leading to breach

13 billion open source component requests annually

11 million developers worldwide

90% of a typical application is is now open source components

46 million vulnerable open source components downloaded annually

PRACTICES

76% don't have meaningful controls over what components are in their applications.

21% must prove use of secure components.

63% have incomplete view of license risk.

COMPONENTS

The Central Repository is used by 83%.

Nexus component managers used 3-to-1 over others

84% of developers use Maven/Jar to build applications.

APP SECURITY

6 in 10 don't track vulnerabilities over time.

77% have never banned a component.

31% suspected an open source breach.

OSS POLICIES

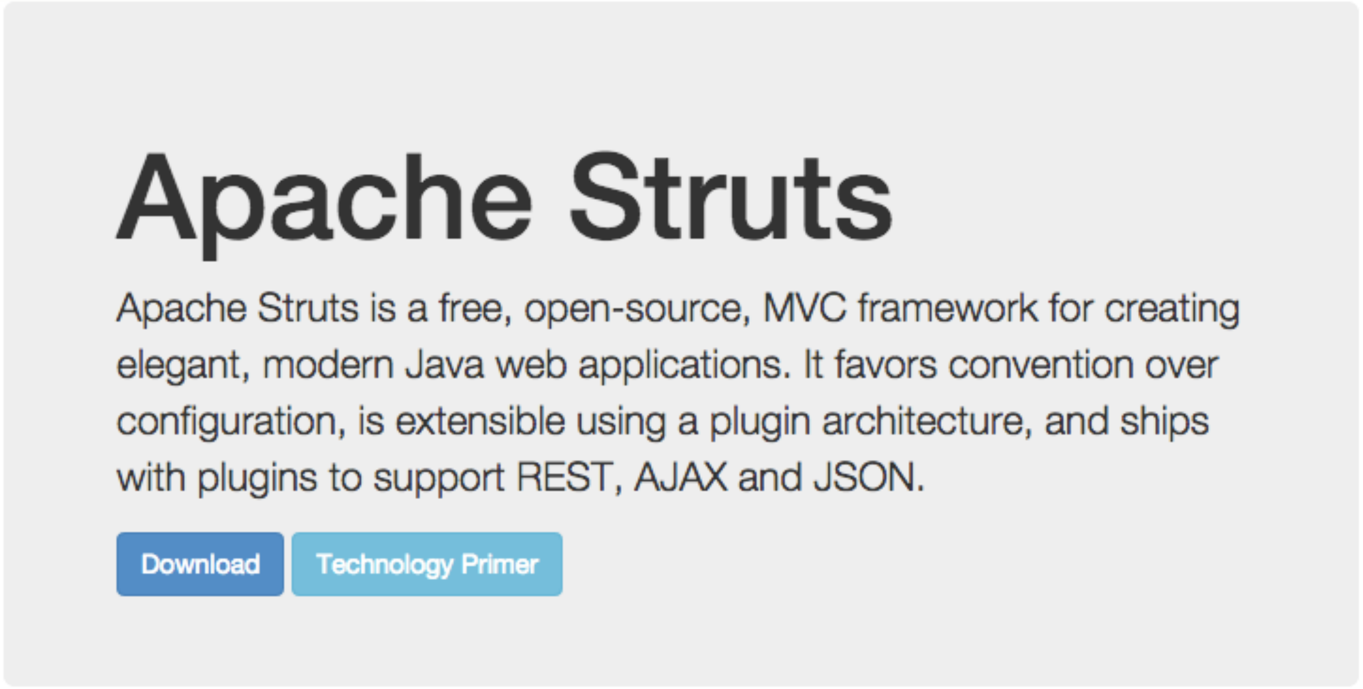
56% have a policy and 68% follow policies.

Top 3 challenges no enforcement/workaround are common, no security, not clear what's expected

Source: 2014 Sonatype Open Source and Application Security Survey

Struts™

The Apache Software Foundation
http://www.apache.org/



Apache Struts

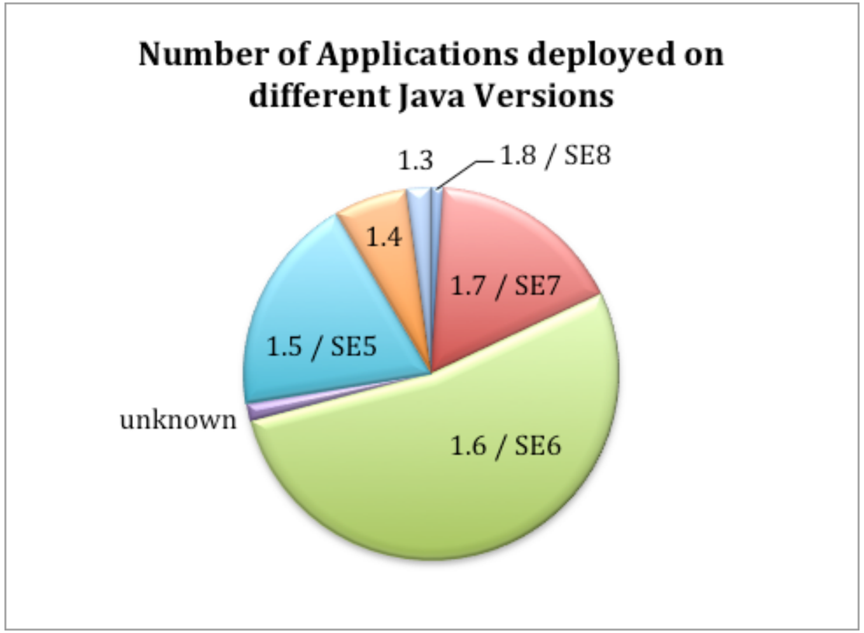
Apache Struts is a free, open-source, MVC framework for creating elegant, modern Java web applications. It favors convention over configuration, is extensible using a plugin architecture, and ships with plugins to support REST, AJAX and JSON.

Struts2 CVEs

The struts2 framework has had 10 CVEs issued in the last two years with a CVSS score of 9 or 10

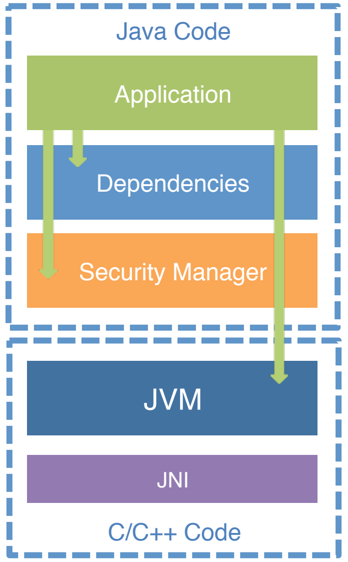
- **CVE** = **C**ommon **V**ulnerabilities and **E**xposures
- **CVSS** = **C**ommon **V**ulnerability **S**coring **S**ystem

Legacy Java



- Most enterprises have large numbers of applications running on older, legacy Java versions
- Updating these apps to the current Java edition is often risky, time consuming, and expensive

JVMs themselves are not secure



The JVM itself performs NO security checks.

All enforcement is through the Java API's and the Security Manager (SM) levels.

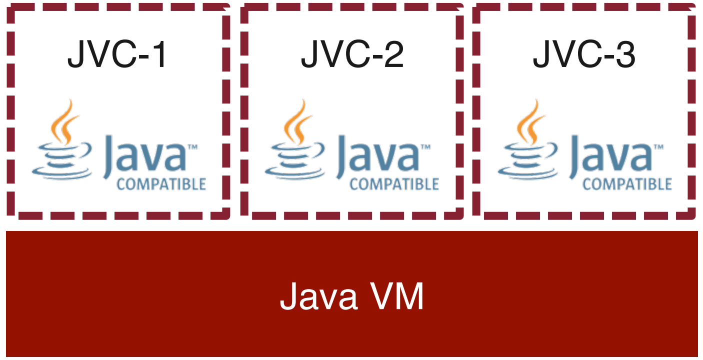
If the SM is compromised or not even enabled then the JVM can be instructed to do ANYTHING.

The JVM doesn't question when your app starts to:

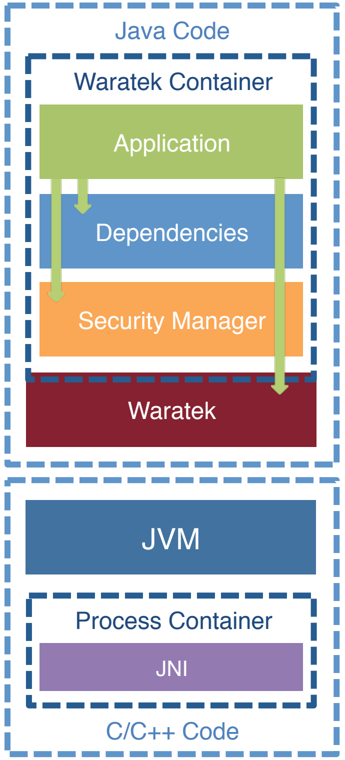
Delete files, open sockets, fork processes, change file permissions, insert SQL parameters, reflect classes etc.

Java Containers

- Waratek provides containment and isolation technology for Java built on top of Oracle Hotspot
- A Java Virtual Container is an in-JVM container with built in security and resource controls
- Java containers boot their own JRE inside them which can be different to the host JRE version
- Java Containers have elastic resources that can be changed at runtime. Resources such as memory, CPU, bandwidth and file I/O.



Java Containers monitor all activity



The container layer gets to inspect every call the application and its dependent libraries need to make.

The calls can then be checked against rules, any suspicious behavior is then logged.

We can also isolate untrusted JNI code by isolating the library within its own process container at the OS level.

Everything is local to the JVM so there is very low overhead.

No agents or code changes are required.

Rules cover all application aspects

Language rules:

- Class loading – class or package level.
- Class linking – any defined class.
- Reflection – from variable to package level.
- Throwable – exception classes.

Input Output rules:

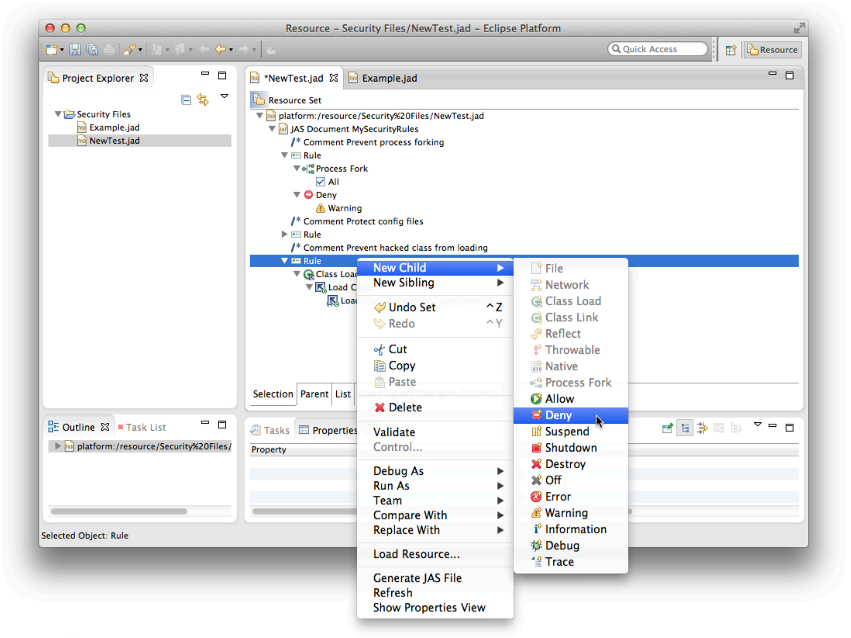
- File – any file I/O.
- Network – any network I/O.

Other rules:

- Native code – accessing JNI.
- Process execution– starting external processes.

External attack vectors:

- SQL injection.



Resource – Security Files/NewTest.jad – Eclipse Platform

```
Resource Set
  platform:/resource/Security%20Files/NewTest.jad
    JAS Document MySecurityRules
      /* Comment Prevent process forking
      Rule
        Process Fork
          All
        Deny
          Warning
      /* Comment Protect config files
      Rule
      /* Comment Prevent hacked class from loading
      Rule
        Class Loa
          Load C
            Loa
```

Selected Object: Rule

Tooling support makes rule customization simple.

What a Container Rule File Looks Like

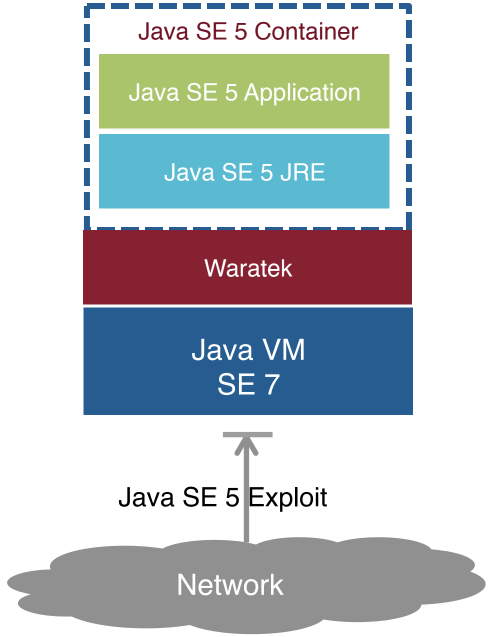
```
# Example file rules
file:read:/etc/:deny:warn
file:read:/etc/passwd:allow:warn
file:exec:*:deny:warn

# Example networks rules
network:connect:www.google.com::deny:warn
network:accept:localhost::deny:warn

# SQL injection mitigation for Oracle PL/SQL
sql:database:oracle:deny:warn

# Virtual patch rule for an Apache Struts2 vulnerability
reflect:class:com.opensymphony.xwork2.ognl.SecurityMemberAccess:method
:setAllowStaticMethodAccess(Z)V:deny:warn
```

Rules can secure any Java version



This approach to legacy Java has two key advantages, first no changes to the application:

- The application does not see an API change.
- Deprecated calls still function.
- Serialized objects still function.
- The application is still using the API it was first tested against.

Second, externally the world can move on:

- Administration is on an up-to-date SUPPORTED platform.
- The surrounding infrastructure can be updated.

SQL Injection

```
String id = request.getParameter("id");
String sql = "SELECT * FROM users WHERE id = '" + id + "'";
```

SQL Injection

```
http://example.com/page.jsp?id=1' OR 1=1--

http://example.com/page.jsp?id=1%27%20OR%201=1--

SELECT * FROM users WHERE id = '1' OR 1=1--'
```

Stopping SQL Injection

- Java Containers automatically perform runtime “taint-tracking” without any changes to application code
- Taint-tracking marks as “untrusted” all user-input data to a Java app (like HTTP request parameters) in real-time
- When “untrusted” user-input data is passed to an SQL query, syntactic analysis is used to accurately and reliably detect SQL injection
  - When SQL injection is detected, the Java Container **gracefully rejects** the unsafe SQL query and the application continues un-exploited
  - No tuning
  - No regex
  - No false positives
  - No code changes

DEMO

Runtime Application Self Protection (RASP)

“Applications should be capable of self-protection — that is, have protection features built into the application runtime environment.”

[RASP solutions] possesses the maximum information necessary to make the most accurate detection or protection decision.

- Gartner

Agenda

1	Securing Legacy Java – *DAP Secure Legacy Java*
2	Virtual Patching – *Waratek's Application Firewall*
3	Application Security – *Lock down the application*

Agenda

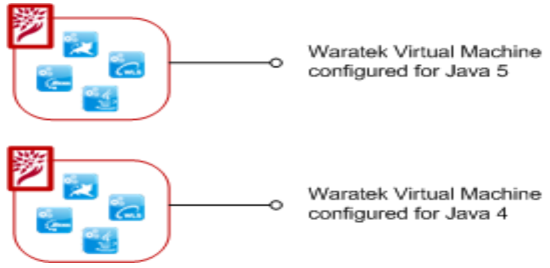
1	Securing Legacy Java – *DAP Secure Legacy Java*
2	Virtual Patching – *Waratek's Application Firewall*
3	Application Security – *Lock down the application*

Securing Legacy Java

Using the Waratek Cloud VM in conjunction with DAP, the immediate risks posed by legacy applications can be mitigated and a route for the upgrade and future sustained management of the application can be established.

Applications will be quarantined without need of code changes or transformation and once the upgrades to new versions of Java have been tested the application is promoted to production.

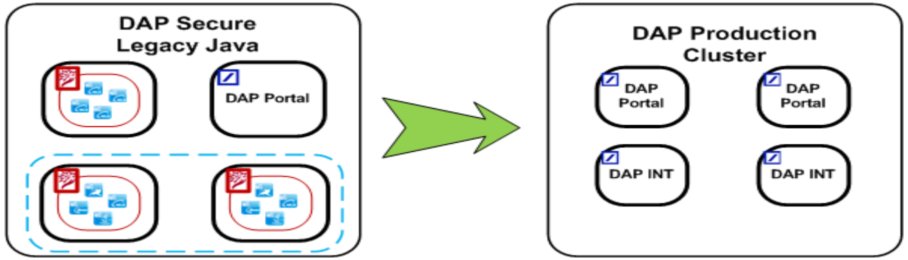
Quarantine with Waratek VM


Waratek Virtual Machine configured for Java 5
Waratek Virtual Machine configured for Java 4

Securing Legacy Java

Applications will be have their legacy JVMs upgraded and tested on the DAP Secure Legacy Java infrastructure and once all tests have successful been passed the application will be promoted to the DAP Production Cluster.

Sustain and Maintain with DAP



Securing Legacy Java

- **No code changes required to deploy legacy Java applications to new version of host Waratek VM.**
- **Application Firewall protects against a large number of attack with just a handful of rules.**
- **Guest host JVM can be patched using Virtual Patching (no downtime or release cycle)**
- **Ultimately this is a more economic, rapid and safer option than traditional upgrade approaches.**

Agenda

Virtual Patching

- **Uses the Waratek Application Firewall to patch applications for new security vulnerabilities that have been identified.**
- **Eliminates the dependency on a full release cycle to patch an application. Applications can be patched dynamically in situ (no application, container or JVM restart).**
- **Applications can be patched far more quickly giving a more rapid response to security vulnerabilities.**
- **Reduces the disruption caused to application teams from unscheduled patching events.**
- **Ultimately this a far more cost effective way to manage the patching of applications for newly identified security vulnerabilities.**

Agenda

Application Security

- **Configure the Waratek Application Firewall to prohibit Java Classes, Packages and APIs.**

- **Lock down disk, network and OS resources with ease in side the JVM.**

- **Warn on indicators of intrusion.**

- **Prevent Cross Site Scripting, SQL Injection and other common attacks.**

- **The costs of remediating identified security vulnerabilities in our applications is greatly reduced by eliminating the need to refactor code.**

Summary

- Secure coding is complex, time consuming and very difficult to get right
- Many applications have potential risks associated with imported components
- Legacy applications running on Legacy Java versions are very widespread

- Application Security controls are ideally placed inside the JVM
- Java Containers enable Run Time Application Self Protection