Using Instrumentation to Optimize Application Security

The road to Self-Protecting Software

Girish Nair, CISSP, CSSLP
Solutions Architect

May 2018

Current state of Application Security

PAIN POINTS

1. Delays
2. Inconvenience
3. Unfriendly
4. …

An early application exploit

- In 1988, Morris Worm exploited a buffer overflow in Unix to spread from machine to machine.
- 3 decades later… buffer overflows still exist.
- Is the developer responsible for the security breach?

Don’t blame the developer!

Possible ways to prevent it?

- One could argue that the technology failed him.
- Why is C/C++ susceptible to buffer overflow attacks?

The JVM and CLR protects against such exploits.

- Java & .NET developers don't create this problem. Are they better developers?

Self-Protecting Application

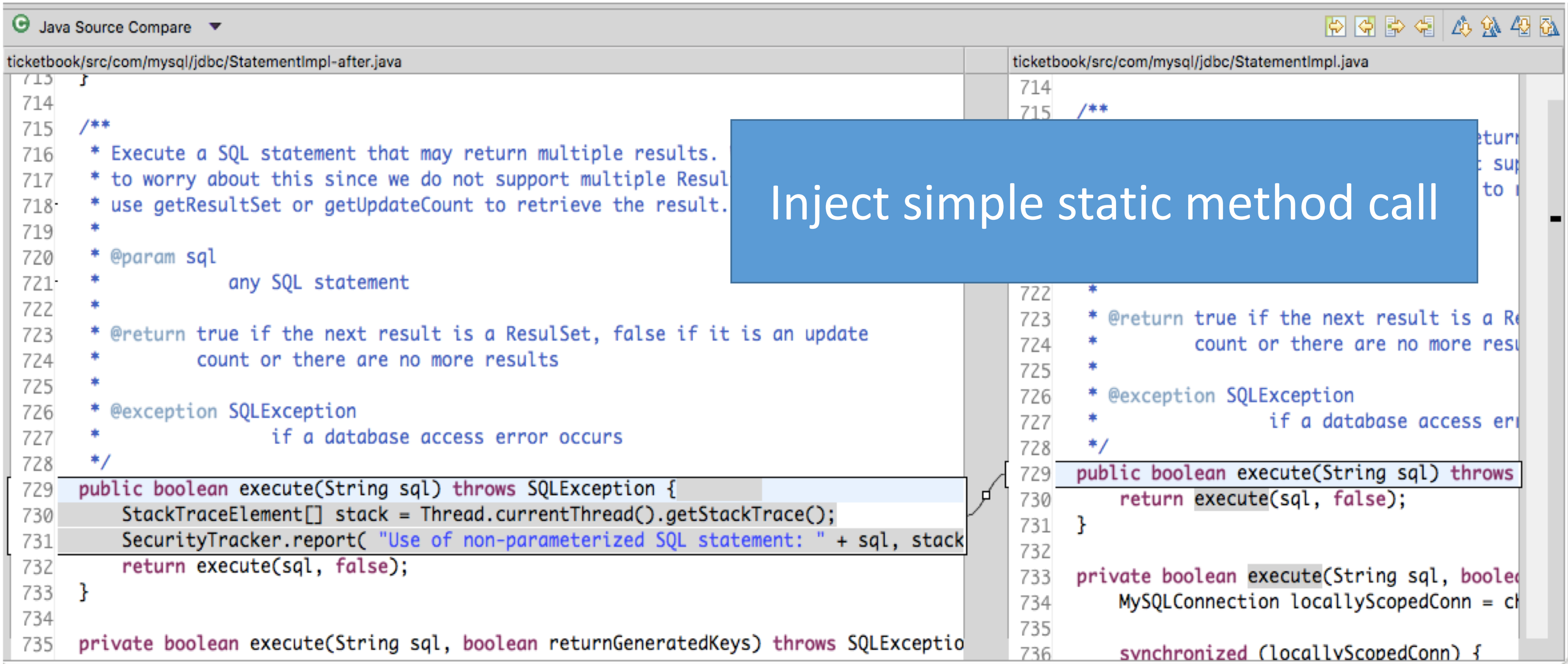
instrumentation, *n.*

Pronunciation: /ˌɪnstruːmɛnˈteɪʃən/

Etymology: < French *instrumentation* (1835 in *Dict. Acad.*), < *instrumenter* : see INSTRUMENT *v.* and -ATION *suffix*. (Show Less)

4. The use of measuring instruments to monitor and control a process. It is the art and science of measurement and control of process variables within a production, laboratory, or manufacturing area.

Source instrumentation



Inject simple static method call

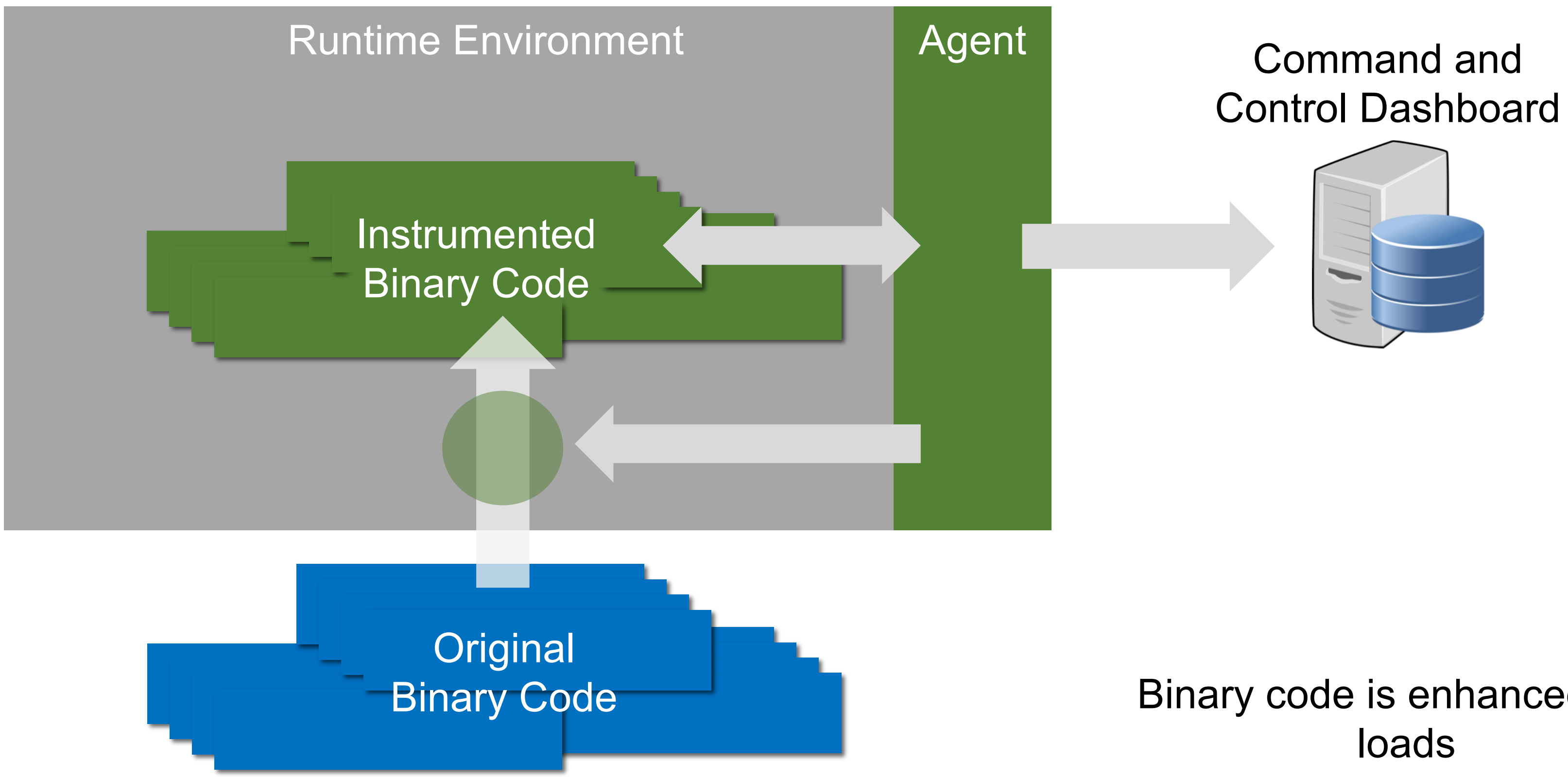
Binary Instrumentation

- Widely used
  - CPU Performance
  - Memory
  - Logging
  - Security
  - …
- Lots of libraries
  - ASM (Java)
  - BCEL (Java)
  - Javassist (Java)
  - MBEL (.NET)
  - RAIL (.NET)
  - …

Bytecode Compare: org.h2.jdbc.JdbcStatement

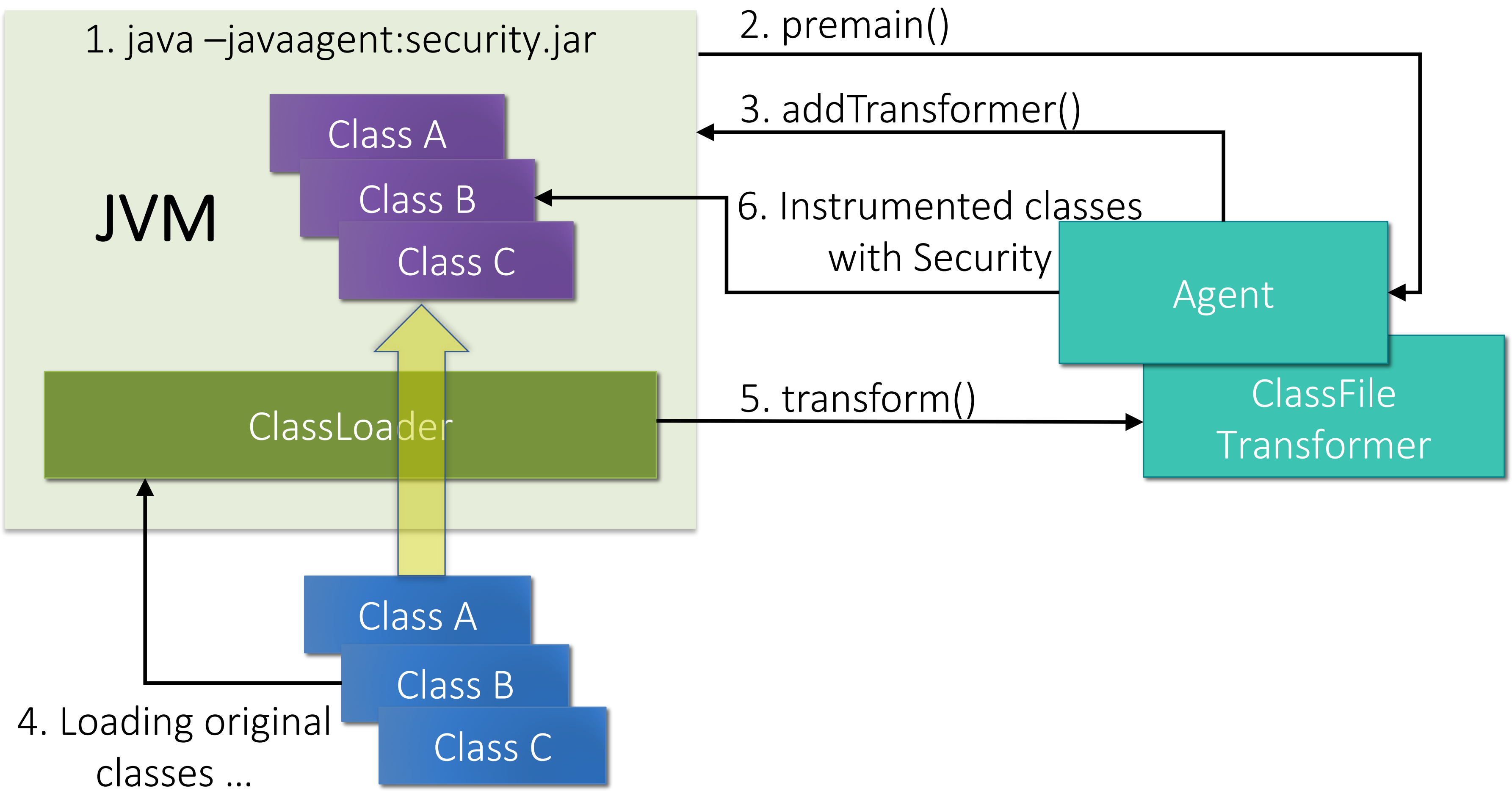
```
57      iload 5
59      putfield boolean JdbcStatement.closedByResultSet
62      return
    }

    public void addBatch(String p0) throws SQLException {
        try-block_start(java.lang.Exception)_0:
        try-block_start(java.lang.Throwable)_0:
0       getstatic NamedScopeTracker EventController.triggerScope
3       ldc String Constant "sql-injection"
5       invokevirtual void NamedScopeTracker.enterScope(String)
        try-block_start(java.lang.Exception)_8:
0       aload_0 0
1       ldc String Constant "addBatch"
9       ldc_w String Constant "addBatch"
3       aload_1 1
4       invokevirtual void JdbcStatement.debugCodeCall(String, String)
7       aload 0 0
```

Dynamic Binary Instrumentation!

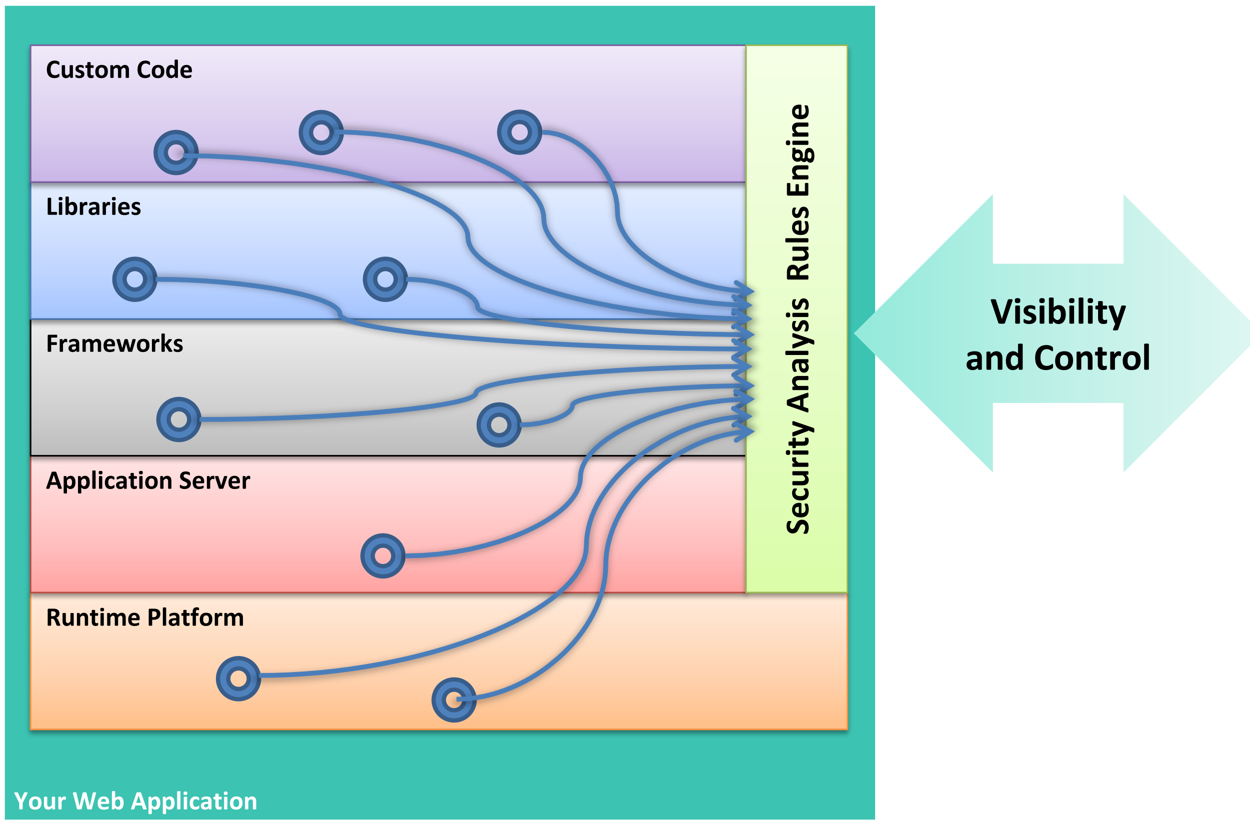


Binary code is enhanced as it loads

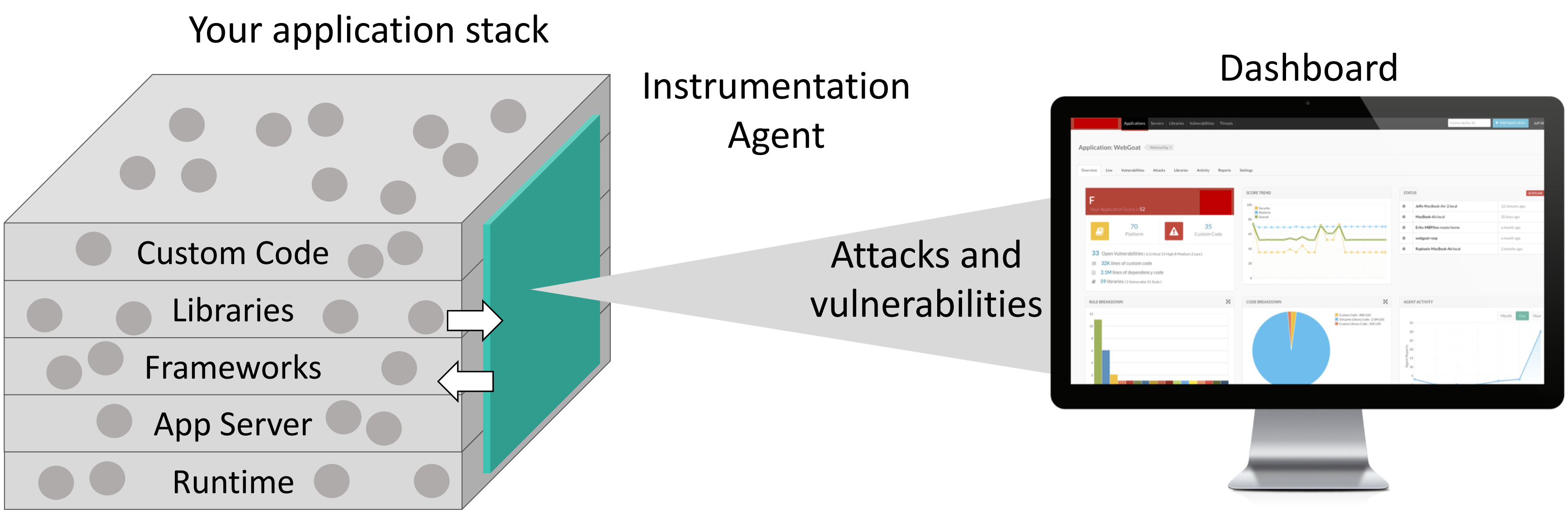
Java Instrumentation API



Sensors are Woven



Instrumentation in Action



1. Add agent
-javaagent:security.jar

2. Agent instruments
running application

3. Agent blocks attacks
and finds vulnerabilities

4. Dashboard provides
visibility and control

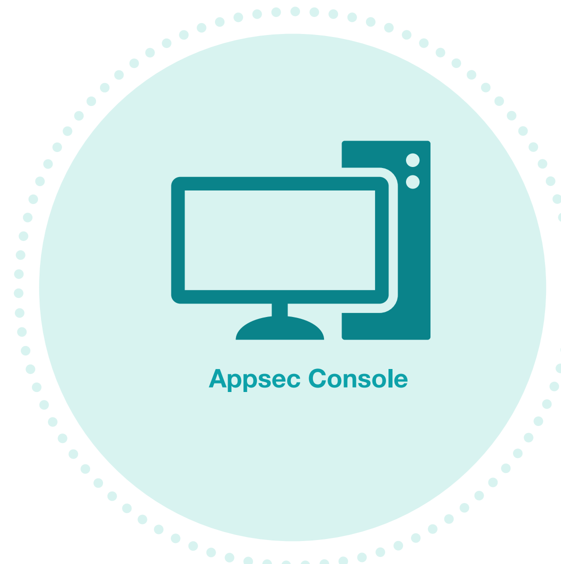
Types of Sensors

Vulnerability Sensors

- Verify Security Configuration
- Verify Library Versions
- Verify Library Vulnerabilities
- Verify Control Flow Patterns
- Verify Data Flow Patterns
- Verify Coding Patterns

Discovery Sensors

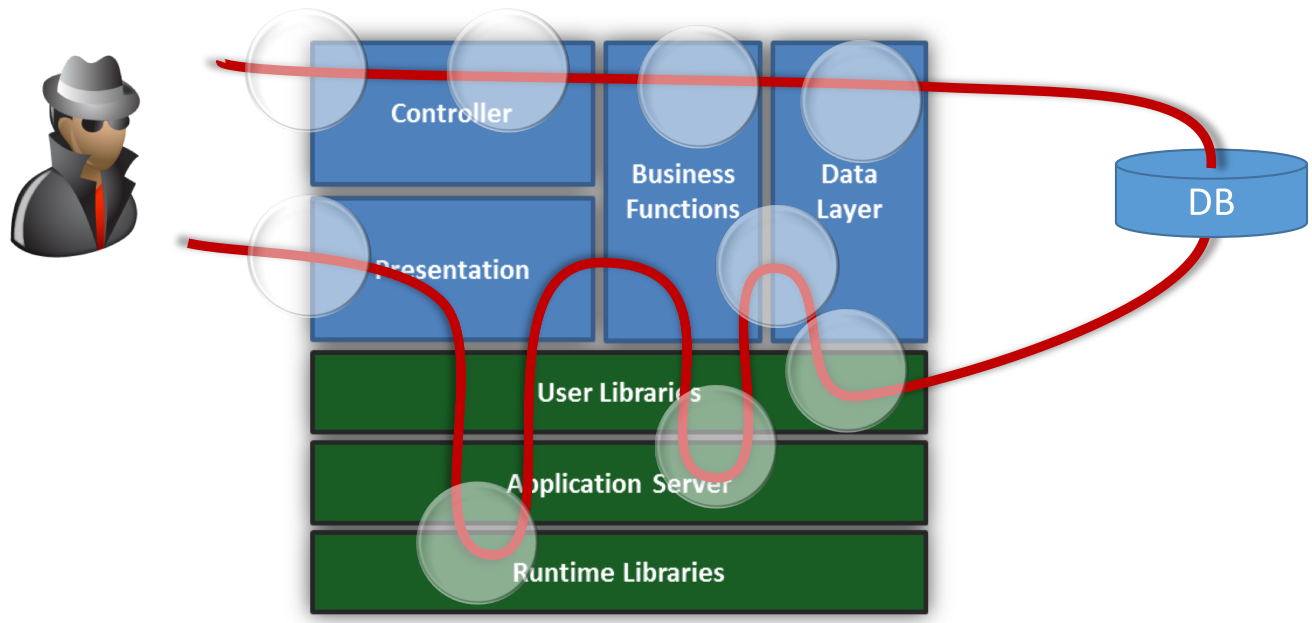
- Identify Architecture
- Identify Connections
- Identify Security Controls
- Profile Application
- Report Technologies In Use
- Measure Lines of Code



What Does a Vulnerability Look Like?

```
conn = pool.get[illegible]nection();
String sql = "[illegible]ct * from [illegible]here
    username='[illegible]sername[illegible]d
    password='"[illegible]sw[illegible]'";
stmt = conn.crea[illegible]();
rs = stmt.executeQ[illegible]);
if (rs.next()) {
    loggedIn = tr[illegible]
        out.pri[illegible]ucc[illegible]ged in");
} else {
        out.p[illegible]n("Invalid [illegible]tials");
}
```

It’s a “Path” Through Your Code!



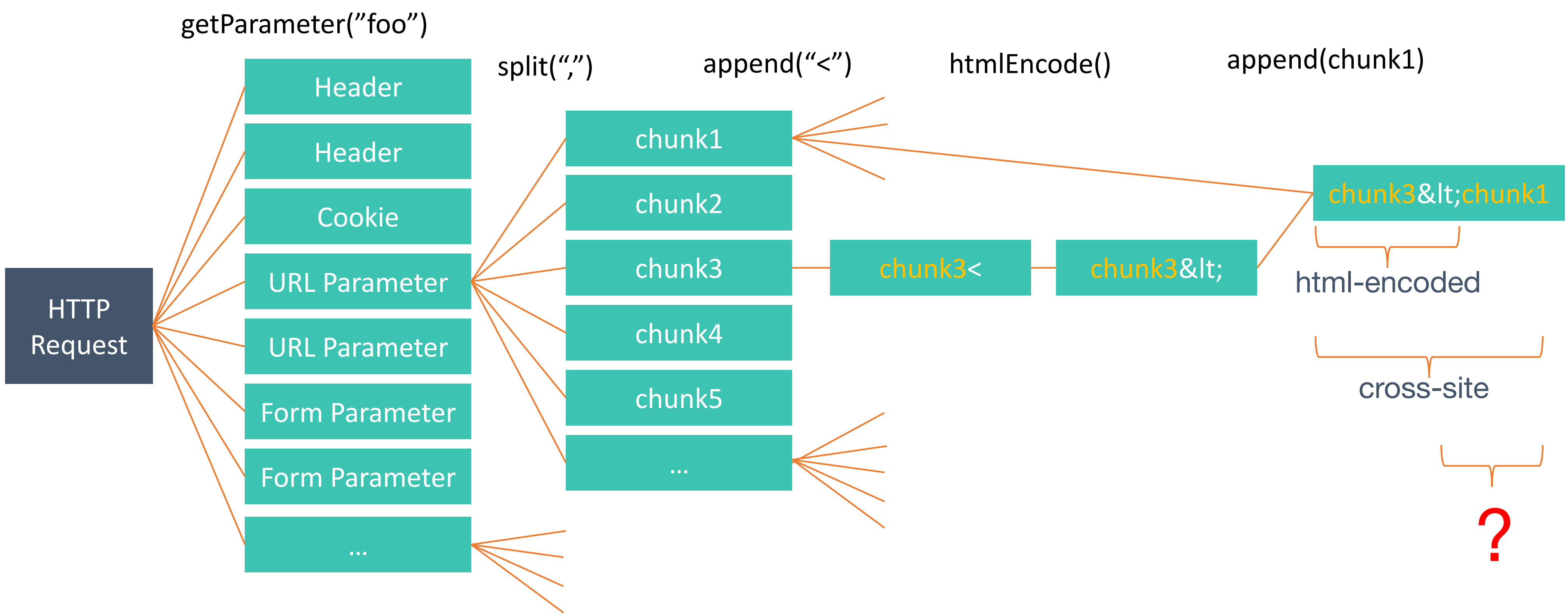
Source	`a = request.getParameter("foo")`
Data Flow	`b = a + "bar";`
Control (Validation)	`pattern.matches("[a-zA-Z0-9 ]");`
Data Flow	`c = b.replaceAll("foo","bar");`
Data Flow	`d = c.getBytes();`
Data Flow	`e = new String(d,"UTF-8");`
Control (Encoding)	`f = ESAPI.encodeForSQL(e);`
Trigger	`stmt.exec("SELECT * FROM " + f);`

Tagging

“user-input”	`a = request.getParameter(“foo”)`
	`b = a + “bar”;`
“limited-chars”	`pattern.matches(“[a-zA-Z0-9 ]”);`
	`c = b.replaceAll(“foo”,“bar”);`
“encoded”	`d = ESAPI.encodeForSQL( c );`
	`stmt.exec(“SELECT * FROM “ + d );`

Safe!

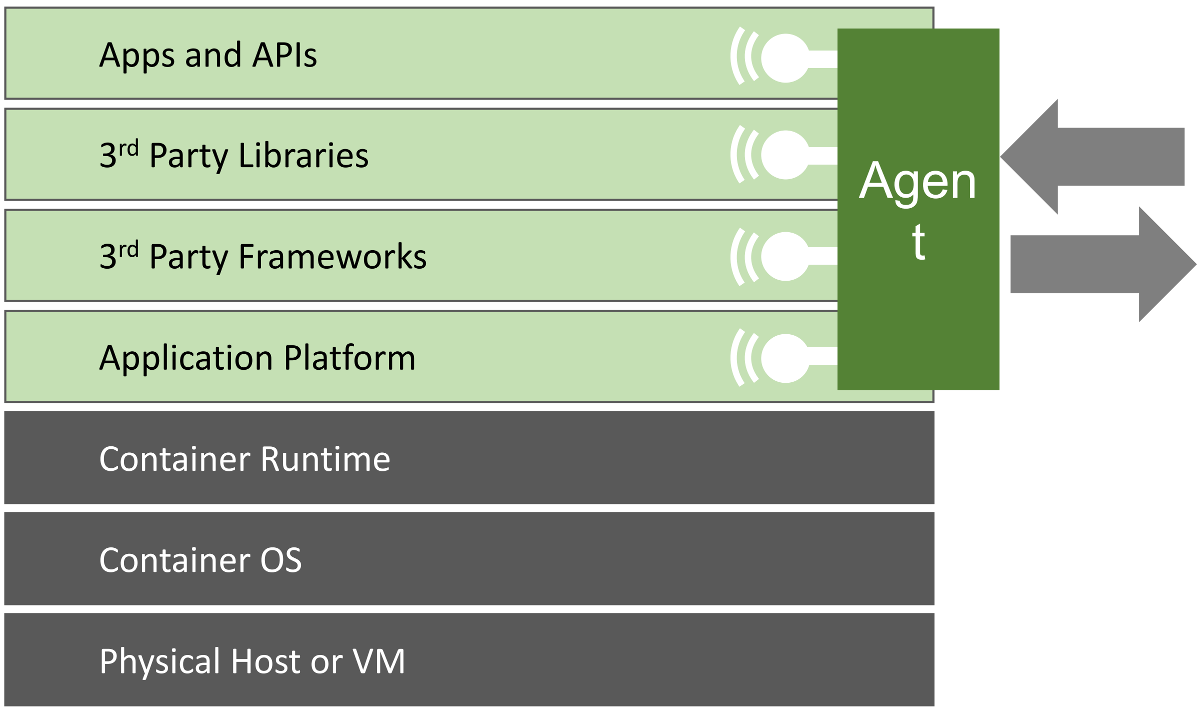
Data flow analysis (aka clusterbomb)



Fine-Grained Tracking

foo	`a = request.getParameter(“foo”)`
foobarfoo	`b = a + “bar” + a;`
	`pattern.matches(“[a-zA-Z0-9 ]”);`
foarfoo	`c = b.replaceAll( “ob”, “” );`
foarfoo	`d = ESAPI.encodeForSQL( c );`
SELECT * FROM ‘foarfoo’	`stmt.exec(“SELECT * FROM “ + d );`

Agent capabilities



Your standard application stack(s)

Examples…

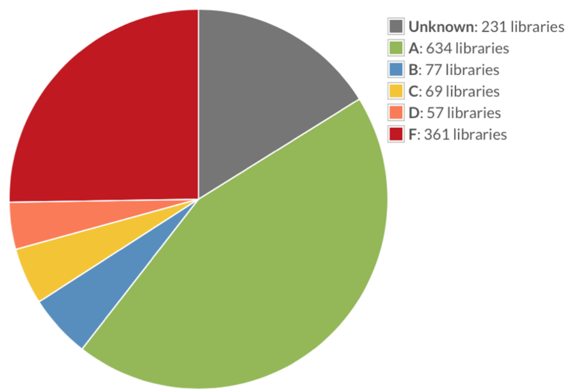
- Analyze configuration files
- Analyze loaded libraries
- Analyze HTTP request
- Analyze HTTP response
- Analyze Backend connections
- Report hardcode credentials
- Report on weak ciphers
- Report injection flaws
- Report vulnerable libraries
- Zero touch logging
- Deploy virtual patches
- Block attacks

Library analysis – 3rd party / CVEs

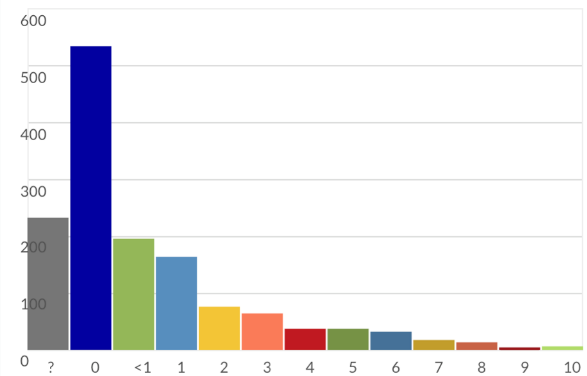
OVERVIEW

- **68 known vulnerabilities** (across 28 libraries)
- **133.71M lines of library code**
- **1429 libraries** (28 Vulnerable 382 Stale)

AVERAGE KNOWN LIBRARY GRADE: F



AVERAGE YEARS OUT OF DATE: 1



commons-dbcp-1.3.jar	A	OpenCms	0	1.3 (02/07/2010)	1.4 (02/07/2010)	19/62
commons-digester-1.8.jar	D	OpenCms	0	1.8 (12/03/2006)	2.1 (09/24/2010)	20/100
commons-email-1.2.jar	F	OpenCms	0	1.2 (11/01/2009)	1.4 (05/18/2015)	2/10
commons-fileupload-1.2.2.jar	F	OpenCms	1	1.2.2 (07/29/2010)	1.3.2 (06/18/2016)	36/44
commons-httpclient-3.1.jar	F	OpenCms	4	3.1 (08/21/2007)	3.1 (08/21/2007)	2/167
commons-io-1.4.jar	F	OpenCms	0	1.4 (01/20/2008)	2.5 (04/30/2016)	0/76
commons-lang-2.4.jar	C	OpenCms	0	2.4 (03/19/2008)	2.6 (01/16/2011)	2/127

Data Flow analysis - SQL injection

SQL Injection from "uname" Parameter, "pass" Parameter on "login.jsp" page

CRITICAL **First Detected:** 12/28/2016 10:52 AM | **Status:** Reported | **ID:** UOVB-LWLN-NYVQ-095Q

Mark as

Overview | How to Fix | HTTP Info | **Details** | Notes | Discussion 0

18 of 19

DANGEROUS DATA RECEIVED	**str = facade.getParameter("uname")** at service() @ login.jsp:3	girish
DATA FLOWED FROM PARAMETER TO OBJECT	**sb = sb.append("girish")** at service() @ HttpJspBase.java:70	select * from members where uname='girish
DANGEROUS DATA RECEIVED	**str = facade.getParameter("pass")** at service() @ login.jsp:4	girish
DATA FLOWED FROM PARAMETER TO OBJECT	**sb = sb.append("girish")** at service() @ HttpJspBase.java:70	select * from members where uname='girish' and pass='girish
DATA FLOWED FROM OBJECT TO RETURN VALUE	**str = sb.toString()** at service() @ HttpJspBase.java:70	select * from members where uname='girish' and pass='girish'
RULE VIOLATION DETECTED	**rs = impl.executeQuery("select * from members whe...girish' and pass='gi rish'")** at service() @ HttpJspBase.java:70	select * from members where uname='girish' and pass='girish'

HTTP response analysis – Clickjacking

Overview | How to Fix | Notes | Discussion 0

3 of 14

Application: DNN
Status: REPORTED
Environments: Development

FEB 17 2017 First Detected

26+ days Window of Exposure

26 days Since Last Detected

We observed 1 page without sufficient anti-clickjacking controls:

/DNN/Install/InstallWizard.aspx

The application doesn't apply anti-framing controls to the given page. Not applying these controls will allow it to be framed by other websites. This can present a few security issues, but the chief concern is that framed pages may be used in Clickjacking attacks.

Clickjacking (also called UI Redress) is an attack whereby an attacker can trick a user into clicking on something different from what it appears they are clicking on. Most attacks will first build a page that entices the user to click on it - for scandalous pictures, monetary rewards, and other promises. The buttons and links on this page are carefully placed so that they overlap with buttons, links or other items on a second page that gets layered on top of the first one.

The problem for the victim in this scenario is that the link that they'd like to click for the promised rewards on the evil page is actually "beneath" a translucent, second page which has been framed into the HTML document. Thus, when they intend to click on **"Collect Your Prize!"**, they're really clicking on **Transfer Your Money!**, or similar.

Numerous real-life Clickjacking attacks have caused damage to Adobe Flash Player, and numerous websites including Facebook, Twitter, YouTube, and others.

There are generally two controls that prevent framing of a web page: the `X-Frame-Options` header, and so-called `frame-busting JavaScript`. Details about these controls can be found in the **Remediation** tab. Neither of these anti-framing controls were found in the HTTP response to the following URL(s):

Configuration analysis – Authentication mode

SSL Not Required For Forms Authentication in \web.config

MEDIUM | **First Detected:** 02/17/2017 10:46 AM | **Status:** Reported | **ID:** XEMH-QJXR-UHJG-4U3G

Mark as

Overview | How to Fix | Notes | Discussion 0

2 of 2

Application	DNN
Status	REPORTED
Environments	Development

FEB 17 2017 First Detected

26+ days Window of Exposure

1 day Since Last Detected

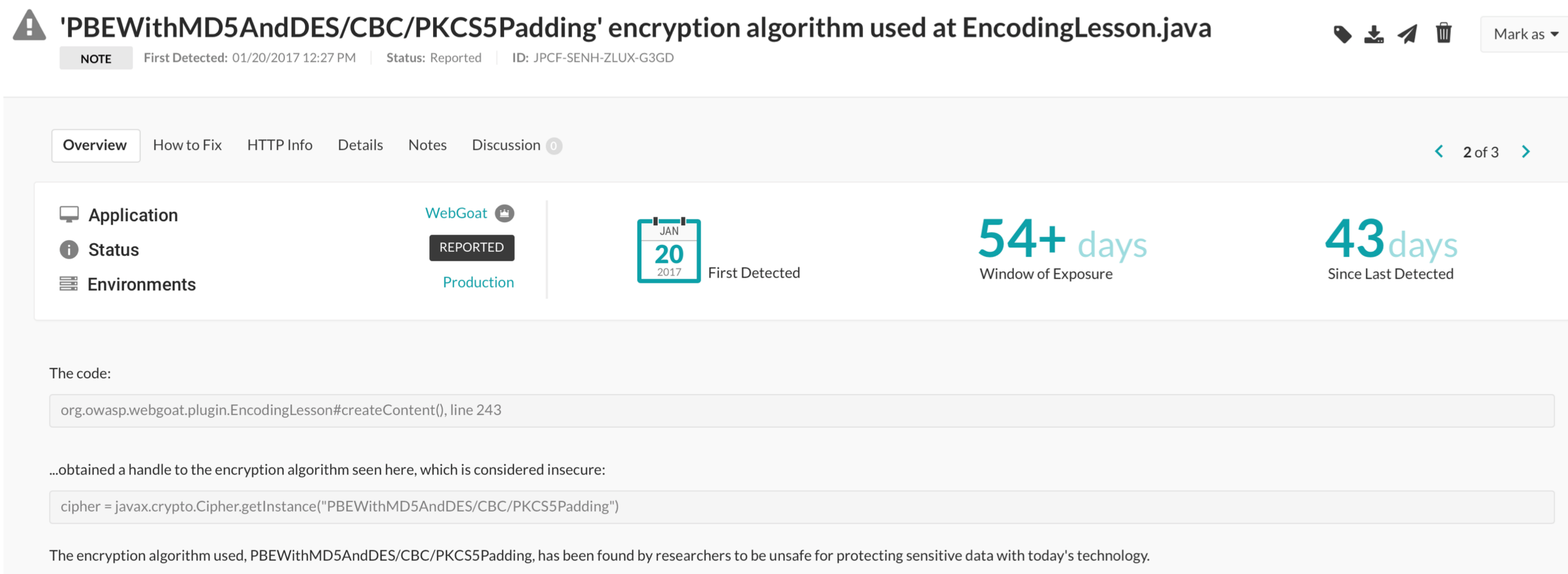
The configuration in \web.config was configured to use forms authentication and `requireSSL` was not set to `true` in the following authentication section:

```
<!-- Forms or Windows authentication -->
<authentication mode="Forms">
  <forms name=".DOTNETNUKE" protection="All" timeout="60" cookieless="UseCookies"/>
</authentication>
<!--
```

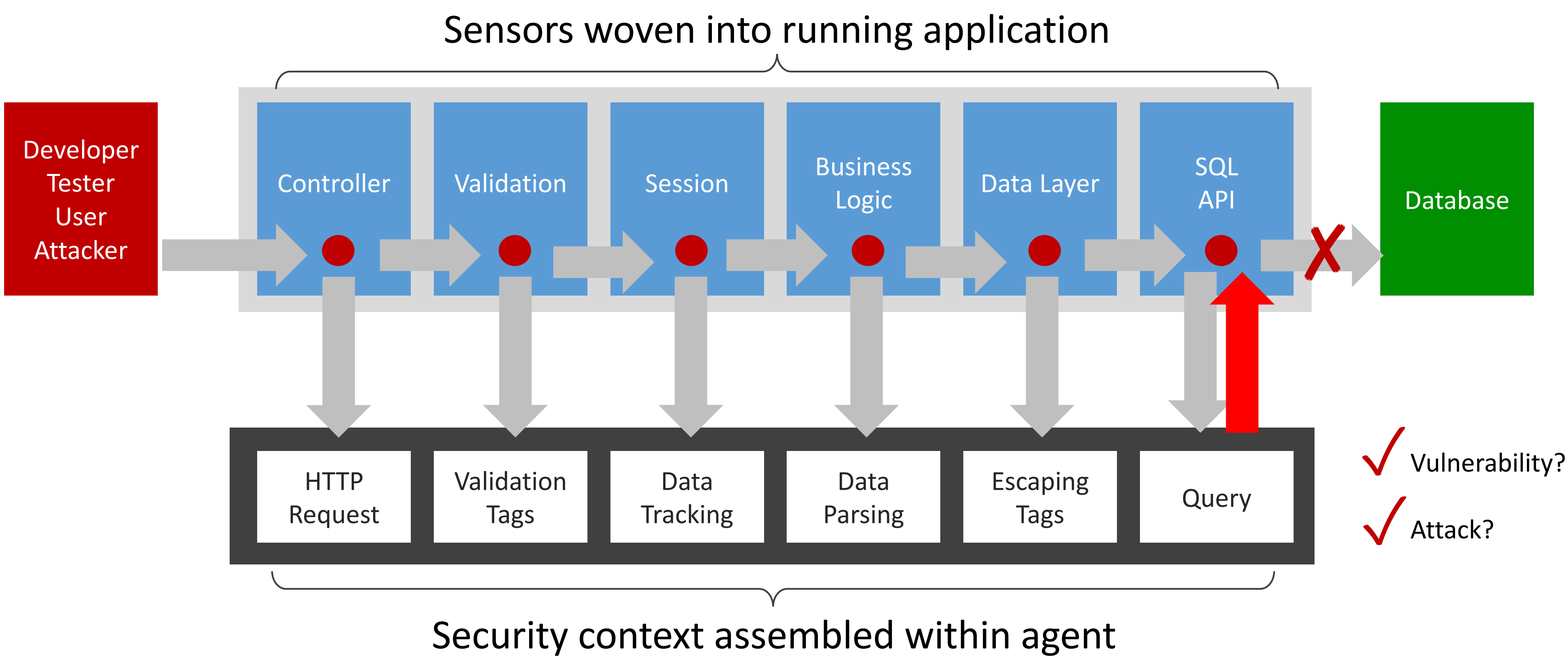
When `requireSSL` is `true`, an SSL connection is required for forms authentication and the forms authentication cookie will have the 'secure' flag which prevents browsers from sending the cookie across unencrypted connections.

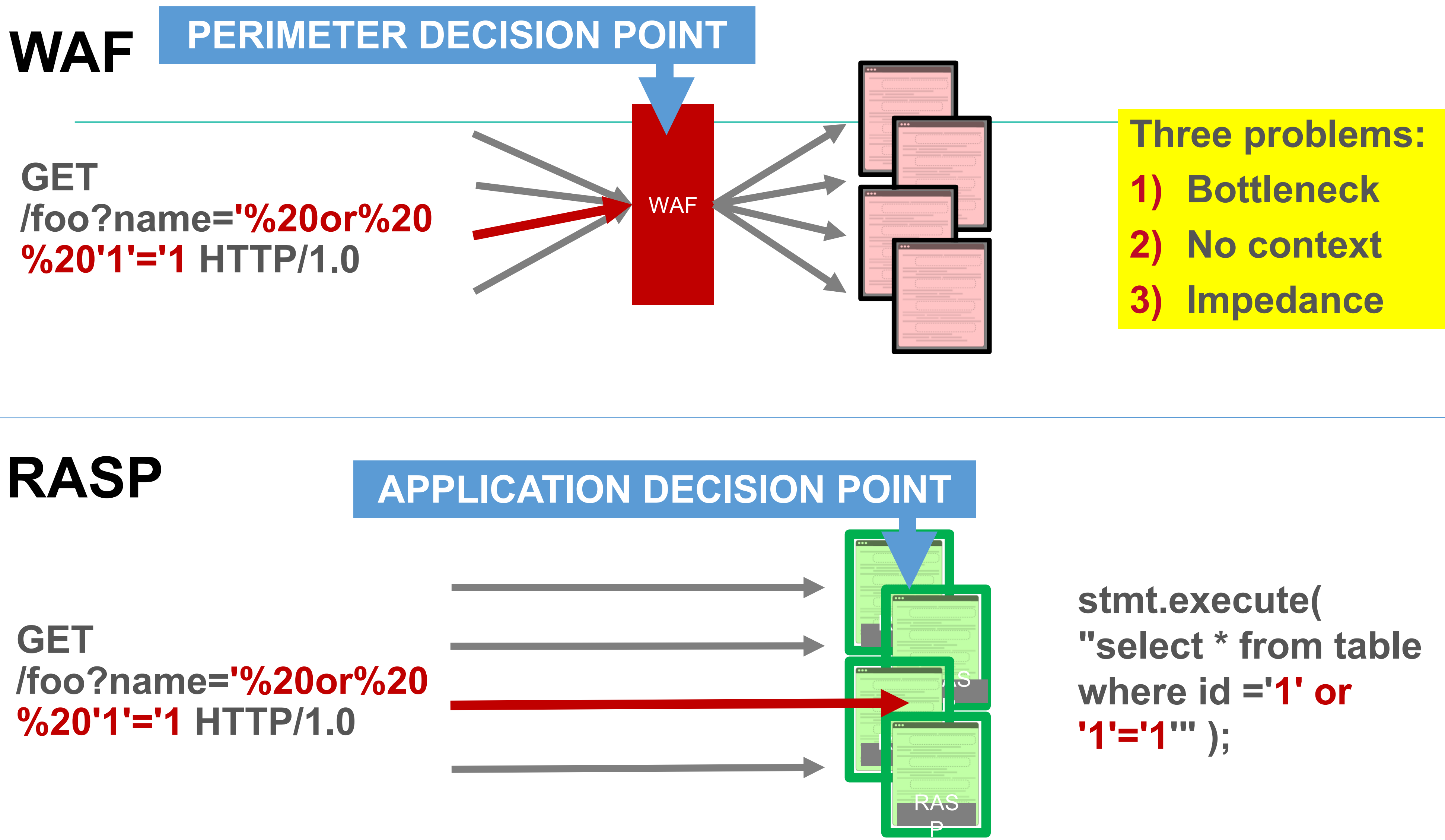
Neither of these protections are used when `requireSSL` is `false`. An attacker could eavesdrop on forms authentication requests sent over HTTP and learn users' credentials as well as the users' forms authentication cookies, leading to the compromise of user accounts.

Execution analysis – Cipher initialization



Blocking attacks




WAF
PERIMETER DECISION POINT
GET /foo?name='%20or%20 %20'1'='1 HTTP/1.0
WAF
Three problems:
1) Bottleneck
2) No context
3) Impedance
RASP
APPLICATION DECISION POINT
GET /foo?name='%20or%20 %20'1'='1 HTTP/1.0
RASP
stmt.execute("select * from table where id ='1' or '1'='1'");

RASP performance – same as code

WebGoat	RASP Processing
Typical traffic	50 microseconds
Mixed traffic	170 microseconds
Heavy attack traffic	230 microseconds

millionths of a second

- Number of applications doesn't matter
- No bottleneck on either bandwidth or CPU

Accuracy, Automation and Scalability

You can't scale appsec without highly accurate tools
(both true positives and true negatives)

Because inaccuracies require experts…

…and experts don't scale.

Concluding Remarks

Instrumentation enables …

- Application security in parallel (background),
- continuously across entire portfolio,
- without scans and bottlenecks,
- on modern software architecture
- at breakneck speed of development.

Thank you!

Girish Nair
girish@contrastsecurity.com

http://contrastsecurity.com