Turning Security into Code with Dynamic Binary Instrumentation

Jeff Williams, Co-founder and CTO
Contrast Security
@planetlevel
We are failing...
You can’t scale appsec without **highly accurate** tools
(both true positives and true negatives)

Because inaccuracies **require** experts...

...and experts don’t scale.
By turning security into code
---> we can get speed, coverage, and accuracy
---> which allows us to scale

Level 3: Management makes informed decisions with detailed security analytics

Level 2: Security experts deliver security as code

Level 1: Development and operations get fully automated security support

Continuous Application Security
How do we turn “security into code”?

**Defend**
Do we have a defense strategy and implementation?

**Assess**
Do we automatically verify defense is present, correct, and used properly everywhere?

**Protect**
Do we automatically detect and block anyone attempting to attack this?
A better way to think about the problem...

How do I get the Information I need to assure what I care about?
Problem: Clickjacking

Defend
Use X-FRAME-OPTIONS header to prevent frames

Assess
Check HTTP responses to ensure they all have X-FRAME-OPTIONS set.

Protect
Tough – looks like expected traffic.

* Image: Igor Abade
Problem: Bypassing Verb-Based Auth’n and Auth’z (VBAAC)

<security-constraint>
<web-resource-collection>
   <url-pattern>/admin/*</url-pattern>
   <http-method>GET</http-method>
   <http-method>POST</http-method>
</web-resource-collection>
<auth-constraint>
   <role-name>admin</role-name>
</auth-constraint>
</security-constraint>
Problem: Insecure Libraries

Defend
Patch and upgrade quickly

Assess
Continuously assess libraries that are actually used for known vulnerabilities.

Protect
Deploy virtual patches that prevent vulnerability from being exploited.
Problem: Weak Crypto Algorithm

‘MD5’ is everywhere
Summary so far...

• Clickjacking -> need HTTP headers
• Bypassable VBAAC -> need web configuration, HTTP to block
• Insecure Libraries -> need libraries, frameworks, servers, platform
• Weak Encryption -> need code, configuration, exceptions
• …
Great – so I have to run 50 tools? No.
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
```java
57  iload 5
59  putfield boolean JdbcStatement.closedByResultSet
62  return
```
```java
public void addBatch(String p0) throws SQLException {
    try-block_start(java.lang.Exception)_0:
    try-block_start(java.lang.Throwable)_0:
    getstatic NamedScopeTracker EventController.triggerScope
    ldc String Constant "sql-injection"
    invokevirtual void NamedScopeTracker.enterScope(String)
    try-block_start(java.lang.Exception)_8:
    aload_0 0
    ldc String Constant "addBatch"
    ldc_w String Constant "addBatch"
    aload_1 1
    invokevirtual void JdbcStatement.debugCodeCall(String, String)
    aload 0 0
```
Dynamic Binary Instrumentation!

- Runtime Environment
- Instrumented Binary Code
- Agent
- Command and Control Dashboard

Binary code is enhanced as it loads:
- Fast
- Safe
- Proven
Problem: Injection (SQL, XSS, etc...)

• Attacker sends data that is passed to an interpreter (SQL, LDAP, EL, ...)

Defend
Use escaping, parameterization correctly everywhere. Right.

Assess
Use automated data flow analysis to track untrusted data to any queries.

Protect
Analyze whether untrusted data flows to a query and modifies its meaning.
Data flow analysis (aka clusterbomb)

HTTP Request
- Header
- Cookie
- URL Parameter
- Form Parameter
- ...
Solution: Instrumentation

Sensors infused into running application

Security context

HTTP Request  Validation Tags  Data Tracking  Data Parsing  Escaping Tags  Query

✓ Assess  ✓ Protect

Database

Controller  Validation  Session  Business Logic  Data Layer  SQL API

Developer
Tester
User
Attacker
Cross-Site Request Forgery

1. Attacker sets the trap on some website on the internet (or simply via an e-mail)

   Hidden `<img>` tag contains attack against vulnerable site

2. While logged into vulnerable site, victim views attacker site

   `<img>` tag loaded by browser – sends GET request (including credentials) to vulnerable site

3. Vulnerable site sees legitimate request from victim and performs the action requested

**Defend**
Add a token to links and forms. Verify token is present on transactions.

**Assess**
Verify non-XHR requests have token on non-idempotent transactions.

**Protect**
Application should detect and block use of unauthorized verbs.
Solution: Instrumentation

- **Sensors infused into running application**

**Vulnerability**
- Is not an XHR request?
- Token check fails
- Non-idempotent transaction

**Attack**
- Add CSRF token to webpages
- Check for tokens on susceptible pages
Three problems:
1) Bottleneck
2) No context
3) Impedance

GET /foo?name='%20or%20%20'1'='1 HTTP/1.0

stmt.execute( "select * from table where id ='1' or '1'='1'" );
OWASP Benchmark – thousands of test cases across a range of true and false vulnerabilities

Free, open, reproducible

Instrumentation speed and accuracy dominates SAST and DAST
Distributed AppSec – In Parallel

Centralized Visibility and Control

Continuous Assessment and Protection
Making DevSecOps Actually Work

DEV

SEC

OPS

Vulnerability Alerts & Guidance

Centralized Visibility & Control

Attack Protection & Threat Intelligence
Instrumentation Powers Continuous AppSec

Analytics across entire portfolio drive budget and priorities

A platform for modeling policy to enforce across portfolio

Instant accurate notification of problems via existing tools

Continuous Application Security
THANK YOU

Jeff Williams  |  jeff.williams@contrastsecurity.com