THE FUTURE OF APPSEC AUTOMATION

WHY YOUR APPSEC EXPERTS ARE KILLING YOU
ARE YOU “SECURE”?

Threats

Expected Security Model (claims)

- Minimal data collection
- Strong encryption in storage and transit
  - All external connections use SSL
  - All internal connections use SSL
  - SSL hardened according to OWASP
  - All highly sensitive data encrypted
  - Encryption uses standard control
  - Encryption uses AES; no CBC or ECB

- Universal authentication
- Pervasive access control
- Injection defenses
  - Strict positive validation of all input
  - Use of parameterized interfaces
  - All paths hardened
  - XML parsers set to not use DOCTYPE
  - Browser set no content sniffing header
  - Etc...
  - Use Hibernate and secure coding
  - Use jQuery and secure coding
  - Etc...

Verified Defenses (evidence)
Claims.

Not evidence.

But still very cool.
36% of security spending is allocated to Application Security.

- Skimmers: 14%
- Insiders: 9%
- Crimeware: 8%
- DOS: 4%
- Espionage: 1%
- Point of Sale: 0.7%

*2014 Verizon DBIR*
The frequency and sophistication of attacks is increasing...  ...as the complexity, connectivity, and criticality of our code is increasing...
“Application security is eating security” – Alex Stamos (Yahoo CISO)
How Are We Doing?

1. Assurance? 22.4
2. Coverage? 10%
3. Process Fit? Mad

*2014 Aspect Security Global Application Security Risk Report*
<table>
<thead>
<tr>
<th>Software</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space Shuttle</td>
<td>400,000</td>
</tr>
<tr>
<td>Spring Framework</td>
<td>1,165,000</td>
</tr>
<tr>
<td>Windows 3.1</td>
<td>2,500,000</td>
</tr>
<tr>
<td>US Tax Code</td>
<td>4,400,000</td>
</tr>
<tr>
<td>Windows NT 3.5</td>
<td>4,500,000</td>
</tr>
<tr>
<td>Firefox</td>
<td>9,500,000</td>
</tr>
<tr>
<td>Windows NT 4.0</td>
<td>12,000,000</td>
</tr>
<tr>
<td>Microsoft Office 2013</td>
<td>44,500,000</td>
</tr>
<tr>
<td>Mac OSX</td>
<td>85,000,000</td>
</tr>
<tr>
<td>Car Software</td>
<td>100,000,000</td>
</tr>
<tr>
<td>Mouse DNA</td>
<td>120,000,000</td>
</tr>
<tr>
<td>Financial Organization</td>
<td>1,000,000,000</td>
</tr>
</tbody>
</table>

The Insane Complexity of Modern Code

- 74,000 Printed Pages
- 16,000,000 Printed Pages
Development organizations interpret DELAYS as DAMAGE and route around them.
CONTEXT IS KING!
A Vulnerability Is a Pattern of Events
Rootkits Aren’t Always Evil!

• “Enterprise Java Rootkits” - BlackHat 2009

• The holy grail of backdoors
• The Java Instrumentation API

Instrumentation

1. `java -javaagent=agent.jar`

2. `premain()` 3. `addTransformer()` 4. `loadClass()` 5. `transform()`

Add sensors.... bytecode is instrumented for security!
DEEP SECURITY INSTRUMENTATION HAS UNFAIR ADVANTAGES…

Deep Security Instrumentation

- Code
- Libraries
- Runtime Data Flow
- Software Architecture
- HTTP Traffic
- Frameworks
- Runtime Control Flow
- Server Configuration
- Backend Connections
- Configuration Data
- Platform Runtime
- Etc…

SAST
- Code
- HTTP Traffic

DAST
- HTTP Traffic

WAF
- HTTP Traffic
Cross-Site Scripting

```html
<p>Cross-Site Scripting</p>
```
Continuous AppSec
“thousands of reports a minute”

http://engineeringblog.yelp.com/2014/09/csp_reports_at_scale.html
“Surface security info for everyone, not just the security team”
APPLICATION SECURITY – TWO SEPARATE WORLDS

Development

Tools to detect vulnerabilities

Operations

Tools to stop attacks
TOMORROW – UNIFIED APPSEC

Development **AND** Operations

UNIFIED APPSEC

RASP

IAST

A single appsec technology across the entire lifecycle
INFUSE SECURITY TECHNOLOGIES INTO APPLICATIONS

- SAST
- DAST
- Agent
- WAF
- IDS/IPS
- Threat

Deep Security Instrumentation

Application
EXAMPLE: CODE VULNERABILITIES (SQL INJECTION)

- Concept
- Architecture
- Design
- Development
- Unit Testing
- Integration
- QA Testing
- Operation
- Monitoring

- Generate accurate security architecture diagrams
- Provide instant vulnerability feedback
- Add security testing to existing test efforts
- Identify and block attacks and exploits
EXAMPLE: THIRD PARTY LIBRARIES (SPRING EL INJECTION)

- **Employee Apps**
  - Automatically collect library inventory

- **Third Party Apps**
  - Notify projects of library vulnerabilities

- **Public Apps**
  - Ensure that developers use libraries safely

- **Cloud Apps**
  - Shield applications from attacks on known vulnerabilities

- **“Rogue” Apps**
“If you have code that’s important enough to deploy, it’s important enough to instrument”

The world’s fastest application security.