Agenda

- Background – Metrics, Distribution of Applications
- Security of Applications
- Third Party Risk
- Summary
Background – Basis for insights

- For over three years, Veracode has been providing automated security analysis of software to large and small enterprises across various industry segments.

- One of the residual effects is the wealth of security metrics derived from the anonymized data across varied industries and types of applications.

- These metrics offer valuable insights on the quality of application security and issues related to the current state-of-practice and maturity of security in software.

- Veracode was founded in 2006 by application security experts from @stake, Guardent, Symantec, and VeriSign.

- Veracode provides automated security assessment capabilities in the cloud. Automated techniques include static binary analysis and dynamic analysis. Manual test data (if performed) is included in the analysis.
The Data Set + Metrics

- **Enterprise**
  - Industry vertical (enumerated)

- **Application**
  - Application Supplier Type (internal, purchased, outsourced, open source)
  - Application Type (Web facing / Non-web)
  - Assurance Level (1 to 5)
  - Language (enumerated)
  - Platform (enumerated)

- **Scan**
  - Scan Number
  - Scan Date
  - Lines of Code

- **Metrics**
  - Flaw Count
  - FlawPercent
  - ApplicationCount
  - First Scan Acceptance Rate
  - Veracode Risk Adjusted Score
  - MeanTimeBetweenScans
  - Days to Remediation
  - Scans to Remediation
  - PCI pass/fail
  - SANS Top25 pass/fail
  - OWASP pass/fail
  - Two flavors: ’04 and ’07

2922 Applications and billions of lines of code
SOSS Volume 2 Data Distribution

Applications by Supplier
- Internally Developed: 71%
- Commercial: 22%
- Open Source: 6%
- Outsourced: 1%

Applications by Language Family
- Java: 50%
- .NET: 29%
- C/C++: 19%
- ColdFusion: <1%
- PHP: 1%

Web versus Non-Web Applications
- Web Applications: 56%
- Non-Web Applications: 44%
Business Criticality (and Application Source)

Figure 2: Application Business Criticality by Supplier
(* small sample size)
Security of Applications

SETEC
ASTRONOMY
76% of the code components of applications that were labeled as internally developed were third-party components (e.g. open source libraries, commercial third-party libraries etc.)
Application Security – Scanning Results (first submission)

The majority of software (provided by customers for scanning)

_______ Secure (Pass)

_______ Insecure (Fail)
More than Half of Software Failed

Supplier Performance on First Submission
(Adjusted for Business Criticality)

- **Overall**: 43% Acceptable, 57% Not Acceptable
- **Outsourced**: 7% Acceptable, 93% Not Acceptable
- **Open Source**: 42% Acceptable, 58% Not Acceptable
- **Internally Developed**: 46% Acceptable, 54% Not Acceptable
- **Commercial**: 35% Acceptable, 65% Not Acceptable

*Figure 3: Supplier Performance on First Submission (Adjusted for Business Criticality)*
Majority compliant with OWASP Top 10?
8 out of 10 Web Apps Do Not Comply with OWASP Top 10

Figure 5: OWASP Top 10 Compliance by Supplier on First Submission

- Open Source: 40% Acceptable, 60% Not Acceptable
- Internally Developed: 12% Acceptable, 88% Not Acceptable
- Commercial: 7% Acceptable, 93% Not Acceptable
Most Prevalent Vulnerability?

Flaw Percent = Flaw Count / Total

• SQL Injection
• Cross-Site Scripting (XSS)
• Cryptographic Issues
• CRLF Injection
• Buffer Overflow
Cross-site Scripting Remains the Most Prevalent

Figure 13: Top Vulnerability Categories (Overall Prevalence)
Which Language Led in Exposure to XSS?

- Java
- .NET

What is the leading issue regarding C/C++?

- Crypto Issues
- Error Handling
- Buffer Overflow
Cross-site Scripting Remains the Most Prevalent

<table>
<thead>
<tr>
<th>Vulnerability Distribution by Language</th>
<th>Java</th>
<th>C/C++</th>
<th>.NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-site Scripting (XSS)</td>
<td>46%</td>
<td>32%</td>
<td>66%</td>
</tr>
<tr>
<td>CRLF Injection</td>
<td>17%</td>
<td>21%</td>
<td>Cryptographic Issues 13%</td>
</tr>
<tr>
<td>Information Leakage</td>
<td>16%</td>
<td>18%</td>
<td>Directory Traversal 8%</td>
</tr>
<tr>
<td>Cryptographic Issues</td>
<td>7%</td>
<td>13%</td>
<td>CRLF Injection 4%</td>
</tr>
<tr>
<td>Directory Traversal</td>
<td>4%</td>
<td>7%</td>
<td>Information Leakage 4%</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>3%</td>
<td>3%</td>
<td>Insufficient Input Validation 2%</td>
</tr>
<tr>
<td>Time and State</td>
<td>2%</td>
<td>2%</td>
<td>SQL Injection 1%</td>
</tr>
<tr>
<td>Untrusted Search Path</td>
<td>2%</td>
<td>1%</td>
<td>Credentials Mgmt 1%</td>
</tr>
<tr>
<td>Credentials Mgmt</td>
<td>1%</td>
<td>&lt;1%</td>
<td>Potential Backdoor &lt;1%</td>
</tr>
<tr>
<td>Encapsulation</td>
<td>1%</td>
<td>&lt;1%</td>
<td>Time and State &lt;1%</td>
</tr>
<tr>
<td>API Abuse</td>
<td>1%</td>
<td>&lt;1%</td>
<td>Error Handling &lt;1%</td>
</tr>
<tr>
<td>Insufficient Input Validation</td>
<td>&lt;1%</td>
<td>&lt;1%</td>
<td>OS Command Injection &lt;1%</td>
</tr>
<tr>
<td>Race Conditions</td>
<td>&lt;1%</td>
<td>&lt;1%</td>
<td>Buffer Overflow &lt;1%</td>
</tr>
<tr>
<td>OS Command Injection</td>
<td>&lt;1%</td>
<td>&lt;1%</td>
<td>Untrusted Search Path &lt;1%</td>
</tr>
<tr>
<td>Dangerous Functions</td>
<td>&lt;1%</td>
<td>&lt;1%</td>
<td>Dangerous Functions &lt;1%</td>
</tr>
</tbody>
</table>

Table 4: Vulnerability Distribution by Language
No single method of application security testing is adequate by itself

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
<th>Manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-site Scripting (XSS)</td>
<td>Information Leakage</td>
<td>Cross-site Scripting (XSS)</td>
</tr>
<tr>
<td>52%</td>
<td>44%</td>
<td>26%</td>
</tr>
<tr>
<td>CRLF Injection</td>
<td>SQL Injection</td>
<td>Information Leakage</td>
</tr>
<tr>
<td>11%</td>
<td>27%</td>
<td>21%</td>
</tr>
<tr>
<td>Information Leakage</td>
<td>Cross-site Scripting (XSS)</td>
<td>Other</td>
</tr>
<tr>
<td>11%</td>
<td>26%</td>
<td>12%</td>
</tr>
<tr>
<td>Cryptographic Issues</td>
<td>Server Configuration</td>
<td>Cryptographic Issues</td>
</tr>
<tr>
<td>6%</td>
<td>2%</td>
<td>11%</td>
</tr>
<tr>
<td>Directory Traversal</td>
<td>OS Command Injection</td>
<td>SQL Injection</td>
</tr>
<tr>
<td>4%</td>
<td>&lt;1%</td>
<td>11%</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>Other</td>
<td>Authorization Issues</td>
</tr>
<tr>
<td>3%</td>
<td>&lt;1%</td>
<td>7%</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>Session Fixation</td>
<td>Authentication Issues</td>
</tr>
<tr>
<td>3%</td>
<td>&lt;1%</td>
<td>5%</td>
</tr>
<tr>
<td>Potential Backdoor</td>
<td>Cryptographic Issues</td>
<td>Insufficient Input Validation</td>
</tr>
<tr>
<td>2%</td>
<td>0%</td>
<td>2%</td>
</tr>
<tr>
<td>Time and State</td>
<td>Insufficient Input Validation</td>
<td>Credentials Mgmt</td>
</tr>
<tr>
<td>2%</td>
<td>0%</td>
<td>2%</td>
</tr>
<tr>
<td>Error Handling</td>
<td>Authentication Issues</td>
<td>Directory Traversal</td>
</tr>
<tr>
<td>1%</td>
<td>0%</td>
<td>1%</td>
</tr>
</tbody>
</table>

Table 5: Vulnerability Distribution by Analysis Type
Applications with the Best First-Scan Acceptance Rate?

• Outsourced
• Open Source
• Internally Developed
• Commercial
Internal Apps have Best First Scan Acceptance Rate

![Supplier Performance on First Submission](chart.png)

*Figure 3: Supplier Performance on First Submission (Adjusted for Business Criticality)*
Shortest Remediation Cycle?

- Outsourced
- Open Source
- Internally Developed
- Commercial
Developers Repaired Security Vulnerabilities Quickly

Figure 4: Remediation Performance by Supplier
Security quality is not commensurate with Business Criticality for Financial Industry applications.

Banks, insurance, and financial services companies have among the best raw security quality scores.
Third-Party Assessments
Suppliers of Cloud/Web Apps Most Frequently Subjected to Third-party Risk Assessments
Three-quarters of all third-party assessments required less than 11 days to achieve acceptable levels of security quality.
Trends and Conclusions

• Lower than average SQL Injection and XSS prevalence in an app is an indicator that the development team understands secure coding.

• Static analysis is being performed in addition to dynamic analysis on web applications.

• First mobile app risks appearing in the wild. Both vulnerabilities such as the PDF iOS 4 vulnerability used by jailbreakme.com and mobile apps with trojan functionality.

• Backdoor (likely intentional) in critical software such as Seimens SCADA product discovered and exploited

• Uptick in cloud based software being tested

• Overall, older platforms getting more mature SDLC as developers take to mobile and cloud
Thank You