Remediation Statistics: What Does Fixing Application Vulnerabilities Cost?

Dan Cornell
Denim Group, Ltd.
Agenda

- An Innocent Question
- Finding a Structure for Remediation Projects
- Methodology
- Remediation Data
- Analysis and Recommendations
- Questions
Fixing a Cross-Site Scripting (XSS) Vulnerability

How long does it take?
A) 9.6 minutes
B) 16.2 minutes
C) 84 minutes
D) It doesn’t matter
E) All of the above
Fixing a Cross-Site Scripting (XSS) Vulnerability

How long does it take?

A) 9.6 minutes – Average fix time for stored XSS (no load)

B) 16.2 minutes – Average fix time for reflected XSS (no load)

C) 84 minutes – Average fix time for stored and reflected (loaded)

D) It doesn’t matter

E) All of the above
Fixing a Cross-Site Scripting (XSS) Vulnerability

How long does it take?

A) 9.6 minutes – Average fix time for stored XSS (no load)

B) 16.2 minutes – Average fix time for reflected XSS (no load)

C) 84 minutes – Average fix time for stored and reflected (loaded)

D) It doesn’t matter

E) All of the above
Remediation Worst Practices

- When the security team:
  - Demands a development team devote time and budget to remediation
  - Provides them with no direction or support
  - Has the development team attempt to make fixes on their own
  - Complies when things don’t work out
Remediation Worst Practices

- Result: No new features and half- or non-fixed vulnerabilities
- Good luck getting your next remediation project approved
Finding a Structure for Remediation Projects

- Desired outcome: predictable and effective remediation projects
  - Predictable: know how long they will take and how much they will cost
  - Effective: targeted vulnerabilities actually get fixed
- A community of stakeholders
  - Security
  - Development
  - IT Audit / Compliance
Remediation Projects

- Inception
- Planning
  - Calculate Risk
  - Agree on Fix and Confirmation Methods
  - Determine Level of Effort
  - Schedule
- Execution
  - Set up Development Environment
  - Fix Vulnerabilities
  - Confirm Fixes and Perform Functional Testing
  - Deploy
Remediation: How To Guide

- Describes methodology for software security remediation projects
- Includes tips and best practices
- **Free** online

[denimgroup.com/howtoguide_download_register.html](http://denimgroup.com/howtoguide_download_register.html)
That’s Great But...

- How long will it actually take me to fix my vulnerabilities?

- Software security remediation projects are software development projects
  - So estimate them as such

- Best practices:
  - Bottom-up estimation
  - Cluster vulnerabilities where possible

- It would be nice to have some data to use as a starting point...
Data!

- Took data from 15 remediated applications
- Two types of analysis:
  - Vulnerability-level (4 applications)
  - Project-level (13 applications)
- Data from Inception and Planning phases was too messy
- Data from Execution phase was useable
The Good (Why This Data Might Be Useful)

- Some data is better than no data
  - As long as you understand potential areas of bias
  - Read “How to Measure Anything” by Douglas W. Hubbard

- Had relatively large sample size for some vulnerability types
The Bad (Some Potential Sources of Bias)

- Relatively small sample size
- Based on a single project type
  - Outsourced software security remediation projects
- Data required cleanup and normalization
- Vulnerability data centered around technical vulnerabilities
  - Most identified by automated static analysis
## Vulnerability-Specific Data (20+ Sample Count)

<table>
<thead>
<tr>
<th>Vulnerability Type</th>
<th>Sample Count</th>
<th>Average Fix (minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dead Code (unused methods)</td>
<td>465</td>
<td>2.6</td>
</tr>
<tr>
<td>Poor logging: system output stream</td>
<td>83</td>
<td>2.9</td>
</tr>
<tr>
<td>Poor Error Handling: Empty catch block</td>
<td>180</td>
<td>6.8</td>
</tr>
<tr>
<td>Lack of Authorization check</td>
<td>61</td>
<td>6.9</td>
</tr>
<tr>
<td>Unsafe threading</td>
<td>301</td>
<td>8.5</td>
</tr>
<tr>
<td>ASP.NET non-serializable object in session</td>
<td>42</td>
<td>9.3</td>
</tr>
<tr>
<td>XSS (stored)</td>
<td>1023</td>
<td>9.6</td>
</tr>
<tr>
<td>Null Dereference</td>
<td>157</td>
<td>10.2</td>
</tr>
<tr>
<td>Missing Null Check</td>
<td>46</td>
<td>15.7</td>
</tr>
<tr>
<td>XSS (reflected)</td>
<td>25</td>
<td>16.2</td>
</tr>
<tr>
<td>Redundant null check</td>
<td>21</td>
<td>17.1</td>
</tr>
<tr>
<td>SQL injection</td>
<td>30</td>
<td>97.5</td>
</tr>
</tbody>
</table>
Some Thoughts and Notes

- Apparently deleting code and changing logging methods are easy

- Cross-Site Scripting
  - Vulnerability count tracks with data from WhiteHat, Veracode, other sources
  - Harder to fix reflected XSS than stored XSS

- Lack of Authorization Check
  - Fix consisted of copy/pasting file include into a number of files

- SQL Injection
  - Surprisingly high
  - Reason: fixes were for more complicated SQL injection vulnerabilities
So If I Have 6 Stored XSS Vulnerabilities...

... my remediation project should take about an hour, right?

But wait!
Remediation Is Not Just About Coding Fixes

- This data is from one of four steps in one of three phases
  - “Fix Vulnerabilities” step in the “Execution” phase

- What about Inception and Planning?
  - No great data available yet

- What about the rest of Execution?
  - Set up Development Environment
  - Fix Vulnerabilities
  - Confirm Fixes and Perform Functional Testing
  - Deploy
  - Overhead
Where Is Time Being Spent?

Indicates the weighted average versus the average of individual projects.
Some Thoughts and Notes

- **Setup Development Environment**
  - Best case: existing development environment or VM
  - Worst case: Safari expedition to recreate environment setup because organization no longer had this knowledge
    - Instructions on setting up a development environment were a deliverable

- **Fix Vulnerabilities**
  - This is what people focus on but there is wide variation
Some Thoughts and Notes (continued)

- Confirm Fixes / QA
  - Sometimes this took more time than the actual fixes
  - Best case: Existing set of automated functional / regression tests

- Deploy
  - Best case: use an existing planned release

- Overhead
  - Surprisingly high in some cases
Using the Data

- I thought you said to estimate bottom-up?
  - Yes. Do that
  - Use the vulnerability data as a guide for estimation
  - Use the project composition data for validation
  - Use the lessons of the data to try and minimize required investment
What Can I Do To Minimize Remediation Costs?

Avoid introducing vulnerabilities into your software

(You are all welcome for this piece of sage advice)
What Can I Do To Minimize Remediation Costs?

- Have ready access to development environments for the developers doing the remediation
- Automated functional / regression testing helps speed security fixes
- Use planned deployments when possible
Which Vulnerabilities Get Fixed and When?

- Use your data-backed, bottom-up WBS for risk management and planning
- Serious vulnerabilities that are easy to fix? Consider an out-of-cycle release
- Otherwise leverage planned releases
The Outlier

- We remediated **one vulnerability** not included in the study that was **more expensive to fix than all vulnerabilities in the study**
  - Authentication issue in a connected system

- Requirements and architecture vulnerability
  - Automated scanners – static or dynamic: powerless to find it

- Should have / would have been caught by even a basic threat modeling or abuse case session
So Where Does This Leave Us

- **Good:**
  - We have a framework
  - We have some data

- **Less good:**
  - The data comes with a number of caveats

- **Given a framework and some data you should be:**
  - Better able to execute successful projects
  - Better able to estimate projects
  - Better able to minimize project costs
Next Steps For Me

- Release a more in-depth report
- Include more data in the analysis
- Perform deeper analysis
  - Impact of size of project (hours)
  - Impact of number of vulnerabilities remediated
  - Impact of platform
  - And so on…
- Include data on logical vulnerabilities
Apply

- Review your existing vulnerability data

- Create a “back of the envelope” plan to address open vulnerabilities
  - Run different scenarios: “All critical and high” “All public-facing apps” and so on

- Talk to developers
  - How do they set up development environments?
  - When do they do planned releases?

- Fix some vulnerabilities!
  - Application-level vulnerabilities persist for a long time
Remediation Resource Center

- Resources for remediating software security vulnerabilities
  - Videos
  - How-to Guide
  - Blog posts

[denimgroup.com/remediation]
Questions?

Dan Cornell
dan@denimgroup.com
Twitter: @danielcornell

www.denimgroup.com
blog.denimgroup.com
www.denimgroup.com/remediation
(210) 572-4400