Secure Development Lifecycle

Security at All Stages of Solution Development

Campbell Murray

Global Head, BlackBerry Cybersecurity Services
Introduction
Meet the Trainer

- **Campbell Murray – Global Head, BlackBerry Cybersecurity Services**
  - Over 20 years of consultancy
  - Pen testing, social engineering, research, development (C, C#, VB, Ruby etc)
  - Joined BlackBerry by acquisition in 2016
**Jargon**

- **Bug**
  - An error that causes a system to behave in an unexpected way.

- **Vulnerability**
  - A bug that affects a system’s security.

- **Threat**
  - A potential condition that may affect a system’s security.

- **Risk**
  - The outcome of a successful attack

- **Mitigation**
  - The elimination or reduction in severity of a vulnerability.

- **Exploit**
  - The use of a vulnerability to cause unintended or unanticipated behavior.
Think like an attacker

- **Understand the architecture of the system**
  - Which component contains what code/functionality.
  - External interfaces exposed by the components
  - Functionality exposed by the external interfaces.

- **Understand the interactions between components and external entities**
  - Authentication and authorization
  - Functionality that is used
  - Data that is passed

- **Look at how the system can be attacked**
  - Try to bypass authentication and authorization checks
  - Mess with data – Look at edge cases
  - Mess with functionality – Look for assumptions made

- **Understand the standard types of vulnerabilities**
Security at All Stages of Solution Development
Security In the Development Lifecycle

- **Variety of methods, tools and approaches**
  - No single activity will produce a secure product
  - Different activities applicable at different times

- **Need to consider security at all stages of the SDL**
  - Finding an issue earlier is easier to address
  - Prevent issues from being introduced if possible
  - Have a plan to address issues that are found
Architecture

- **External entities – users, administrators, devices, etc.**
  - What components in the system will they access?
  - How will those components authenticate and authorize the entities?

- **Components – processes and data stores**
  - What accounts will the processes run as?

- **Data flows**
  - What components communicate with each other?
  - How do the different components authenticate and authorize each other?
  - How is the confidentially and integrity of the data protected in transit?
Threat Modeling Methodologies

- STRIDE
- The Process for Attack Simulation and Threat Analysis (PASTA)
- Trike
- VAST - Visual, Agile, and Simple Threat modeling
## STRIDE

<table>
<thead>
<tr>
<th>Threat</th>
<th>Desired property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Authenticity</td>
</tr>
<tr>
<td>Tampering</td>
<td>Integrity</td>
</tr>
<tr>
<td>Repudiation</td>
<td>Non-repudiability</td>
</tr>
<tr>
<td>Information disclosure</td>
<td>Confidentiality</td>
</tr>
<tr>
<td>Denial of Service</td>
<td>Availability</td>
</tr>
<tr>
<td>Elevation of Privilege</td>
<td>Authorisation</td>
</tr>
</tbody>
</table>
Threat Modelling

1. Identify threats
   - For each external entity, process, data store and data flow think about:
     - Bypassing authentication / authorization
     - Messing with data
     - Messing with functionality
     - Use STRIDE to classify threats

2. Rate threats
   - CVSS or another scoring method

3. Mitigate threats
   - Redesign the system to eliminate the threats
   - Apply standard mitigations
   - Build a custom mitigation
   - Transfer the risk – Warn the user
   - Accept the threat
Task 1
Task 2
Verification

- **Review the attack surface**
  - Validate the interfaces present match the architecture
    - Open ports
    - IPC end points
    - etc.
  - Validate authentication and authorization checks are present

- **Perform a vulnerability assessment**
  - Range of static and dynamic testing for known vulnerability types
  - Code review of high risk areas

- **Run automated tools**
  - Internal Tools – Nessus, OpenVAS
  - Static Analysis – Klocwork, Fority, etc.
  - Fuzzing
Release

- **Monitor**
  - Logs
    - Brute forcing
    - Strange queries to APIs
    - CSP logs show XSS attempts
  - Security issues reported
  - Security vulnerabilities in third party software

- **Sustainment engineering**
  - Have the ability to patch security issues and release to customers

- **Third party software**
  - Incorporate the latest stable versions in each release.
History of SDL

At Microsoft:
• Bill Gates Trustworth Computing Memo (2002)
• SQL Slammer Worm (2003)
• Expansion and publication of Microsofts security response team and secure windows initiative

BlackBerry Secure Development History
• Security was one of the three pillars
• Always had dedicated security research teams
• We based our process on in-house work and combined elements of the MS SDL
• Continued to develop with new standards and we now apply strong maturity models
1. ESTABLISH CORE SECURITY TRAINING

In Practise

- Internal Learning Management System
- Directed training for development teams with key issues
- Security Champions within the development team
2. ESTABLISH SECURITY AND PRIVACY REQUIREMENTS

In Practise

• First do no harm – PII
• Second consider regulatory requirements
• Third consider technology options and limitations
• Prioritise systems
  • Is it widely accessible?
  • Would it actually be attacked? Who would attack?
3. CREATE QUALITY GATES / BUG BARS

In Practice

- Power to block a product release for security concerns
- Classification of vulnerabilities
- Prioritisation of systems/data
Setting CVSS Based Bug Bars

SDR Severity
The Severity field of an SDR is assigned based in part on the CVSS score of the vulnerability. Other factors, such as business and customer impact, should also be taken into account on a case by case basis.

<table>
<thead>
<tr>
<th>CVSS score</th>
<th>Security Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 - 10</td>
<td>Critical</td>
</tr>
<tr>
<td>6 – 7.9</td>
<td>Severe or higher</td>
</tr>
<tr>
<td>2 – 5.9</td>
<td>Moderate or higher</td>
</tr>
<tr>
<td>0.1 – 1.9</td>
<td>Low or higher</td>
</tr>
</tbody>
</table>
4. PERFORM SECURITY AND PRIVACY RISK ASSESSMENTS

In Practise

• All the data exists from the previous two activities
• This isn’t always a formal phase in software development
• Generally prioritisation can be done on the key areas to provide a deeper risk assessment.
• Light touch is the key
5. ESTABLISH DESIGN REQUIREMENTS

In Practise

- Developer Guidance Notes (DGNs)
- Common core design rules
- OSS Policies
6. PERFORM ATTACK SURFACE REDUCTION/ANALYSIS

In Practise

- Back to our secure design principles
- Review regularly to catch changes creeping in
- Do we really need 7 copies of OpenSSL in one product?
7. USE THREAT MODELLING

In Practise

• Assets
• Attackers
• Software
• STRIDE
8. USE APPROVED TOOLS

In Practise

- Approved tools and approved options
- Compilers or configurations which default to more secure options
9. DEPRECATE UNSAFE FUNCTIONS

In Practise

• Not just MS unsafe APIs list
• Maybe your library has weaknesses
• Maybe your previous product version needs an API for compatibility but it shouldn’t be used in new code.
10. PERFORM STATIC ANALYSIS

In Practise

- Seems to be first security practise implemented
- Should not be relied upon
- Huge numbers of false positives can be tricky to deal with
- Should be a developer led task with security support.
- It usually takes less time to fix a potential issue than argue over whether it is exploitable
11. PERFORM DYNAMIC ANALYSIS

In Practise

• Runtime monitoring tools
• Usually requires custom developed harnesses
• Include it in your unit testing
• Use specific testing tools
12. PERFORM FUZZ TESTING

In Practise

- Off the shelf fuzzers are often more difficult than writing a custom harness
- Just use a fuzzing framework and tailor it to your software
- Beware of code coverage
13. CONDUCT ATTACK SURFACE REVIEW

In Practise

• Review all components
• Feedback into threat model
• Include code complexity as an attack surface
14. CREATE INCIDENT RESPONSE PLAN
15. CONDUCT FINAL SECURITY REVIEW
16. CERTIFY RELEASE AND ARCHIVE
17. EXECUTE INCIDENT RESPONSE PLAN

In Practise

- You need a response team to handle this for any significant business size
- Public Relations is as important as the technical response
- Development team should be prepared to implement priority fixes
Thank You

Campbell Murray
cmurray@blackberry.com
Global Head, BlackBerry Cybersecurity Services