Avoiding The Top 10 Software Security Design Flaws

Summary of Report

Boston .NET Architecture Group

Robert Hurlbut

Presented 9/17/2014

Notes on Summary

- This is a summary of the report “Avoiding The Top 10 Software Security Design Flaws”.
- The slides, headings, and bullet points in this summary primarily come directly from the report as reviewed by the presenter.
- This summary is not endorsed by or affiliated in any way with the IEEE Computer Society or the Center for Secure Design (CSD).

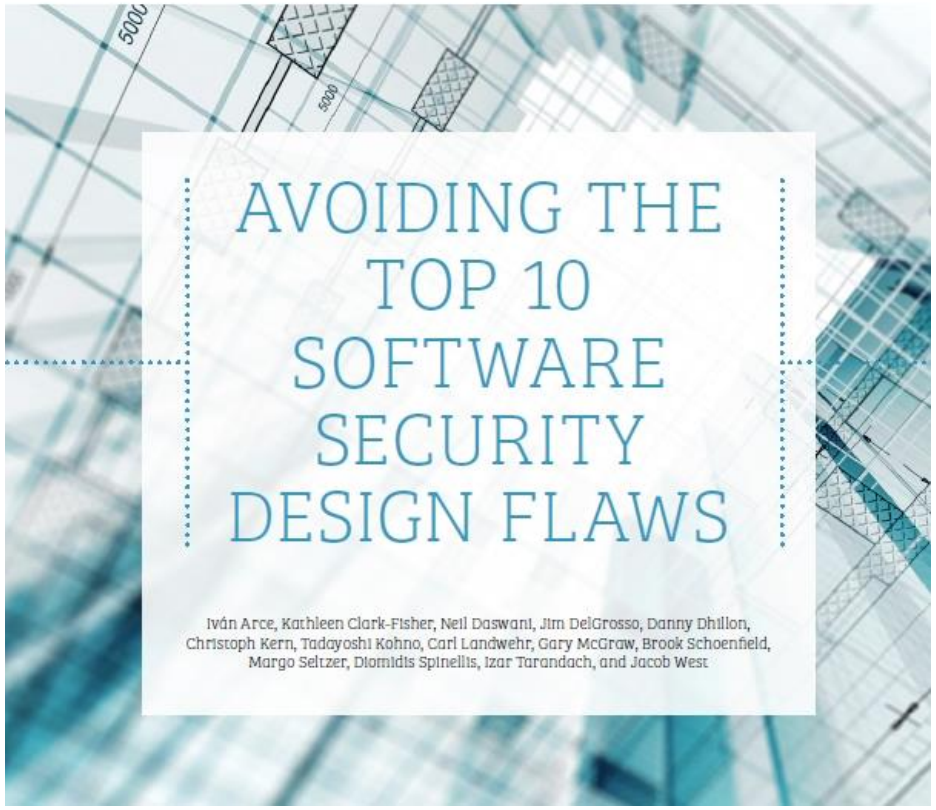
Center for Secure Design (CSD)

- The IEEE Computer Society created the Center for Secure Design (CSD) with a foundational workshop in April, 2014
- The purpose of the workshop was to bring together software security experts who are real practitioners from industry, academia and government to address the problem of secure design.
- Participants in the CSD foundational workshop included experts from major corporations such as RSA, EMC, HP, Google, Twitter, and Intel, along with key academics in the field.

The CSD Mission

- *The IEEE Computer Society's CSD will gather software security expertise from industry, academia and government. The CSD provides guidance on:*
  - *Recognizing software system designs that are likely vulnerable to compromise.*
  - *Designing and building software systems with strong, identifiable security properties.*
- *The CSD is part of the IEEE Computer Society's larger cybersecurity initiative, launched in 2014.*

Report



http://cybersecurity.ieee.org/images/files/images/pdf/CybersecurityInitiative-online.pdf

Bugs vs Flaws

- While a system may always have implementation defects or “bugs,” we have found that the security of many systems is breached due to design flaws or “flaws.”
- We believe that if organizations design secure systems, which avoid such flaws, they can significantly reduce the number and impact of security breaches.
- A bug is an implementation-level software problem.
- A flaw, by contrast, is a problem at a deeper level … it is the result of a mistake or oversight at the design level.

Secure Design Flaws

1. Incorrect trust assumptions
2. Broken authentication mechanisms that can be bypassed or tampered with
3. Neglecting to authorize after authentication
4. Lack of strict separation between data and control instructions, and as a result processing control instructions received from an untrusted source
5. Not explicitly validating all data
6. Misuse of cryptography
7. Failure to identify sensitive data and how they should be handled
8. Failure to consider the users
9. Misunderstanding how integrating external components change an attack surface
10. Brittleness in the face of future changes made to objects and actors

Secure Design Recommendations

1. Earn or give, but never assume, trust
2. Use an authentication mechanism that cannot be bypassed or tampered with
3. Authorize after you authenticate
4. Strictly separate data and control instructions, and never process control instructions received from untrusted sources
5. Define an approach that ensures all data are explicitly validated
6. Use cryptography correctly
7. Identify sensitive data and how they should be handled
8. Always consider the users
9. Understand how integrating external components changes your attack surface
10. Be flexible when considering future changes to objects and actors

1. Earn or give, but never assume, trust

- Confirm sensitive material really does need to be stored on the client.
- If IP or sensitive material must be stored or sent to the client, the system should be designed to cope with potential compromise.
- Make sure all data received from an untrusted client are properly validated before proceeding (more on this later).

2. Use an authentication mechanism that cannot be bypassed or tampered with

- Use of authentication techniques that don't fall into the category of something you know (i.e. password), something you are (i.e. biometric), or something you have (i.e. smartphone) may allow users to access a system or service they shouldn't.
- A single authentication mechanism leverage one or more factors as per an application's requirements – serves as a logical "choke point".
- Authentication credentials have limited lifetimes, be unforgeable, and be stored so that if stolen they can not be used by the thief to pose as legitimate users.

3. Authorize after you authenticate

- Always follow this order. Don’t assume authorization automatically after authentication.
- Use a common infrastructure (e.g. system library or back end) to authorize users.

4. Strictly separate data and control instructions, and never process control instructions received from untrusted sources

- Consider control-flow integrity and segregation of control and potentially untrusted data as important design goals.
- A design that relies on ability to transform data into code should be careful of:
  - Eval functions
  - Query languages
  - Exposed reflection

5. Define an approach that ensures all data are explicitly validated

- Design or use centralized validation mechanism.
- Transform data into canonical form.
- Use common libraries of validation primitives.
- Input validation requirements are often state-dependent.
- Explicitly re-validate assumptions “nearby” code that relies on them.
- Use implementation-language-level types to capture assumptions about data validity.

6. Use cryptography correctly

- Avoid common pitfalls of using cryptography:
  - Rolling your own cryptographic algorithms or implementations
  - Misuse of libraries and algorithms
  - Poor key management
  - Randomness that is not random
  - Failure to centralize cryptography
  - Failure to allow for algorithm adaptation and evolution
- Make use of proven algorithms and libraries

7. Identify sensitive data and how they should be handled

- Identify sensitive data and determine how to protect it appropriately.
- Data sensitivity is context-sensitive dependent on regulation, company policy, contractual obligations, and user expectation.
- Identify trust boundaries for when data sets transit between systems and implement proper data protection policies.

8. Always consider the users

- Create designs that facilitate secure configuration and use by those interested in doing so.
- Use designs that motivate and incentivize secure use among those not particularly interested in software security.
- Use designs that prevent or mitigate abuse from those who intend to weaken or compromise the system.

9. Understand how integrating external components changes your attack surface

- It is a common adage of software security that whenever possible, functionality should be achieved by the reuse of tried-and-true pieces of previously tested and validated software, instead of developing from scratch every time.
- Isolate external components as much as your required functionality permits; use containers, sandboxes, and drop privileges before entering uncontrolled code.
- Document everything.
- Design for flexibility.

10. Be flexible when considering future changes to objects and actors

- Software security must be designed for change, rather than being fragile, brittle, and static.
- Design for secure updates.
- Design for security properties changing over time; for example, when code is updated.
- Design with the ability to isolate or toggle functionality.
- Design for changes to objects intended to be kept secret.
- Design for changes in the security properties of components beyond your control.
- Design for changes to entitlements.

Get Involved

- As stated in the mission statement, the IEEE Computer Society Center for Secure Design will provide guidance on:
  - Recognizing software system designs that are likely vulnerable to compromise.
  - Designing and building software systems with strong, identifiable security properties.
    - This document is just one of the practical artifacts that the Center for Secure Design will deliver.
- Interested in keeping up with Center for Secure Design activities? Follow @ieeecsd on Twitter, catch up with us via cybersecurity.ieee.org, or contact Kathy Clark-Fisher, Manager, New Initiative Development (kclark-fisher@computer.org).

Resources

- http://cybersecurity.ieee.org/
- http://searchsecurity.techtarget.com/opinion/McGraw-on-the-IEEE-Center-for-Secure-Design