RSAConference2015
San Francisco | April 20-24 | Moscone Center

CHANGE
Challenge today's security thinking

SESSION ID: ASD-T09

How to Avoid the Top Ten Software Security Flaws

Gary McGraw, Ph.D.

Chief Technology Officer
Cigital
@cigitalgem

#RSAC

IEEE CENTER FOR SECURE DESIGN

hp

cigital

Google

University of Washington Computer Science & Engineering

Cyber Security Policy and Research Institute
THE GEORGE WASHINGTON UNIVERSITY

fundación SADOSKY
Investigación y Desarrollo en TIC

EMC²

ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

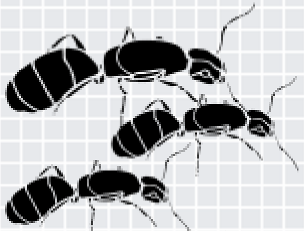
cigital

IEEE CENTER FOR SECURE DESIGN

IEEE CSD Mission

- The IEEE 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.

On Bugs, Flaws, and Defects



`gets()`

`attacker in the middle`

BUGS — FLAWS

- Architectural risk analysis
- Customized static rules
- Commercial SCA tools: Fortify, Ounce Labs, Coverity, Cigital SCA

Avoiding the Top Ten Flaws

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

YES

- Make sure all data from an untrusted client are validated
- Assume data are compromised

- Avoid authorization, access control, policy enforcement, and use of sensitive data in client code

2. Use an authentication mechanism that can't be bypassed

YES

- Prevent the user from changing identity without re-authentication, once authenticated.
- Consider the strength of the authentication a user has provided before taking action
- Make use of time outs

- Do not stray past the big three
  - Something you are
  - Something you have
  - Something you know
- Avoid shared resources like IP numbers and MAC addresses
- Avoid predictable tokens

3. Authorize after you authenticate

YES

- Perform authorization as an explicit check
- Re-use common infrastructure for conducting authorization checks

- Authorization depends on a given set of privileges, *and* on the context of the request
- Failing to revoke authorization can result in authenticated users exercising out-of-date authorizations

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

YES

- Utilize hardware capabilities to enforce separation of code and data
- Know and use appropriate compiler/linker security flags
- Expose methods or endpoints that consume structured types

- Co-mingling data and control instructions in a single entity is bad
- Beware of injection-prone APIs
  - XSS, SQL injection, shell injection
- Watch out for (eval)

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

YES

- Ensure that comprehensive data validation actually takes place
- Make security review of the validation scheme possible
- Use a centralized validation mechanism and canonical data forms (avoid strings)

- Watch out for assumptions about data
- Avoid blacklisting, use whitelisting

6. Use cryptography correctly

YES

- Use standard algorithms and libraries
- Centralize and re-use
- Design for crypto agility
- Get help from real experts

- Getting crypto right is VERY hard
- Do not roll your own
- Watch out for key management issues
- Avoid non-random "randomness"

7. Identify sensitive data and how they should be handled

YES

- Know where your sensitive data are
- Classify your data into categories
- Consider data controls
  - File, memory, database protection
- Plan for change over time

- Do not forget that data sensitivity is often context sensitive
- Confidentiality is not data protection
- Watch out for trust boundaries

8. Always consider the users

YES

- Think about: deployment, configuration, use, update
- Know that security is an emergent property of the system
- Consider user culture, experience, biases, …
- Make things secure by default

- Security is not a feature!
- Don’t impose too much security
- Don’t assume the users care about security
- Don’t let the users make security decisions

9. Understand how integrating external components changes your attack surface

- Test your components for security
- Include external components and dependencies in review
- Isolate components
- Keep an eye out for public security information about components

- Composition is dangerous
- Security risk can be inherited
- Open source is not secure
- Don’t trust until you have applied and reviewed controls
- Watch out for extra functionality

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

YES

- Design for change
- Consider security updates
- Make use of code signing and code protection
- Allow isolation and toggling
- Have a plan for "secret compromise" recovery

- Watch out for fragile and/or brittle security
- Be careful with code signing and system administration/operation
- Keeping secrets is hard
- Crypto breaks

Avoiding the Top Ten Flaws

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

Center for Secure Design Contributors

Organization	Individual
Athens University of Economics and Business	Diomidis Spinellis
Cigital	Jim DelGrosso
Cigital	Gary McGraw
EMC	Izar Tarandach
George Washington University	Carl Landwehr
Google	Christoph Kern
Harvard University	Margo Seltzer
HP	Jacob West
McAfee, Part of Intel Security Group	Brook Schoenfield
RSA	Danny Dhillon
Sadosky Foundation	Iván Arc
Twitter	Neil Daswani
University of Washington	Tadayoshi Kohno

SearchSecurity + Silver Bullet

www.searchsecurity.com

No-nonsense monthly security column by Gary McGraw

TechTarget

SearchSecurity

www.cigital.com/~gem/writing

www.cigital.com/silverbullet

Apply Slide

- Download the IEEE CSD document: http://bit.ly/ieee-CSD
- Adapt the flaw avoidance advice to your organization
  - Copy Twitter
  - Copy Google
- Create design patterns that eradicate classes of bugs
- Join the Center for Secure Design!