Today’s session will be recorded.

An archive of today’s talk will be available at: [www.sercuarc.org/serc-talks/](http://www.sercuarc.org/serc-talks/)

Use the Q&A box to queue questions, reserving the chat box for comments, and questions will be answered during the last 5-10 minutes of the session.

If you are connected via the dial-in information only, please email questions or comments to Ms. Mimi Marcus at mmarcus@stevens.edu.

Any issues? Use the chat feature for any technical difficulties or other comments, or email Ms. Mimi Marcus at mmarcus@stevens.edu.
The Systems Engineering Research Center (SERC) is a federally funded University Affiliated Research Center managed by Stevens Institute of Technology.

Any views, opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense, ASD(R&E), nor the SERC.

No Warranty. This Stevens Institute of Technology Material is furnished on an “as-is” basis. Stevens Institute of Technology makes no warranties of any kind, either expressed or implied, as to any matter including, but not limited to, warranty of fitness for purpose or merchantability, exclusivity, or results obtained from use of the material. Stevens Institute of Technology does not make any warranty of any kind with respect to freedom from patent, trademark, or copyright infringement.

This material has been approved for public release and unlimited distribution.
Avoiding the Top Ten Software Security Flaws

Gary McGraw, Ph.D.
Vice President, Security Technology
Synopsys
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.


• https://cybersecurity.ieee.org/center-for-secure-design/
On Bugs, Flaws, and Defects

- Commercial SAST Tools: HP Fortify, Coverity, etc
- Architectural risk analysis
- Customized static rules

gets()
Avoiding the Top Ten Flaws

- Earn or give, but never assume, trust
- Use an authentication mechanism that cannot be bypassed or tampered with
- Authorize after you authenticate
- Strictly separate data and control instructions, and never process control instructions received from untrusted sources
- Define an approach that ensures all data are explicitly validated

- Use cryptography correctly
- Identify sensitive data and how they should be handled
- Always consider the users
- Understand how integrating external components changes your attack surface
- Be flexible when considering future changes to objects and actors
1. Earn or give, but never assume, trust
Earn or give, but never assume, trust

✔ 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
Use an authentication mechanism that can’t be bypassed

✓ 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
Authorize after you authenticate

- 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
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
Define an approach that ensures all data are explicitly validated

- 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)

_WARNING_:
- Watch out for assumptions about data
- Avoid blacklisting, use whitelisting
6. Use cryptography correctly
Use cryptography correctly

- 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
Identify sensitive data and how they should be handled

- 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
Always consider the users

- 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
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
Be flexible when considering future changes to objects and actors

- 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
# Center for Secure Design Early Contributors

<table>
<thead>
<tr>
<th>Organization</th>
<th>Individual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Athens University of Economics and Business School</td>
<td>Diomidis Spinellis</td>
</tr>
<tr>
<td>Synopsys</td>
<td>Jim DelGrosso</td>
</tr>
<tr>
<td>Synopsys</td>
<td>Gary McGraw</td>
</tr>
<tr>
<td>EMC</td>
<td>Izar Tarandach</td>
</tr>
<tr>
<td>George Washington University</td>
<td>Carl Landwehr</td>
</tr>
<tr>
<td>Google</td>
<td>Christoph Kern</td>
</tr>
<tr>
<td>Harvard University</td>
<td>Margo Seltzer</td>
</tr>
<tr>
<td>HP</td>
<td>Jacob West</td>
</tr>
<tr>
<td>McAfee, Part of Intel Security Group</td>
<td>Brook Schoenfield</td>
</tr>
<tr>
<td>RSA</td>
<td>Danny Dhillon</td>
</tr>
<tr>
<td>Sadosky Foundation</td>
<td>Iván Arc</td>
</tr>
<tr>
<td>Twitter</td>
<td>Neil Daswani</td>
</tr>
<tr>
<td>University of Washington</td>
<td>Tadayoshi Kohno</td>
</tr>
</tbody>
</table>
Where to learn more
Silver Bullet + Writings


- Writings, Blogs, Music garymcgraw.com
The Book + BSIMM8

• How to DO software security
  – Best practices
  – Tools
  – Knowledge
  – bsimm.com

• Cornerstone of the Addison-Wesley Software Security Series:
  www.swsec.com
Build Security In


• Send me e-mail: gem@cigital.com

• @cigitalgem
SERC 2017 Annual Events

**TUESDAY NOVEMBER 7 2017**
Time: 12:00 - 5:00PM
Reception to immediately follow at 5pm

**5TH ANNUAL SERC DOCTORAL STUDENTS FORUM**

**WEDNESDAY NOVEMBER 8 2017**
Time: 8:00AM - 5:00PM

**9TH ANNUAL SERC SPONSORED RESEARCH REVIEW**

**MARK YOUR CALENDAR & JOIN US**

LOCATION: FHI360 CONFERENCE CENTER
1825 CONNECTICUT AVE NW, 8TH FLOOR, WASHINGTON, DC 20009

REGISTER NOW

For more information or any questions regarding this event, please contact:
Ms. Monica Brito or Ms. Megan Clifford
UPCOMING TOPICS:

Cybersecurity Series

“The Dilemmas of Cybersecurity -- Why is Everything Broken?”
Dr. William Scherlis, Institute for Software Research, Carnegie Mellon University
November 1, 2017 | 3:00 pm ET

Thank you for joining us!
Please check back on the SERC website for today’s recording and future SERC Talks information!