Software Assurance (SwA)
• See anything wrong with this simple C-language program?

```c
#define BUFSIZE 256
int main(int argc, char **argv) {
    char buf[BUFSIZE];
    strcpy(buf, argv[1]);
    printf(buf);
}
```

• Later in the presentation, we will enumerate the flaws
• What is Software Assurance (SwA)?
• Why is SwA important?
• When are SwA principles applied?
• What are some SwA best practices?
• May I see some examples, please?

• Summary
• Q&A
Question 1: What is Software Assurance (SwA)?
Let’s Begin with Information Assurance (IA)

Information Assurance (IA) – Measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation. These measures include providing for restoration of information systems by incorporating protection, detection, and reaction capabilities.

- **Confidentiality** – Prevent unauthorized disclosure of information
- **Integrity** – Ensure that information is not altered in an unauthorized manner
- **Availability** – Make information available to authorized users, even when under attack

\[ C = \text{Confidentiality} \]
\[ I = \text{Integrity} \]
\[ A = \text{Availability} \]

- **Identification** – Who are you?
- **Authentication** – Prove your identity
- **Authorization** – What are you allowed to do?
- **Accountability** – Record all security relevant activities for subsequent review
- **Non-repudiation** – You cannot prove that a security relevant event was not undertaken by an entity bearing your identity
Some IA Framework Domains – SwA is One

Written, distributed, and enforced IA policies and procedures
Risk management
  Risk assessment
  Control selection
  Control implementation
  Control monitoring
Physical security
Operational security
  Subjects (both people and software): identification, authentication, authorization
  Objects: access control
  Crypto key management
  Continuous monitoring (IDS, IPS, DLP) leading to situational awareness
  Incident management & disaster recovery
  Configuration management (operational environment)
  Patch management
Personnel security
  Education, training, and awareness
  Clearances
  Separation of duties
  Rotation of duties
  Activity monitoring
Component security
  HW counterfeit protection
  HW anti-tamper
  Software assurance
Supply chain assurance
  Supplier practices
  Product assurance
  Configuration management (development environment)
What is Application Security?

Application Security IS NOT:

• Network Security, Firewalls, Intrusion Detection Systems, Operating System Hardening, Database Hardening, etc.
  - These are all useful and necessary to ensure the overall security of the Enterprise, but they address different risks
• Common myths
  - “We have a firewall!”
  - It’s an “Internal Application”
  - It’s protected by SSO
• An afterthought
• Out of scope or a “nice to have”
What is Application Security?

Application Security IS:

• Providing reliable, confidential, and valid information at the application layer
  - e.g., Securing the “custom code” that drives a web application

• “Built In” not “Bolted On”
  - Application Security needs to be built into the application throughout the software development lifecycle.
  - It should not be an afterthought after designing, developing, and deploying a solution

• A requirement and a responsibility
Synonymous with Application Security, Secure Software, and Secure Application Development…

SwA is a lifecycle endeavor

CNSSI 4009, “National Information Assurance (IA) Glossary”, Entry for Software Assurance (SwA)
- “Level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle and that the software functions in the intended manner.”
- In other words, the software does what it’s supposed to do
  - Meets all requirements
- But the software does not do things it’s not supposed to do
  - No Easter eggs, no backdoors
  - No vulnerabilities introduced by poor coding practices
SAFECode is a non-profit organization that promotes the best practices of proven software assurance methods. They define the three pillars of SwA as follows:

- **Security** – Software is developed in accordance with secure lifecycle development principles.

- **Authenticity** – The recipient of the software is provided a level of confidence that the software originated with the expected source.

- **Integrity** – The recipient of the software is provided means to verify that the software was not altered in any way by any party during transit through the supply chain.
DHS Software and Supply Chain Assurance Build Security In

SwA Addresses

- **Conformance**: Planned and systematic set of multi-disciplinary activities that ensure software processes and products conform to requirements, standards / procedures.
- **Predictable Execution**: Justifiable confidence that software, when executed, functions as intended.
- **Trustworthiness**: No exploitable vulnerabilities exist, either maliciously or unintentionally inserted.

https://buildsecurityin.us-cert.gov
SwA Addresses

DHS Software and Supply Chain Assurance
Build Security In

https://buildsecurityin.us-cert.gov

Predictable Execution
Justifiable confidence that software, when executed, functions as intended

Conformance
Planned and systematic set of multi-disciplinary activities that ensure software processes and products conform to requirements, standards / procedures

Trustworthiness
No exploitable vulnerabilities exist, either maliciously or unintentionally inserted
Planned and systematic set of multi-disciplinary activities that ensure software processes and products conform to requirements, standards / procedures.

Conformance

Predictable Execution

Trustworthiness

Justifiable confidence that software, when executed, functions as intended.

No exploitable vulnerabilities exist, either maliciously or unintentionally inserted.

DHS Software and Supply Chain Assurance
Build Security In

https://buildsecurityin.us-cert.gov
It is also important to understand that software assurance is the responsibility of many disciplines, including:

- Program Management
- Systems Engineering
- Software Engineering
- Security Engineering
- Supply Chain
Question 2: Why is SwA important?
Software Applications - The New Attack Target

• Application security has often been ignored, in part because of the faulty assumption that firewalls and other perimeter defenses can protect the functional code.

• The problem is further compounded as application developers without specific security training are typically unaware of the ways their software, while meeting functional requirements, could be compromised.

• As the operating system and network security vulnerabilities have been reduced over time, applications have become the next attack target.

Today the attackers are targeting the applications
Here is an example of vulnerability risks, even with perimeter defenses in place. The diagram shows several firewalls constructed at different points of access with a network.

Vulnerable applications expose the internal network.
The consequences of a security breach are detrimental
- Information is a valuable asset to the Company and we need to protect the way we access and modify it
- Application Security has a direct effect on the Company's profit
- Application Security has a direct effect on the Company's reputation
Question 3: When are SwA principles applied?
SwA Spans The Entire Development Lifecycle

Training
- Software Assurance for Managers
- Software Assurance Practitioner
- Cyber Professional
- CSSLP

Initiation, Development and Acquisition
- Threat, Attack Surface, and Misuse Case Modeling
- Requirements Management
- IBM Rational Doors

Implementation and Assessment
- Static Source Code Analysis
- Static Binary Code Analysis
- Dynamic Binary Analysis
- 3rd Party Reviews
- Secure Coding Standard

Operations and Maintenance Disposal
Certified Software Management
- Continuous Assessment
- Patching
- Configuration Management
- 3rd Party Reviews

Whole System Monitoring
- Detection
- Correlation
- Visualization
- Response

It’s NOT just about writing code!
Pay Me Now ($), or Pay Me Later ($$$)

Hours to Fix Defects by Development Phase

<table>
<thead>
<tr>
<th></th>
<th>Design and architecture</th>
<th>Implementation</th>
<th>Integration testing</th>
<th>Customer beta test</th>
<th>Postproduct release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>1X*</td>
<td>5X</td>
<td>10X</td>
<td>15X</td>
<td>30X</td>
</tr>
</tbody>
</table>

Question 4: What are some SwA best practices?
Software developers should be trained in assured software development practices. Software managers should be trained to understand and advocate software assurance. Software assurance should be treated as an integrated, required part of the software development life cycle process. Software assurance should be included in program reviews for projects that require developed software.
Train SW leads, develop community of SwA SMEs
Available Training

• Government
  - DHS / DoD Build Security In Web Site

• Industry
  - SAFECeCode One Hour Training Videos
  - Lone Star Application Security Conference (LASCON)
    ▪ Held in Austin TX, sponsored by Austin OWASP Chapter
    ▪ Focus on web application security
    ▪ Often preceded by 1-3 days of relevant SwA training
Lots of Reference Help is Available

• For Awareness and Guidance
  - DHS / DoD Build-Security-In Pocket Guide series

• For Developers
  - MITRE CWE
    - Common Weakness Enumeration
    - http://cwe.mitre.org
  - SANS / CWE Top 25 Most Dangerous Software Errors
    - See MITRE CWE
    - http://www.sans.org/top25-software-errors
  - OWASP Top Ten
    - Focused on web applications (increased attack surface)
There’s functional security requirements… and then there’s the “ilities”
Common General Concepts

- **Economy of Mechanism**
  - Keep the design as simple and small as possible. (Keep it simple stupid)

- **Least Common Mechanism**
  - Minimize the amount of mechanism common to more than one user and depended on by all users. (Opposite of Economy of Mechanism.)

- **Complete Mediation**
  - Every access to every object must be checked for authority.

- **Failing Securely**
  - Base access decisions on permission rather than exclusion.

- **Psychological Acceptability**
  - It is essential that the human interface be designed for ease of use, so that users routinely and automatically apply the protection mechanisms correctly.
SAFE Code Secure Design Principles

- Threat Modeling
- Use Least Privilege
- Implement Sandboxing
- Secure Coding Practices
- Validate Input and Output to Mitigate Common Vulnerabilities
- Use Robust Integer Operations for Dynamic Memory Allocations and Array Offsets

- Use Anti-Cross Site Scripting (XSS) Libraries
- Use Canonical Data Formats
- Avoid String Concatenation for Dynamic SQL Statements
- Eliminate Weak Cryptography
- Use Logging and Tracing

Threat Analysis and Attack Surface

Training
- Software Assurance for Managers
- Software Assurance Practitioner
- Cyber Professional
- CSSLP

Initiation, Development and Acquisition
- Threat, Attack Surface, and Misuse Case Modeling
- Requirements Management
  - IBM Rational Doors

Implementation and Assessment
- Static Source Code Analysis
- Static Binary Code Analysis
- Dynamic Binary Analysis
- 3rd Party Reviews
- Secure Coding Standard

Operations and Maintenance Disposal
Certified Software Management
- Continuous Assessment
- Patching
- Configuration Management
- 3rd Party Reviews

Whole System Monitoring
- Detection
- Correlation
- Visualization
- Response

Essential during architecture and high-level design
Threat Analysis

- Who? – Adversary characterization
- What? – Threat categories
- Where? – Attack surface analysis
- How? – Attack trees
- All the above? – Attack Patterns

Know Your Enemy and Think Like An Attacker
Who – Threat Actors

- Script kiddies – Low-skilled, unfunded, just for fun
- Identity thieves – Highly skilled, business model, financial motives
- Nation states – Very highly skilled, very highly funded, political and military objectives, theft of corporate IP
- Hactivists – Ranging skills, lightly funded, highly motivated by social or political objectives
STRIDE is an acronym that stands for the six threat categories documented in Howard and Leblanc’s *Writing Secure Code*. STRIDE stands for the following:

- **Spoofing**: Can the software or end user be tricked into seeing some data as something other than what it actually is? Examples include packet spoofing and user interface spoofing.
- **Tampering with data**: Can someone that shouldn’t have access to modifying or deleting the data do so?
- **Repudiation**: Is it possible to prove which actions a user has taken? For example, are important actions logged and are the logs accurate?
- **Information disclosure**: Is only necessary information given to users? Can the additional information reveal information about the target system or users?
- **Denial of service**: Is an attacker able to prevent legitimate users from accessing the application?
- **Elevation of privilege**: Is an attacker able to perform actions that only higher-privileged users should be allowed to perform?
• Attack Surface = Accessible entry points and exit points of an application or system
  - Entry points are the inputs to the application through interfaces, services, protocols, and code
  - Exit points are the outputs from the application, including error messages produced by the application in response to user interaction

• Goal: Reduce the attack surface of the software application or system
  - Disable unused services and protocols
  - Generic error messages
The aim for secure development is to reduce the overall risk by reducing the attack surface area.

One can reduce the attack surface area by:
- Reducing the amount of code executing
  - Turn off features
- Reducing the volume of code accessible to users
  - Least privilege
- Limit the damage if the code is exploited
How – Attack Trees

- Provides a formal method of modeling threats against a computer system or SW application
- Root node = Goal
- Leaf nodes are different ways of achieving the goal
- Child nodes are Sub-Goals - Ex. “Learn Password”
- All sub-goals must be satisfied before the main goal is achieved
• The MITRE Corporation
  - Government sponsored non-profit organization
  - Federally Funded Research and Development Center (FFRDC)

• Software Assurance Strategic Initiative
  - Department of Homeland Security
  - Co-sponsored by the National Cyber Security Division

• Database of attack methods used by hackers to exploit software

• Understand how weaknesses and vulnerabilities lead to a compromise
CAPEC – Common Attack Pattern and Enumeration Classification

- Probing an application through its error messages
  - Fuzzing input parameters to cause errors
  - Learn about the system
    - Error Messages
    - System behavior

Description
Attack Prerequisites
Typical Likelihood of Exploit
Examples – Instances
Attacker Skills or Knowledge Required
Resources Required
Indicators – Warnings of Attack
Obfuscation Techniques

* Solutions and Mitigations *
Misuse / Abuse Case Modeling

Training
- Software Assurance for Managers
- Software Assurance Practitioner
- Cyber Professional
- CSSLP

Initiation, Development and Acquisition
- Threat, Attack Surface, and Misuse Case Modeling
  - Requirements Management
    - IBM Rational Dawn

Implementation and Assessment
- Static Source Code Analysis
- Static Binary Code Analysis
- Dynamic Binary Analysis
- 3rd Party Reviews
  - Secure Coding Standard

Operations and Maintenance Disposal
- Certified Software Management
  - Continuous Assessment
  - Patching
  - Configuration Management
  - 3rd Party Reviews

Whole System Monitoring
- Detection
- Correlation
- Visualization
- Response

Think like an attacker
Misuse Case Diagram

Start with use cases (white hat)

“Think like a bad guy” (black hat)

For each use case, ask “How might an attacker try to exploit this use case?
SwA is not all about writing code… but it is a lot about writing code.
CERT Top 10 Secure Coding Practices

- Validate Input
- Heed compiler warnings
- Architect and design for security policies
- Keep it simple
- Default deny
- Adhere to the principle of least privilege
- Sanitize data sent to other systems
- Practice defense in depth
- Use effective quality assurance techniques
- Adopt a secure coding standard
• Software Engineering Institute (SEI) – Carnegie Mellon University
  - Federally Funded Research and Development Center (FFRDC)

• Computer Emergency Response Team (CERT)
  - The Morris Worm 11/1988
    ▪ The first Internet worm
    ▪ Wide mainstream media attention

• Secure coding standards
  - C, C++, Java and Perl

The CERT Oracle Secure Coding Standard for Java

CERT and Oracle are developing The CERT Oracle Secure Coding Standard for Java.

The rules and recommendations are not globally editable, but anyone is able to add comments, and qualified individuals can be added as editors.

We are depending on the active involvement of the Java community (you) to make this effort a success. We invite you to participate in this effort by reviewing content in the Java space and providing comments, or by contributing new rules and recommendations for secure Java coding. These can be included as comments or emailed to secure-coding at cert dot org.

Java is a trademark or registered trademark of Oracle Corporation, in the US and other countries.

Java Concurrency Guidelines TR Released
CERT has released the Java Concurrency Guidelines technical report that documents the portion of the CERT Oracle Secure Coding Standard for Java that are related to concurrency.
SAST – Static Analysis Security Tool

Training
- Software Assurance for Managers
- Software Assurance Practitioner
- Cyber Professional
- CSSLP

Initiation, Development and Acquisition
- Threat, Attack Surface, and Misuse Case Modeling
- Requirements Management
- IBM Rational Doors

Implementation and Assessment
- Static Source Code Analysis
- Static Binary Code Analysis
- Dynamic Binary Analysis
- 3rd Party Reviews
- Secure Coding Standard

Operations and Maintenance Disposal
- Certified Software Management
- Continuous Assessment
- Patching
- Configuration Management
- 3rd Party Reviews

Whole System Monitoring
- Detection
- Correlation
- Visualization
- Response

Coverity, HP-Fortify, Klocwork, Parasoft, Veracode, AdaCore (Ada)
• The 2011 CWE/SANS Top 25 Most Dangerous Programming Errors is a list of the most widespread and critical programming errors that can lead to serious software vulnerabilities.

<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>93.8</td>
<td>CWE-99</td>
<td>Improper Neutralization of Special Elements used in an SQL Command (&quot;SQL Injection&quot;)</td>
</tr>
<tr>
<td>2</td>
<td>83.3</td>
<td>CWE-79</td>
<td>Improper Neutralization of Special Elements used in an OS Command (&quot;OS Command Injection&quot;)</td>
</tr>
<tr>
<td>3</td>
<td>79.0</td>
<td>CWE-190</td>
<td>Buffer Copy without Checking Size of Input (&quot;Classic Buffer Overflow&quot;)</td>
</tr>
<tr>
<td>4</td>
<td>77.7</td>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation (&quot;Cross-site Scripting&quot;)</td>
</tr>
<tr>
<td>5</td>
<td>76.9</td>
<td>CWE-306</td>
<td>Missing Authentication for Critical Function</td>
</tr>
<tr>
<td>6</td>
<td>76.8</td>
<td>CWE-862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>7</td>
<td>75.0</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>8</td>
<td>75.0</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>9</td>
<td>74.0</td>
<td>CWE-844</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>10</td>
<td>74.0</td>
<td>CWE-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
<tr>
<td>11</td>
<td>73.1</td>
<td>CWE-230</td>
<td>Execution with Unnecessary Privileges</td>
</tr>
<tr>
<td>12</td>
<td>70.1</td>
<td>CWE-332</td>
<td>Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>13</td>
<td>69.3</td>
<td>CWE-222</td>
<td>Improper Limitation of a Pathname to a Restricted Directory (&quot;Path Traversal&quot;)</td>
</tr>
<tr>
<td>14</td>
<td>68.5</td>
<td>CWE-494</td>
<td>Download of Code Without Integrity Check</td>
</tr>
<tr>
<td>15</td>
<td>67.8</td>
<td>CWE-863</td>
<td>Incorrect Authorization</td>
</tr>
<tr>
<td>16</td>
<td>66.0</td>
<td>CWE-929</td>
<td>Inclusion of Functionality from Untrusted Control Sphere</td>
</tr>
<tr>
<td>17</td>
<td>65.5</td>
<td>CWE-232</td>
<td>Incorrect Permission Assignment for Critical Resource</td>
</tr>
<tr>
<td>18</td>
<td>64.6</td>
<td>CWE-676</td>
<td>Use of Potentially Dangerous Function</td>
</tr>
<tr>
<td>19</td>
<td>64.1</td>
<td>CWE-327</td>
<td>Use of a Broken or Risky Cryptographic Algorithm</td>
</tr>
<tr>
<td>20</td>
<td>62.4</td>
<td>CWE-131</td>
<td>Incorrect Calculation of Buffer Size</td>
</tr>
<tr>
<td>21</td>
<td>61.5</td>
<td>CWE-307</td>
<td>Improper Restriction of Excessive Authentication Attempts</td>
</tr>
<tr>
<td>22</td>
<td>61.1</td>
<td>CWE-601</td>
<td>URL Redirection to Untrusted Site (&quot;Open Redirect&quot;)</td>
</tr>
<tr>
<td>23</td>
<td>61.0</td>
<td>CWE-114</td>
<td>Uncontrolled Format String</td>
</tr>
<tr>
<td>24</td>
<td>60.3</td>
<td>CWE-190</td>
<td>Integer Overflow or Wraparound</td>
</tr>
<tr>
<td>25</td>
<td>59.9</td>
<td>CWE-798</td>
<td>Use of a One-Way Hash without a Salt</td>
</tr>
</tbody>
</table>

Source: http://cwe.mitre.org
1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross Site Request Forgery (CSRF)
9. Using Components with Known Vulnerabilities
10. Unvalidated Redirects and Forwards

The Open Web Application Security Project (OWASP) is a 501(c)(3) worldwide not-for-profit charitable organization focused on improving the security of software.

https://www.owasp.org/index.php/Top_10_2013-Top_10
Third Party Manual Code Reviews

**Training**
- Software Assurance for Managers
- Software Assurance Practitioner
- Cyber Professional
- CSSLP

**Initiation, Development and Acquisition**
- Threat, Attack Surface, and Misuse Case Modeling
- Requirements Management
- IBM Rational Doors

**Implementation and Assessment**
- Static Source Code Analysis
- Static Binary Code Analysis
- Dynamic Binary Analysis
- 3rd Party Reviews
- Secure Coding Standard

**Operations and Maintenance Disposal**
- Certified Software Management
- Continuous Assessment
- Patching
- Configuration Management
- 3rd Party Reviews

**Whole System Monitoring**
- Detection
- Correlation
- Visualization
- Response

Sorry, it cannot all be automated
Executable code now required
Types of Security Testing

- **White Box Testing**
  - Typically have access to design docs, source code etc.
  - Static source code analysis
  - Dynamic source code analysis

- **Black Box Testing**
  - Limited to no knowledge of the design
  - No access to source code

- **Fuzzing**
  - Send random and malformed data, inputs etc to the application

- **Penetration Testing**
  - Typically performed after an application has been deployed
  - Actual attack that attempts to compromise an application

- and others…
SwA activity does not end until system End-Of-Life (EOL)
Question 5: May I see some examples, please?
So What’s So Bad about This Small C-program?

- Remember this simple C-language program?

```c
#define BUFSIZE 256
int main(int argc, char **argv) {
    char buf[BUFSIZE];
    strcpy(buf, argv[1]);
    printf(buf);
}
```

- Code does not check argc for number of arguments... potential null pointer dereference (CWE-476)
- Buffer of 256 bytes does not allocate space for null-terminator (CWE-193)
- Code does not validate content of argv[1] in any way (CWE-20)
- Buffer copy without checking size of destination buffer (CWE-120)
- Attacker can exploit code that does not provide a format string as the first argument to printf() functions... (CWE-134)
```c
#define BUFSIZE 256
int main(int argc, char **argv) {
    char buf[BUFSIZE];
    strcpy(buf, argv[1]);
}
```

“Smashing the Stack for Fun and Profit”
Cross-Site Scripting – The Beginning

• Consider the URL:

    http://www.foo.com/hello.jsp?name=John+Doe

• What hello.jsp looks like:

    <html><body>
    Hello <%=request.getAttribute("name");%>
    </body></html>

• And the returned page is:

    <html><body>
    Hello John Doe
    </body></html>
Cross-Site Scripting – User Turns Mischievous

• And if the URL is:

```html
http://www.foo.com/hello.jsp?
name=<script>alert(document.cookie)</script>
```

• And the returned page is:

```html
<html><body>
Hello <script>alert(document.cookie)</script>
</body></html>
```

• Then the user’s cookies for www.foo.com will pop up in an alert box on their screen
• Who cares if someone can pop windows up on your screen – This is no big deal right?
• What if the URL is:

```html
http://www.foo.com/hello.jsp?
```

• And the returned page is:

```html
<html><body>
</body></html>
```

• The user’s cookies have just been stolen by the malicious user at www.bar.com through Cross-Site Scripting and a “web bug”
Examine a very simple example of SQL Injection by considering the URL:

http://www.foo.com/edit.jsp?uid=test&password=secret

For the purposes of this application lets say it is used to edit the account details for a given user and retrieves the details for the user with the query:

```
SELECT * FROM usertable WHERE uid='test' AND password='secret';
```
Now consider a slightly different query designed with malicious purposes in mind:

```
http://www.foo.com/edit.jsp?uid=admin’;--&password=whatever
```

The consequences of adding a SQL comment (--) may be devastating, allowing the user to access the admin account with the evaluated SQL below:

```
SELECT * FROM usertable WHERE uid='admin';--' AND password='whatever';
```

Ignoring the comment (“--“ and all that follows), the SQL below is equivalent and gets all attributes for user “admin”… possibly including its hashed password!

```
SELECT * FROM usertable WHERE uid='admin';
```
Summary
• Software Assurance is becoming a requirement, especially for US DoD contracts

**BE PREPARED!**

• The challenge is yours – will you commit to developing software in a secure fashion to protect the interests of customers, our nation, and all who use your software?
Questions?
John Whited, Principal Engineer, Raytheon, has 5 years of experience in Cybersecurity with expertise in Software Assurance (SwA) and secure development life cycles (SDLC). Prior to joining Raytheon, he was a software and a systems engineer in commercial telephony, holding five US patents on Intelligent Networks. He is also a CISSP and a CSSLP. He is a graduate of Texas Tech University with a Bachelors of Science and a Masters of Science in Electrical Engineering. He has made two joint presentations at the RSA Security Conference (2010 and 2012).

E-mail: john.whited@raytheon.com