



# *Ground System Architectures Workshop*

## *Tutorial E*

### *Part 2*

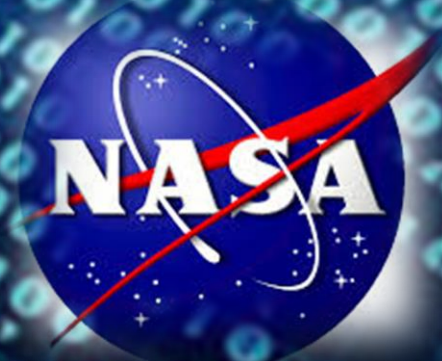
A Proven Methodology for Developing Secure  
Software and Applying It to Ground Systems

2/29/16

Brandon Bailey

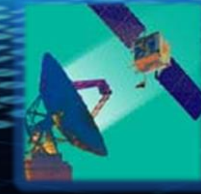
[brandon.t.bailey@nasa.gov](mailto:brandon.t.bailey@nasa.gov)

304-629-8992



*NASA's IV&V Program*  
*Safety and Mission Assurance (SMA) Office*  
*Information Assurance/Cybersecurity Support*  
<http://www.nasa.gov/centers/ivv>

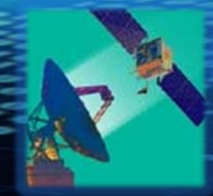
# Agenda/Outline



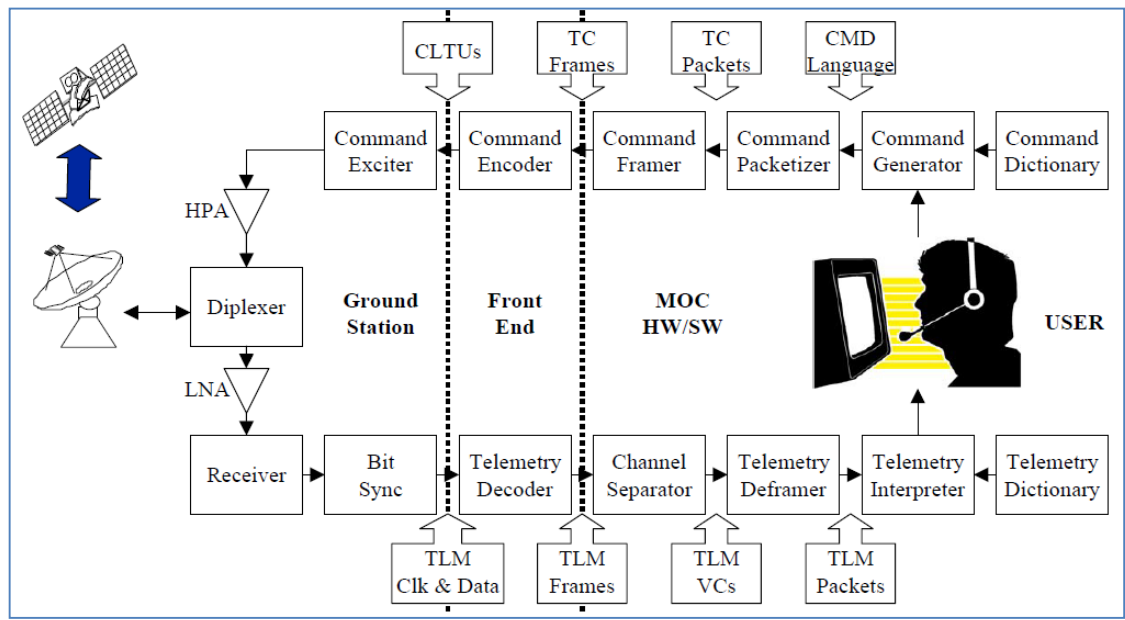
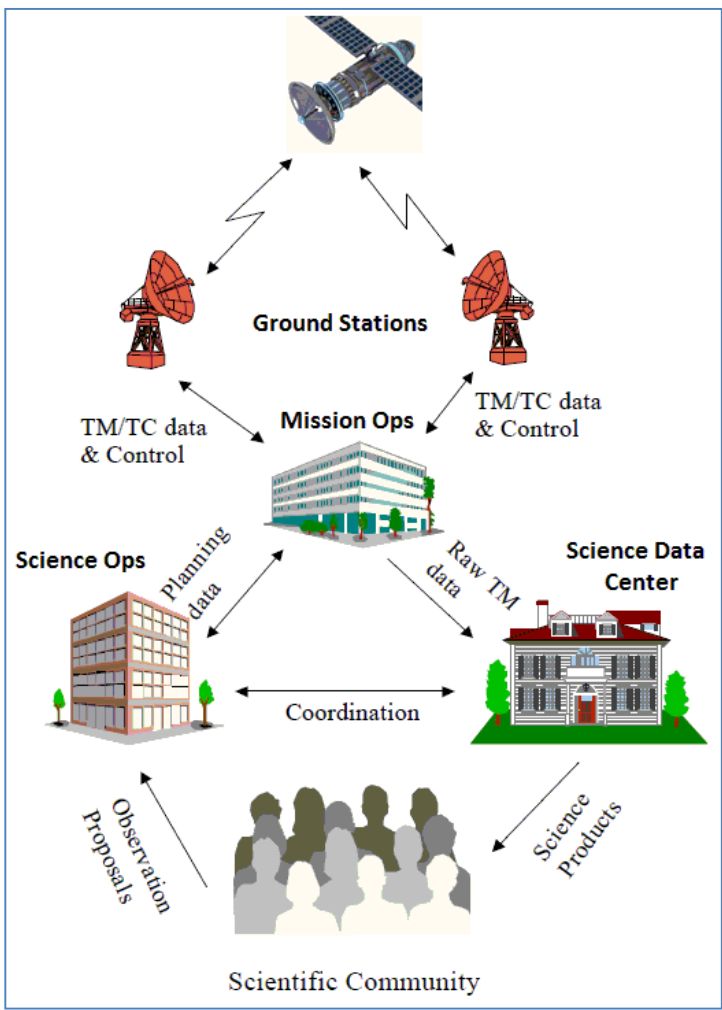
GSAW  
2016

- **Tutorial E Part 1:**
  - Section 1: Cyber Threat – Who, What and Why
  - Section 2: Defense-In-Depth
  - Section 3: Secure Software Engineering Steps
  - Section 4: Errors, Weaknesses and Exploits
  - Section 5: Threat Modeling
  - Section 6: Testing
  - Section 7: Resources
- **Tutorial E Part 2:**
  - Section 1: Ground Systems Overview
  - Section 2: Secure Software Development
  - Section 3: Defense in Depth for Ground Systems
  - Section 4: What Now?

# Defining "Ground Systems"

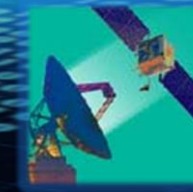


GSAW  
2016

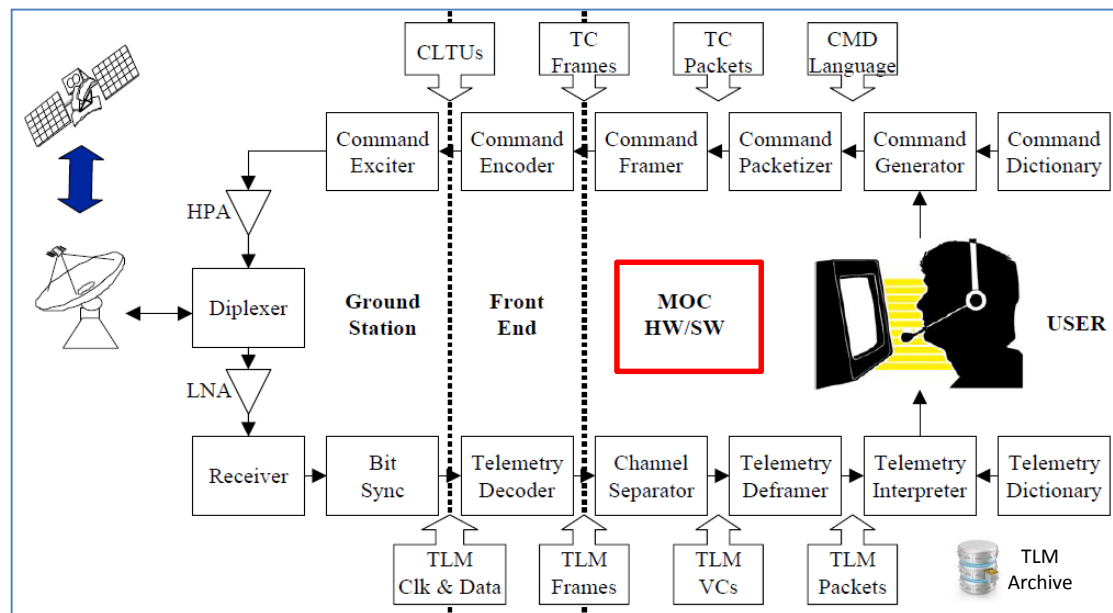
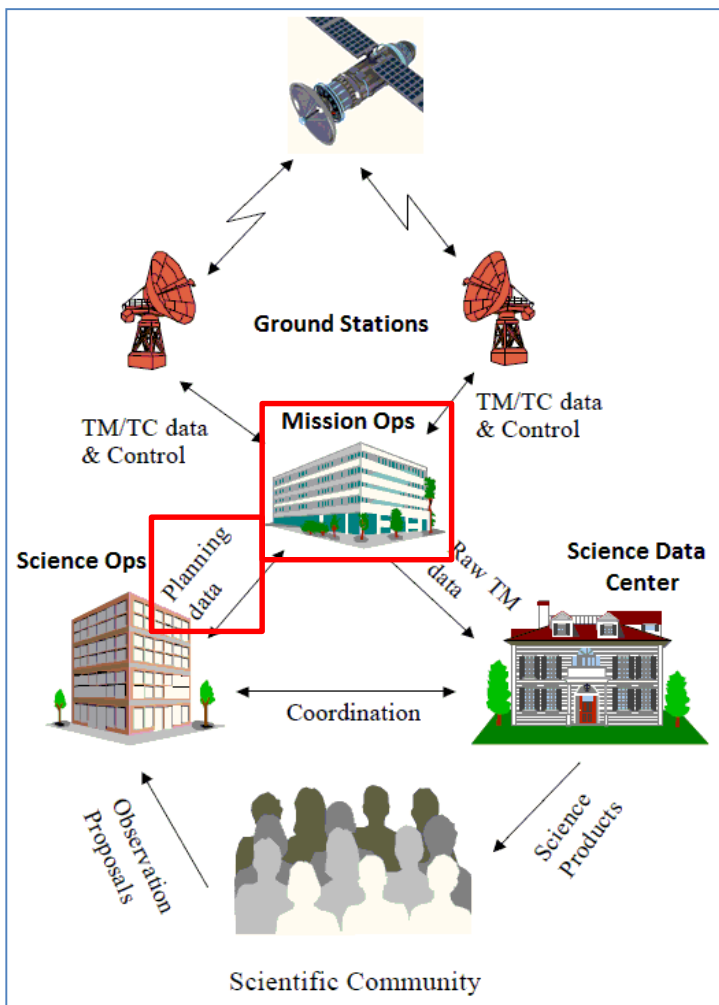


Spacecraft Ground Systems encompasses the entire system, beginning with issuing the command from the MOC up until it emits from the antenna to the reception of radio signals down at the antenna to displaying telemetry on the MOC computer

# Scope...

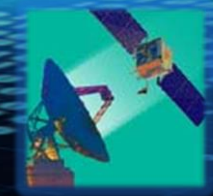


GSAW  
2016



- Tutorial will focus on the software developed for the
  - Mission Operations Center (MOC)
  - Mission planning area
  - Software development environment

# Threats for Space Missions

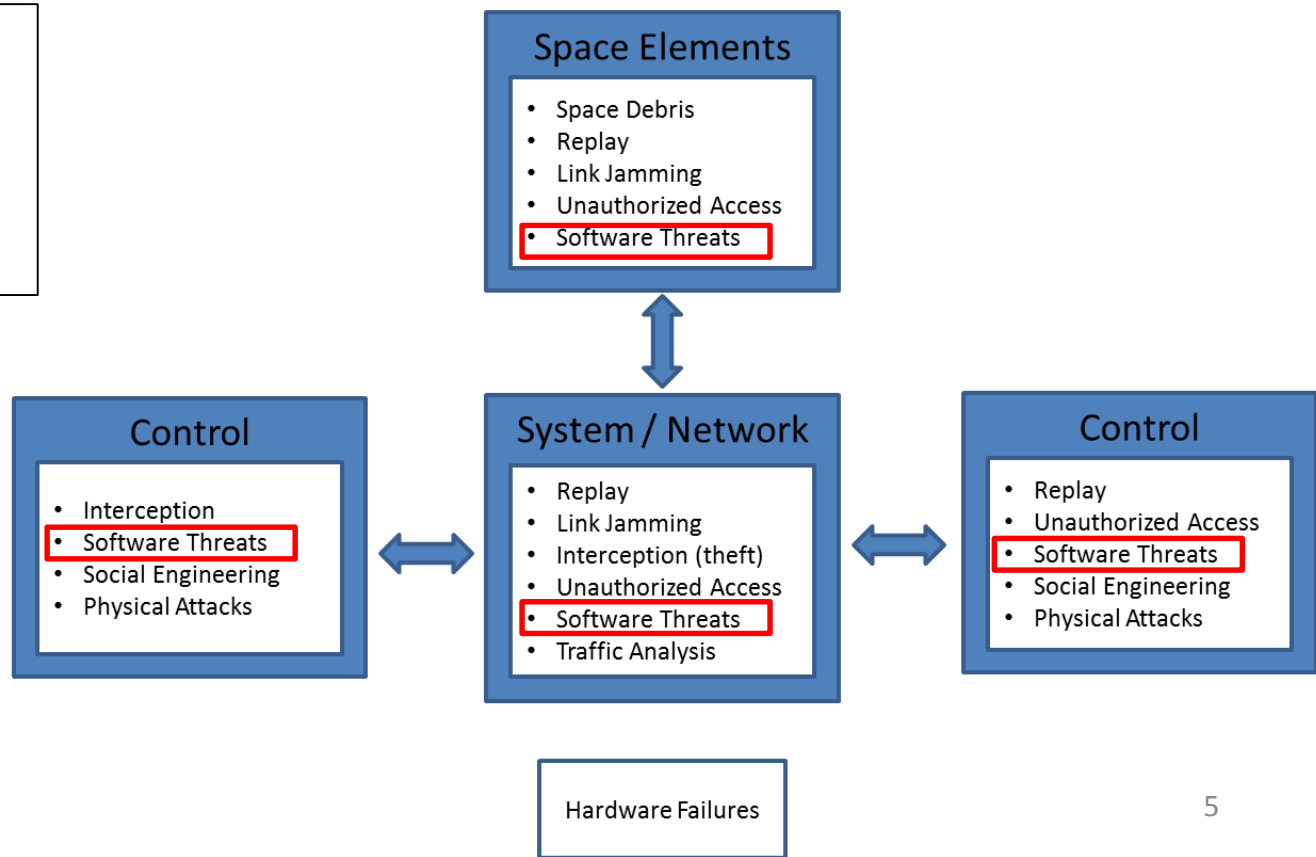


GSAW  
2016

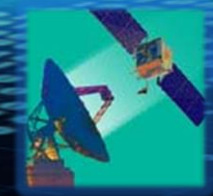
SECURITY THREATS AGAINST SPACE MISSIONS  
CCSDS 350.1-G-1  
March 2015

CCSDS was founded in 1982 by the [major space agencies of the world](#), the CCSDS is a [multi-national forum](#) for the development of communications and data systems standards for spaceflight. 60+ standards published serving 500+ missions

**Security Threats Against Space Missions** was developed to provide mission planners with an overview on threat assessment as well as the common threats and threat sources that exist for various categories of civilian space missions.



# Threats for Space Missions

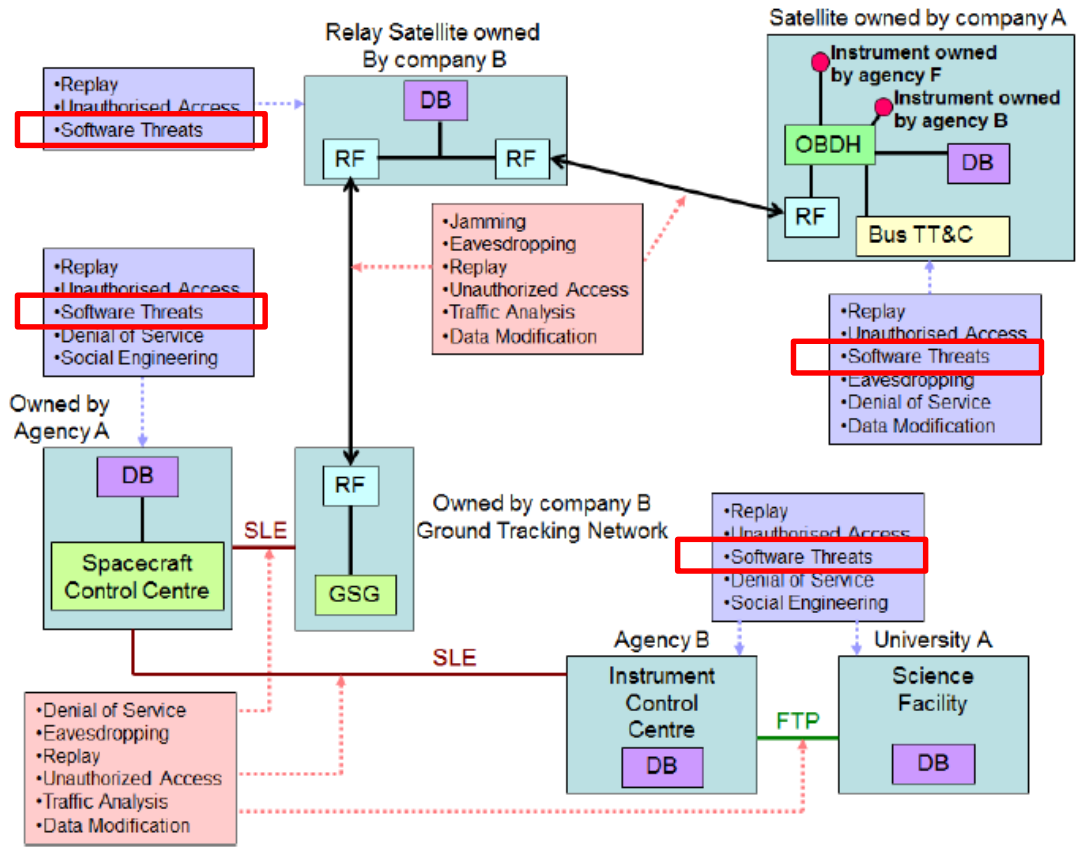


GSAW  
2016

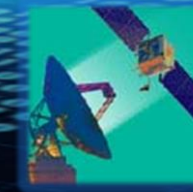
SECURITY THREATS AGAINST SPACE MISSIONS  
CCSDS 350.1-G-1  
March 2015

CCSDS was founded in 1982 by the [major space agencies of the world](#), the CCSDS is a [multi-national forum](#) for the development of communications and data systems standards for spaceflight. 60+ standards published serving 500+ missions

**Security Threats Against Space Missions** was developed to provide mission planners with an overview on threat assessment as well as the common threats and threat sources that exist for various categories of civilian space missions.



# Threats in Space



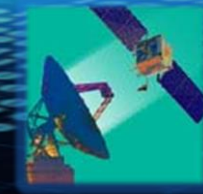
**GSAW  
2016**

Applicable Threats to Space Missions	Impacts	Could Software Be Involved?
<b>Data Corruption</b>	<ul style="list-style-type: none"> <li>• Modification of information</li> <li>• System damage</li> </ul>	Yes; SW attacks could result in data corruption
<b>Ground Facility Physical Attack</b>	Loss of command, control and data	No
<b>Interception</b>	Loss of sensitive data	No
<b>Jamming</b>	<ul style="list-style-type: none"> <li>• Loss of Command telemetry link</li> <li>• Loss of access to resources</li> </ul>	No
<b>Denial-of-Service</b>	Loss of access to resources	Yes; SW DoS attacks are common and can affect both ground, flight and web applications
<b>Masquerade</b>	<ul style="list-style-type: none"> <li>• Potential to disrupt operations (uplink)</li> <li>• Potential to receive false information (downlink)</li> </ul>	Yes; SW protections can be placed to prevent
<b>Replay</b>	System damage (possible safety of life issues)	Yes; SW protections can be placed to prevent
<b>Software threats</b>	<ul style="list-style-type: none"> <li>• <b>Undesirable events</b></li> <li>• <b>System damage</b></li> <li>• <b>Enable other threats (i.e. Jamming, DoS)</b></li> </ul>	<b>Yes</b>
<b>Unauthorized Access</b>	<ul style="list-style-type: none"> <li>• Disruption of operations</li> <li>• System damage (possible safety of life issues)</li> </ul>	Yes; SW protections can be placed to prevent or SW can be used to gain unauthorized access
<b>Tainted Hardware Components</b>	<ul style="list-style-type: none"> <li>• Hidden, Malicious capabilities</li> <li>• System instability</li> <li>• System damage</li> <li>• Undesirable System effects</li> </ul>	No

# Cybersecurity in the space domain

## Isn't ONLY an IT function

*Security is a part of Mission Success*



GSAW  
2016



- Web sites/servers, email, workstation patching, etc.
- CIO infrastructure focused

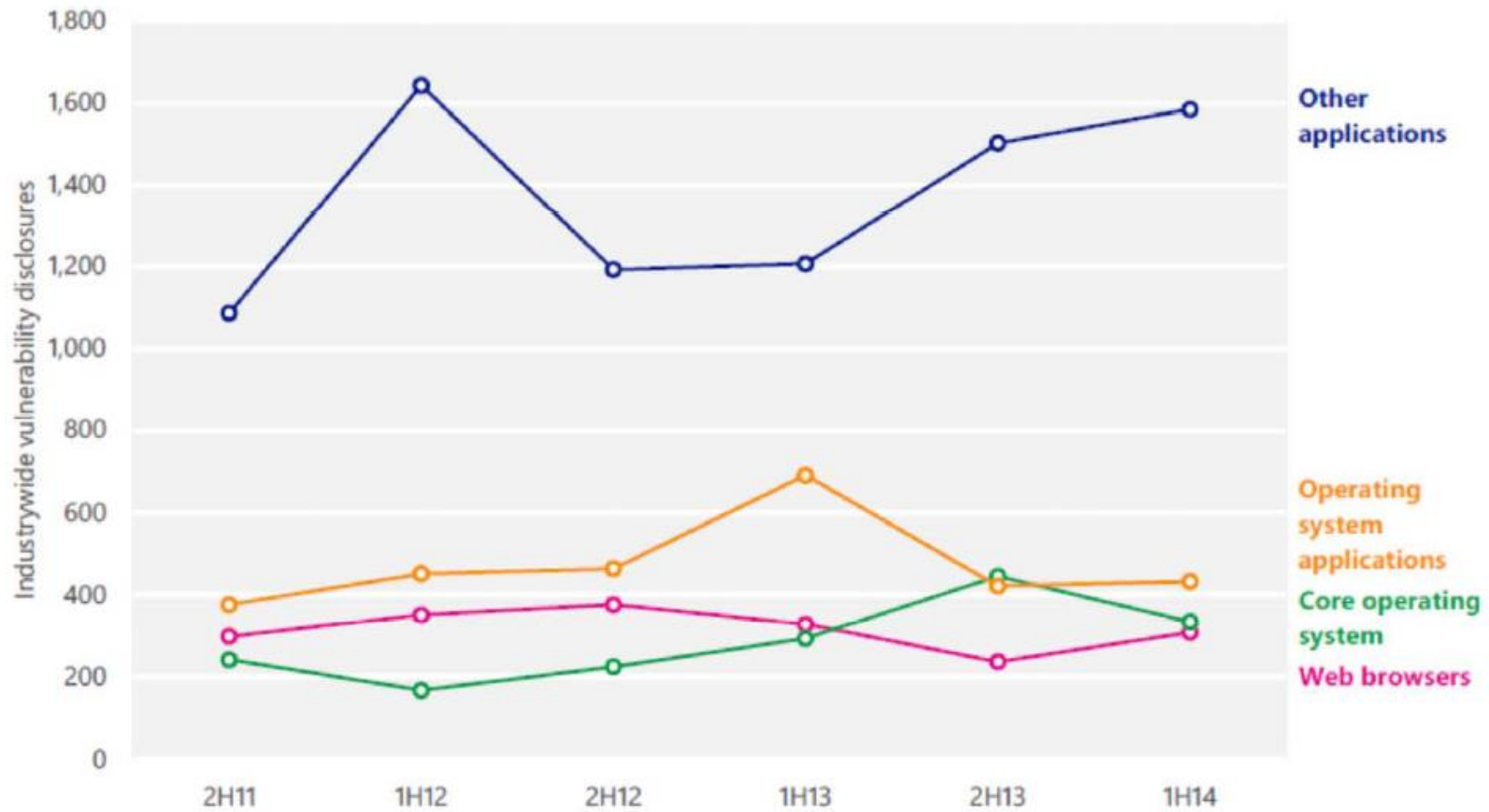
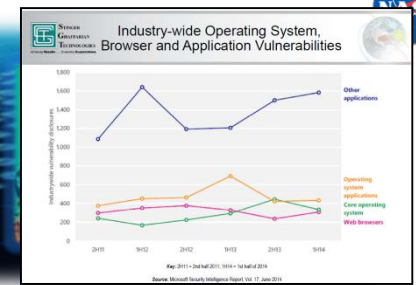
***Must counteract the threat landscape for Mission environments with Defense in Depth***

- Mission Targets / Enterprise Risk
  - **Software Security (COTS, FOSS, Custom, etc.)**
  - Network Layer (Routers, Firewalls, etc.)
  - Computer Network Defense (IPS/IDS, Sensors, Continuous Monitoring, etc.)
  - Industrial Control Systems (ICS)
  - Supply Chain...
- Multiple stakeholders (CIOs, Network Engrs, SW Developers, Project Managers, etc.)





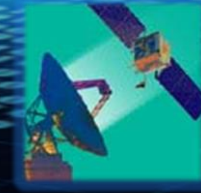
# Custom SW – Gets Exploited!



Key: 2H11 = 2nd half 2011; 1H14 = 1st half of 2014

Source: Microsoft Security Intelligence Report, Vol. 17, June 2014

# Reducing SW Risk



GSAW  
2016

Multiple vulnerabilities could adversely impact Mission Operations (Architecture, SW, IT, etc.)

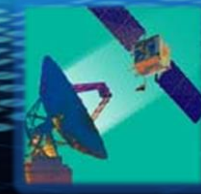
## ○ Preventing vulnerabilities

- Levying requirements from the top in policy, contracts, etc.
  - PPPs, A&A process, SW development, etc.
- During mission design/planning
  - Designing security in
  - Secure software development
  - Rigorous mission assurance (SW Assurance, IV&V, etc.)
  - Awareness, training, tooling
- Supply chain – know the parts you are building with...
  - Hardware
  - Software (i.e. COTS and Open Source)

## ○ Discovering vulnerabilities

- Once vulnerabilities are introduced into operation – then what?
  - Continuous monitoring
  - Vulnerability assessments
  - Penetration testing

# Requirements...




GSAW  
2016

- Examples of requirements government agencies may invoke
  - DOD
    - [Program Protection & System Security Engineering](#)
  - NASA
    - NPRs [2810](#), [7150.2B](#), [7120.5E](#), and the SW Assurance Standard/Handbook (under draft)
  - NIST [800-53](#)
    - Example control for SW:
      - [SA-11](#) Developer Security Testing and Evaluation
      - [RA-5](#) Vulnerability Scanning
  - European Space Agency (ESA) - (under draft)
    - ESSB-ST-E-008 - Secure Software Engineering Std
    - ESSB-HB-E-007 – Secure Software Engineering Handbook
- Other resources to help identify requirements
  - [Security Quality Requirements Engineering \(SQUARE\)](#)
  - [Microsoft Security Development Lifecycle](#)

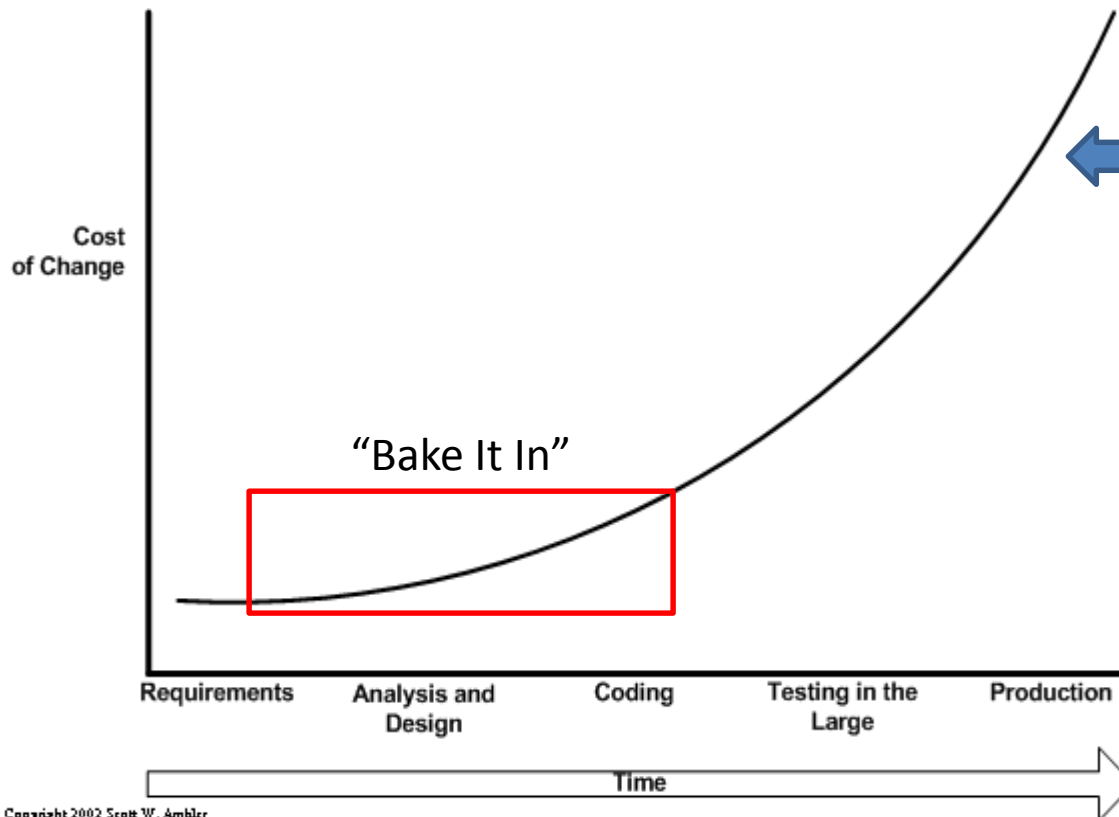
# Not Baking in Security

**Effects of Building Security in from the Get-Go**

- **Better Code Quality**
  - Testing up front reduces cost
  - Easier to meet schedule – less rework
- **Better Security Posture**
  - Increases Cybersecurity
- **Smaller Attack Surface**
  - Reduces the number of vulnerabilities
  - Reduces overall risk to system
- **Total Lifecycle Cost is Reduced**
  - It is estimated that bolting-on security post-development, costs roughly three times more than the cost of built-in security.



- Traditional cost of change curve depicts how discovering defects last impacts cost

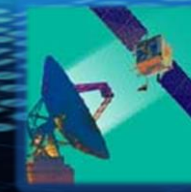


If Bug=Exploited  
damage could be  
more than monetary

**Loss of Mission Obj(s)**  
**Loss of Mission**  
**Loss of Life**

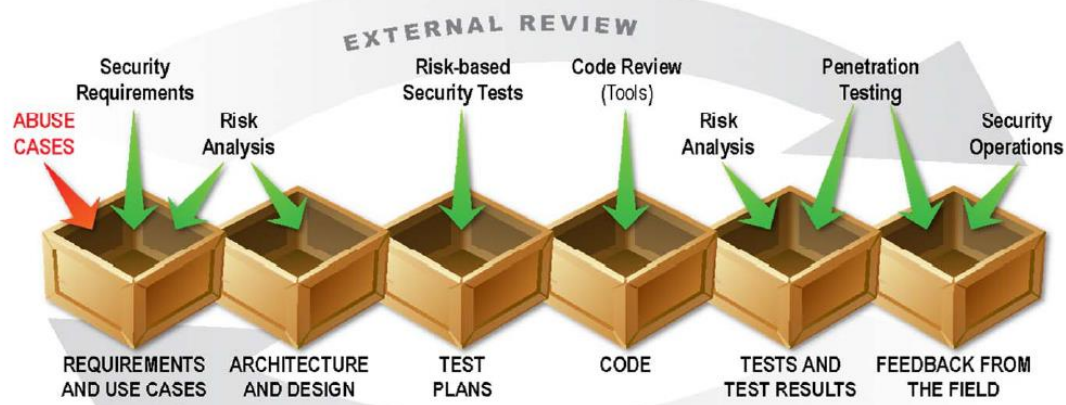
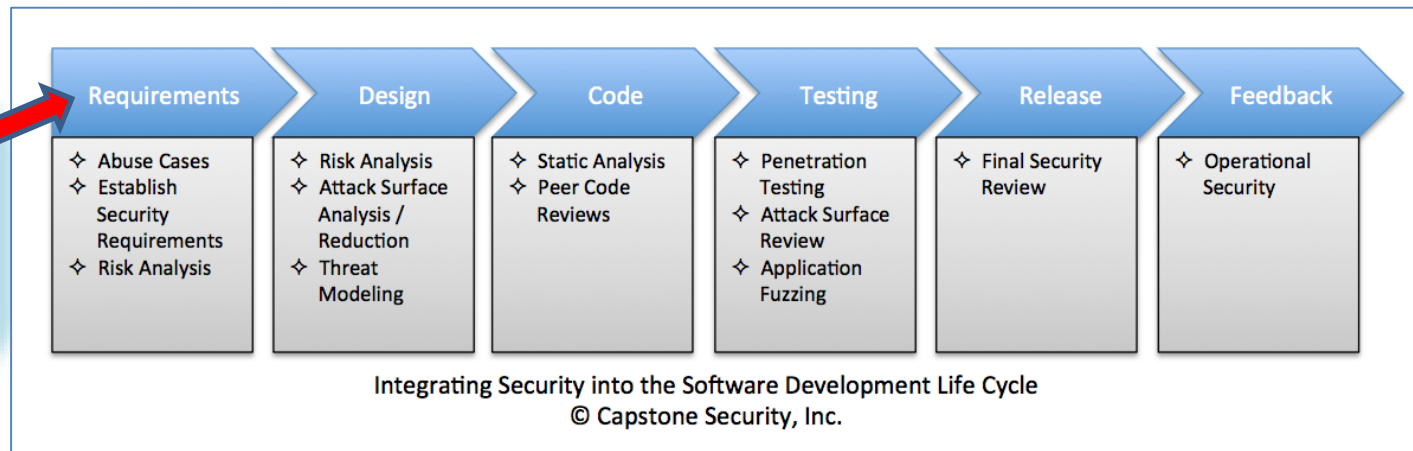


# Baking It In



GSAW  
2016

Secure software development begins where all software begins!



Secure software is an end-to-end development concept, not patchwork



# Secure Development

- Utilize [Best Practices](#)
  - List is from NASA's Secure Coding Portal
- Coding Standards (Ex. CERT [C](#), [C++](#) or [JAVA](#) Stds)
  - Ex: Don't use unsafe functions ([Flawfinder](#))
- Integrate tools into development environment
  - Code Analyzers (i.e. [Klockwork](#), [Fortify](#), [Flexelint](#), [CodeSonar](#), [Sonatype](#), [BlackDuck](#), etc.)
  - Great resource for identifying tools
    - [Report](#) | [Spreadsheet](#)

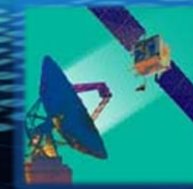
Applies to all SW development!!  
**Not** just ground systems

# Secure Development (cont.)

- Use information from DHS:
  - [Common Weakness Enumeration \(CWE\)](#), [Common Vulnerabilities and Exposures \(CVE\)](#), and [Common Attack Pattern Enumeration and Classification \(CAPEC\)](#)
  - Plan for Defense in Depth and not solely on protective perimeter
    - Historically developers depend/plan for Firewalls to protect vice designing in SW
    - Securing the development environment (i.e. prevent injecting of malicious code)
- Training
  - Free:
    - [FedVTE](#) Ex: Software Assurance Executive Course (SAE)
    - [SAFECode](#)
    - [Secure Coding and Standards Tutorial](#) (NASA Only)
  - Paid: (Ex: [Cigital](#), [Pluralsight](#))

# Not Baking in Security

## Ex: Actual Ground Software



GSAAW  
2016

- Insecure random number generator was used to generate passphrases that control access to VPNs and other network resources
- Could enable someone to monitor or interfere with a system and be undetectable
  - If this code was deployed with weak symmetric keys, the supposedly "secure" data-links between these devices would be vulnerable to a "man-in-the-middle" attack.
- There were several instances throughout the code
  - Klockwork discovered these during static code analysis

```

for (int i = 0; i < randomStringLength; i++)
{
    // randomly select a char for the random list
    int selChar = (int) (Math.random() * (randomList.length() - 1));

    if (allowSpecialCharacters) {...}

    // make sure we pick a non special character
    while (specialCharacters.indexOf(randomList.charAt(selChar)) != -1)
    {
        selChar = (int) (Math.random() * (randomList.length() - 1));
    }

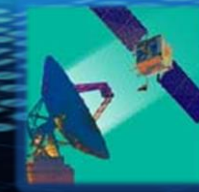
    // make sure we pick a special character from the list
    while (specialCharacters.indexOf(randomList.charAt(selChar)) < 0)
    {
        selChar = (int) (Math.random() * (randomList.length() - 1));
    }
}
    
```

Resource Describing Math.random:  
<http://franklinta.com/2014/08/31/predicting-the-next-math-random-in-java>



# Not Baking in Security

## Ex: Actual Ground Software



GSAAW  
2016

- Code calls a generic exception handler
  - Typically is done when a developer assumes they can only get known types of exceptions
  - However, depending on the source of the exception (input stream for example) someone can try to cause a different exception resulting in unpredictable behavior (i.e. DoS)
  - Also with a ground system, you want to fail-fast
    - Catching and ignoring fatal exceptions makes a program less robust since it will try to carry on as if nothing happened in the worst of conditions
    - Immediately report at its interface any failure
    - Don't pretend like nothing happened, because it's going to get worse
- Klockwork discovered these during static code analysis

```

* @throws Exception
*/
@PostConstruct
public void initializer() throws Exception
{

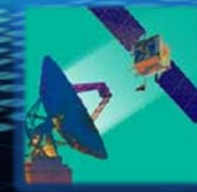
* @throws Exception
*/
@PostConstruct
public void initializer() throws Exception
{

* @throws Exception
*/
@PreRemove
public void onPreRemove() throws Exception
{

```

# Not Baking in Security

## Ex: Actual Ground Software



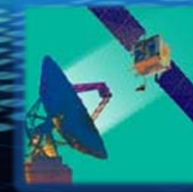
GSAW  
2016

- Lacking the appropriate code in a finally block (java exception handling)
  - Using something and then call a close, doesn't mean it will actually close if an exception is encountered either in the use or the close call
  - A finally block helps assure proper closure and deallocation
  - This can be for any type of resources (file, database, etc.)
- Resource leak could use up all resources, causing the system to become unresponsive after excessive or continued use, reducing dependability (i.e. DoS)
- Klockwork discovered these during static code analysis

```
public ICommunicationProcessorXXX connectUsingRemoteXXX()  
    throws GeneralSecurityException, NamingException  
{  
    try  
    {  
        InitialContext context = new InitialContext(properties);  
        m_ctmProcessorRemote =  
            (ICommunicationProcessorXXX) context.lookup(CommunicationProcessorXXXClientUtils.PROCESSOR_REMOTE_JNDI_NAME);  
    }  
    catch (NamingException e) {...}  
  
    return m_ctmProcessorRemote;  
}
```

# Low Hanging Fruit

## Unsafe Functions

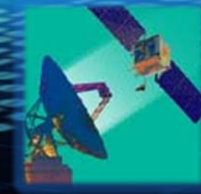


GSAW  
2016

- Stop using known unsafe functions and always do bounds checking if you are copying to a buffer
  - Even if you think you know what you are copying from and it's limited, defensive coding is best.
- Some samples of unsafe functions due to allowed writing with no regard to buffer size
 

memset	sprintf
memcpy	strncpy
strcat	_iota
strcmp	sscanf
strcpy	wcslon
strlen	
- Most of these are unsafe due to allowed writing with no regard to buffer size
  - strncpy, \_iota, sscanf, & wcslon have safer \_s varieties (ex. *\_iota\_s*) that require a buffer size to be specified
    - Resource: [Security Development Lifecycle \(SDL\) Banned Function Calls](#)
    - Resource: [Stack Overflow Post](#)
- Free tool to help find unsafe functions - [Flawfinder](#)

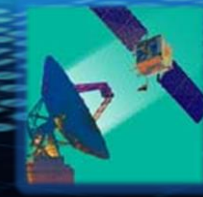
# Demo



GSAW  
2016

- Demo Flawfinder

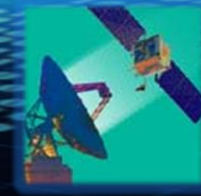
# Low Hanging Fruit CERT Rules



GSAW  
2016

- For legacy code:
  - MSC00-C. Compile cleanly at high warning levels
    - The process of fixing compiler warnings will probably quash some other vulnerabilities.
  - ERR33-C. Detect and handle standard library errors
    - Include any program functions that give some kind of error indication
      - If a function returns some special value on error, such as NULL, your calls to that function should always check its return value

# Low Hanging Fruit CERT Rules (cont.)



GSAW  
2016

- For new code
  - ERR00-C. Adopt and implement a consistent and comprehensive error-handling policy
    - This is where programs fail the most easily. They fail to check for errors because the developers don't know what to do if an unexpected error occurs.
  - MEM00-C. Allocate and free memory in the same module, at the same level of abstraction
    - A design issue, but not following it will get your code into hot water quickly.
  - MEM12-C. Consider using a goto chain when leaving a function on error when using and releasing resources
    - More specifically, make sure your code frees resources even if errors occur.
- For both new and existing code: execute static code analysis tools to determine weaknesses
  - Free ones are a good place to start; See slide 14 for commercial ones
    - [Cppcheck](#)
    - [Rosecheckers](#)
    - [Splint](#)
    - [Find Bugs](#)
    - [RATS](#)
    - [Flawfinder](#)
    - [SWAMP](#) ★

# Info from DHS



Services sponsored by Department of Homeland Security and managed by Mitre

## CWE:

- Serves as a common language for describing software security weaknesses in architecture, design, or code
- Provides a:
  - Standard measuring stick for software security tools targeting these weaknesses
  - Common baseline standard for weakness identification, mitigation, and prevention efforts
- Utilize CWE to better understand, identify, fix, and prevent weaknesses and vulnerabilities

## CVE:

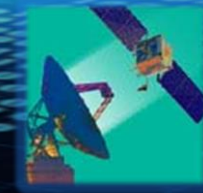
- Identifies publicly known information security vulnerabilities and assign them a CVE\_ID.
- Scored 1 to 10 on CVSS scale

## CAPEC:

- Community-developed list of common attack patterns
- Comprehensive schema and classification taxonomy
- International in scope

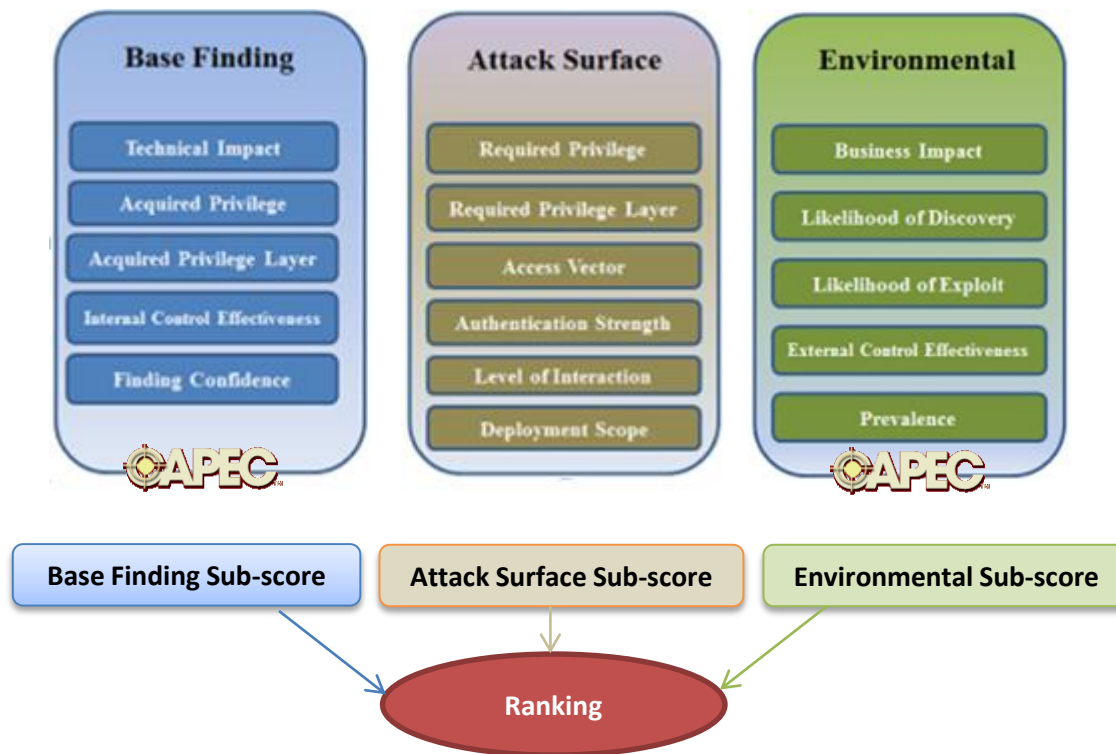
Taking into account attack pattern and any other factors to generate list of CWEs that are critical. Tools report findings in CVEs (known) and CWEs (potential) -> Identify then Fix!

# CWEs & Ground Systems



GSAW  
2016

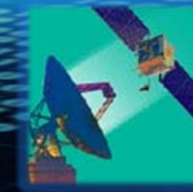
- For NASA, research & analysis has been performed by the IV&V Program to identify the Top 25 CWEs for Ground Systems
- The following categories are part of the formula for CWSS



Each factor in the category is assigned a value. These values are converted to associated weights and a category sub-score is calculated. The three sub-scores are multiplied together, which produces a Common Weakness Scoring System (CWSS) score. Higher the score, higher it ranks.



# Top 25 CWEs Ground Systems v2.0



GSAAW  
2016

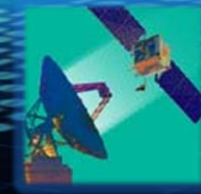
Rankings are ***currently*** under peer review.

Version 2.0 of Top 25 now includes Common Attack Patterns



Rank	CWE ID	CWE Title	Rank	CWE ID	CWE Title
1	<a href="#">312</a>	Cleartext Storage of Sensitive Information	13	<a href="#">403</a>	Exposure of File Descriptor to Unintended Control Sphere ('File Descriptor Leak')
2	<a href="#">88</a>	Argument Injection or Modification	14	<a href="#">314</a>	Cleartext Storage in the Registry
3	<a href="#">77</a>	Improper Neutralization of Special Elements used in a Command ('Command Injection')	15	<a href="#">835</a>	Loop with Unreachable Exit Condition ('Infinite Loop')
4	<a href="#">23</a>	Relative Path Traversal	16	<a href="#">833</a>	Deadlock
5	<a href="#">73</a>	External Control of File Name or Path	17	<a href="#">764</a>	Multiple Locks of a Critical Resource
6	<a href="#">798</a>	Use of Hard-coded Credentials	18	<a href="#">421</a>	Race Condition During Access to Alternate Channel
7	<a href="#">353</a>	Missing Support for Integrity Check	19	<a href="#">119</a>	Improper Restriction of Operations within the Bounds of a Memory Buffer
8	<a href="#">732</a>	Incorrect Permission Assignment for Critical Resource	20	<a href="#">318</a>	Cleartext Storage of Sensitive Information in Executable
9	<a href="#">22</a>	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	21	<a href="#">242</a>	Use of Inherently Dangerous Function
10	<a href="#">78</a>	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	22	<a href="#">497</a>	Exposure of System Data to an Unauthorized Control Sphere
11	<a href="#">290</a>	Authentication Bypass by Spoofing	23	<a href="#">772</a>	Missing Release of Resource after Effective Lifetime
12	<a href="#">20</a>	Improper Input Validation	24	<a href="#">681</a>	Incorrect Conversion between Numeric Types
			25	<a href="#">192</a>	Integer Coercion Error

# Demo



GSAW  
2016

- Demo Fortify w/ CWE Reporting

# Origin Analysis: Secure SW Supply Chain



Can You Answer:



- What open source components are being used?
- Where are these components being used?
- What is the “*Bill of Materials*” for my application and/or software we have purchased?
- Are we using open source components with known vulnerabilities?
- What are the security, licensing and quality risks for each component in my application?

- From Institute for Defense Analyses (IDA) [SOAR Report](#) – “*Origin analyzers are tools that analyze source code, bytecode, or binary code to determine their origins (e.g., pedigree and version).*”
- Origin Analysis can be used to reduce the software supply chain risk
  - Identifies **CVEs** that may be present in re-used open source libraries/code
  - Also identifies potentially licensing issues
- Examples of tools
  - [Sonatype](#)
    - Binary scanner; Works best on JAVA
  - [Black Duck HUB](#)
    - Provides binary and source tree scanning; Support C/C++ as well has JAVA
  - [OWASP Dependency Check](#)
    - Currently Java, .NET, Ruby, Node.js, and Python projects are supported; additionally, limited support for C/C++ projects is available for projects using CMake or autoconf.



# Examples from Ground Systems

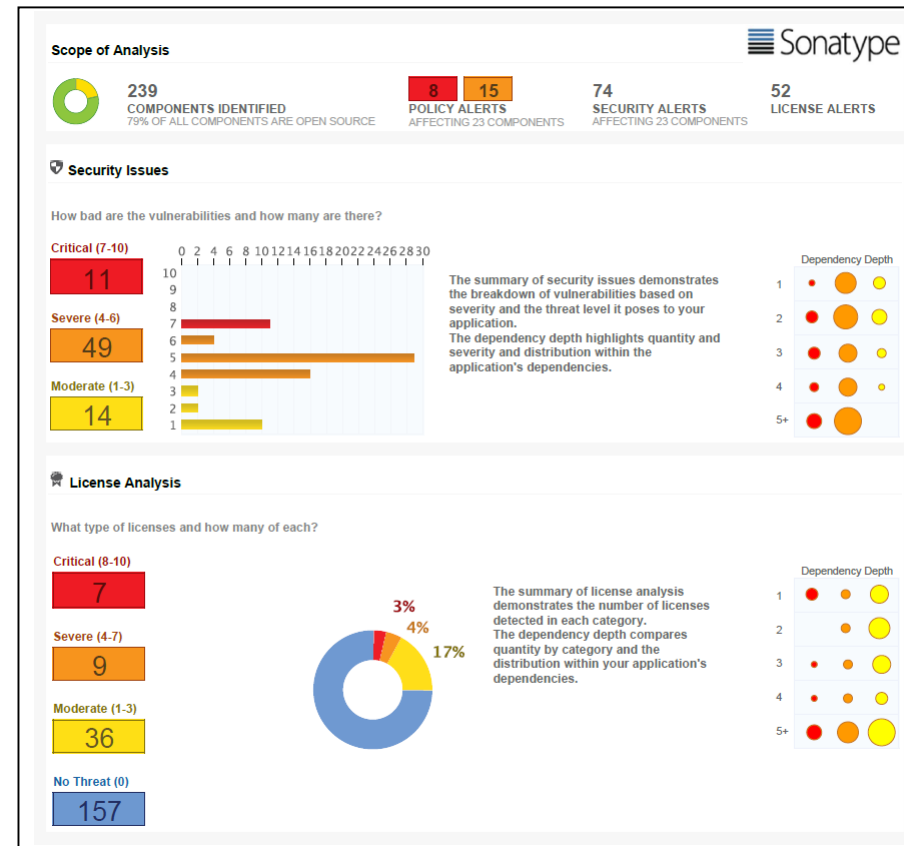


Can You Answer:

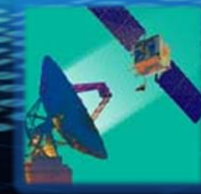


- What open source components are being used?
- Where are these components being used?
- What is the "Bill of Materials" for my application and/or software we have purchased?
- Are we using open source components with known vulnerabilities?
- What are the security, licensing and quality risks for each component in my application?

Vulnerability	Affected File	Mitigation
CVE-2014-0003: Allows remote attackers to execute arbitrary Java methods via a crafted message.	camel-core-1.5.4.0-fuse.jar	Upgrade Jar file to 2.11.4 or newer
CVE-2009-4611: Allow remote attackers to modify a window's title, or possibly execute arbitrary commands or overwrite files, via an HTTP request	jetty-6.1.14.jar; jetty-util-6.1.14.jar	Upgrade Jar file to 6.1.25 or newer
CVE-2011-2730: Allows remote attackers to obtain sensitive information	spring-web-2.5.5.jar	Upgrade Jar file to 3.2.9 or newer
CVE-2014-0107: Allows remote attackers to bypass expected restrictions and load arbitrary classes or access external resources via a crafted messages	xsltc.jar; xalan.jar	Upgrade Jar file to 2.7.2 or newer
CVE-2013-4002: Allows remote attackers to affect availability via unknown vectors.	Xerces2.6.2_xercesImpl.jar; xercesImpl.jar	N/A (new versions exist but also contain vulnerabilities). Implement host based restrictions (i.e., IP tables, file integrity detection, Host based IDS)
CVE-2010-1244: Allows remote attackers to hijack the authentication of unspecified victims	activemq-web-5.2.0.2-fuse.jar	Upgrade Jar file to 5.9.0 or newer



# Demo

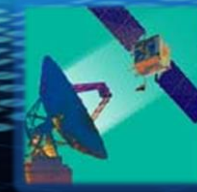


GSAW  
2016

- Demonstrate Sonatype

# Example: Heartbleed

## What is it?

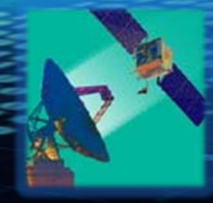


GSAW  
2016

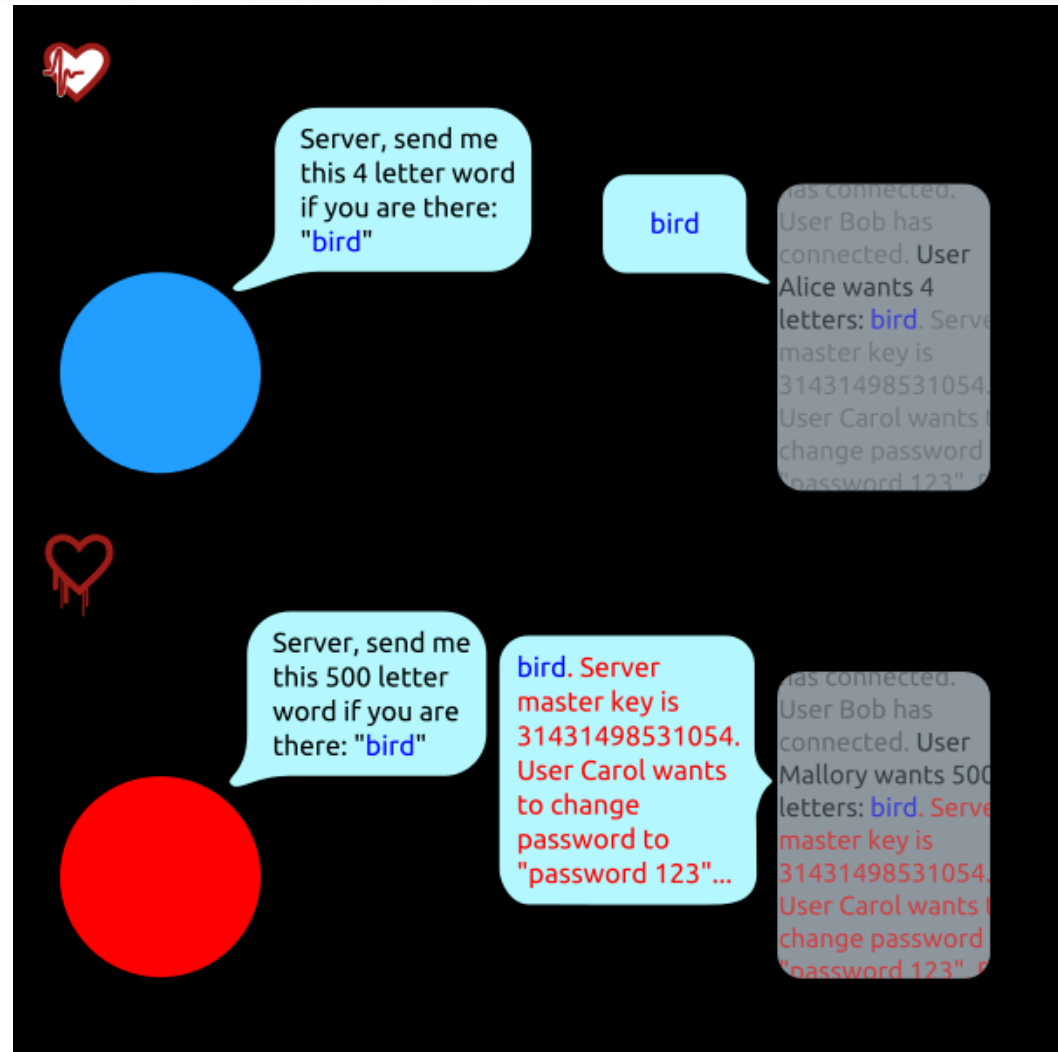
- **OpenSSL** is an [open-source](#) implementation of the [SSL and TLS](#) protocols. The core [library](#), written in the [C programming language](#), implements the basic [cryptographic](#) functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.
- Its free!
- **As of 2014 two thirds of all web servers use it.**
- From Heart**beat** to Heart**bleed**
  - Defect could be used to reveal up to 64 [kilobytes](#) of the application's memory with every [heartbeat](#)
  - The affected versions of OpenSSL allocate a memory buffer for the message to be returned based on the length field in the requesting message, without regard to the size of actual payload in that message.

# Example: Heartbleed

## How does it work?

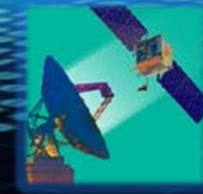


GSAW  
2016



# Example: Heartbleed

## How do I find it?



GSAW  
2016

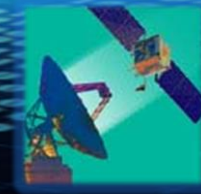
- One way to find it would be to execute Origin Analyzer across source tree that includes your open source code as well



Identifier ^	Published ⇅	Base Score ⇅	Exploitability ⇅	Impact ⇅
> <span>VulnDB</span> 107729	Jun 5, 2014	6.8	8.6	6.4
> <span>VulnDB</span> 113251	Oct 14, 2014	2.6	4.9	2.9
> <span>NVD</span> CVE-2014-0160	Oct 21, 2015	5	10	2.9
> <span>NVD</span> CVE-2015-1788	Jun 15, 2015	4.3	8.6	2.9
> <span>NVD</span> CVE-2015-1789	Jun 15, 2015	4.3	8.6	2.9
> <span>NVD</span> CVE-2015-1790	Jun 15, 2015	5	10	2.9
> <span>NVD</span> CVE-2015-1791	Jun 15, 2015	6.8	8.6	6.4
> <span>NVD</span> CVE-2015-1792	Jun 15, 2015	5	10	2.9
> <span>NVD</span> CVE-2015-4000	Jul 22, 2015	4.3	8.6	2.9



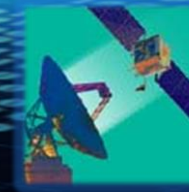
# Demo



GSAW  
2016

- Demonstrate BlackDuck Hub

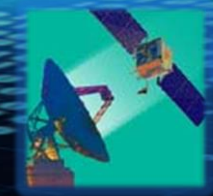
# Finding Vulnerabilities in SW



GSAW  
2016

- It is difficult to determine what types of tools and techniques exist for analyzing software, and where their use is appropriate.
  - Institute for Defense Analyses (IDA) created the SOAR [report](#) and [matrix](#) to assist
    - NASA has slight modified version to include tool names <-> contact [brandon.t.bailey@nasa.gov](mailto:brandon.t.bailey@nasa.gov)
  - Ideally developers will institute static source code and binary analysis to assist in identifying weaknesses
    - Development activities should include analyzing source code before it is compiled to detect coding errors, non-secure coding constructs, and other indicators of security vulnerabilities or weaknesses that are detectable at the source code level
  - Developers should perform software evaluations throughout the software development lifecycle to address potential security vulnerabilities early in the process
  - Use research from [NSA's CAS](#) and [Institute for Defense Analyses](#) to establish a blend of tools that will provide the most value





**GSAW  
2016**

# Demo

- Demo SOAR Spreadsheet

SOAR\_Matrix(2).xlsx - Microsoft Excel

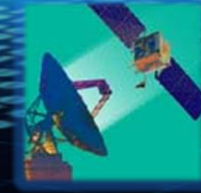
File Home Developer Insert Page Layout Formulas Data Review View Acrobat Livelink

Clipboard Font Alignment Number Styles Cells Editing

C1 Source code quality analyzer

	C	D	E
1	Source code quality analyzer	Source code weakness analyzer	Context-configured source code weakness analyz
2	Supplier: Coverity Product Name: Coverity Quality Advisor Languages Supported: C, C++, Java, C# in progress for advisor URL: <a href="http://www.coverity.com/">http://www.coverity.com/</a>	Supplier: Coverity Product Name: Coverity Security Advisor Languages Supported: C, C++, Java, C# in progress for advisor URL: <a href="http://www.coverity.com/">http://www.coverity.com/</a>	N/A  Supplier: IBM Product Name: Rational Asset Analyst URL: <a href="http://www-03.ibm.com/software/proc">http://www-03.ibm.com/software/proc</a>
3	Supplier: Klocwork Product Name: Truepath® Refactoring® Languages Supported: Truepath: C/C++, Java and C# Refactoring: C/C++ URL: <a href="http://www.klocwork.com/products/insight/index.php">http://www.klocwork.com/products/insight/index.php</a>	Supplier: Grammatech Product Name: CodeSonar Languages Supported: C, C++ URL: <a href="http://www.grammatech.com/products/codesonar/overview.html">http://www.grammatech.com/products/codesonar/overview.html</a>	Supplier: Micro Focus Product Name: Enterprise Analyzer URL: <a href="http://www.microfocus.com/products/">http://www.microfocus.com/products/</a>
4	Supplier: Sonar Product Name: SonarSource Languages Supported: C, C++, Java, C#, VB.net, PL/I, COBOL, PHP, Python, VB6, Natural, Javascript, XML, etc. ( caveat: when leveraging their open source orchestration engine, only some languages are open source and licensed as such) Commercial license support includes a broader set of languages. URL: <a href="http://www.sonarsource.org/">http://www.sonarsource.org/</a>	Supplier: HP Fortify Product Name: Static Code Analyzer (SCA) Languages Supported: C, C++, C#, .NET languages, COBOL, Java, JavaScript/ AJAX, PHP, PL/SQL, Python, T-SQL URL: <a href="http://www8.hp.com/us/en/software/solutions/software.htm?compURI=1338812#UXEF-MX-NM">http://www8.hp.com/us/en/software/solutions/software.htm?compURI=1338812#UXEF-MX-NM</a>	
5	Supplier: CAST Product Name: Application Intelligence Platform Languages Supported: Includes .NET, Java, COBOL URL: <a href="http://www.castsoftware.com/products/the-applicationintelligence-platform">http://www.castsoftware.com/products/the-applicationintelligence-platform</a>	Supplier: Checkmarx Product Name: Checkmarx Languages Supported: Java, C#, .NET, PHP, C, C++, Visual Basic 6.0, VB.NET, Flash, APEX, Ruby, JavaScript, ASP, Android, Objective C, Perl URL: <a href="http://www.checkmarx.com/">http://www.checkmarx.com/</a>	
6		Supplier: IBM (formerly Ounce Labs) Product Name: AppScan Source Languages Supported: C, C++, Java, VB.NET, C# URL: <a href="http://www-01.ibm.com/software/awdtools/appscan/">http://www-01.ibm.com/software/awdtools/appscan/</a>	
7			

# Break

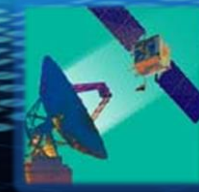


GSAW  
2016

- **Tutorial E Part 1:**
  - Section 1: Cyber Threat – Who, What and Why
  - Section 2: Defense-In-Depth
  - Section 3: Secure Software Engineering Steps
  - Section 4: Errors, Weaknesses and Exploits
  - Section 5: Threat Modeling
  - Section 6: Testing
  - Section 7: Resources
- **Tutorial E Part 2:**
  - Section 1: Ground Systems Overview
  - Section 2: Secure Software Development
  - Section 3: Defense in Depth for Ground Systems
  - Section 4: What Now?




# Compounding Problem




GSAW  
2016

- Work with Network Engineers to implement enclaves/network zoning and/or encryption
  - Build a “zero trust” architecture
    - Vulnerabilities injected by Mission X may affect Mission Y
  - Network layer encryption
- Understand and eliminate pivot points
  - From networking perspective, software security perspective, host level security
- Increase attack depth or eliminate all together

Utilize tools like RedSeal Networks, Skybox, etc. to understand network topology and threat exposures


 Changing the Current Mindset

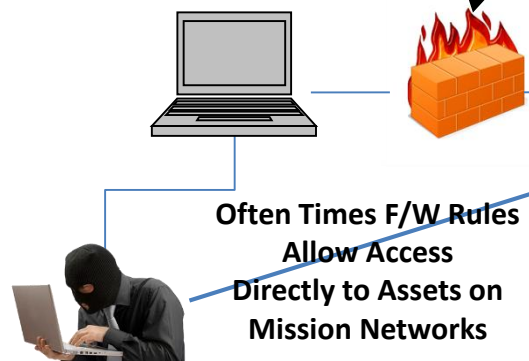
- Cybersecurity is not just about firewalls
  - Software Teams must now bear the brunt of security
- Essential that security is factored in
  - Can't be an afterthought
- 'Hardening' the O/S complements secure coding
  - It does not replace it



# Example SW Impacting Mission

Developers can't assume protection from Firewall. Need "Defense in Depth". Can't assume if knocking on door, that they are supposed to be there.

Signs onto Rogue Wifi, Click Phishing Link, Etc.  
**Then Signs onto VPN**



Mission Asset



Exploits Custom S/W

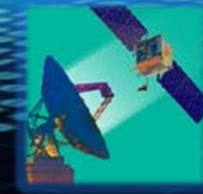
**Establishes persistent foothold on Mission Asset**

Launch Attacks (DoS, Brute Force, Extract Data, etc.)

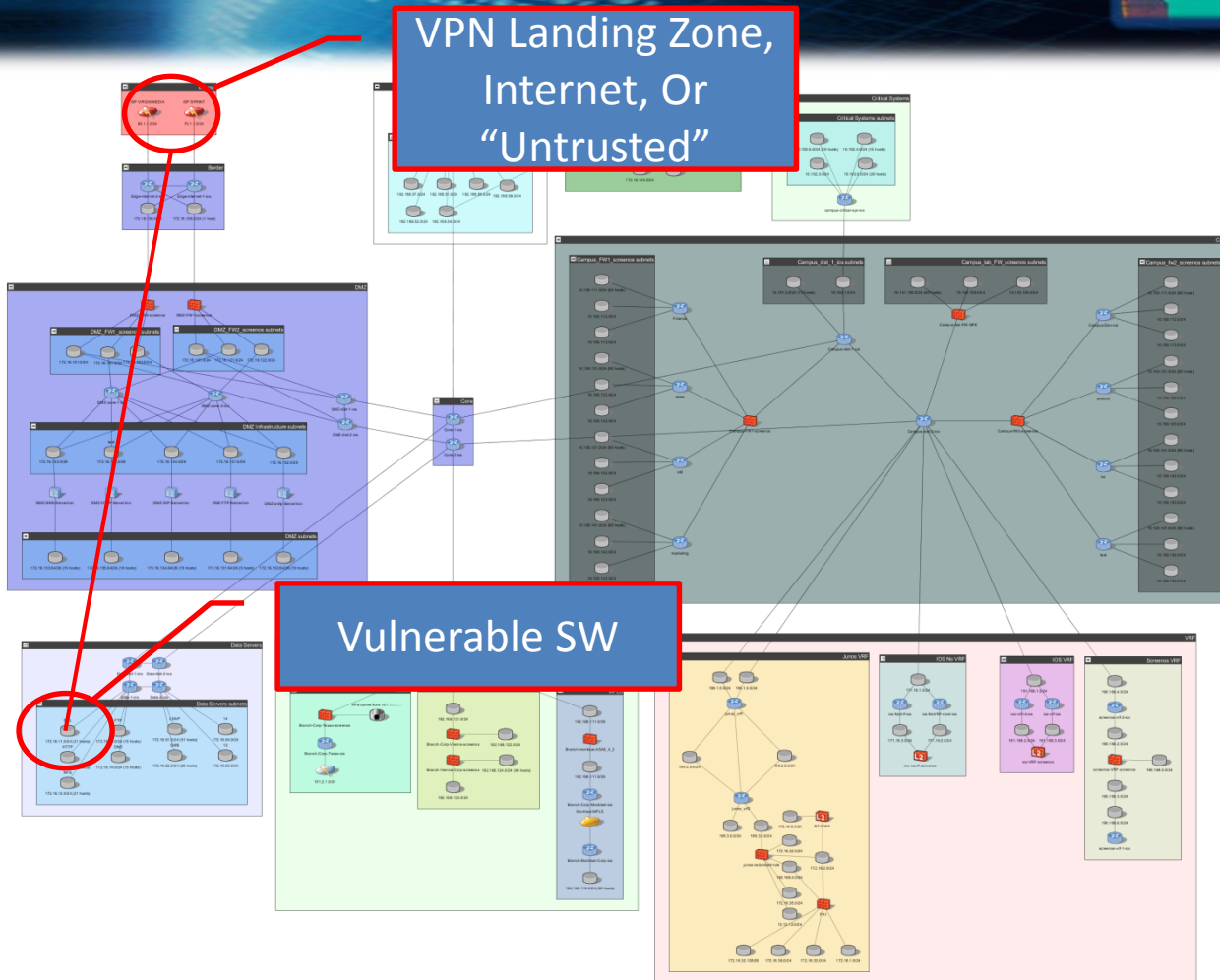


This example will depict how vulnerable software within a network can potentially impact critical mission assets

# Sample Exposure



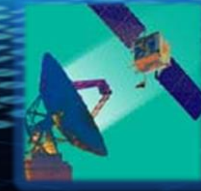
GSAW  
2016



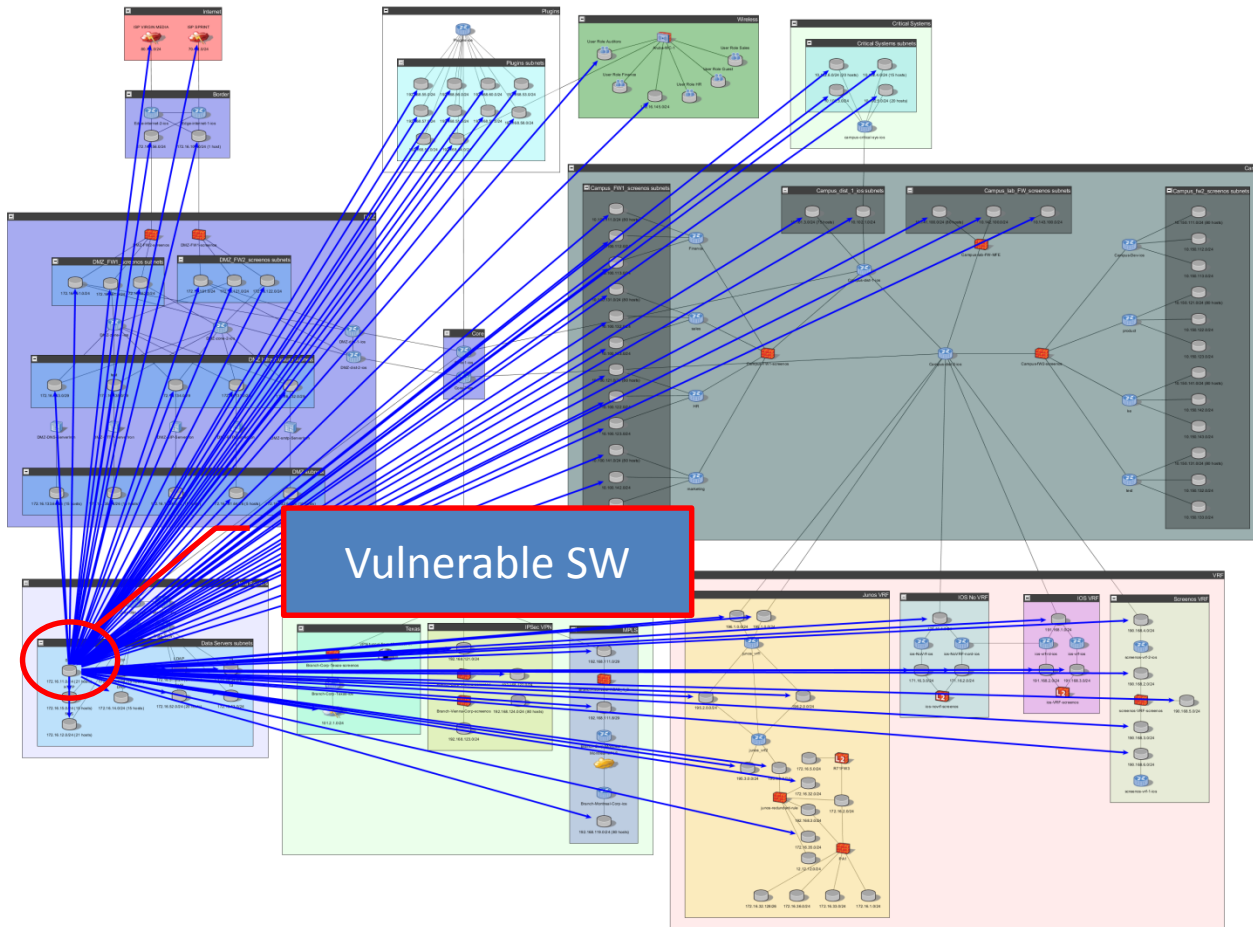
**Demonstrates that a pathway exists from the VPN Landing Zone, Internet, Or Untrusted to a vulnerable piece of software**



# Sample Exposure

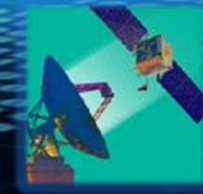


GSAW  
2016

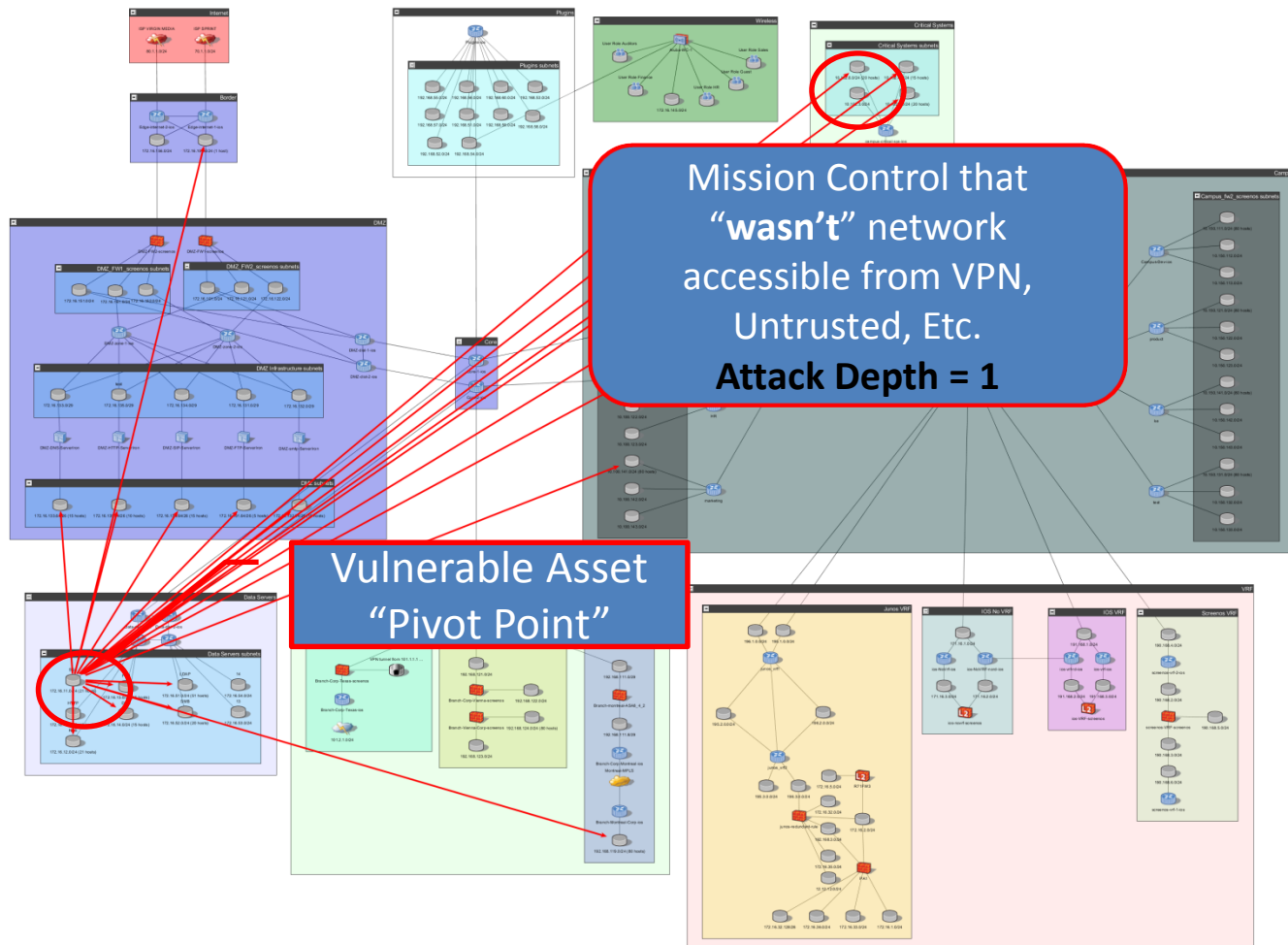


Demonstrates all outbound access paths (**Pivoting**) from the vulnerable asset

# Sample Exposure

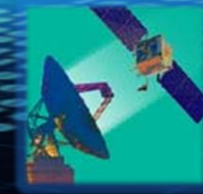


GSAW  
2016



Demonstrates potential vulnerabilities that could be exploited from this server

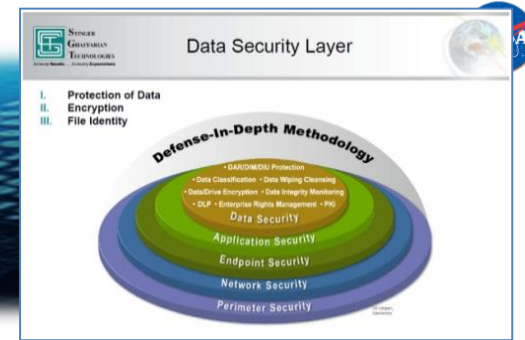
# Defense in Depth (DiD) (cont.)



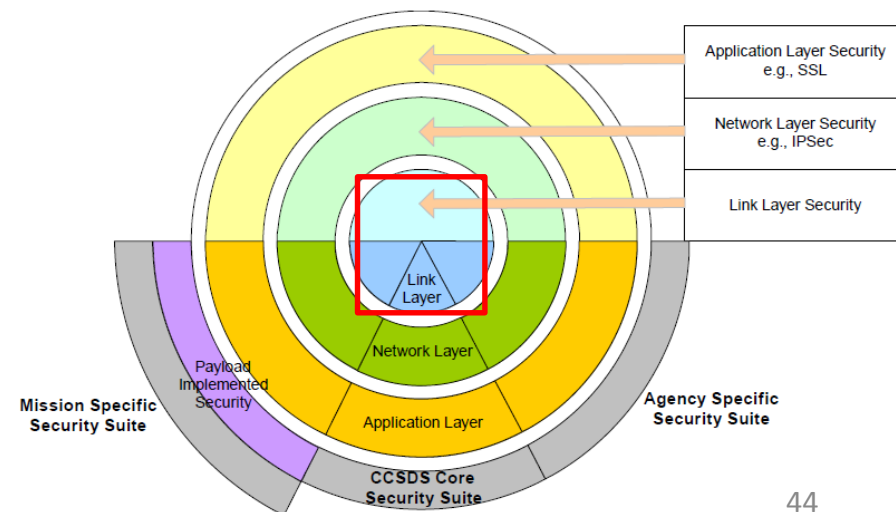
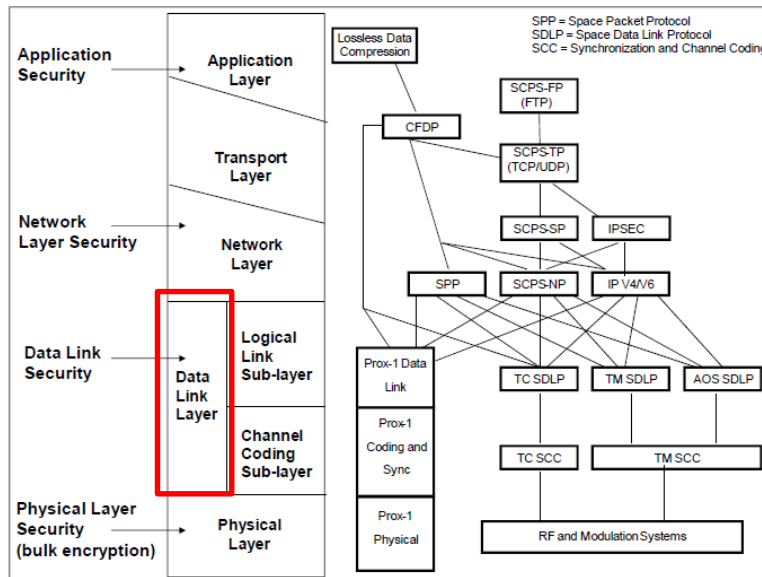
GSAW  
2016

- Example depicted how vulnerable software in a network that **doesn't** employ zero-trust architecture can expose mission assets
- Network encryption is another layer of defense that provides protection
  - Protocol machinery below the Network Layer including the Network Layer protocol (e.g., IP, Data Link, Physical) is exposed
- Data Link Layer services can provide additional protection
  - Upper-layer protocols become opaque
  - Data Link Layer security may be useful when a threat assessment indicates a heightened risk of exposure of the underlying protocols across an RF link or when traffic analysis is a concern.
  - Data link layer is **usually** under complete control of the Mission and vulnerabilities within the shared architecture can be mitigated by added this layer of defense

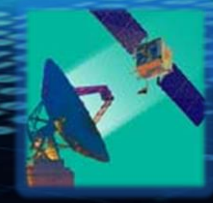
# Data Link Layer



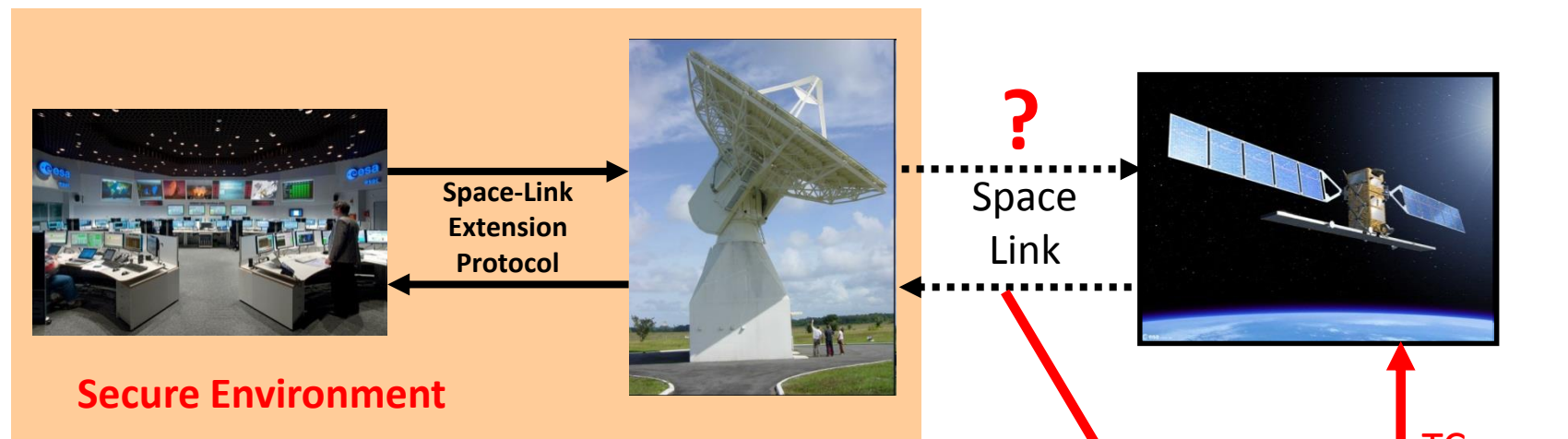
- DiD: Network layer unsecure/non-encrypted or frame data is sensitive post network layer (i.e. RF)
- Secure at data link layer
  - Data Link Layer security services may be able to provide all of the mission’s security needs, which could include authentication, integrity, and confidentiality, **but only on the specific link over which the security services are provided.**



# Securing Space Data Link: Space-Link Threat Analysis

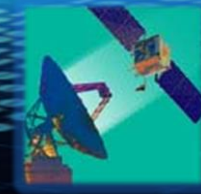


GSAW  
2016



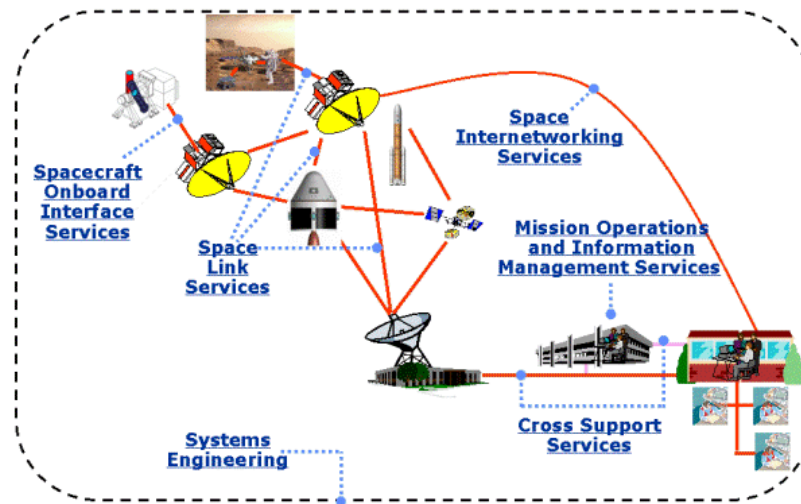
- Without protection:
  - Spacecraft are vulnerable to spoofing attacks from rogue ground stations
  - Telemetry could be received and processed by rogue ground stations
  - ...

# CCSDS Space-Link Security

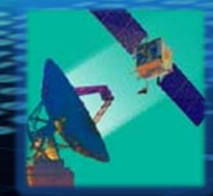


GSAW  
2016

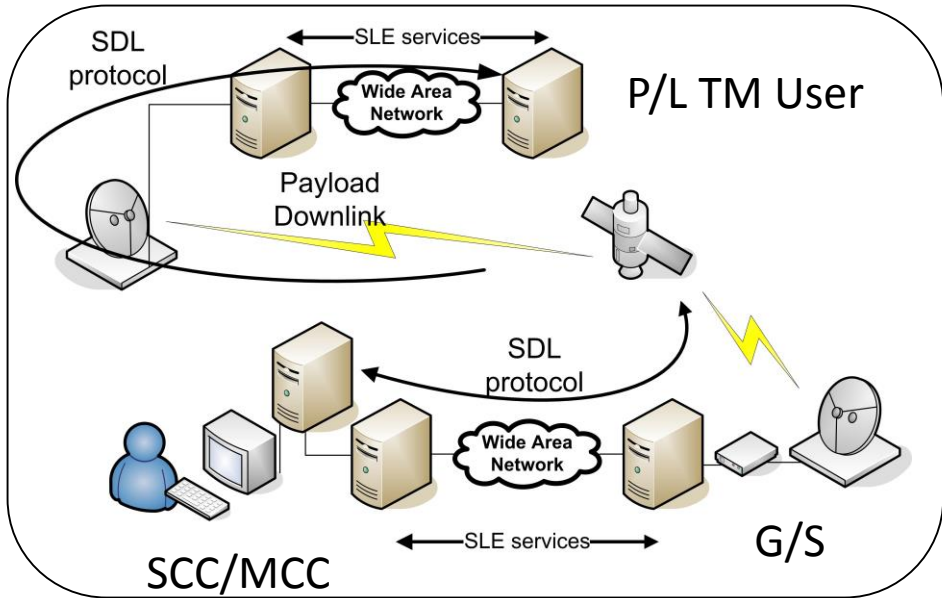
- CCSDS realized that a standardized protocol to integrate security into space missions with a simple network topology could be proposed at the data-link layer
  - Space Data-Link Layer Security Protocol (SDLS)
- Space Data Link Security (SDLS) [Blue Book](#) Published September 2015
  - *Protections implemented via software!*



# SDLS Capabilities



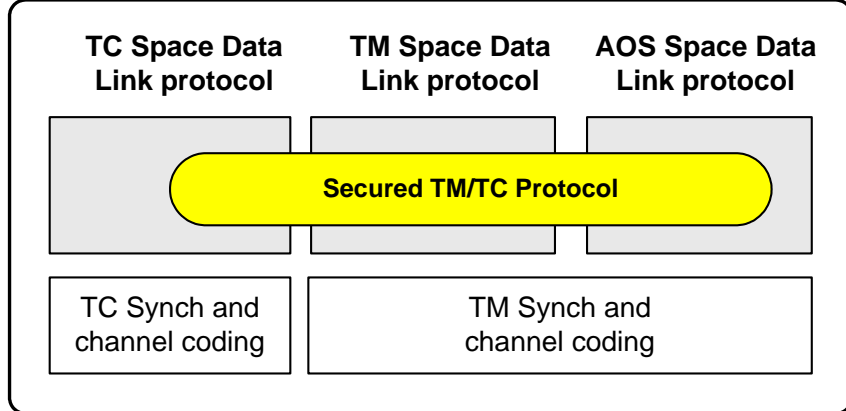
GSAW  
2016



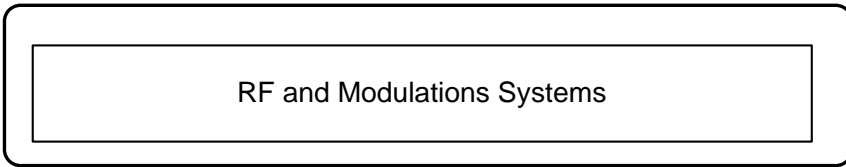
## Network Layer

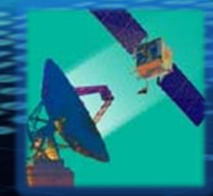


## Data Link Layer



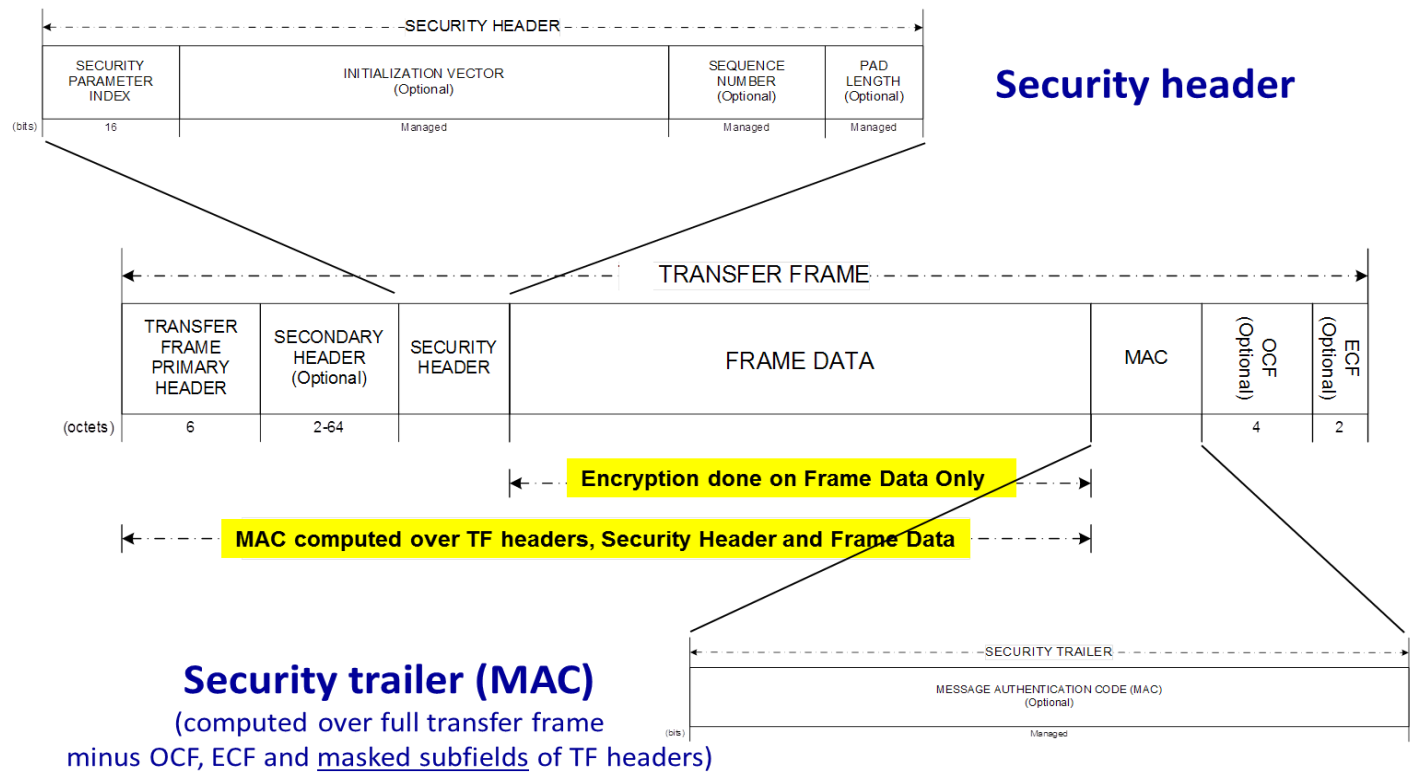
## Physical Layer





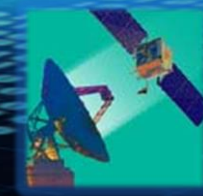
# SDLS in a Nutshell

- SDLS supports two main security services
  1. Authentication only – providing authentication and integrity
  2. Authenticated encryption – Adding confidentiality



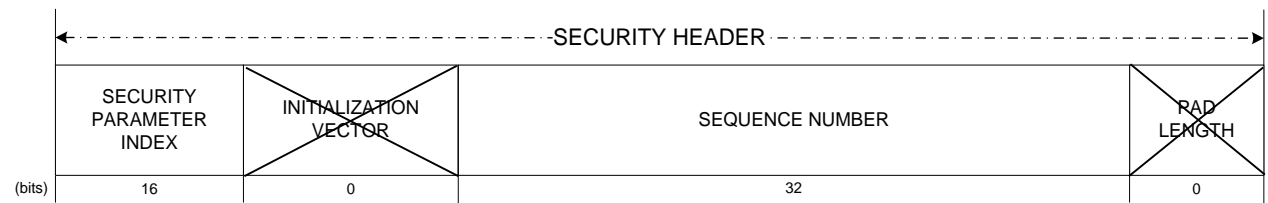
**Security trailer (MAC)**  
 (computed over full transfer frame  
 minus OCF, ECF and masked subfields of TF headers)



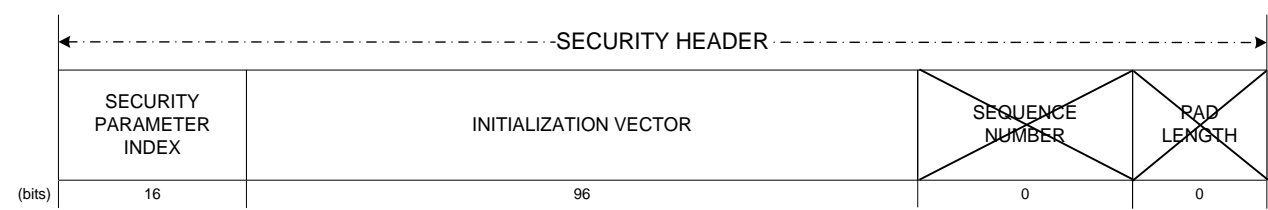


# SDLS Baseline Mode

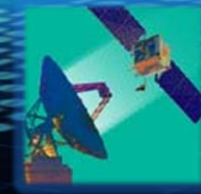
- To promote multi-mission implementations and interoperability:
  - 3 recommended profiles have been defined covering security requirements of most missions w.r.t. TC, TM and AOS links
- Baseline mode for TC
  - **Authentication only**, using AES/CMAC, 128-bit key, 32-bit ARC, 128-bit MAC (22-octet overhead (8%))



- Baseline mode for TM and AOS
  - **Authenticated encryption**, using AES/GCM, 128-bit key, 96-bit initialization vector, no seq. # needed, 128-bit MAC (30-octet overhead (2.5%))



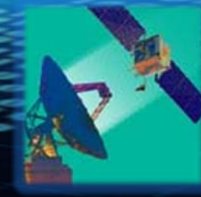
# What To Do Now?



GSAW  
2016

- In mission environments (esp. mission with extended ops) you may not be able to patch code; therefore for vulnerable code that can't be fixed the "host" owner can
  - Harden the servers and hosts by disabling all ports, protocols and services that are not explicitly required for operations
  - Install file integrity software (i.e., TripWire, Aide) to alert to changes made to the file system
  - Install and finely tune a host-based IDS that will alert to any anomalous traffic
  - Utilize IP tables/IPFilters to limit data flow to specific IP addresses, ports, protocols and services

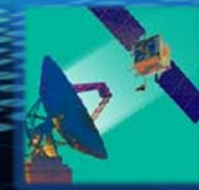
# What To Do Now?



GSAW  
2016

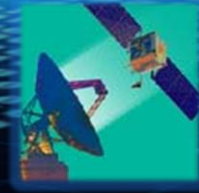
- To prevent future deployments of vulnerable code
  - Participate in secure code training
    - Educate developers, PMs, Authorizing Officials, Security Personnel (ISSO, ISO, etc.) on the importance of eliminating vulnerable code from architecture
  - Pick the low hanging fruit (see slides 20/21)
  - Utilize Best Practices and Secure Coding Standards
    - Ex: [Best Practices](#) from NASA's Secure Coding Portal
    - Ex: Coding Standards (Ex. CERT [C](#), [C++](#) or [JAVA](#) Stds)
  - Institute static source code and binary analysis to assist in identifying weaknesses - [https://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)
    - Apply the tools within the development activity (i.e., as an add-on to the developer's Integrated Development Environment (IDE)) as well as in the Independent Test and Evaluation (IT&E) activities
    - Top 25 CWEs for Ground Systems
      - Use NASA's or create you own based on your mission and threats

# Help Assure Ground System Security



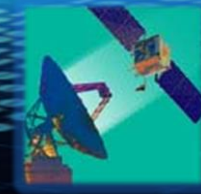
GSAW  
2016

- Expand independent assessments of ground systems for vulnerabilities using latest technologies
  - Perform “Red” and “Blue” Teams across the entire ground system – End to End
- Promote integration of security early in acquisition and development life-cycles
- Integrate cyber security activities to dependably ***Know, Prevent, Detect, Respond, and Recover***



# Backup Slides

# Links



GSAW  
2016

Slide 5/6:

- [major space agencies of the world](http://public.ccsds.org/participation/member_agencies.aspx) - [http://public.ccsds.org/participation/member\\_agencies.aspx](http://public.ccsds.org/participation/member_agencies.aspx)
- [multi-national forum](http://cwe.ccsds.org/) - <http://cwe.ccsds.org/>

Slide 11:

- [Program Protection & System Security Engineering](http://www.acq.osd.mil/se/initiatives/init_pp-sse.html) - [http://www.acq.osd.mil/se/initiatives/init\\_pp-sse.html](http://www.acq.osd.mil/se/initiatives/init_pp-sse.html)
- [2810](http://nodis3.gsfc.nasa.gov/npg_img/N_PR_2810_001A_/N_PR_2810_001A_.pdf) - [http://nodis3.gsfc.nasa.gov/npg\\_img/N\\_PR\\_2810\\_001A\\_/N\\_PR\\_2810\\_001A\\_.pdf](http://nodis3.gsfc.nasa.gov/npg_img/N_PR_2810_001A_/N_PR_2810_001A_.pdf)
- [7150.2B](http://nodis3.gsfc.nasa.gov/npg_img/N_PR_7150_002B_/N_PR_7150_002B_.pdf) - [http://nodis3.gsfc.nasa.gov/npg\\_img/N\\_PR\\_7150\\_002B\\_/N\\_PR\\_7150\\_002B\\_.pdf](http://nodis3.gsfc.nasa.gov/npg_img/N_PR_7150_002B_/N_PR_7150_002B_.pdf)
- [7120.5E](https://foiaelibrary.gsfc.nasa.gov/_assets/doclibBidder/tech_docs/1.N_PR_7120_005E_.pdf) - [https://foiaelibrary.gsfc.nasa.gov/\\_assets/doclibBidder/tech\\_docs/1.N\\_PR\\_7120\\_005E\\_.pdf](https://foiaelibrary.gsfc.nasa.gov/_assets/doclibBidder/tech_docs/1.N_PR_7120_005E_.pdf)
- [800-53](http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf) - <http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf>
- [SA-11](https://web.nvd.nist.gov/view/800-53/Rev4/control?controlName=SA-11) - <https://web.nvd.nist.gov/view/800-53/Rev4/control?controlName=SA-11>
- [RA-5](https://web.nvd.nist.gov/view/800-53/Rev4/control?controlName=RA-5) - <https://web.nvd.nist.gov/view/800-53/Rev4/control?controlName=RA-5>
- [Security Quality Requirements Engineering \(SQUARE\)](http://www.cert.org/cybersecurity-engineering/products-services/square.cfm) - <http://www.cert.org/cybersecurity-engineering/products-services/square.cfm>
- [Microsoft Security Development Lifecycle](https://www.microsoft.com/en-us/sdl/) - <https://www.microsoft.com/en-us/sdl/>

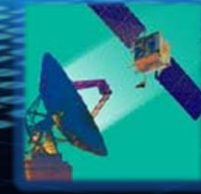
Slide 14:

- [C](https://www.securecoding.cert.org/confluence/display/c/SEI+CERT+C+Coding+Standard) - <https://www.securecoding.cert.org/confluence/display/c/SEI+CERT+C+Coding+Standard>
- [C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637) - <https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637>
- [JAVA](https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java) - <https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java>
- [Klockwork](http://www.klocwork.com/products/insight) - <http://www.klocwork.com/products/insight>
- [Fortify](http://www8.hp.com/us/en/software-solutions/software-security/) - <http://www8.hp.com/us/en/software-solutions/software-security/>
- [Flexelint](http://www.gimpel.com/html/flex.htm) - <http://www.gimpel.com/html/flex.htm>
- [CodeSonar](http://www.grammatech.com/codesonar) - <http://www.grammatech.com/codesonar>
- [Sonatype](http://www.sonatype.com/) - <http://www.sonatype.com/>
- [BlackDuck](https://www.blackduckssoftware.com/products/black-duck-hub) - <https://www.blackduckssoftware.com/products/black-duck-hub>
- [Report](http://www.acq.osd.mil/se/docs/P-5061-software-soar-mobility-Final-Full-Doc-20140716.pdf) - <http://www.acq.osd.mil/se/docs/P-5061-software-soar-mobility-Final-Full-Doc-20140716.pdf>
- [Spreadsheet](http://www.acq.osd.mil/se/docs/P-5061-AppendixE-soar-sw-matrix-v9-mobility.xlsx) - <http://www.acq.osd.mil/se/docs/P-5061-AppendixE-soar-sw-matrix-v9-mobility.xlsx>

Slide 15:

- [Common Weakness Enumeration \(CWE\)](https://cwe.mitre.org/) - <https://cwe.mitre.org/>
- [Common Vulnerabilities and Exposures \(CVE\)](https://cve.mitre.org/) - <https://cve.mitre.org/>
- [Common Attack Pattern Enumeration and Classification \(CAPEC\)](https://capec.mitre.org/) - <https://capec.mitre.org/>
- [FedVTE](https://fedvte.usalearning.gov/) - <https://fedvte.usalearning.gov/>
- [SAFECode](https://training.safecode.org/) - <https://training.safecode.org/>
- [Secure Coding and Standards Tutorial](https://www.safaribooksonline.com/self-registration/nasatutorials/) - <https://www.safaribooksonline.com/self-registration/nasatutorials/>
- [Cigital](https://www.cigital.com/services/training/elearning/) - <https://www.cigital.com/services/training/elearning/>
- [Pluralsight](https://www.pluralsight.com/search?q=security&categories=course) - <https://www.pluralsight.com/search?q=security&categories=course>

# Links (cont.)



GSAW  
2016

Slide 16:

- <http://franklinta.com/2014/08/31/predicting-the-next-math-random-in-java>

Slide 19:

- [Security Development Lifecycle \(SDL\) Banned Function Calls](https://msdn.microsoft.com/en-us/library/bb288454.aspx) - <https://msdn.microsoft.com/en-us/library/bb288454.aspx>
- [Stack Overflow Post](http://stackoverflow.com/questions/6747995/a-complete-list-of-unsafe-string-handling-functions-and-their-safer-replacements) - <http://stackoverflow.com/questions/6747995/a-complete-list-of-unsafe-string-handling-functions-and-their-safer-replacements>
- [Flawfinder](http://www.dwheeler.com/flawfinder/) - <http://www.dwheeler.com/flawfinder/>

Slide 22:

- [Cppcheck](http://cppcheck.sourceforge.net/) - <http://cppcheck.sourceforge.net/>
- [Rosecheckers](http://sourceforge.net/projects/rosecheckers/) - <http://sourceforge.net/projects/rosecheckers/>
- [Splint](http://www.splint.org) - <http://www.splint.org>
- [RATS](https://code.google.com/p/rough-auditing-tool-for-security) - <https://code.google.com/p/rough-auditing-tool-for-security>
- [Flawfinder](http://www.dwheeler.com/flawfinder) - <http://www.dwheeler.com/flawfinder>
- [SWAMP](https://continuousassurance.org) - <https://continuousassurance.org>
- [Find Bugs](http://findbugs.sourceforge.net/) - <http://findbugs.sourceforge.net/>

Slide 23:

- [CWE](https://cwe.mitre.org/) - <https://cwe.mitre.org/>
- [CVE](https://cve.mitre.org/) - <https://cve.mitre.org/>
- [CAPEC](https://capec.mitre.org/) - <https://capec.mitre.org/>

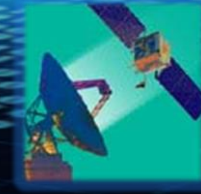
Slide 27:

- [SOAR Report](http://www.acq.osd.mil/se/docs/P-5061-software-soar-mobility-Final-Full-Doc-20140716.pdf) - <http://www.acq.osd.mil/se/docs/P-5061-software-soar-mobility-Final-Full-Doc-20140716.pdf>
- [Sonatype](http://www.sonatype.com/) - <http://www.sonatype.com/>
- [Black Duck HUB](https://www.blackducksoftware.com/products/black-duck-hub) - <https://www.blackducksoftware.com/products/black-duck-hub>
- [OWASP Dependency Check](https://www.owasp.org/index.php/OWASP_Dependency_Check) - [https://www.owasp.org/index.php/OWASP\\_Dependency\\_Check](https://www.owasp.org/index.php/OWASP_Dependency_Check)

Slide 30/31:

- <http://en.wikipedia.org/wiki/Heartbleed>

# Links (cont.)



GSAW  
2016

Slide 34:

- [report](http://www.acq.osd.mil/se/docs/P-5061-software-soar-mobility-Final-Full-Doc-20140716.pdf) - <http://www.acq.osd.mil/se/docs/P-5061-software-soar-mobility-Final-Full-Doc-20140716.pdf>
- [matrix](http://www.acq.osd.mil/se/docs/P-5061-AppendixE-soar-sw-matrix-v9-mobility.xlsx) - <http://www.acq.osd.mil/se/docs/P-5061-AppendixE-soar-sw-matrix-v9-mobility.xlsx>
- [NSA's CAS](http://samate.nist.gov/docs/CAS_2011_SA_Tool_Method.pdf) - [http://samate.nist.gov/docs/CAS\\_2011\\_SA\\_Tool\\_Method.pdf](http://samate.nist.gov/docs/CAS_2011_SA_Tool_Method.pdf)
- [Institute for Defense Analyses](http://www.acq.osd.mil/se/docs/P-5061-software-soar-mobility-Final-Full-Doc-20140716.pdf) - <http://www.acq.osd.mil/se/docs/P-5061-software-soar-mobility-Final-Full-Doc-20140716.pdf>

Slide 46:

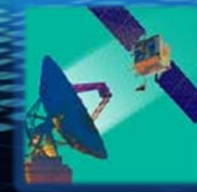
- [Blue Book](http://public.ccsds.org/publications/archive/355x0b1.pdf) - <http://public.ccsds.org/publications/archive/355x0b1.pdf>

Slide 51:

- [C](https://www.securecoding.cert.org/confluence/display/c/SEI+CERT+C+Coding+Standard) - <https://www.securecoding.cert.org/confluence/display/c/SEI+CERT+C+Coding+Standard>
- [C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637) - <https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=637>
- [JAVA](https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java) - <https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java>
- [https://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_static\\_code\\_analysis](https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis)



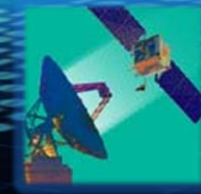
# Best Practices from NASA's Secure Coding Portal



GSAW  
2016

1. **Validate input.** Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files.
2. **Heed compiler warnings.** Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code.
3. **Use Code Analysis Tools.** Use static and dynamic analysis tools to detect and eliminate additional security flaws. Dynamic analysis is the testing and evaluation of an application during runtime. Static analysis is the testing and evaluation of an application by examining the code without executing the application. Many software defects that cause memory and threading errors can be detected both dynamically and statically. The two approaches are complementary because no single approach can find every error. The primary advantage of dynamic analysis: It reveals subtle defects or vulnerabilities whose cause is too complex to be discovered by static analysis. Dynamic analysis can play a role in security assurance, but its primary goal is finding and debugging errors. The primary advantage of static analysis: It examines all possible execution paths and variable values, not just those invoked during execution. Thus static analysis can reveal errors that may not manifest themselves until weeks, months or years after release. This aspect of static analysis is especially valuable in security assurance, because security attacks often exercise an application in unforeseen and untested ways.
4. **Use Binary Analysis Tools.** Binary analysis creates a behavioral model by analyzing an application's control and data flow through executable machine code – the way an attacker sees it. Unlike source code tools, this approach accurately detects issues in the core application and extends coverage to vulnerabilities found in 3rd party libraries, pre-packaged components, and code introduced by compiler or platform specific interpretations.

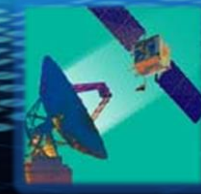
# Best Practices from NASA's Secure Coding Portal



GSAW  
2016

5. **Architect and design for security policies.** Create software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set.
6. **Keep it simple.** Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex.
7. **Default deny.** Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted.
8. **Adhere to the principle of least privilege.** Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should be held for a minimum time. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges.
9. **Sanitize data sent to other systems.** Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem.
10. **Practice defense in depth.** Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment.

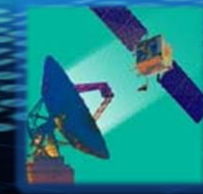
# Best Practices from NASA's Secure Coding Portal



GSAAW  
2016

11. **Use effective quality assurance techniques.** Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions.
12. **Adopt a secure coding standard.** Develop and/or apply a secure coding standard for your target development language and platform.
13. **Define security requirements.** Identify and document security requirements early in the development life cycle and make sure that subsequent development artifacts are evaluated for compliance with those requirements. When security requirements are not defined, the security of the resulting system cannot be effectively evaluated.
14. **Model threats.** Use threat modeling to anticipate the threats to which the software will be subjected. Threat modeling involves identifying key assets, decomposing the application, identifying and categorizing the threats to each asset or component, rating the threats based on a risk ranking, and then developing threat mitigation strategies that are implemented in designs, code, and test cases.
15. **Don't trust services.** Many organizations utilize the processing capabilities of third party partners, who more than likely have differing security policies and posture than you. It is unlikely that you can influence or control any external third party, whether they are home users or major suppliers or partners. Therefore, implicit trust of externally run systems is not warranted. All external systems should be treated in a similar fashion.

# Best Practices from NASA's Secure Coding Portal



GSAW  
2016

16. **Separation of duties.** A key fraud control is separation of duties. For example, someone who requests a computer cannot also sign for it, nor should they directly receive the computer. This prevents the user from requesting many computers, and claiming they never arrived. Certain roles have different levels of trust than normal users. In particular, administrators are different to normal users. In general, administrators should not be users of the application.
17. **Software Supply Chain.** IT managers should create and preserve a bill of materials, or a list of ingredients, for the components used in a given piece of software. The complexities and interdependencies of the IT ecosystem require software suppliers to not only be able to demonstrate the security of products they produce, but also evaluate the integrity of products they acquire and use. Ultimately this should lead to greater confidence through integrity checks incorporated in a defined secure development lifecycle.
18. **Avoid security by obscurity.** Security through obscurity is a weak security control, and nearly always fails when it is the only control. This is not to say that keeping secrets is a bad idea, it simply means that the security of key systems should not be reliant upon keeping details hidden. For example, the security of an application should not rely upon knowledge of the source code being kept secret. The security should rely upon many other factors, including reasonable password policies, defense in depth, business transaction limits, solid network architecture, and fraud and audit controls. A practical example is Linux. Linux's source code is widely available, and yet when properly secured, Linux is a hardy, secure and robust operating system.
19. **Fix security issues correctly.** Once a security issue has been identified, it is important to develop a test for it, and to understand the root cause of the issue. When design patterns are used, it is likely that the security issue is widespread amongst all code bases, so developing the right fix without introducing regressions is essential.