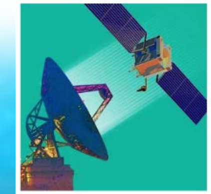


GSAW 2019
Feb. 25-28, 2019
Renaissance Los Angeles Airport Hotel
"Creating Smarter Ground Systems"

Ground System Architectures Workshop
Tutorial D

Improving Security of Ground System Software

February 25, 2019

Brandon Bailey

Chief Technology Officer – TMC Technologies

brandon.t.bailey@nasa.gov | brandon.bailey@tmctechnologies.com

304-629-8992

NASA's IV&V Program

Safety and Mission Assurance (SMA) Office

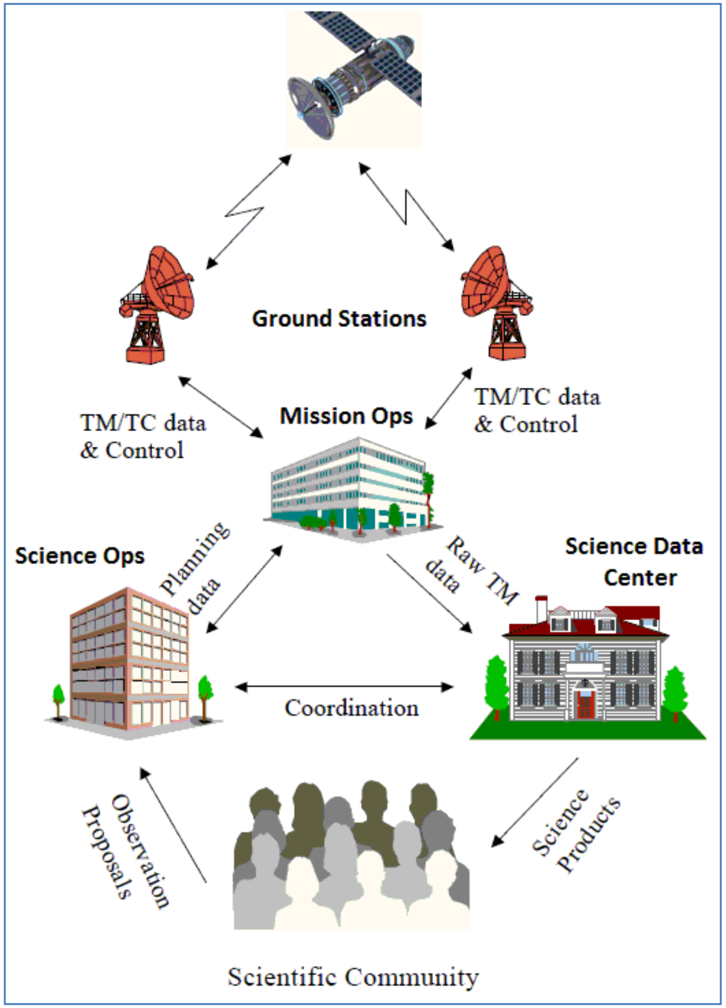
Information Assurance/Cybersecurity Support

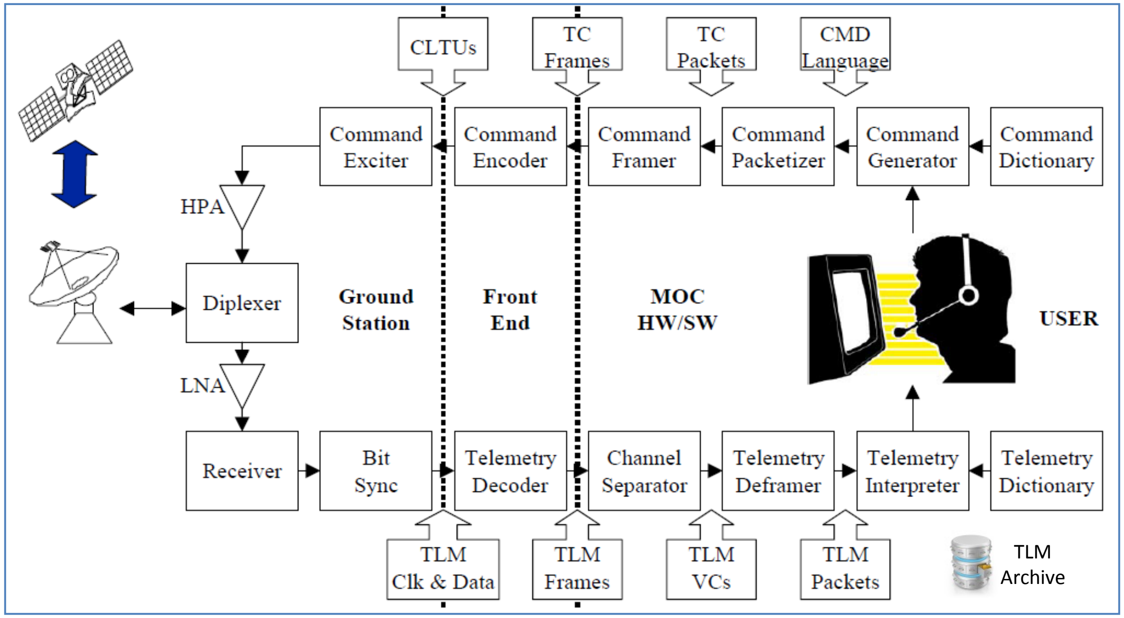
http://www.nasa.gov/centers/ivv

Agenda/Outline

- **Tutorial D Outline:**
  - Getting on the Same Page with Ground Systems
  - Threat Landscape w/ Software
  - What is SW in a Ground System?
  - SW Security is Required but Barriers Exist
  - What about NIST?
  - Approach for Secure and Resilient Software
    - System Threat Modeling
    - Sample Process for Developing Secure Software
    - Alphabet Soup - VA, SCA, OA, CWE, CVE, CWSS
  - Ground Software Example: FEPs
  - Near Term Goals and What to do Now?
  - Trends and Lessons Learned
  - Future: Cloud and DevSecOps

Defining "Ground Systems"





Spacecraft Ground Systems encompass 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

Evolving Threatscape for Space Missions

GSAW 2019

Threats are both becoming more frequent and more malicious

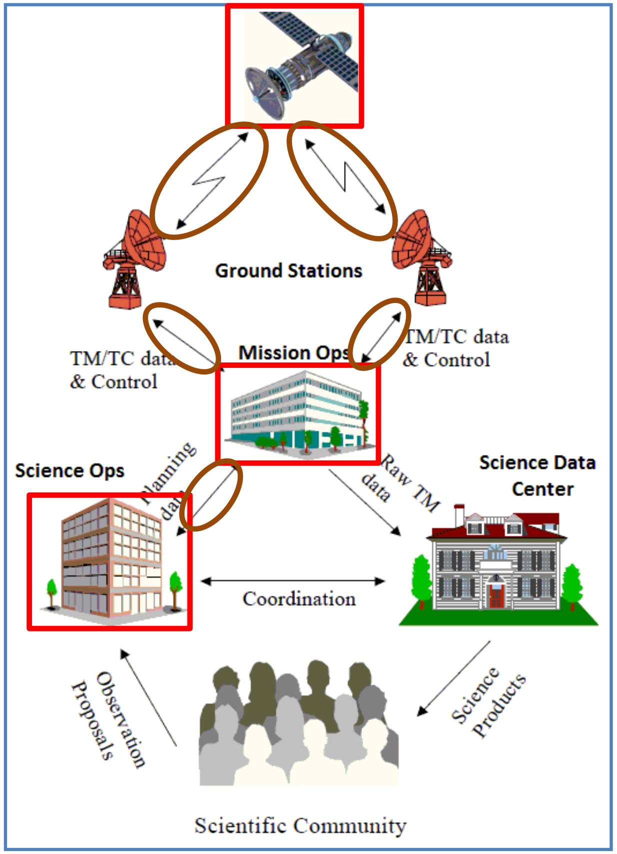
Past:
- Known Vulnerabilities and Attack Vectors
- Out of box Security

Current:
- Emerging Threats
- Phishing
- Insider Threat
- Advanced Persistent Threats (APT)
- Zero-Day Threats

Current/Future:
- Unknown Vulnerability and/or Threat
- Vulnerability at creation
- Supply Chain
- Others…

Satellite System vulnerabilities to Threats

- ***Custom software*** located throughout the system present potential vulnerabilities to software threats
  - Spacecraft
  - Mission Operations Center (MOC)
  - Mission planning area
  - Software development environment
- Software interfaces throughout the system, present potential vulnerabilities – both insider and external threats
- Software resiliency to vulnerabilities and weaknesses
  - Security architecture
  - Software controls against credible threats
  - Common Weakness Enumerations (CWEs)
  - Common Vulnerabilities and Exposures (CVEs)



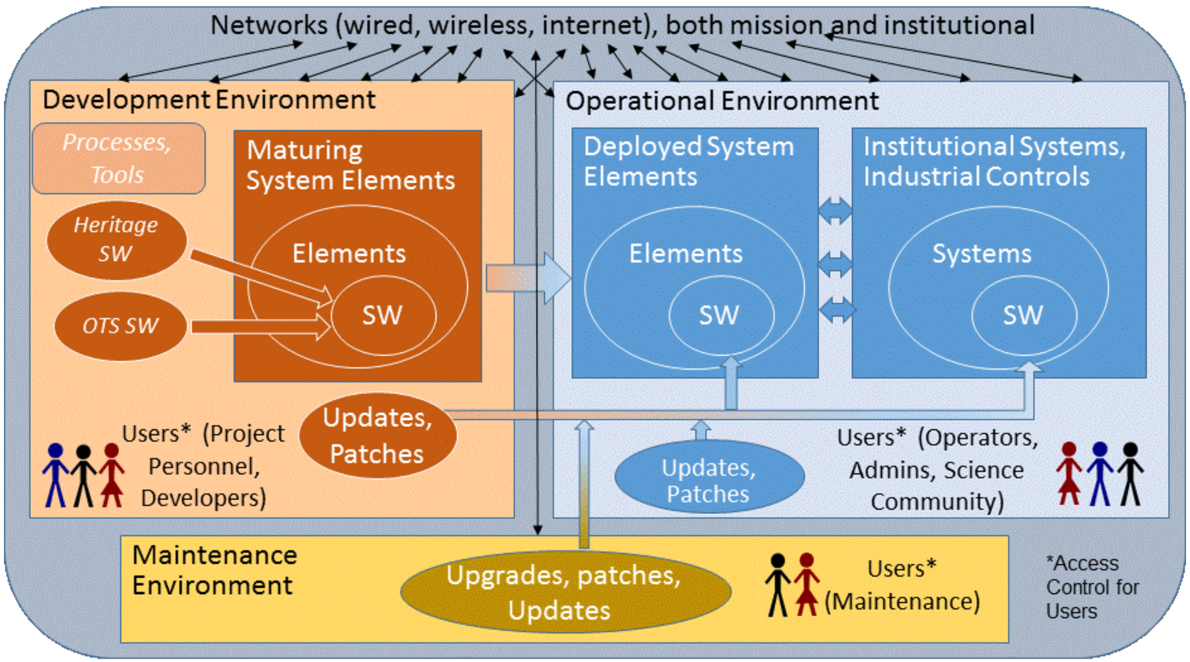
Back to the Basics

GSAW 2019

- In Ground Systems….What is Software?

A. Custom developed?
B. Commercial-off-the-Shelf (COTS) Software?
C. Government-off-the-Shelf (GOTS) Software?
D. Free and Open Source Software (FOSS) ?
E. Industrial Control System (ICS) Software

Answer: All of the Above

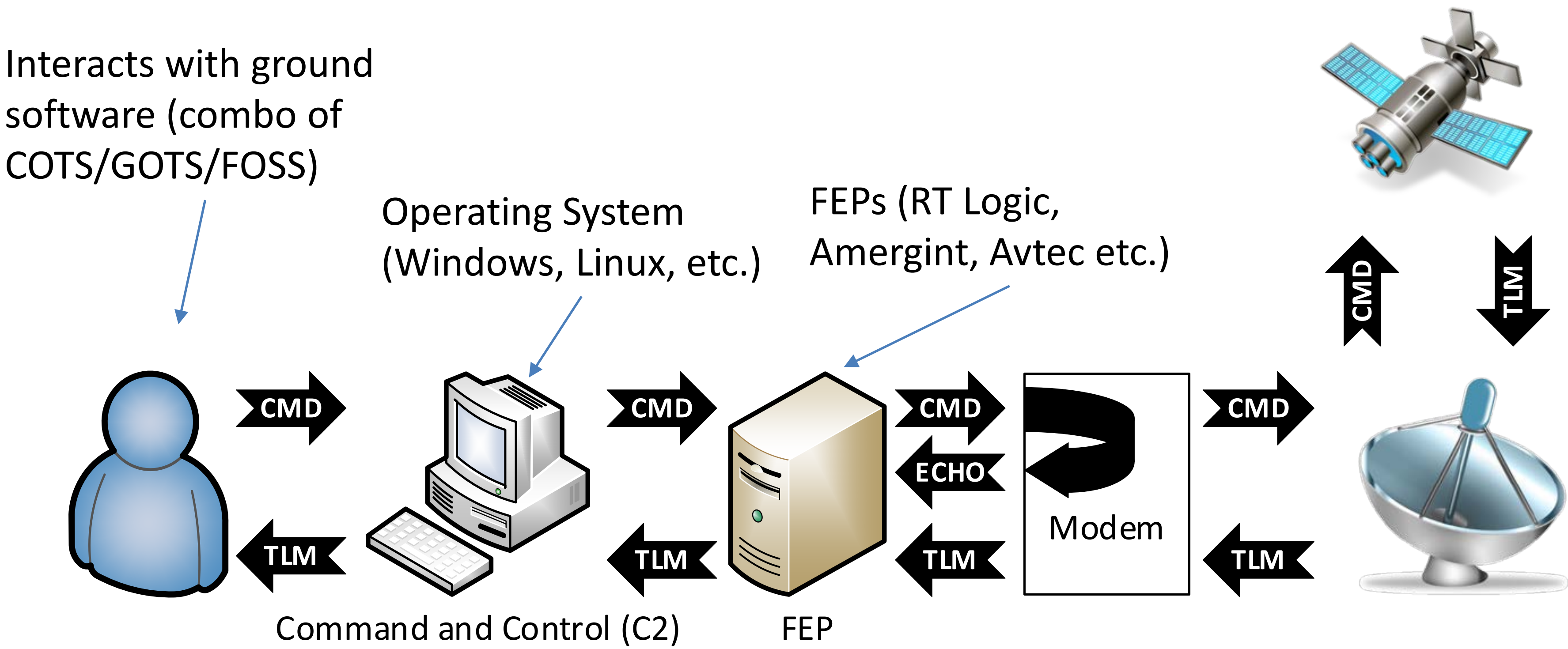


Cybersecurity and Information Assurance risk needs to be assessed and addressed across the entire mission system architecture – including development and operations

- In Ground Systems….Where is Software Used?

Answer: Everywhere

Scope for this Discussion…



Software Security

- Why
  - SW controls mission critical activities such as command sequencing, scheduling, satellite tracking, launch control and payload operations
- What
  - With any system or system of systems, the software is a critical component and the security of said software is equally important
- How
  - Designing in security (e.g. threat modeling) and using secure coding practices (e.g. coding standards and tools)

But Where are the Requirements?

GSAW 2019

FISMA requires each agency to use a risk-based approach to develop, document, and implement an agency wide security program for the information and information systems that support the operations and assets of the agency, including those provided or managed by another agency, contractor, or other source.

- OMB-130 -- “Security of Federal Automated Information Systems”
- Executive Order 13800, ***Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure***,
- Federal agency directives (DoD 8510.01, NASA NPR 2810, etc.)
- DoDI 5000.02 and DoDI 5200.39
- …

Flowing Down…

- How do these directives, EOs, policies, etc. prevent software weaknesses and vulnerabilities (e.g. buffer overflows and un-sanitized input)?
  - SW developers do not develop to these requirements which is a barrier

Other Barriers to Reducing SW Risk

Barrier	Detail
Security as a Technical (Systems Engineering) function	Programs/Systems may choose to comply with baseline controls in the NIST 800 series compared to performing the mission security analysis using risks and threats
Evolving Threatscape	The evolving threatscape entails full understanding of current and future threats that can exploit system vulnerabilities
Security is more than IT	The perception that Information Technology (IT) protects (e.g. border firewalls) a mission environment is no longer adequate in the evolving threatscape
Complex Supply Chains	System complexity leads to large supply chains, including delivery of various products using varying processes
Belief "This will not happen to me"	Given the history of success of NASA/DoD missions, a cavalier attitude is possible. This is not secure, given the evolving threatscape. Hope is not the security strategy, any more than it is for Safety.

Other Barriers to Reducing SW Risk

GSAW 2019

Barrier	Detail
Culture of Openness	Security control of information is counter to some cultures of openness and sharing with International Partners and the Public (e.g. NASA).
Traditional Systems Engineering approach led to stovepipe elements	The top-down elaboration and allocation process has successfully led to complex systems being developed, including infrastructure and legacy systems. The advent of security has a unique architecture view to traditional systems engineering approaches
Security as a Priority	The priority of security must be emphasized at an Agency, Program, Center/Installation, and Project level.
Governance and Organizations	To achieve an appropriate security posture, organizations such as the Protection Programs, Chief Information Officers, System Engineers, Operators, Institutional Systems, Programs, and SMA need to work together.
Terminology	An outcome of the multiple organizations is that each may have slightly unique vernacular. Arriving at a common terminology enables a shared strategy, implementation and operation.

NIST Can Help….

- If implemented and governed properly NIST can help but usually NIST is thought to be “compliance” only
- The security control structure is made up of the following sections:
  - Control section
  - Supplemental guidance section
  - Control enhancements section
  - References section
  - Priority and baseline allocation section
- Remember! NIST provides guidance not requirements
- NIST intentionally presents controls written at a very high level of abstraction
  - System Specification Requirements:
    - Developed by translating the abstract controls into specific requirements
      - These would be further decomposed from the system level

Example: SI-10 (NIST 800-53 Rev 4)

GSAW 2019

SI-10 INFORMATION INPUT VALIDATION

Control: The information system checks the validity of [*Assignment: organization-defined information inputs*].

Supplemental Guidance: Checking the valid syntax and semantics of information system inputs (e.g., character set, length, numerical range, and acceptable values) verifies that inputs match specified definitions for format and content. Software applications typically follow well-defined protocols that use structured messages (i.e., commands or queries) to communicate between software modules or system components. Structured messages can contain raw or unstructured data interspersed with metadata or control information. If software applications use attacker- supplied inputs to construct structured messages without properly encoding such messages, then the attacker could insert malicious commands or special characters that can cause the data to be interpreted as control information or metadata. Consequently, the module or component that receives the tainted output will perform the wrong operations or otherwise interpret the data incorrectly. Prescreening inputs prior to passing to interpreters prevents the content from being unintentionally interpreted as commands. Input validation helps to ensure accurate and correct inputs and prevent attacks such as cross-site scripting and a variety of injection attacks.

Control Enhancements:

(1) .....

(2) ......

(3) *INFORMATION INPUT VALIDATION | PREDICTABLE BEHAVIOR*

The information system behaves in a predictable and documented manner that reflects organizational and system objectives when invalid inputs are received.

Supplemental Guidance: A common vulnerability in organizational information systems is unpredictable behavior when invalid inputs are received. This control enhancement ensures that there is predictable behavior in the face of invalid inputs by specifying information system responses that facilitate transitioning the system to known states without adverse, unintended side effects.

(4) ....

(5) ....

References: None.

Priority and Baseline Allocation:

P1	**LOW** Not Selected	**MOD** SI-10	**HIGH** SI-10

NIST Security Controls that Apply to Software

GSAW 2019

- Compiled an initial selection of NIST 800-53r4 controls that relate to software or software control
- 113 of 343 “High” Baseline controls and enhancements implemented by software

ID	FAMILY	Relates to software	Total
AC	Access Control	24	43
AU	Audit and Accountability	19	28
CM	Configuration Management	8	31
IA	Identification and Authentication	20	24
MP	Media Protection	1	12
RA	Risk Assessment	3	8
SC	System and Communications Protection	22	30
SI	System and Information Integrity	16	27

Software Related Security Controls

- Note: Additional controls or enhancements may be brought into focus while following the evidence in support of an analysis finding

NIST Too High Level?

- NIST can be at too high of a level and abstract for SW developers
- Common Weakness Enumeration (CWE) prevention is a more implementable "requirement"
- For the same SI-10 NIST Control the following CWEs apply
  - 77, 134, 22, 23, 20, 73, 79, 78, 119, 787, 805, 131, 170
- Whatever your method, requirements need to be clear and understood
  - Requirement to have "secure code" is not good enough
    - Quote: "SW must not have CAT I or CAT II findings"
  - Requirement to implement and be compliant with NIST is not good enough without thorough **technical governance**

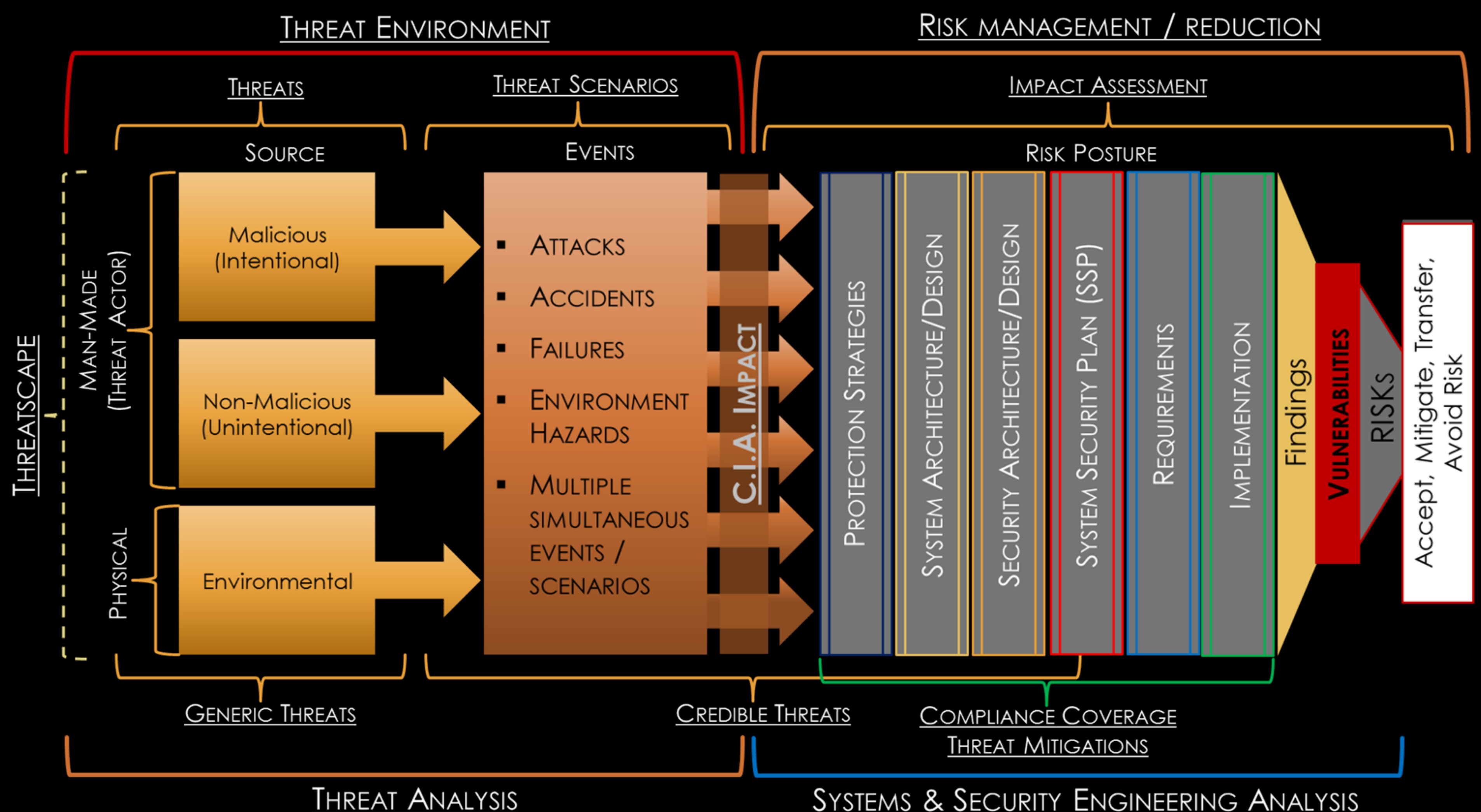
An Approach for Secure & Resilient SW

GSAW 2019

- Not “the” approach but “an” approach to help solve this problem
  - We do agree a problem exists, right?
- Need secure designs **and** secure code
  - Is there a difference?
  - CWE prevention != Secure Design & vice versa
- “An” approach to secure design = Threat Modeling, Best Practices (see backup) Adherence, etc.
- “An” approach to secure code = Coding Standards & CWE prevention (oh….and don’t forget CVE prevention either)

High Level Threat Modeling

GSAW 2019



Generalized Process to Develop Secure Software

GSAW 2019

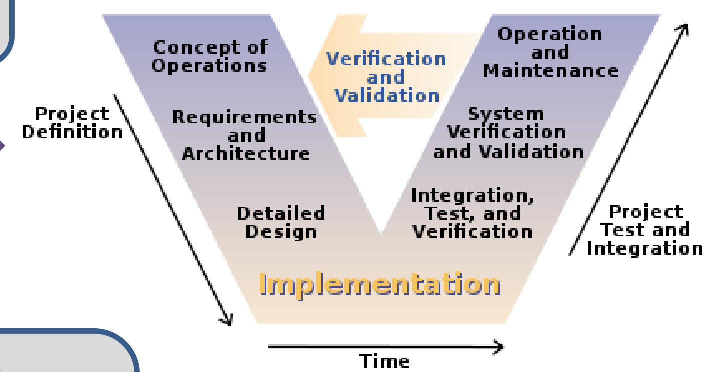
- Systems Engineering Process to design out security risk
- Establish credible threats and vulnerabilities, and designs in software controls, following NIST guidelines
- Once security implementation approach is established (System Security Plan), development proceeds

Part 1: Assess Mission for Credible Threats, and Vulnerabilities
- Credible threats based on situational environment
- Vulnerabilities assessed by establishing security risk to system
- Preliminary @ KDP 0 (~SRR); Baseline @ KDP 1 (~SDR)

Part 2: Develop Security Strategy
- Develop security architecture and ConOps
- Capture in Project Protection Plan
- Preliminary @ SDR; Baseline @ PDR

Part 3: The Security Plan is a pivotal artifact that captures security strategy, presents controls and sets the basis for implementation

Part 3: Select and Tailor Security Controls
- Many controls software based
- Preliminary @ SDR, Baseline @ PDR



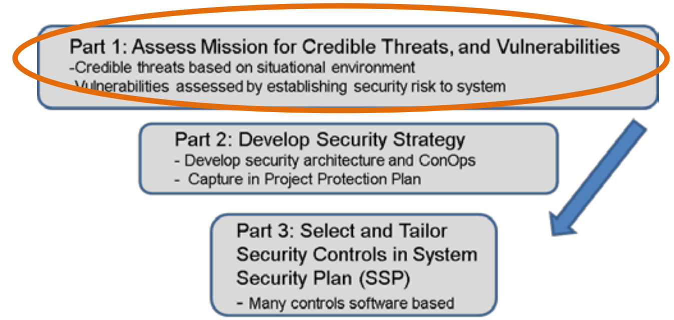
Part 4: Implement and Test Security Strategy and Controls
- Defined controls become basis for system and software requirements
- Implement in accordance with traditional lifecycle development
- System level tests consider threat scenarios

Products: Verified and Validated Secure Software

Lifecycle development occurs based on the SSP and secure coding practices

Part 1: System Security Threat Understanding

- Development of the Project Protection Plans (PPP) require an understanding of credible threats
- Developing credible threats for identified mission
  - General information in [CCSDS green book]
  - Leverage all intel sources at all levels
  - Threat Summary can be classified Top Secret
- The key project inputs for the threat summary process are:
  - Mission overview
  - Lifecycle phase
  - CONOPS
  - Communication links
- Evolving Threat Summary process – work with all stakeholders and other agencies to identify credible threats in order to develop the PPP.

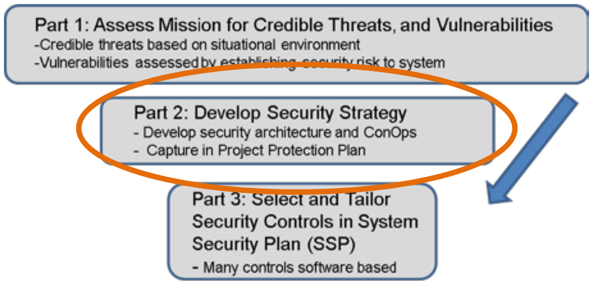


Threat Summary: Documents the threat environment that a space system/constellation or aircraft is most likely to encounter as it reaches operational capability

Part 2: Develop Security Strategy

GSAW 2019

The key elements of the Project Protection Plan (PPP):



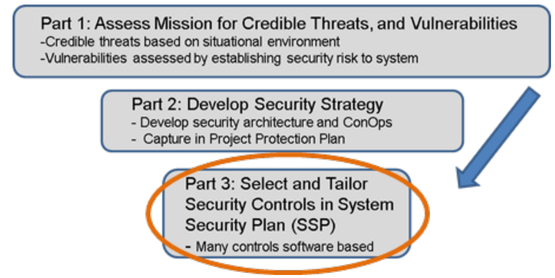
- Vulnerabilities Analysis
  - What will prevent the system from reaching mission requirements due to threats exploiting vulnerabilities?
- Risk Analysis
  - Sufficient detail must be documented in the risk analysis for senior decision makers to approve the project at key decision points (KDPs). The risk analysis must answer all the vulnerabilities driven by the threat and potential countermeasures and mitigations.
  - Also in the risk analysis, document what risks will not be addressed and the rationale behind that decision.
  - Consider Defense in Depth and the Evolving Threatscape
- Likely a classified document and should have information such as

PPP

- Mission Overview
- Mission Support Elements
  - e.g., Comm networks, ground systems, navigations and tracking systems, enterprise security
- Threat Overview
- System Criticality and Susceptibilities
  - Architecture – critical elements and nodes
  - CONOPS – critical processes
- Mission Vulnerabilities and Risks
- Protection Strategies

Part 3: The System Security Plan

GSAW 2019



- In order to select controls, begin by specifying and documenting the information system's:
  - Categorization per FIPS-199
  - Information types
  - Security impact levels for
    - Confidentiality
    - Integrity
    - Availability
  - Security boundary and interfaces

INFORMATION TYPE (Derived from NIST SP 800-60)	D11 – Transportation	
INFORMATION SUB-TYPE	D11.4 – Space Operations	
Confidentiality Impact Level	NIST: Low	OWNER: Moderate
Integrity Impact Level	NIST: High	OWNER: High
Availability Impact Level	NIST: High	OWNER: High
Justification for any deviation from the NIST recommended impact level	Business functions involve proprietary information	

- Each information system has its own SSP (multiple per mission) per the strategy provided in the Project Protection Plan. Risk assessment captured in companion document, Risk Assessment Report (RAR).

- *NIST = National Institute of Standards and Technology*
- *FIPS = Federal Information Processing Standard*
- *FIPS Publications are standards issued by NIST after approval*

Part 3: Select and Tailor Security Controls

GSAW 2019

To select **security controls**, engineers must:

Select all the security controls based on the security categorization process

Tailor by applying scoping, parameterization, and compensating control guidance

Supplement with Agency supplemental security controls for selected controls

Document in the SSP

Specify the minimum control requirements

Identify from this set which of the security controls are common controls or controlled by another organization

Baseline of Security Controls

Tailored and Scoped Security Controls

Supplemented Security Controls

Security Controls, in SSP

Iterate and evolve

Iterate and evolve

Part 1: Assess Mission for Credible Threats, and Vulnerabilities
-Credible threats based on situational environment
-Vulnerabilities assessed by establishing security risk to system

Part 2: Develop Security Strategy
- Develop security architecture and ConOps
- Capture in Project Protection Plan

Part 3: Select and Tailor Security Controls in System Security Plan (SSP)
- Many controls software based

System Security Plan
- Information types
- Security impact levels for Confidentiality, Integrity, Availability
- Security boundary and interfaces
- Security controls

- Risk based process
- Engineering Analysis
- Iterative in nature
- Continuous monitoring

Part 3: Security Controls Families (NIST 800-53)

GSAW 2019

ID	FAMILY
★ AC	Access Control
AT	Awareness and Training
★ AU	Audit and Accountability
CA	Security Assessment and Authorization
★ CM	Configuration Management
CP	Contingency Planning
★ IA	Identification and Authentication
IR	Incident Response
MA	Maintenance
★ MP	Media Protection
PE	Physical and Environmental Protection
PL	Planning
PS	Personnel Security
★ RA	Risk Assessment
SA	System and Services Acquisition
★ SC	System and Communications Protection
★ **SI**	**System and Information Integrity**
PM	Program Management

★ *= relates to software or software control*

Within a Control Family, analyze controls based on
1) Required controls, based on FIPS-199 classification

CNTL NO.	CONTROL NAME	PRIORITY	INITIAL CONTROL BASELINES		
			LOW	MOD	HIGH
	System and Information Integrity				
SI-1	System and Information Integrity Policy and Procedures	P1	SI-1	SI-1	SI-1
SI-2	Flaw Remediation	P1	SI-2	SI-2 (2)	SI-2 (1) (2)
SI-3	Malicious Code Protection	P1	SI-3	SI-3 (1) (2)	SI-3 (1) (2)
...					
SI-10	Information Input Validation	P1	Not Selected	SI-10	SI-10
...					
SI-17	Fail-Safe Procedures	P0	Not Selected	Not Selected	Not Selected

2) Evaluation of supplemental controls, enhancements that are not explicitly specified

Example SI-10 (3)

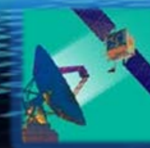
SI-10 Information Input Validation.
Enhancement (3) Information input validation | Predictable behavior The information system behaves in a predictable and documented manner that reflects organizational and system objectives when invalid inputs are received.

Supplemental Guidance: …This control enhancement ensures that there is predictable behavior in the face of invalid inputs by specifying information system responses that facilitate transitioning the system to known states without adverse, unintended side effects.

Part 4: Secure Software Development

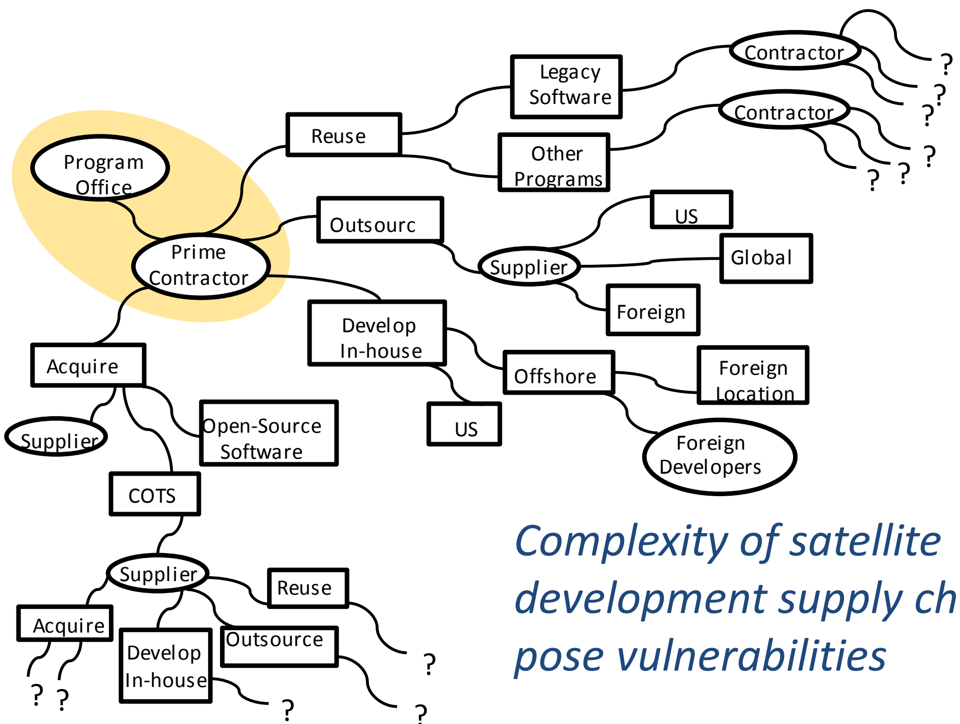
Part 4: Implement and Test Security Strategy and Controls
- Defined controls become basis for system and software requirements
- Implement in accordance with traditional lifecycle development
- System level tests consider threat scenarios

Products: Verified and Validated Secure Software



GSAW 2019

Software Threats Description (CCSDS Green Book, Section 3.4.9)	Mitigations/Controls
Users, system operators, and programmers often make mistakes that can result in security problems. Users or administrators can install **unauthorized or un-vetted software**, which might contain bugs, viruses, spyware, or which might simply result in system instability. System operators might configure a system incorrectly resulting in security weaknesses. Programmers may introduce **logic or implementation errors** which could result in system vulnerabilities or instability.	• **Unauthorized/Un-Vetted SW**: Provide appropriate focus on Supply Chain risks • **Logic/Implementation Errors:** Utilize Coding Standards and integrate tools into development environment (e.g. VA, OA, SCA, Threat Modeling) • Plan for Defense in Depth and secure the development environment



Complexity of satellite development supply chains pose vulnerabilities

Example Supply Chain Risks

- Undefined security requirements, policies, and practices limiting overarching security considerations
- Insecure software delivery mechanisms, leading to theft or malware injection
- Code and design defects that lead to vulnerable software
- Integration of insecure 3rd party libraries.

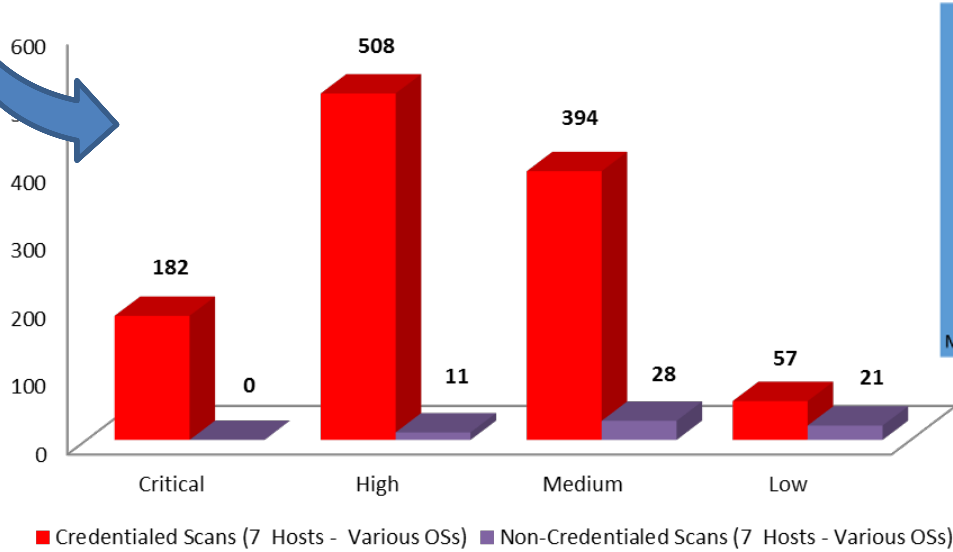
Secure Software Development Static Tools: Stds vs VA vs SCA vs OA

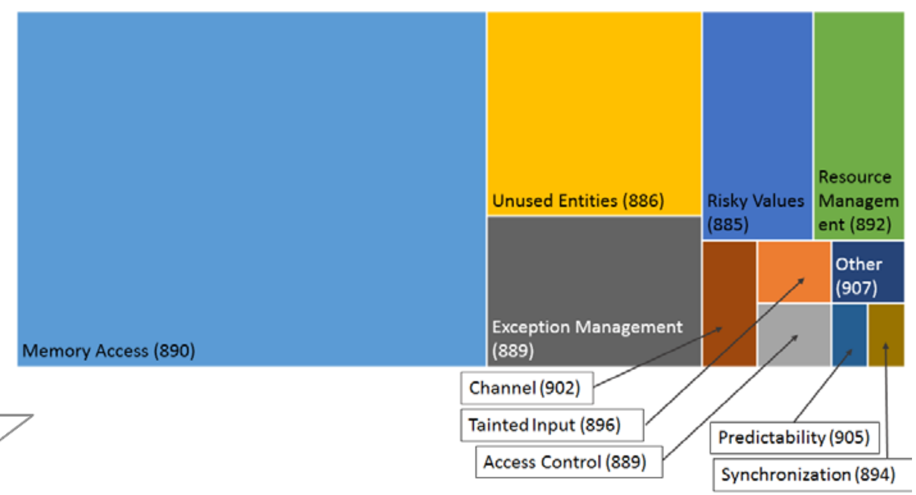
GSAW 2019

- Coding Standards
  - Secure coding standards are practices that are implemented to prevent the introduction of security vulnerabilities, such as bugs and logic laws. By following secure coding standards, programs can significantly reduce vulnerabilities before deployment
- Vulnerability Assessment (VA)
  - Running of tool(s) to identify known vulnerabilities and/or configuration settings that could lead to an impact to confidentiality, integrity or availability. VA identifies Common Vulnerabilities and Exposures (CVEs) or non-compliance with compliance regulations (e.g. STIGs)
- Static Code Analysis (SCA)
  - Running of tools that attempt to highlight possible weaknesses within 'static' (non-running) source code by using techniques such as taint analysis and data flow analysis. SCA identifies Common Weakness Enumerations (CWEs).
- Origin Analysis (OA)
  - OA fingerprints the binaries and folder structures, which discovers the third-party components used by the developer of the software, and creates a "bill of materials". Based on each identified component and its version, the tool then crosschecks its database for known vulnerabilities and software licenses associated with the component and categorize each as potential security or operational risks respectively. OA identifies Common Vulnerabilities and Exposures (CVEs) and risks with open source license usage.

We "should" be doing this already!

- The requirements for secure SW & security testing are present in existing guidance (e.g. STIGs, NIST Control RA-5)
  - Knowledge, tool availability, oversight and governance could be improved which puts government at risk
  - Credentialed vulnerability scanning, static code analysis, origin analysis and dynamic analysis of software is needed to adequately reduce software risk





Secure and Resilient Code

GSAW 2019

Common Weakness Enumerations (CWE):
Serves as a common language for describing software security weaknesses in architecture, design, or code. Protection is important for Ground SW, less vulnerabilities/threats for Flight SW. Originated by MITRE.

- 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

with

Common Weakness Scoring System (CWSS) of CWEs

- High impact within **our system**
- Values will be different for flight and ground (system dependent)

\+

Assess CWEs against common attack pattern enumeration and classification (CAPEC):

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

= **Top/Most Dangerous CWEs**

Assess SW against Common Vulnerabilities and exposure (CVE) {i.e. open source supply chain}:

- Identifies publicly known information security vulnerabilities and assign them a CVE_ID.
- Scored 1 to 10 on CVSS scale
- Operating Systems, Applications, FOSS, etc.

CWEs may already be addressed through good coding practices including use of static code analyzers with appropriate checkers (e.g. buffer overflow), coding standards, code walkthroughs, etc.

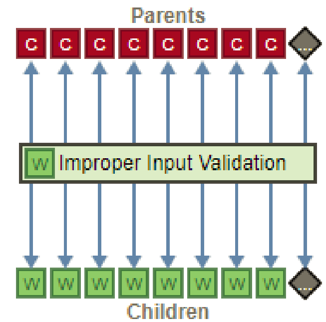
Let’s Break that Down…

GSAW 2019

- In order to provide assurance from a secure code perspective we need to establish:
  - The weaknesses in the software we deem most important within the context of the system
    - These could in turn be “requirements”
  - What coding standards need to be in place to eliminate weaknesses?
  - A link between the tools used for analysis and the most important weaknesses (trust but verify!)
  - Create a plan to maximize coverage with respect to static code analysis coverage

CWE Rack and Stack

- Source of weaknesses
  - Common Weakness Enumeration
    - Ex: CWE 20: Improper Input Validation
    - Weakness parents / children
    - Impacts to CIA
    - Examples

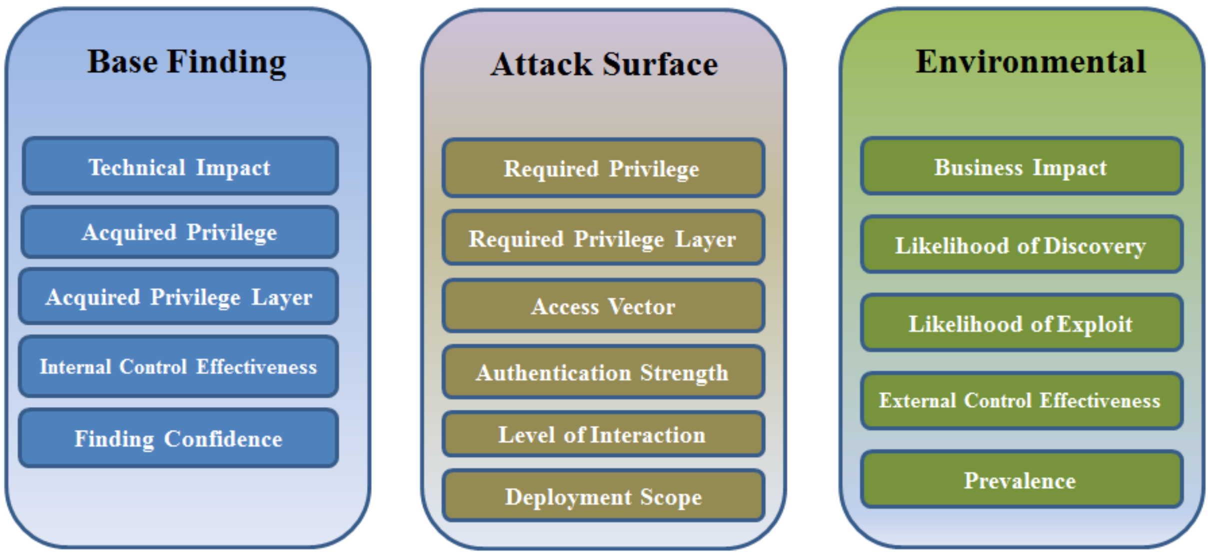


Scope	Impact
Availability	**Technical Impact:** *DoS: Crash, Exit, or Restart; DoS: Resource Consumption (CPU); DoS: Resource Consumption (Memory)* An attacker could provide unexpected values and cause a program crash or excessive consumption of resources, such as memory and CPU.
Confidentiality	**Technical Impact:** *Read Memory; Read Files or Directories* An attacker could read confidential data if they are able to control resource references.
Integrity Confidentiality Availability	**Technical Impact:** *Modify Memory; Execute Unauthorized Code or Commands* An attacker could use malicious input to modify data or possibly alter control flow in unexpected ways, including arbitrary command execution.

- Which ones do we care most about?
  - High impact within **our system**
  - Broad attack surface (many patterns, low technical barrier)
  - Evidence of real world exploitation (i.e. threat intel)
- Will have to use a combination of objective and subjective inputs

CWSS evaluation

- CWSS can help determine the CWEs with high impact within our system
- https://cwe.mitre.org/cwss/cwss_v1.0.1.html



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.

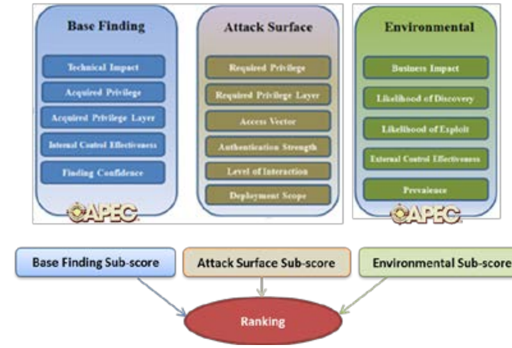
- Values will be different for each system (e.g. spacecraft and ground)
  - Realistically this should be performed on a **per mission / system** basis

Let's Add in CAPEC

- Common Attack Pattern Enumeration and Classification
  - https://capec.mitre.org
- 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.

Combining it All

- Calculates Scoring based on CWSS
  - CWSS = *BaseFindingScore * AttackSurfaceScore * EnvironmentScore*
  - Subjective due to system dependability
- Maintain ranking of CAPEC scores
  - Will have to use your own ranking system
  - More objectivity
- Maintain relationship between tools used and CWEs
  - Easily demonstrate which CWEs are covered
  - Can be used to develop future tools (Config generators, etc.)
- Process = Near complete picture of the top CWEs
- Subjective and Objective measures
  - Subjective - CWSS
  - Objective - CVE
  - Hybrid - CAPEC



Disclaimer

DISCLAIMER

Using mapping from tool vendors on their CWE coverage. Verification and Validation has not been performed!

Research being performed at [SAMATE] & CMU-SEI to help with this problem.

Rapid Expansion of Classification Models to Prioritize Static Analysis Alerts for C

https://resources.sei.cmu.edu/asset_files/Presentation/2017_017_001_506534.pdf

Results

- Peer reviewed most dangerous list of CWEs for system
  - Perfect ? **No**
  - Good enough ? **Yes**
  - Better than blindly accepting tool vendor criticality? **Yes**

- A link between the tools available and the most important weaknesses
  - Associate tool checks with CWEs
  - Mapped to secure coding standards/guidelines

Know what you are trying to prevent before selecting coding standards and tools

Simple Use Case #1

CWE 311: Missing Encryption of Sensitive Data

- Btw also NIST SC-8 Transmission Confidentiality and Integrity

- Adhere CERT Rules
  - MSC00-J
  - MSC18-C
  - WIN04-C
- HP Fortify static code analyzer has checkers for this which can reduce likelihood of being in code

Simple Use Case #2

GSAW 2019

CWE 119: Improper Restriction of Operations within the Bounds of a Memory Buffer

- Btw also NIST SI-10 Information Input Validation

- Adhere CERT Rules
  - ARR38-C, STR32-C, STR31-C, FIO37-C, EXP39-C, EXP33-C, ENV01-C, CTR50-CPP, ARR30-C, ARR00-C, ARR38-C, ARR00-C, CTR52-CPP, ARR30-C, STR32-C, CTR50-CPP, CTR52-CPP, EXP33-C, STR31-C, EXP39-C, FIO37-C, ENV01-C
- Fortify does not have a checker mapped to this

  - But Klockwork static code analyzer does
    - ABV.ANY_SIZE_ARRAY, ABV.GENERAL, ABV.ITERATOR, ABV.STACK, ABV.TAINTED, NNTS.MIGHT, NNTS.MUST, SV.STRBO.BOUND_SPRINTF, SV.STRBO.UNBOUND_COPY, SV.STRBO.UNBOUND_SPRINTF, SV.TAINTED.LOOP_BOUND

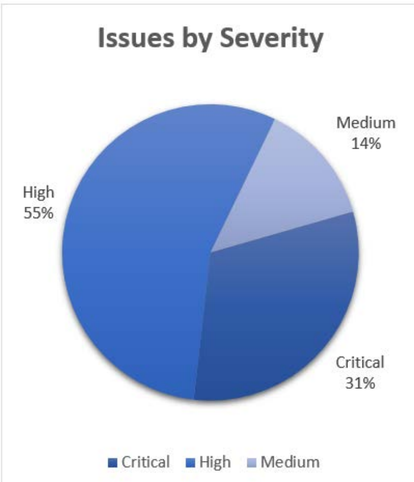
Takeaway

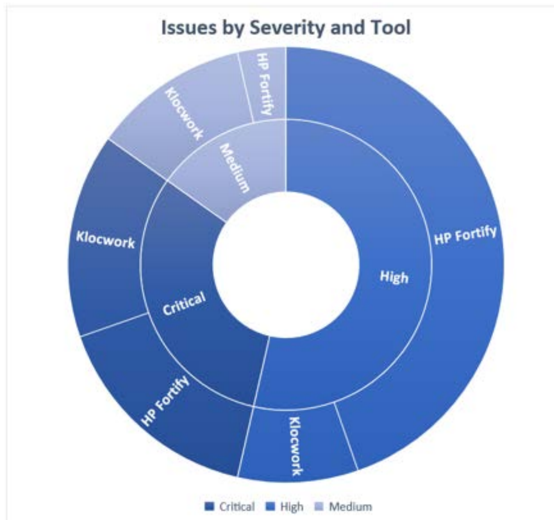
- One SCA tool is not going to ensure code is secure
- Just running tools is not enough, false positive analysis is super critical to value of SCA
- For real security assurance, must know what you want to prevent
  - What risk am I reducing in my system/software?
- Now pick the rules/guidelines and tools to help reduce that risk
- Great resource for identifying tools
  - Institute for Defense Analyses (IDA) Report | Spreadsheet
  - NASA also maintains matrix for mapping Top CWEs to tools to CERT rules
  - CERT's Wiki now has CWE to CERT Rule (Ex: https://wiki.sei.cmu.edu/confluence/display/cplusplus/MITRE+CWE)

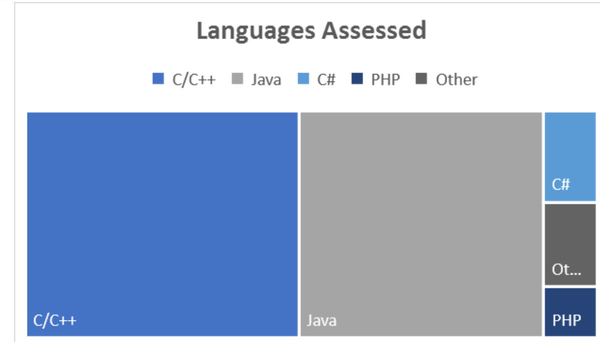
SCA: Real World Example

GSAW 2019

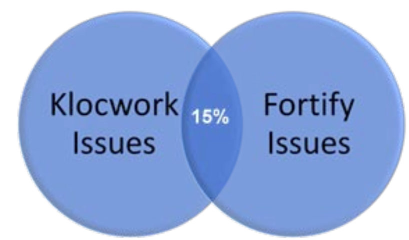
- 13 million lines of ground SW analyzed
- Klocwork and Fortify executed







Overlap of defects was 15%



- Surprised?
  - Not surprising given that the tools only have a 22% overlap in the ability to detect the same defects from NASA's most dangerous CWE list

DISCLAIMER

SCA: Real World Example (cont.)

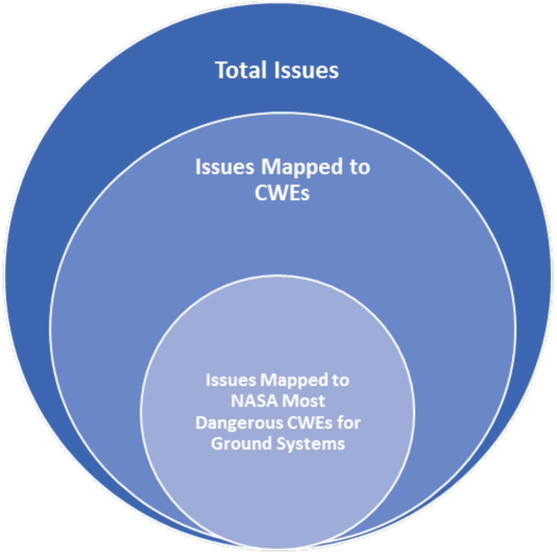
GSAW 2019

- Of the 49 most dangerous CWEs in ground systems
  - Klocwork against C/C++ = 47% coverage
  - Adding HP Fortify increases coverage by almost 35%
  - Giving the ability to detect 82% of the CWEs in C/C++
  - Side Note: About 50% of most dangerous ground CWEs have CERT coding standards associated
- Similarly, if HP Fortify is the only tool used then the tool only has the ability to detect 57% in C/C++, but by adding Klocwork an increase of 25% is realized, resulting in 82% coverage

DISCLAIMER

Targeted Metrics

- NASA's Most Dangerous Common Weakness Enumerations (CWEs) were used as a basis for evaluation as an additional overlay to what the tools report as Critical/High/Medium
  - NASA's most dangerous CWEs is a list published by NASA's Secure Coding Portal (SCP) team, which classifies the most dangerous weaknesses for ground software (similar to SANS Top 25 software errors)
  - Subset of weakness that mapped to the most dangerous ground system CWEs



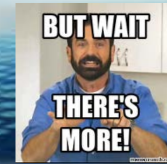
Takeaway

GSAW 2019

- If a program's security approach was simply to execute one SCA tool, that would be a good start but not good enough
- If a program's security approach was to simply invoke coding standard (CERT's std.), there could also be presence of CWEs
  - Some CWEs don't have CERT rules and some CWEs don't have checkers in Fortify/Klockwork but are CERT rules
- In the previous example, if one tool was used there's a risk that ~ 50% of the dangerous CWEs would be in the SW
- Takeaway: **Do all the above, multiple SCA tools, coding standards, etc.**

DISCLAIMER

But Wait There’s More



- Don’t forget….
  - [Common Vulnerabilities and Exposures (CVE)]
- Two flavors to worry about
  - COTS CVEs (Windows, Linux, Intel, etc.)
    - Installed on end points
  - FOSS CVEs (Struts, Xerces, Apache, etc.)
    - Embedded within custom code or installed on end points
- Different tools for detection
  - Vulnerability Assessment vs Origin Analysis

Origin Analysis: Secure SW Supply Chain

- 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

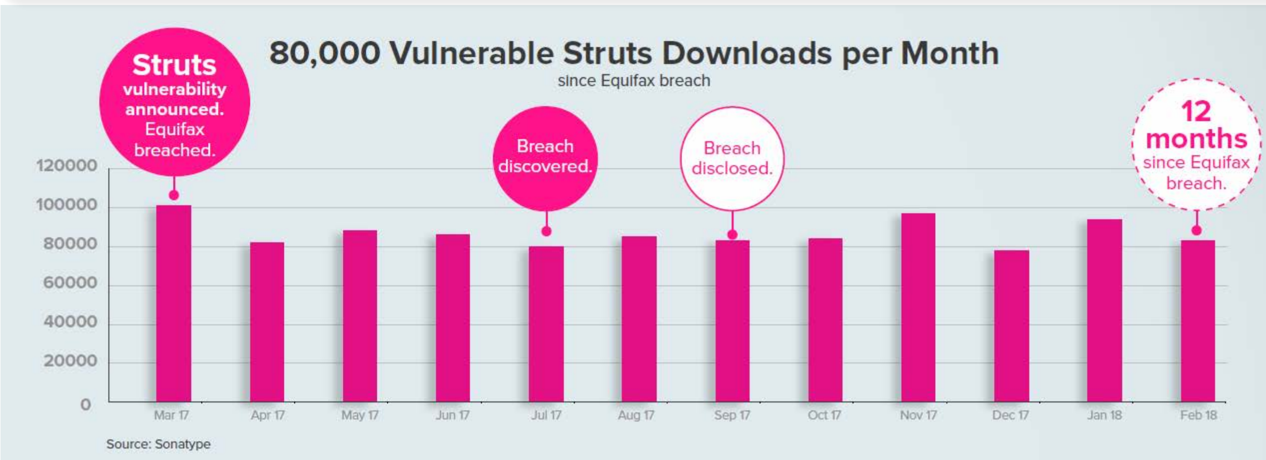
- Examples of tools
  - Sonatype
    - Binary scanner; Works best on JAVA
  - Black Duck HUB
    - Provides binary and source tree scanning; Support C/C++ as well has JAVA
  - OWASP Dependency Check
    - Currently Java, .NET, Ruby, Node.js, and Python projects are supported; additionally, limited support for C/C++ projects is available for projects using CMake or autoconf.

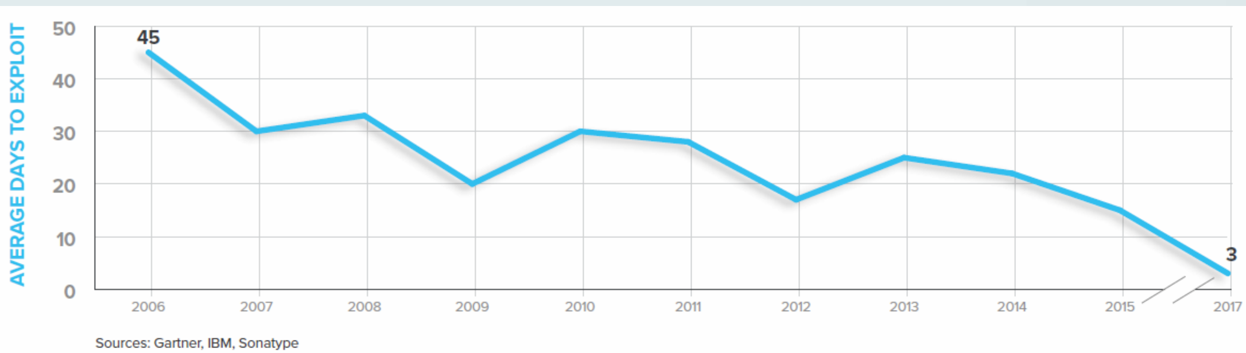
OSS Vulnerabilities Present Major Challenge

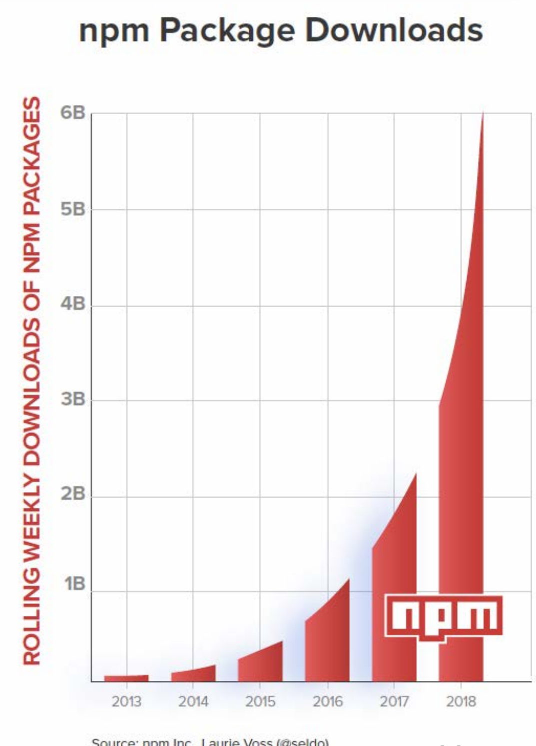
GSAW 2019

2018 State of Software Supply Chain (Sonatype)

- Open source vulnerabilities increased 120% YoY and their mean time to exploit compressed by 93.5%.
- Public vulnerability databases lack information on more than 1.3 million open source security advisories.
- Suspected or known open source breaches increased 55% YoY. Up 121% since 2014.



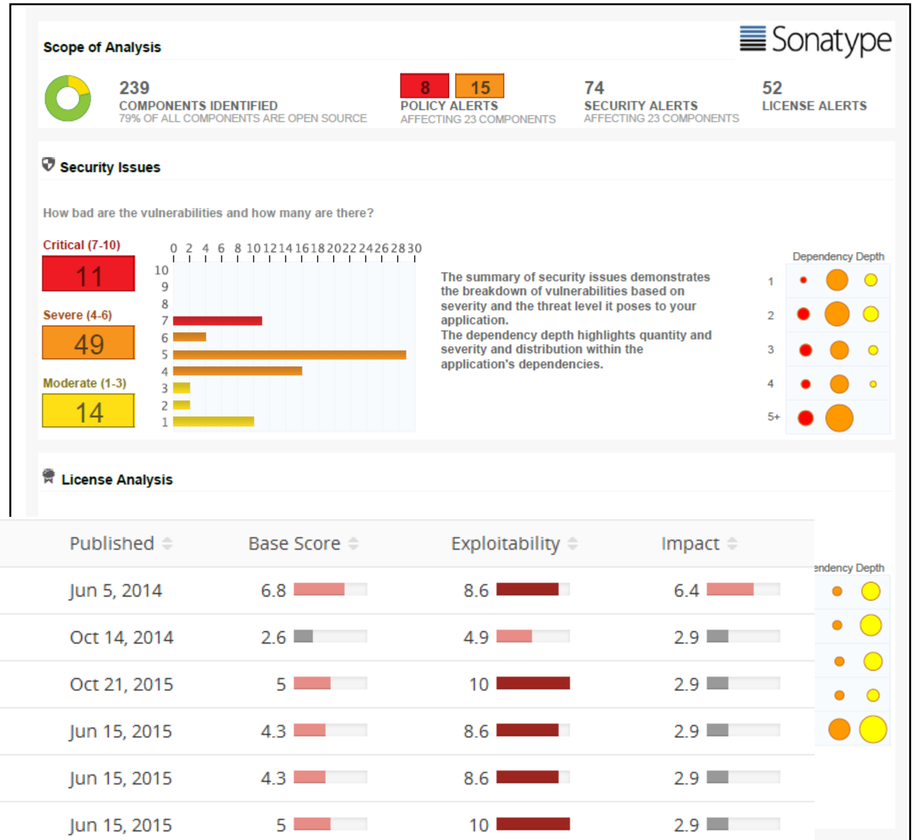




OA: Examples from Ground Systems

GSAW 2019

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 contain Implemen (i.e., IP detection,
CVE-2010-1244: Allows remote attackers to hijack the authentication of unspecified victims	activemq-web-5.2.0.2-fuse.jar	Upgrade J



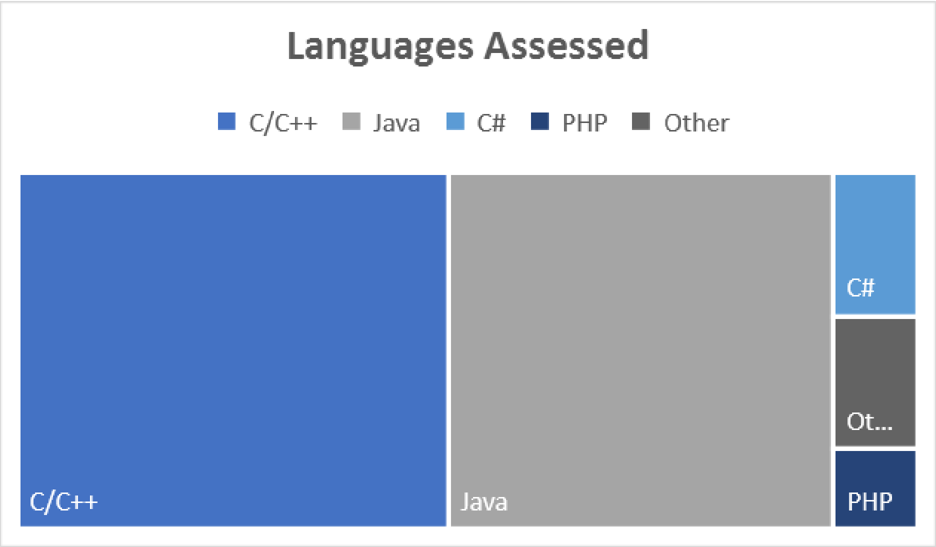
Identifier	Published	Base Score	Exploitability	Impact
VulnDB 107729	Jun 5, 2014	6.8	8.6	6.4
VulnDB 113251	Oct 14, 2014	2.6	4.9	2.9
NVD CVE-2014-0160	Oct 21, 2015	5	10	2.9
NVD CVE-2015-1788	Jun 15, 2015	4.3	8.6	2.9
NVD CVE-2015-1789	Jun 15, 2015	4.3	8.6	2.9
NVD CVE-2015-1790	Jun 15, 2015	5	10	2.9
NVD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

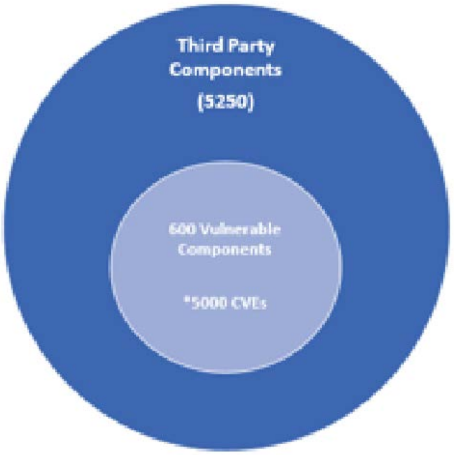
BLACKDUCK

OA: Real World Example

GSAW 2019

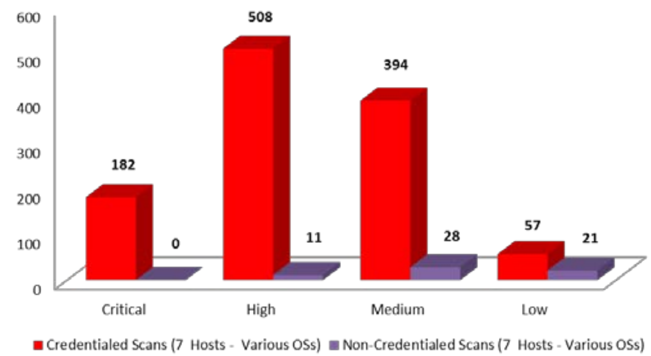
- Analyzed ~13 million lines of custom developed ground software using the OA tools
  - Mostly C/C++ and Java
    - » Identified 600 (11%) out of 5,250 third party components contained a combined 5,000 CVEs in addition to some risky open source licenses.





Vulnerability Assessment/Scanning

- Vulnerability scanning uses tools like Nessus, Foundstone, AlienVault, OpenVAS, Retina, SCAP, CIS Benchmarks
  - Don't confuse VA tools for SCA or OA tools
  - Identifies CVEs, misconfigurations, and compliance issues
  - Must be credentialed!!!!



- Example

HP Data Protector 7.0x < 7.03 build 108 / 8.1x < 8.15 / 9.0x < 9.06 Multiple Vulnerabilities (HPSBGN03580) (Bar Mitzvah)	The version of HP Data Protector installed on the remote host is 7.0x prior to 7.03 build 108, 8.1x prior to 8.15, or 9.0x prior to 9.06. It is, therefore, affected by the following vulnerabilities : - A security feature bypass vulnerability exists, known as Bar Mitzvah, due to improper combination of state data with key data by the RC4 cipher algorithm during the initialization phase. A man-in-the-middle attacker can exploit this, via a brute-force attack using LSB values, to decrypt the traffic. (CVE-2015-2808) - A flaw exists due to a failure to authenticate users, even with Encrypted Control Communications enabled. An unauthenticated, remote attacker can exploit this to execute arbitrary code. (CVE-2016-2004)	Upgrade to HP Data Protector 7.03 build 108 (7.03_108) / 8.15 / 9.06 or later per the vendor advisory.	CVE-2015-2808 CVE-2016-2004 CVE-2016-2005 CVE-2016-2006 CVE-2016-2007 CVE-2016-2008 OSVDB:117855 OSVDB:137412 OSVDB:137413 OSVDB:137414 OSVDB:137415 OSVDB:137416 CERT:267328 EDB-ID:39858 IAVA:2016-A-0110

Demos

- Have several tool demos for those interested
  - FortifyCLI
  - BlackDuck
  - Flawfinder
  - CPPCheck
  - FortifyScanWizard
  - OWASPDepCheck
  - Sonatype
- Have several CWE / bad code demos
  - Buffer Overflow
  - SQL Injection
  - String Injection
  - Integer Overflow
  - OS Command Injection

Lesson 1 – Buffer Overflow:

This lab is an example of a buffer overflow. The lesson uses a C program that has a vulnerability where an input string's length is not checked, allowing the string to overwrite other data.

About this CWE:

CWE-120: "A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold, or when a program attempts to put data in a memory area outside of the boundaries of a buffer. The simplest type of error, and the most common cause of buffer overflows, is the "classic" case in which the program copies the buffer without restricting how much is copied." Additional details (https://cwe.mitre.org/data/definitions/120.html)

About this CERT Rule:

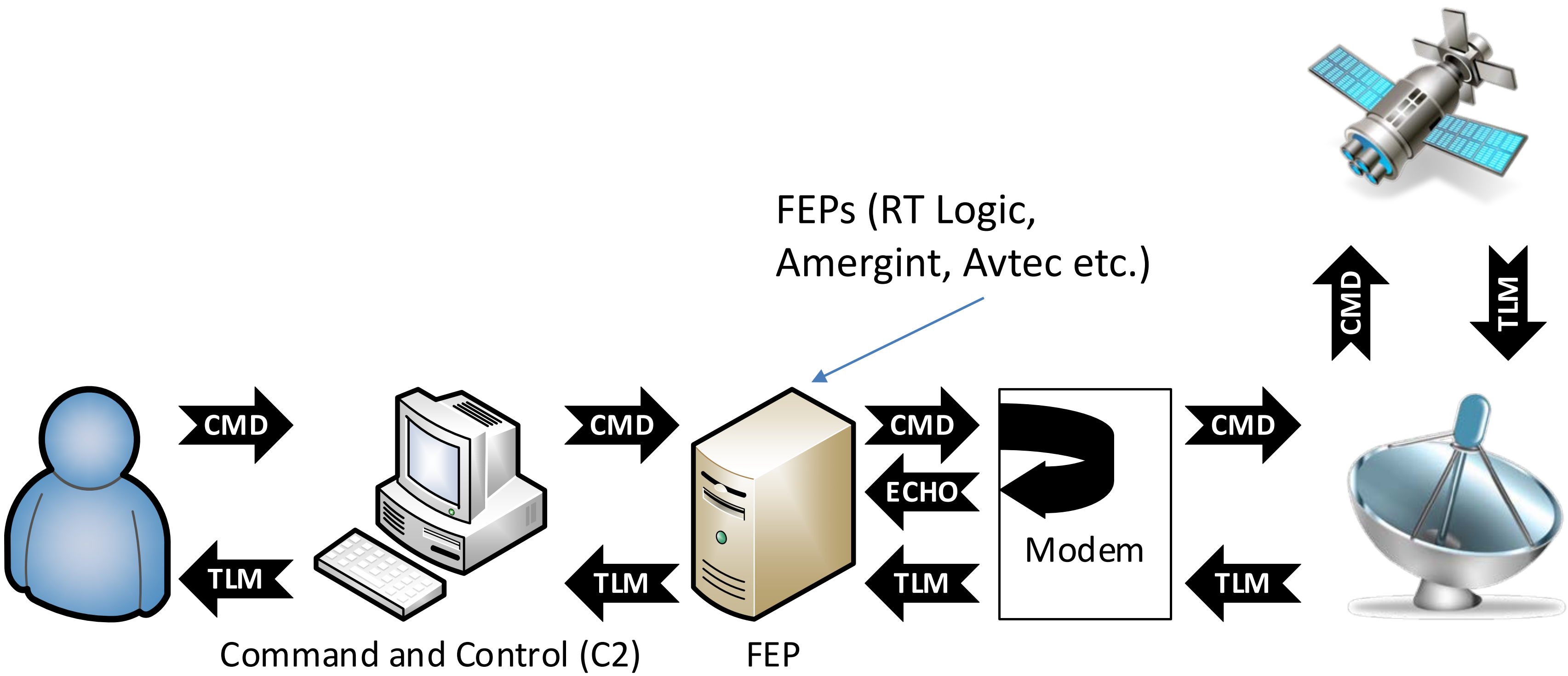
STR31-C: "Guarantee that storage for strings has sufficient space for character data and the null terminator. Copying data to a buffer that is not large enough to hold that data results in a buffer overflow." Additional details (https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator)

Real Life Example

Front End Processors

Unsecure Design Example

Scope for this Example



FEP: Commanding & Telemetry

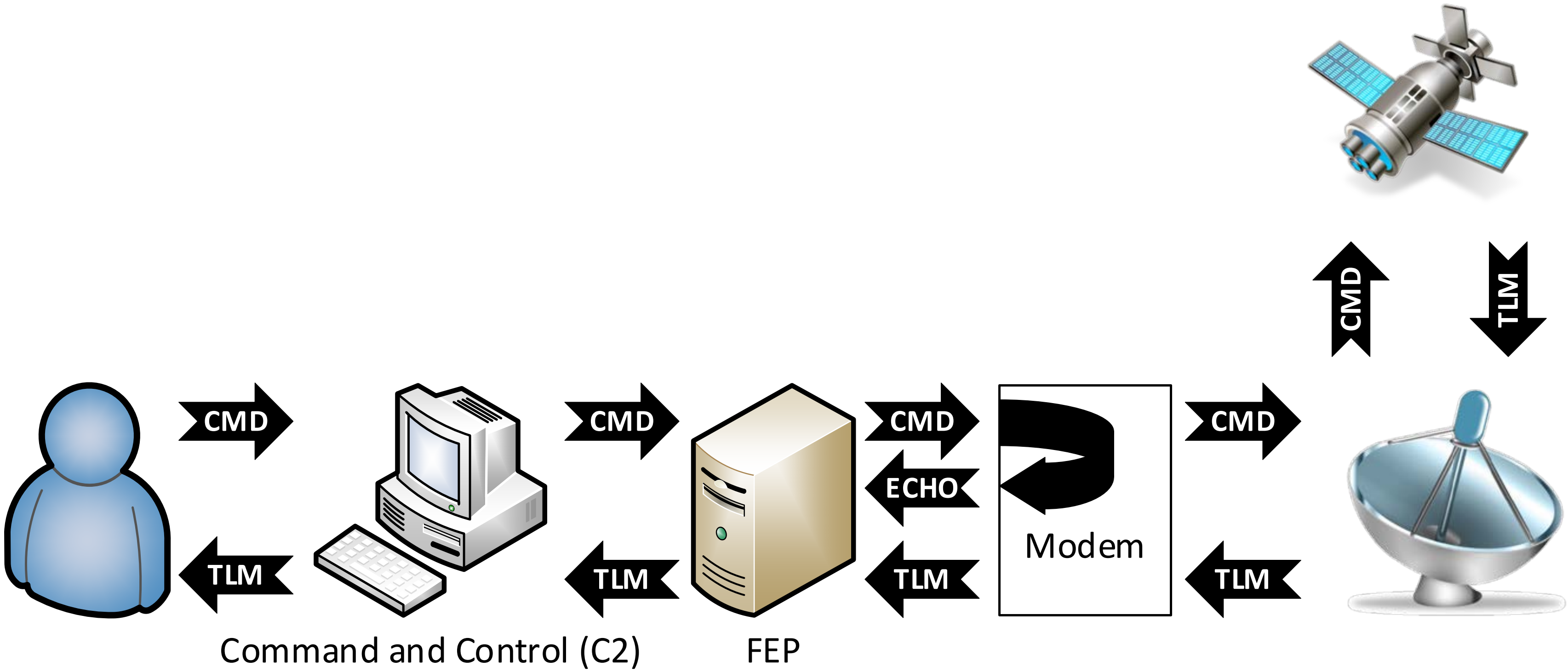
GSAW 2019

- Commanding
  - Command and Control (C2) Systems automate user processes:
    - Send command sequences
    - Translate mnemonics to binary commands
    - Set limits on commanding
    - Store logs of commands sent and telemetry received
  - C2 controls the FEP
  - Modem converts digital signal to analog signal (modulation)
  - Transmitter amplifies and transmits RF signal
- Telemetry
  - Receiver collects and amplifies RF signal.
  - Modem converts analog signal to digital signal (demodulation)
  - Command and Control (C2) Systems automate user processes:
    - Translate frames/sub frames of telemetry into calibrated data (decomm)
    - Set limits on telemetry
    - Store logs of commands sent and telemetry received

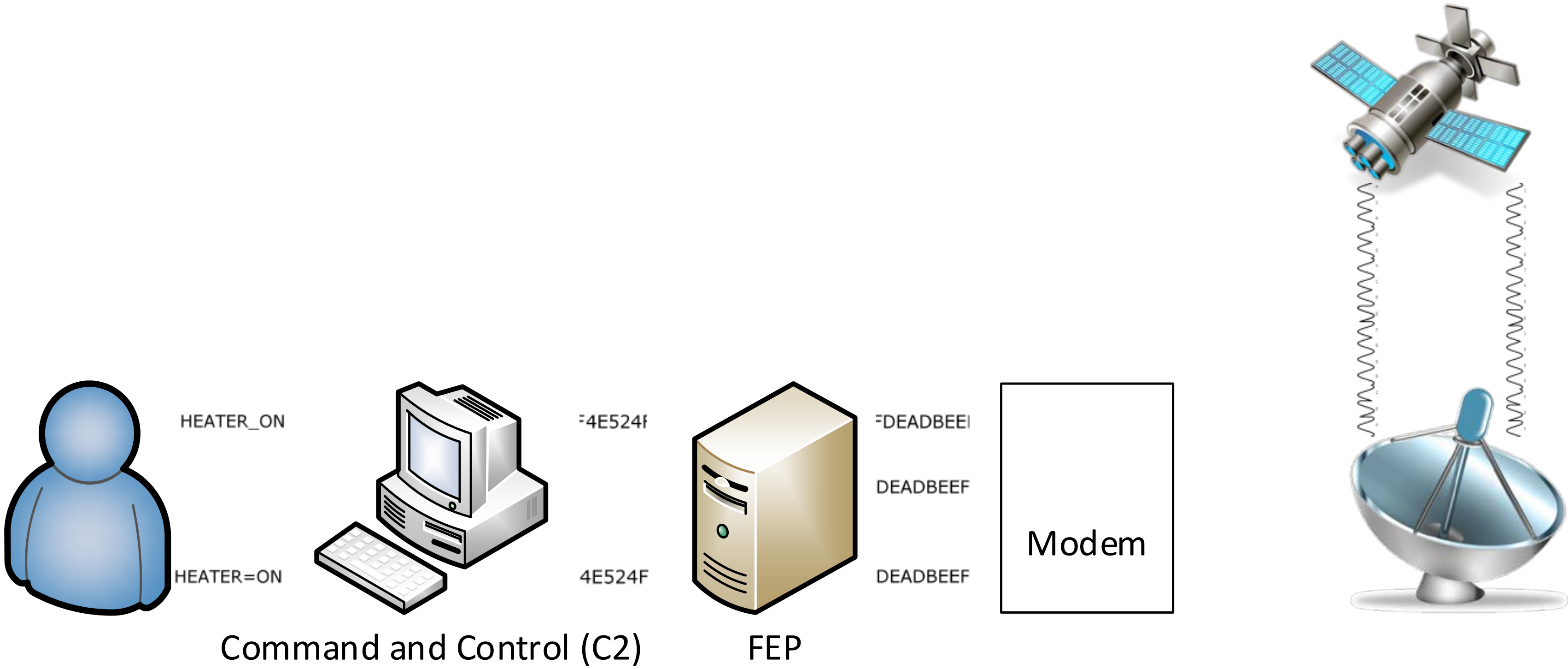
Example: FEP Providers

- RT Logic (1997, Colorado Springs, CO)
  - T501 Front-End Processor
- Amergint (2008, Colorado Springs, CO)
  - SoftFEP
- Avtec (1990, Fairfax, VA)/Ingenicomm (2010, Chantilly, VA)
  - Programmable Telemetry Processor
- GDP Space Systems
  - Components
- Acromamatics Telemetry Systems (1971, Santa Barbara, CA) /Delta Information Systems, Inc. (1976, Horsham, PA)
  - Model 2900AP PCI Telemetry System
  - Model 2900AP - Lightweight Rackmount PCI Telemetry System
  - Model 3022P - "Lunchbox" PCI Telemetry Data Processing System
  - Model 4000 - Compact "quick-look" Telemetry System
- Aventas Inc. (2002, Richardson, TX)

Command and Telemetry



Command and Telemetry



FEP: Threats & Mitigations

GSAW 2019

- Threats
  - The connectivity between a FEP and a modem varies between programs. It potentially contains many media and signal conversions.
  - Isolating issues to a FEP or the related infrastructure can be difficult.
  - The FEP and the related infrastructure is complex and functionality becomes prioritized over change management.
  - Defense of a FEP is expected on the boundaries, so they tend to have minimal end-point protection.
  - Testing of FEPs centers on functionality and requirements verification, not resiliency or reliability.
- Mitigations
  - Basic hardening produces significant gains in security posture.
  - FEPs have a relatively regular operations, meaning anomalous behavior should be relatively easy to recognize.
  - FEPs and the related infrastructure have a lot of redundancy and sparing.

Sample Attack #1 during PenTest

GSAW 2019

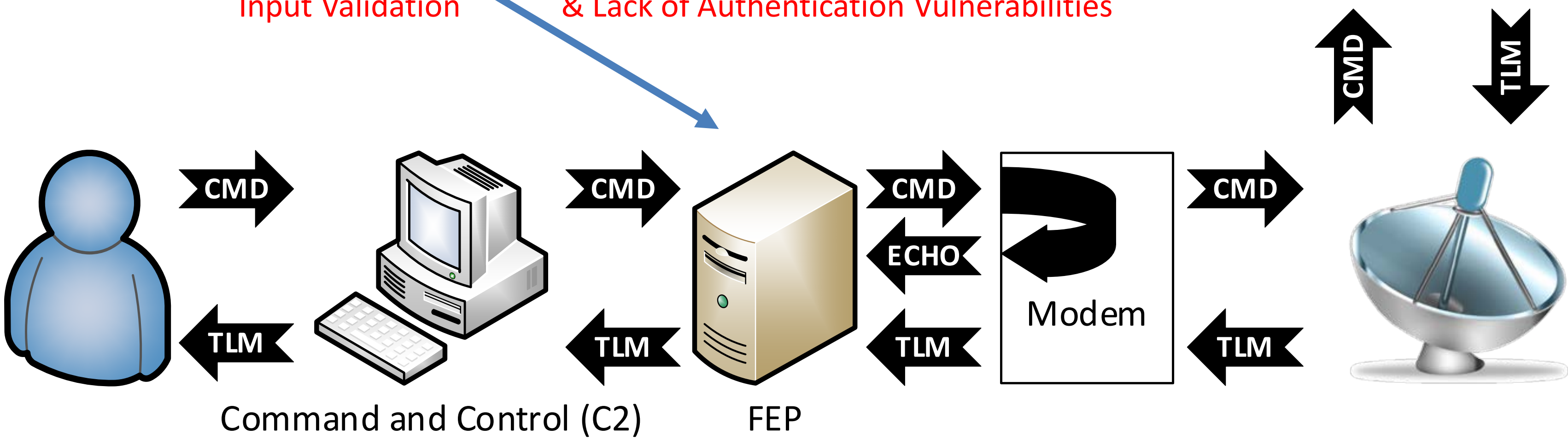
NIST SI-10

The software performs actions in the server's operating system using calls build in the "Python" scripting language. Several scripts exist in the URLs that execute tasks in the OS and return the output to the application.

NIST IA-3

The calls performed by these scripts are passed to the OS without the use of **<u>input validation</u>** or **<u>any authentication</u>** at the application/OS level. The use of these scripts creates a semi-shell environment where a user can execute many OS commands through the web browser.

Input Validation & Lack of Authentication Vulnerabilities



Sample Attack #2 during PenTest

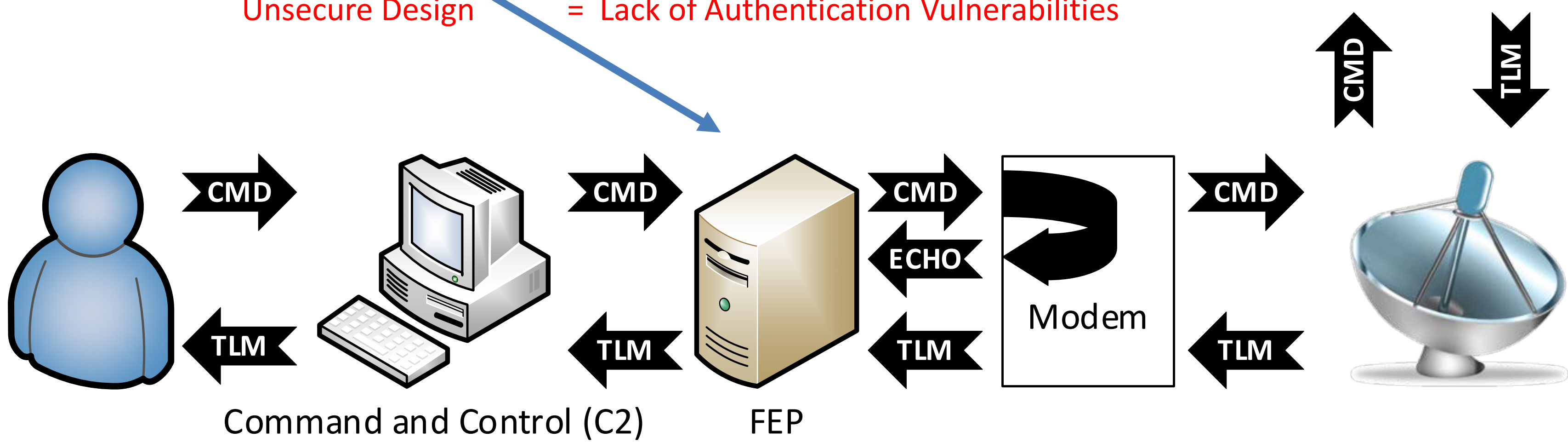
GSAW 2019

FEP intended design…. *"Just write the message to the socket, and read the reply. In fact, if you are so inclined, you can telnet to port xxxxx and enter the messages directly."* – Vendor Docs

NIST IA-3

Therefore, anyone with access to the network has the capability to send commands to these ports and reconfigure the FEP **unauthenticated**. If used as an attack vector, it affects the availability and integrity of the FEP system.

Unsecure Design = Lack of Authentication Vulnerabilities



Near Term Goals

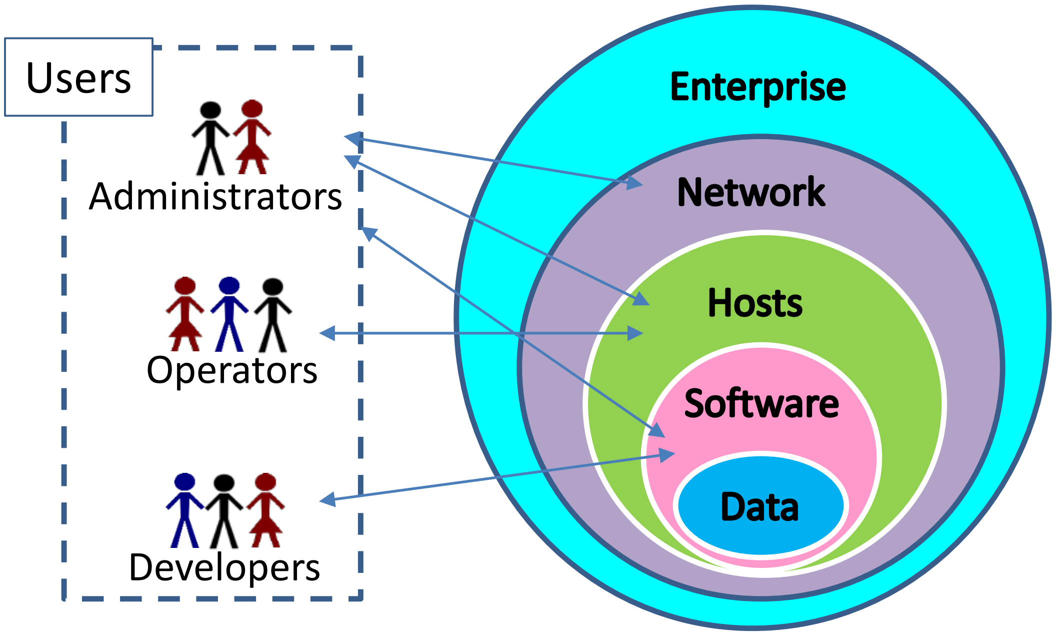
- You can't boil the ocean
  - Threat modeling takes time
  - Classifying CWEs takes time
    - Free to use NASA's list as a starter, NASA can share their customizable Access DB
  - Procuring VA, SCA, & OA tools takes time

- Discussion has been geared around how to reduce risk starting from inception of system
  - What about legacy systems? Let's discuss....

Near Team Goals (cont.)

GSAW 2019

- Promote Defense-in-Depth



Services Provided, Received

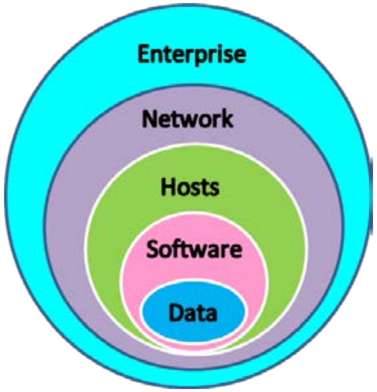
- Software runs on a Host
- Hosts are interconnected via the Network
- Developers code the software builds, updates, and patches in a non-operational environment
- Operators use the Hosts to interact with the Network and Software appropriately
- Administrators manage the Hosts and Networks while installing/configuring Software

Additionally:

- Software handles Data
- Mission runs within an Enterprise

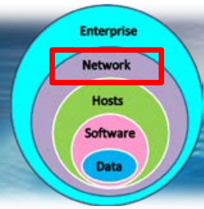
Defense in Depth (DiD)

- Secure software development is extremely important but DiD is key to protecting mission assets
- In space mission environments, DiD can be difficult
  - Older architectures/technology
    - Unsupported operating systems, older hardware, etc.
  - Shared architectures/technology
    - Mission X doesn't own all layers of the defense



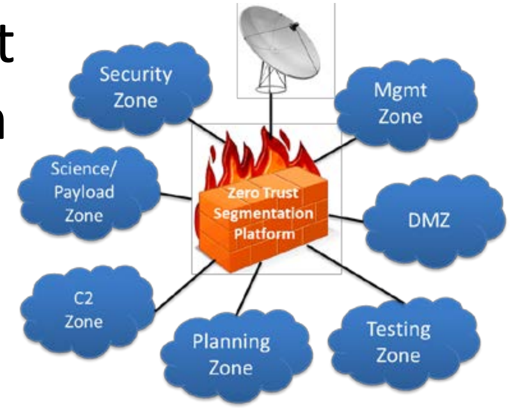
- Sometimes vulnerable software depends on something that is out of their control to protect it
  - Do you trust the Network Engineers? Should you?
  - Do you control the host level configuration?

DiD (cont.)



GSAW 2019

- Work with Network Engineers to implement enclaves/network zoning and/or encryption
  - Migrate to a "zero trust" architecture
    - Vulnerabilities injected by Mission X may affect Mission Y
  - Investigate software defined networking



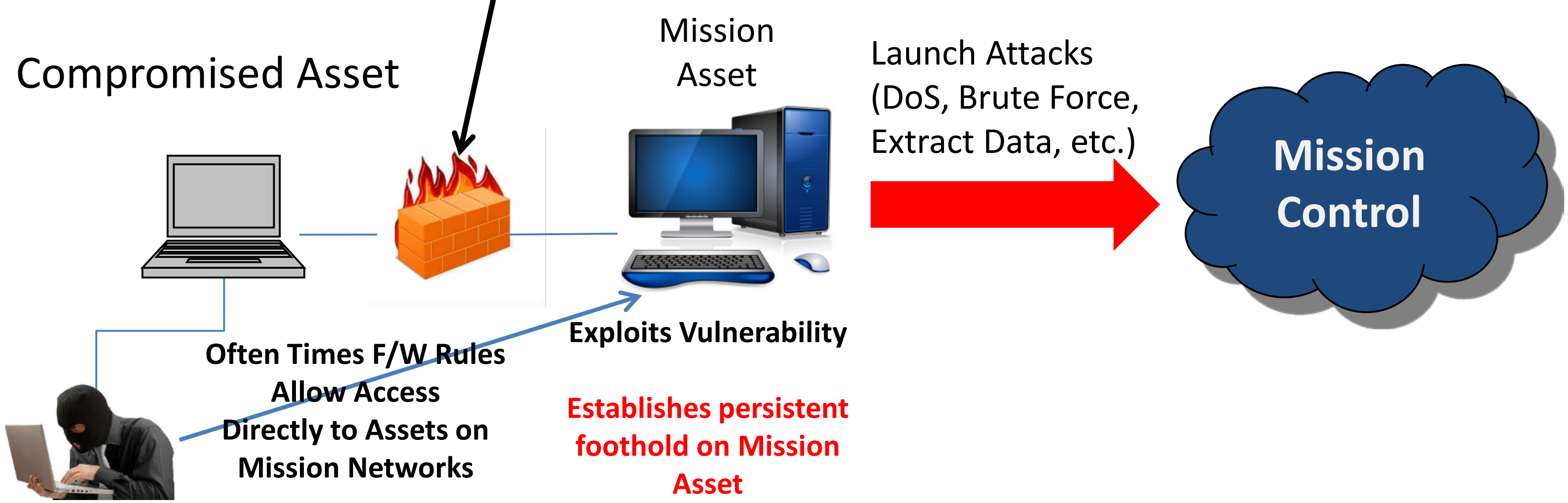
- Understand and eliminate pivot points
  - From networking perspective, software security perspective, host level security
- Increase attack depth or eliminate all together

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

Example SW Impacting Mission

GSAW 2019

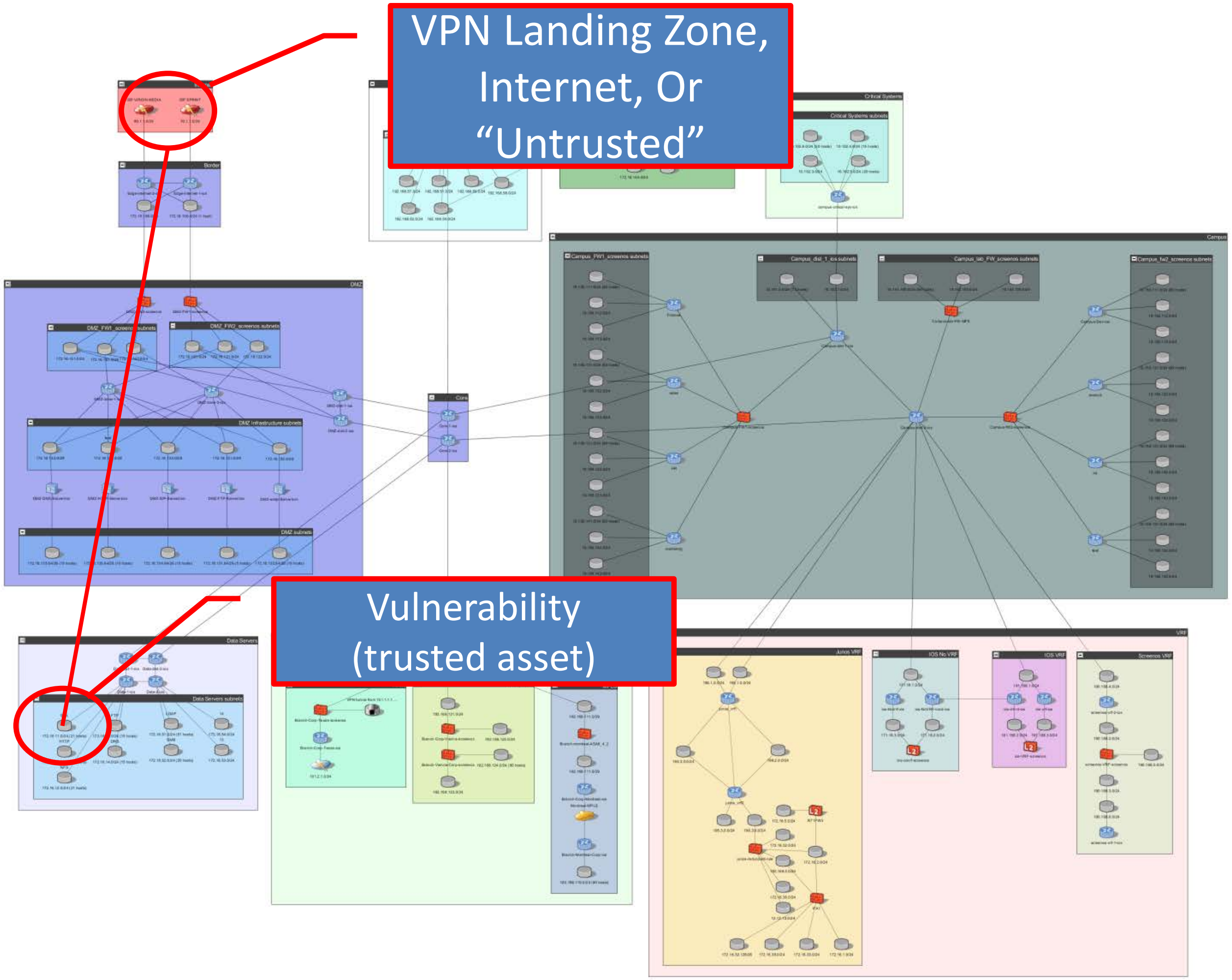
Can't assume protection from Firewall. Need "Defense in Depth". Can't assume if knocking on door, that they are supposed to be there.



This example will depict how vulnerability on non-critical (trusted) asset within a network can potentially impact critical mission assets

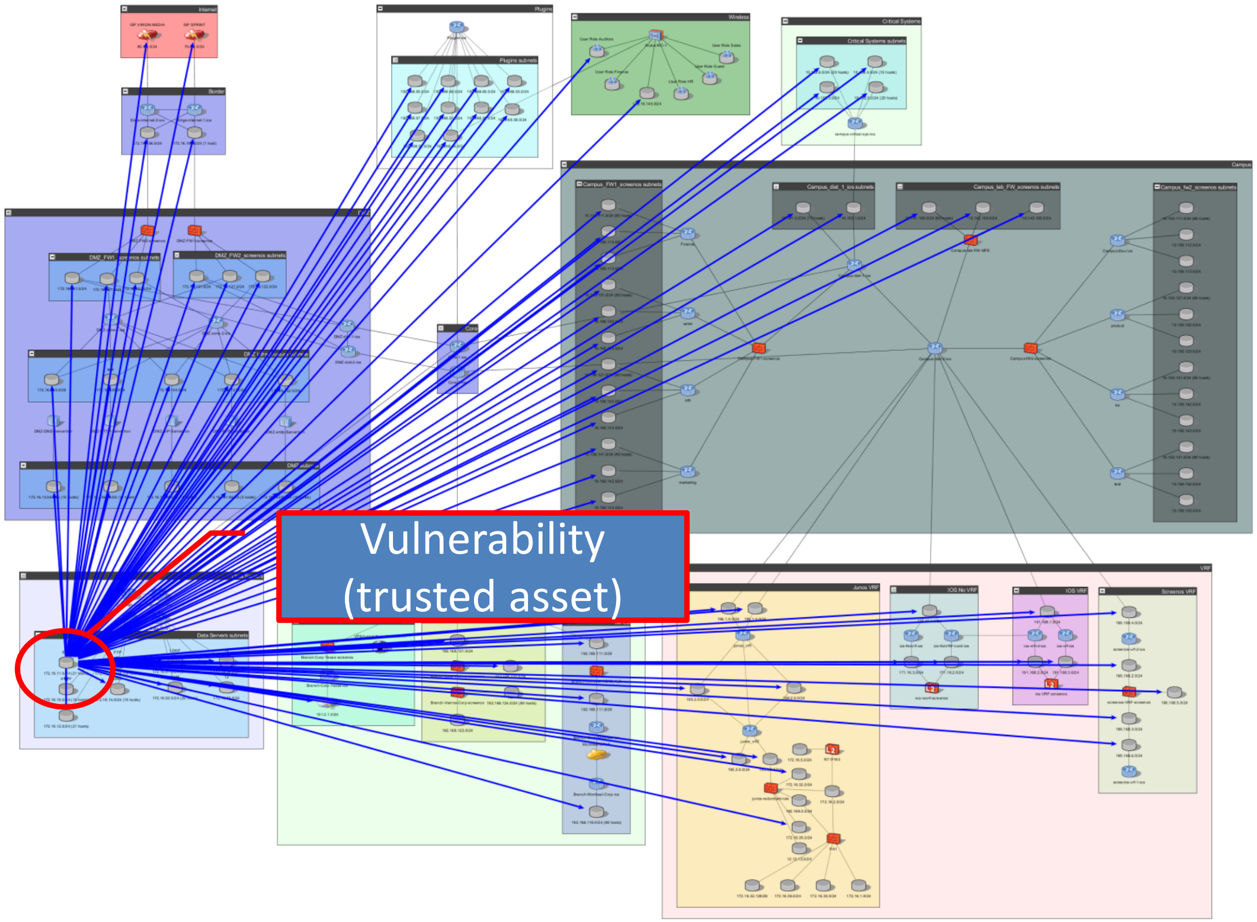
Sample Exposure

GSAW 2019



Demonstrates that a pathway exists from the VPN Landing Zone, Internet, Or Untrusted to a vulnerable asset in non-zero trust network

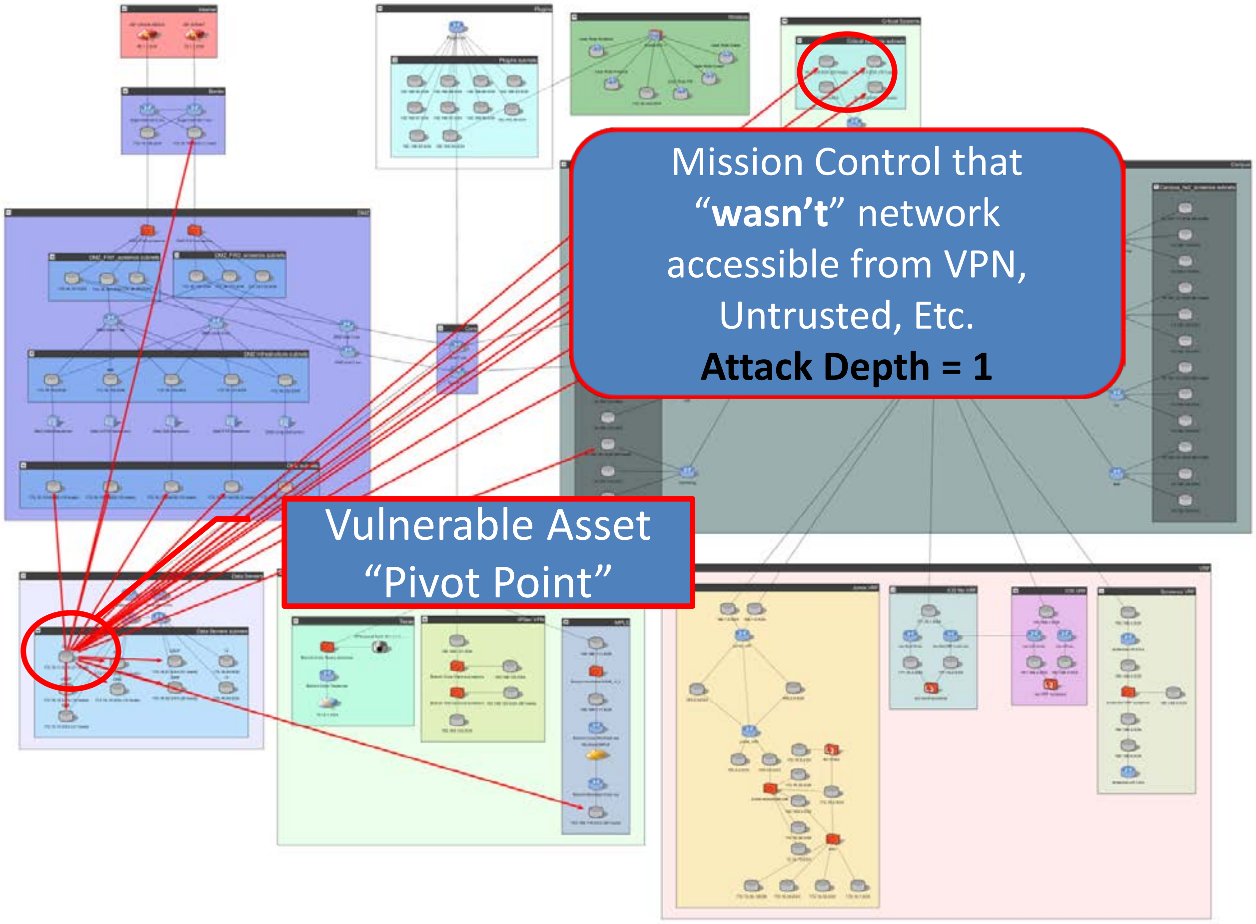
Sample Exposure



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

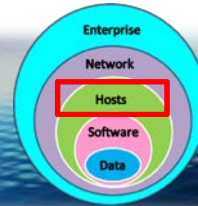
Sample Exposure

GSAW 2019



Demonstrates potential vulnerabilities that could be exploited from this server

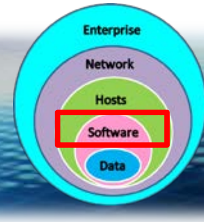
What To Do Now?



GSAW 2019

- In space 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 2019

- 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 next slides)
  - Utilize Best Practices (see backup slides) 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
    - 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
    - Classify most dangerous CWEs for Ground Systems
      - Use NASA's or create you own based on your mission and threats

Low Hanging Fruit - Unsafe Functions

GSAW 2019

- 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
memcpy
strcat
strcmp
strcpy
strlen

sprintf
strncpy
_iota
sscanf
wcslen

- Most of these are unsafe due to allowed writing with no regard to buffer size
  - strncpy, _iota, sscanf, & wcslen 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

Low Hanging Fruit - CERT Rules

GSAW 2019

- 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.)

- 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
      - Cppcheck
      - Rosecheckers
      - Splint
      - Find Bugs
      - RATS
      - Flawfinder
      - SWAMP ★

Current Trends in the Field

GSAW 2019

- Lack of Defense in Depth (DiD) – Layered Security
  - Border protection (i.e. Firewalls) is depended on too much
- Network management and insight is insufficient
  - Lack of ground-truth topology
  - Lack of monitoring, alerting and knowing what is required or “normal”
- Industrial Control Systems are Vulnerable
  - Not designed or operated with cyber resiliency in mind
- Patching and Security Testing is not a Priority
  - Mission trumps all and patching/testing is delayed or never done
  - Lack of vulnerability scanning, code analysis, & dynamic analysis
    - Vulnerable COTS, Open Source, and Custom Code on networks
- Limited Staffing Investment
  - Lacking appropriate training on technology/tools and knowledge
  - Staff is overtasked with non cyber activities
- Programs are waiting for Continuous Diagnostics and Mitigation (CDM) Phases 1 – 3 deployment to provide “security”

Phase 1:

- HWAM – Hardware Asset Management
- SWAM – Software Asset Management
- CSM – Configuration Settings Management
- VUL – Vulnerability Management

Phase 2: Least Privilege and Infrastructure Integrity

- TRUST –Access Control Management (Trust in People Granted Access)
- BEHV – Security-Related Behavior Management
- CRED – Credentials and Authentication Management
- PRIV – Privileges

Phase 3: Boundary Protection and Event Management for Managing the Security Lifecycle

- Plan for Events
- Respond to Events
- Generic Audit/Monitoring
- Document Requirements, Policy, etc.
- Quality Management
- Risk Management
- Boundary Protection (Network, Physical, Virtual)

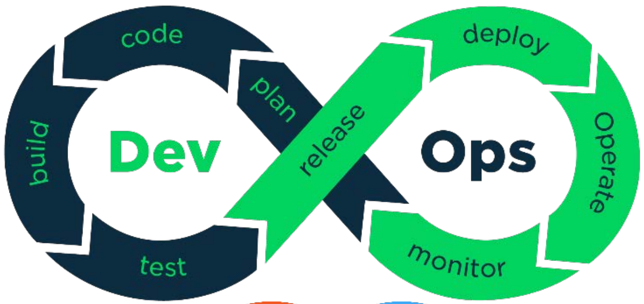
Where are we going.....

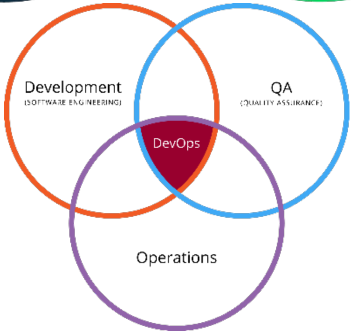
Cloud & DevOps

GSAW 2019

- Cloud and DevOps joined at the hip
  - Major cloud vendors have numerous native services to support DevOps
  - Services are organized into reference architecture
- From customer perspective everything is software
  - AWS API has thousands of commands or hooks
  - Facilitates automation but destroys Configuration Management
  - Monitoring becomes mitigation
- Cloud introduces new vectors
  - Misconfiguration leading to inadvertent disclosure (i.e. S3 buckets, Drive/Box)
  - Token based attacks (i.e. GitHub harvesting)

DevOps is the combination of **cultural philosophies**, **practices**, and **tools** that increases an organization's ability to deliver applications and services at **high velocity**: evolving and improving products at a **faster pace** than organizations using traditional software development and infrastructure management processes. This **speed** enables organizations to better serve their customers and **compete more effectively in the market**.





We need SecDevOps

GSAW 2019

- SecDevOps / DevSecOps / DevOpsSec is the process of integrating secure development best practices and methodologies into development and deployment processes which DevOps makes possible
- Needs: Tooling, Processes, Culture
  - Automation (SCA, OA, VA, Dynamic Testing, etc.)
  - Need defined cyber roles within DevOps
  - Separation of duties – eliminate COI where developers fully maintain development and sometimes ops environment
  - Testers/validators must work together with DevOps team and must have access to development environment tools to write security focused tests
  - Incentivized to create secure software

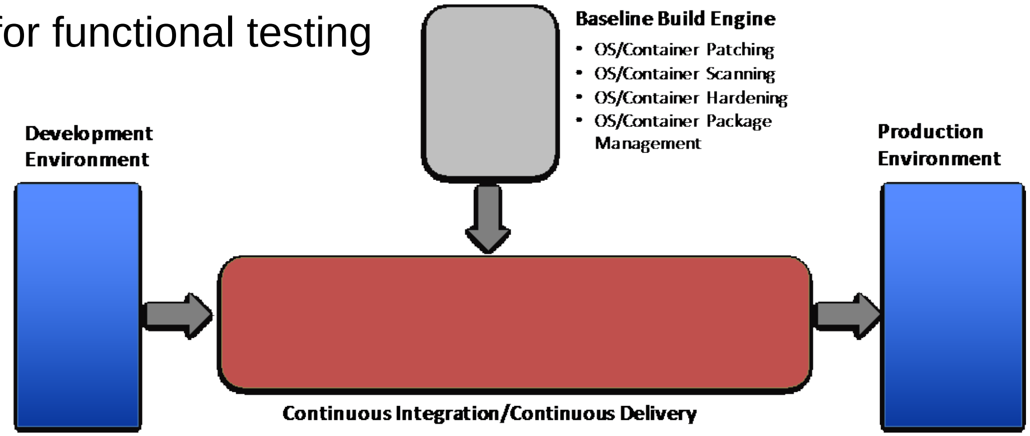
How Do We Integrate Security

GSAW 2019

- **Organization**
  - Create the Cyber Security Developer role
  - Create the Cyber Security Tester role
- **Development**
  - Architecture must be documented
  - Security functionality and design patterns should be abstracted in the architecture for developers to leverage
    - Encryption
    - Data Validation
    - Logging
    - Error Handling
    - Authentication
    - Access Control
- **Validation**
  - DevOps environments need to be documented and assessed
  - CI/CD Pipelines need to be documented and assessed
    - DevOps environments are not usually considered in scope of security assessments (i.e. they are not "in production")

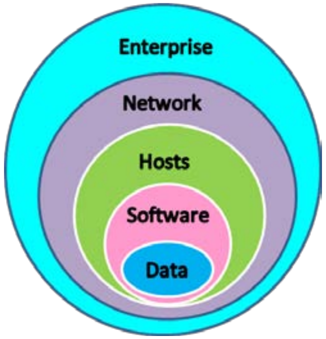
Some Places to Start

- Low-level security requirements would have the most impact but would require large scale culture change
- Two examples of where to start
  - Example #1: CI/CD Reference Architecture
    - Library Standardization (e.g. approved repos, block vulnerable libraries {Nexus Firewall}, etc.)
    - Automated Scanning/Patching (e.g. SCA, VA, OA, dynamic)
    - Automated Configuration Compliance Check (e.g. STIGs)
  - Example #2: Hardened Baseline
    - Requires centralized distribution (e.g. approved containers, VMs, etc.)
    - Requires high level policy on what is acceptable/required
    - Should be starting point for functional testing



Summary: Shortlist of takeaways

GSAW 2019



- Network defense is too highly depended on, so build defense in depth
  - Where changes can't be made (e.g. patching or SW updates) build protections everywhere around software
- For old and new SW, refer to low hanging fruit slides
  - Integrate SCA, OA, VA – don't have to fix everything but prioritize (e.g. top CWEs)
  - Slowly integrate best practices. Don't attempt to boil the ocean
- Incentivize secure SW development
  - Culture change on cheapest solution is "best" solution
- DevOps needs to migrate to SecDevOps
  - Eliminate COI where developers fully maintain development and/or operating environment
    - Testers/validators must work together with DevOps team and must have access to development environment tools to write security focused tests

Backup Slides

References / Links

- Zero Trust
  - http://csrc.nist.gov/cyberframework/rfi_comments/040813_forrester_research.pdf
  - http://www.ndm.net/firewall/pdf/palo_alto/Forrester-No-More-Chewy-Centers.pdf
- NIST 800-53
  - http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf
- Space Security
  - http://www.spacesafetymagazine.com/aerospcae-engineering/cyber-security/cyber-crime-cyber-space-outer-space/
  - http://www.nbcnews.com/tech/security/hacked-space-are-satellites-next-cybersecurity-battleground-n658231
  - http://www.homelandsecuritynewswire.com/dr20160922-space-cybersecurity-s-final-frontier
  - Security Threats: https://public.ccsds.org/Pubs/350x1g2.pdf
  - https://www.chathamhouse.org/sites/files/chathamhouse/publications/research/2016-09-22-space-final-frontier-cybersecurity-livingstone-lewis.pdf
- Misc.:
  - DoD: http://www.cyberdefensereview.org/2015/12/10/mission-command-primer/
  - NASA Networks: http://www.gao.gov/new.items/d104.pdf
  - CIS Top 20: https://www.sans.org/media/critical-security-controls/SANS_CSC_Poster.pdf

Links

GSAW 2019

CCSDS

- major space agencies of the world - http://public.ccsds.org/participation/member_agencies.aspx
- multi-national forum - http://cwe.ccsds.org/

Policies and such

- Program Protection & System Security Engineering - 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
- 7150.2B - 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
- 800-53 - 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
- 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?
- Microsoft Security Development Lifecycle - https://www.microsoft.com/en-us/sdl/

SCA/OA

- C - 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
- JAVA - https://www.securecoding.cert.org/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java
- Klockwork - http://www.klocwork.com/products/insight
- Fortify - http://www8.hp.com/us/en/software-solutions/software-security/
- Flexelint - http://www.gimpel.com/html/flex.htm
- CodeSonar - http://www.grammatech.com/codesonar
- Sonatype - http://www.sonatype.com/
- BlackDuck - https://www.blackducksoftware.com/products/black-duck-hub
- Report - 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

Info and Training

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

Links (cont.)

GSAW 2019

- Security Development Lifecycle (SDL) Banned Function Calls - 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
- Flawfinder - http://www.dwheeler.com/flawfinder/
- Cppcheck - http://cppcheck.sourceforge.net/
- Rosecheckers - http://sourceforge.net/projects/rosecheckers/
- Splint - http://www.splint.org
- RATS - https://code.google.com/p/rough-auditing-tool-for-security
- Flawfinder - http://www.dwheeler.com/flawfinder
- SWAMP - https://continuousassurance.org
- Find Bugs - http://findbugs.sourceforge.net/

Mitre Links

- CWE - https://cwe.mitre.org/
- CVE - https://cve.mitre.org/
- CAPEC - https://capec.mitre.org/

Tools

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

Links (cont.)

IDA Work

- report - 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
- NSA's CAS - 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

Standards

- C - 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
- 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

Acronym List

Acronym List			
ACL	Access Control Lists	NIST	National Institute for Standards and Technology
C2	Command and Control	OPM	Office of Personal Management
CIS	Center for Internet Security	PIM	Privileged Identity Management
CND	Computer Network Defense	SANS	
DiD	Defense in Depth	SIEM	Security Incident and Event Manager
DLP	Data Loss Prevention	SPAN	Switch Port for Analysis
DMZ	Demilitarized Zone	SSH	Secure Shell
HW	Hardware	SSL	Secure Sockets Layer
IDS	Intrusion Detection System	SW	Software
IONet	Internet Protocol Operation Network	TAP	Test Access Point
IP	Internet Protocol	TC	Telecommands
IPS	Intrusion Protection System	TM	Telemetry
IT	Information Technology	VPN	Virtual Private Network
MOC	Mission Operations Center	WSC	White Sands Complex
NASA	National Aeronautics and Space Administration		

Some Secure Coding Best Practices

GSAW 2019

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.

Some Secure Coding Best Practices

GSAW 2019

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.

Some Secure Coding Best Practices

GSAW 2019

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.

Some Secure Coding Best Practices

GSAW 2019

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.

Back

IA/Cyber Lessons Learned in Space Systems

GSAW 2019

Area	Challenge Faced	Potential Solutions
Overall Approach	• Adding security to in-process developments • Incorporating security into existing processes • Newness of artifacts to Development process, variations in artifact quality	Work together to incorporate as part of engineering and risk process
SSP	• Using FIPS categorization to baseline control set without supplementation for mission-specific threats • Defining customizations based on as-is design vs. identifying control substitutions or other mitigating factors—identification / documentation of residual risk • Definition of SSPs around development of the ground segment (e.g. workstations, servers) instead of system/mission • Sometime there are no SSPs for the spacecraft system	Projects ensure that asset protection is part of the engineering process, with results captured in the SSP. Promote best practices and lessons learned across projects
Security Allocation to Requirements	• Security is not a distinct domain • Requirements defined prior to availability of SSP, PPP, or Threat Summary	Ensure a top-down approach to addressing security. Focus on intent of controls and not compliance

Example Security Analysis (Part 1)
Threats and Vulnerabilities

Part 1: Assess Mission for Credible Threats, and Vulnerabilities
-Credible threats based on situational environment
-Vulnerabilities assessed by establishing security risk to system

Part 2: Develop Security Strategy
- Develop security architecture and ConOps
- Capture in Project Protection Plan

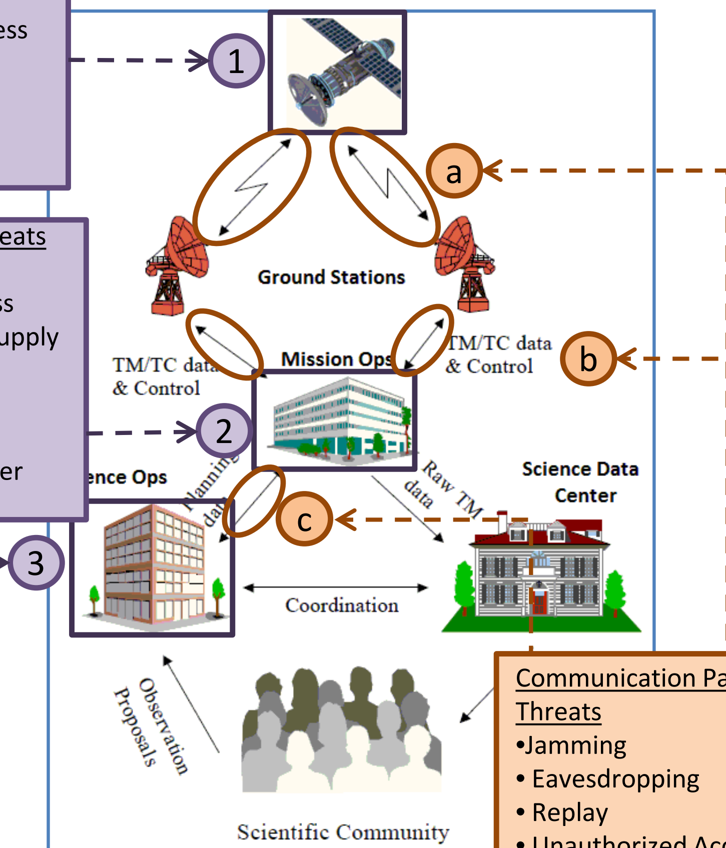
Part 3: Select and Tailor Security Controls in System Security Plan (SSP)
- Many controls software based

GSAW 2019

- Document credible threat environment, identify vulnerabilities

<u>Satellite Threats</u>
- Replay
- Unauthorized Access
- Software Threats
- Eavesdropping
- Denial of Service
- Data Modification

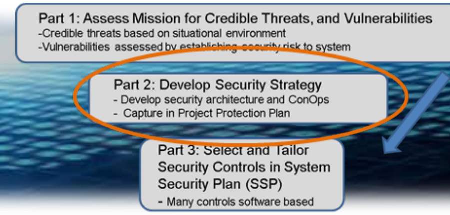
<u>Ground Element Threats</u>
- Replay
- Unauthorized Access
- Software Threats/Supply Chain
- Denial of Service
- Social Engineering
- Threat-Agent/Insider Threat



<u>Communication Path Threats</u>
- Jamming
- Eavesdropping
- Replay
- Unauthorized Access
- Traffic Analysis
- Data Modification
- Supply Chain

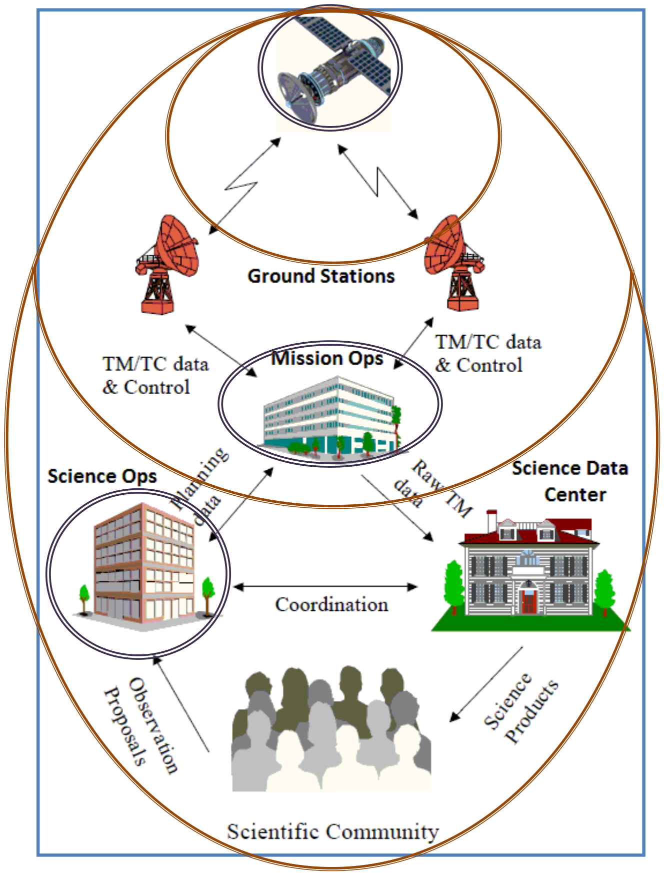
- *Credible threat environment (notional)*
  - *Satellite*
  - (1) *Mission Ops*
  - (2) *Science Ops*
  - (3) *Ground station – Satellite Links*
  - (a) *Mission Ops - Ground Stations*
  - (b) (c) *Science Ops (evaluate all points of entry) – Ground Station +*
- *Three types of threat groups identified*
  - *Communication paths*
  - *Ground elements*
  - *Satellite*
- *Establish risk using Confidentiality, Integrity and Availability*
  - *Assess that communications paths and ground elements pose high risk*
  - *Assess that satellite poses low-moderate risk (assuming other system aspects are secure)*

Example Security Analysis (Part 2)
Security Strategy



GSAW 2019

- Project survivability strategy against credible threats, vulnerabilities, and acknowledge evolving threat environment
- Strategy defined in terms of interfaces and information types (establish security perimeters and how strong they need to be)



- Security strategy is at element level <u>and</u> at system level to arrive at acceptable risk posture
  - For example, if the Mission Ops and command interface into a spacecraft is secure, perhaps less security is needed within the satellite
- *Candidate security strategy for SC FSW*
  - *Protect the commanding path*
  - *Perform command authentication*
  - *Command traffic analysis*
  - *Provide satellite software resiliency to common weakness enumerations*

Example Security Analysis (Part 3)
Security Controls

Part 1: Assess Mission for Credible Threats, and Vulnerabilities
-Credible threats based on situational environment
-Vulnerabilities assessed by establishing security risk to system

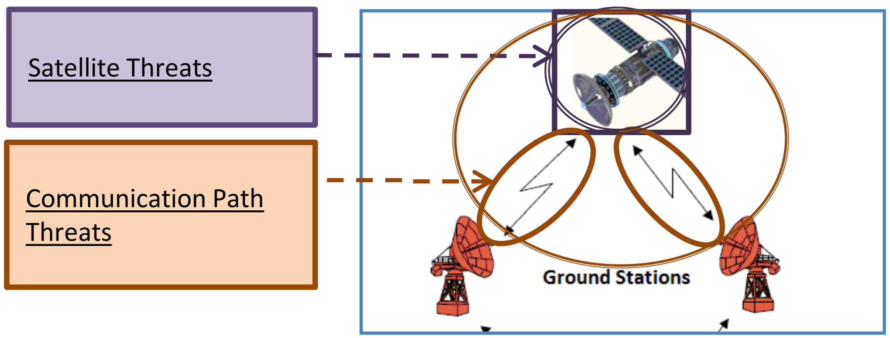
Part 2: Develop Security Strategy
- Develop security architecture and ConOps
- Capture in Project Protection Plan

Part 3: Select and Tailor Security Controls in System Security Plan (SSP)
- Many controls software based

GSAW 2019

- Once threats, perimeters (interfaces and information types established), engineering process to select controls and tailor accordingly

Candidate SC FSW Threats, Perimeters



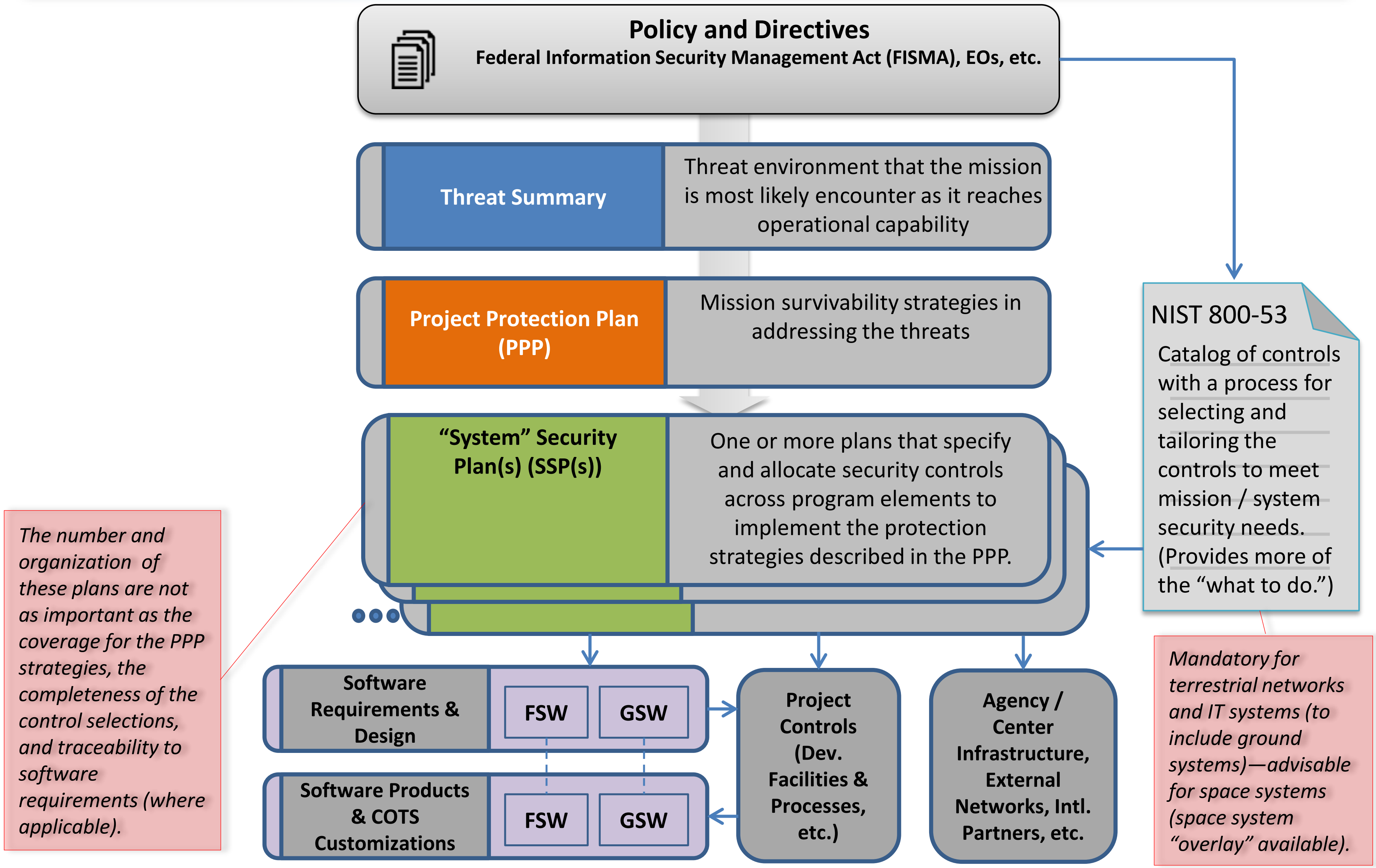
Candidate security controls based on planned strategy

Strategy	Candidate Security Control	SW
Command Path	Encryption	X
Cmd Authentication	Protocol	X
Command Traffic	Monitoring	
Software Resiliency	Coding Standards	X

- Establish security categorization
- Select Controls, based on 800-53 analysis, system specific tailoring
  - Required Controls
  - Supplemental Controls
- Consider
  - Data in Motion, Data at Rest, Data in Use
  - Strength of the control, pervasiveness of threats
- Hints:
  - Sometimes one control addresses multiple threats, collateral security
  - For spacecraft software, SC and SI are the most relevant control families
  - Controls may already be addressed through design or fault management (e.g., SI-10(3)), e.g. applying a robust set of security controls may simply require taking credit for what is already being done

Governance / Relationships Between Expected Artifacts / Decomposition of Security Requirements

GSAW 2019

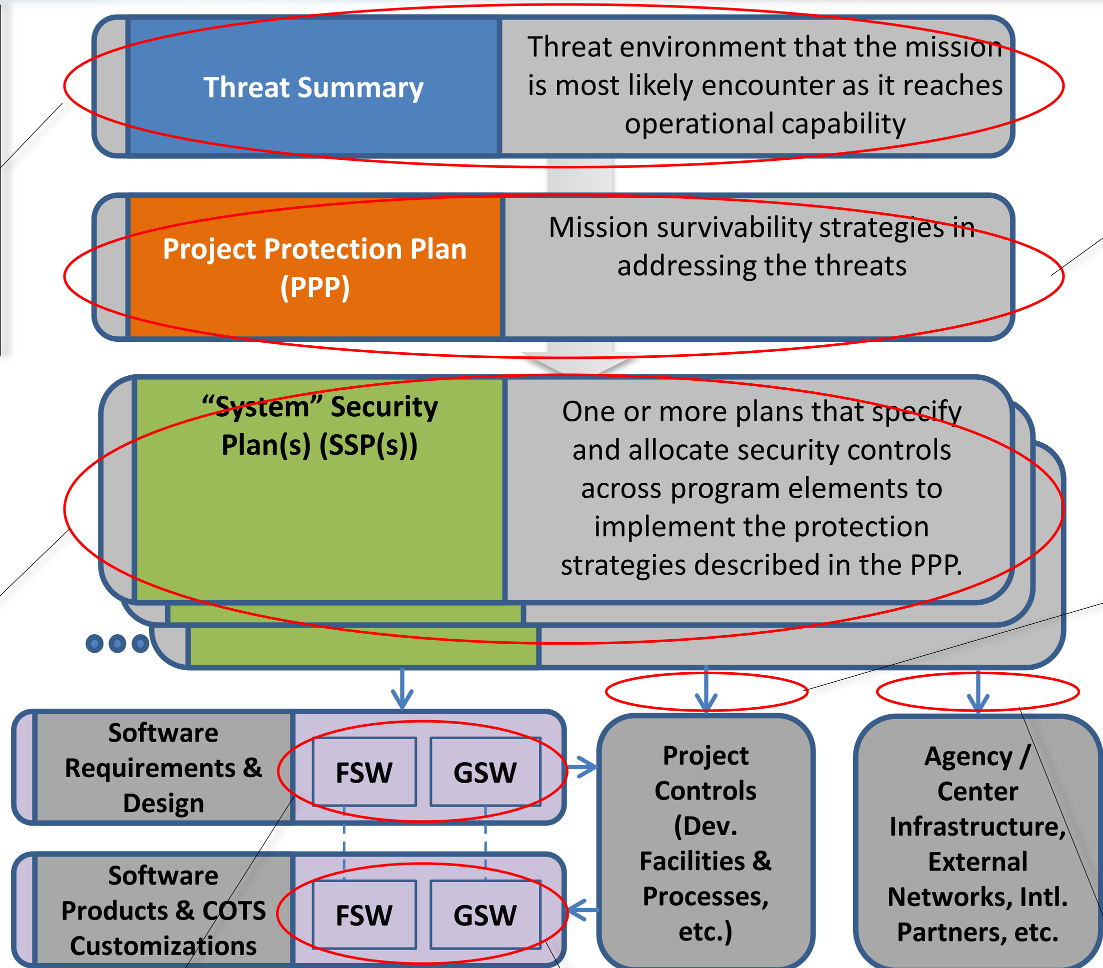


Points of Assurance

GSAW 2019

The project has a Threat Summary—or the PPP contains information—that indicates the project has taken into account the full range of threats appropriate to its mission type, capabilities, and assets.

System-level plans fully integrate the protection strategies from the PPP are traceable to control selection, allocation tailoring decisions at all levels of the system design along with any corresponding system specifications. Additionally, these decisions are based on an appropriate categorization of the specific data and assets being protected in each instance ensuring risk is mitigated to a level consistent with the project's risk tolerance (as defined in the PPP).



The PPP contains a comprehensive set of project survivability and protection strategies addressing the full range of threats and vulnerabilities that exist or are likely to exist throughout its lifecycle. Also, it contains an assessment of risk showing how the strategies mitigate the project's risk to an acceptable level.

Plans and specifications for programmatic controls such as secure development and acquisition processes, physical and personnel security, change control, and routine plan maintenance are complete and consistent with PPP project protection strategies and risk tolerance.

Use of outside systems, networks, and controls are fully described with supplemental controls applied as needed to mitigate risk.

Controls allocated to software are traceable down to specific software modules and completely and correctly specify the control

Controls implemented in software perform as specified. Software products are robust and free from:

- *Defects that many induce additional vulnerabilities or bypass controls (CWEs)*
- *Undocumented / unspecified functionality*