Trusted Software Development:
"A Pathway to More Assured Software"

Dr. Ben Calloni, P.E.,
CISSP, OCRES-AP
Lockheed Martin Fellow,
Software Security

Published by the DHS Software Assurance Forum with permission

© 2010 LOCKHEED MARTIN CORPORATION

Agenda

- **Part 1 – Security and Trusted Development**
  - ***Concepts & Definitions***
  - ***Background, Scope and Applicability***
  - ***Security-Relevant Requirements/Trusted Software***
  - ***Trusted Software Process***

- **Part 2 –Implementation Guidance**
  - ***Software Security***
  - ***Secure Software Development Principles***
  - ***Software Vulnerabilities***
  - ***Static Code Analysis***

- **Wrap up**

© 2010 LOCKHEED MARTIN CORPORATION

Safety and Security Requirements

- **Question 1 – If a safety critical component, such as flight control or mission system, calculated the correct data value each iteration but delivered that value meeting the timing deadline requirement only 50% of the time, can the system be trusted to behave in a safe manner?**
- **Question 2 - If the mission systems calculates the correct data values but does not meet the given security requirements, can the system be trusted to behave in a secure manner?**
- **Too much of the commercial enterprise market and software only focuses on correct functionality!**
- **Security, like safety, has to be Built-in, not bolted on to meet Certification and Accreditation requirements (DITSCAP, DIACAP, Common Criteria)**

© 2010 LOCKHEED MARTIN CORPORATION

Objectives

Provide information to:

1. *Achieve a common level of understanding of Information Systems Security (INFOSEC) concepts*
2. *Increase awareness of requirements and responsibilities associated with trusted software development*
3. *Provide consistent application of Trusted Software Development Standards*
4. *Encourage dialogue and interaction with Information System Security Professionals (ISSP) and the Software Developers*
5. ***Increase likelihood of achieving certification and ultimately Government approval to operate***

© 2010 LOCKHEED MARTIN CORPORATION

Concepts and Definitions

© 2010 LOCKHEED MARTIN CORPORATION

Imprecise Yet Intertwined Terminology

- **Is your home safe?**
  - ***FROM What***
    - Fire
    - Wind
    - Hail
    - Flood
    - Burglary
    - etc.
- **If the Threat Agent is an Act of Nature, Mechanical Failure, or Poor Engineering – Safe!**
- **If the Threat Agent is a malicious entity – Secure!**

© 2010 LOCKHEED MARTIN CORPORATION

World's Safest Automobile?

- **Reliability, Availability, Dependability, etc**
  - ***Safety is not necessarily achieved by these "…ilities"***
- ***Pure safety must be balanced against useful functionality!***

© 2010 LOCKHEED MARTIN CORPORATION

A Commercial Offering of a Secure Car!

Start with a "safe" COTS automobile with great 'ilities!

BMW 760LI High Security

© 2010 LOCKHEED MARTIN CORPORATION

A Secure Car! BMW 760 LI High Security

Withstands Armour Piercing Bullets, Grenade Blasts, gas attacks.

http://www.firstdefense.com/html/Armoured_BMW_Series.htm

- **Certified to meet B6/B7 requirements of German Federal Crime Office.**
  - ***Source:*** **www.carat-duchatelet.be/s-class/ballb7.htm**
- **Withstands fire from weapons as powerful as an M16 or Kalashnikov AK47 rifle**
- **Can travel a long distance at 80 kmph (50 mph) even if tires are burst by rifle fire.**
- **Withstands the detonation of two hand grenades under the driver and rear passenger seats (body guard is expendable!?!?!?)**
- **If attacked with tear gas, the cabin is hermetically sealed and its passengers supplied with oxygen.**
- **Remote control engine start ensures no explosives are wired up to the ignition.**
- **Automatic or manual actuation of onboard fire extinguishers**
- **Distinguishable only to the trained eye by its toughened glass and wider tires.**

© 2010 LOCKHEED MARTIN CORPORATION

Security$_1$

- **“Security”**
  - ***Protection of information systems against***
    - ***Unauthorized access to or modification of information***
      - Stored, processed or transmitted
    - ***Denial of service to authorized users***
      - Measures necessary to detect, document, and counter threats

- **There is no such thing as perfect security!**
- **It’s about managing risk:**
  - *Identify Threats*
  - *Assess Vulnerabilities*
  - *Implement Countermeasures (aka Controls)*

© 2010 LOCKHEED MARTIN CORPORATION

Security$_2$

- **Fundamental aspects of the term “Security”:**
  - ***Confidentiality***
    - The characteristic of information being disclosed only to authorized persons, entities, and processes at authorized times and in the authorized manner
  - ***Integrity***
    - The characteristic of information being accurate and complete and the information systems’ preservation of accuracy and completeness
  - ***Availability***
    - The characteristic of information and supporting information systems being accessible and usable on a timely basis in the required manner

© 2010 LOCKHEED MARTIN CORPORATION

Information Assurance[1]

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

as defined in:

CNSS Instruction No. 4009

Revised May 2003

© 2010 LOCKHEED MARTIN CORPORATION

Information Assurance$_2$

- **Practical aspects of the term "Information Assurance":**
  - ***Need to Know:*** *The necessity for access to, knowledge of, or possession of specific information required to carry out official duties.* ***(Confidentiality Only)***

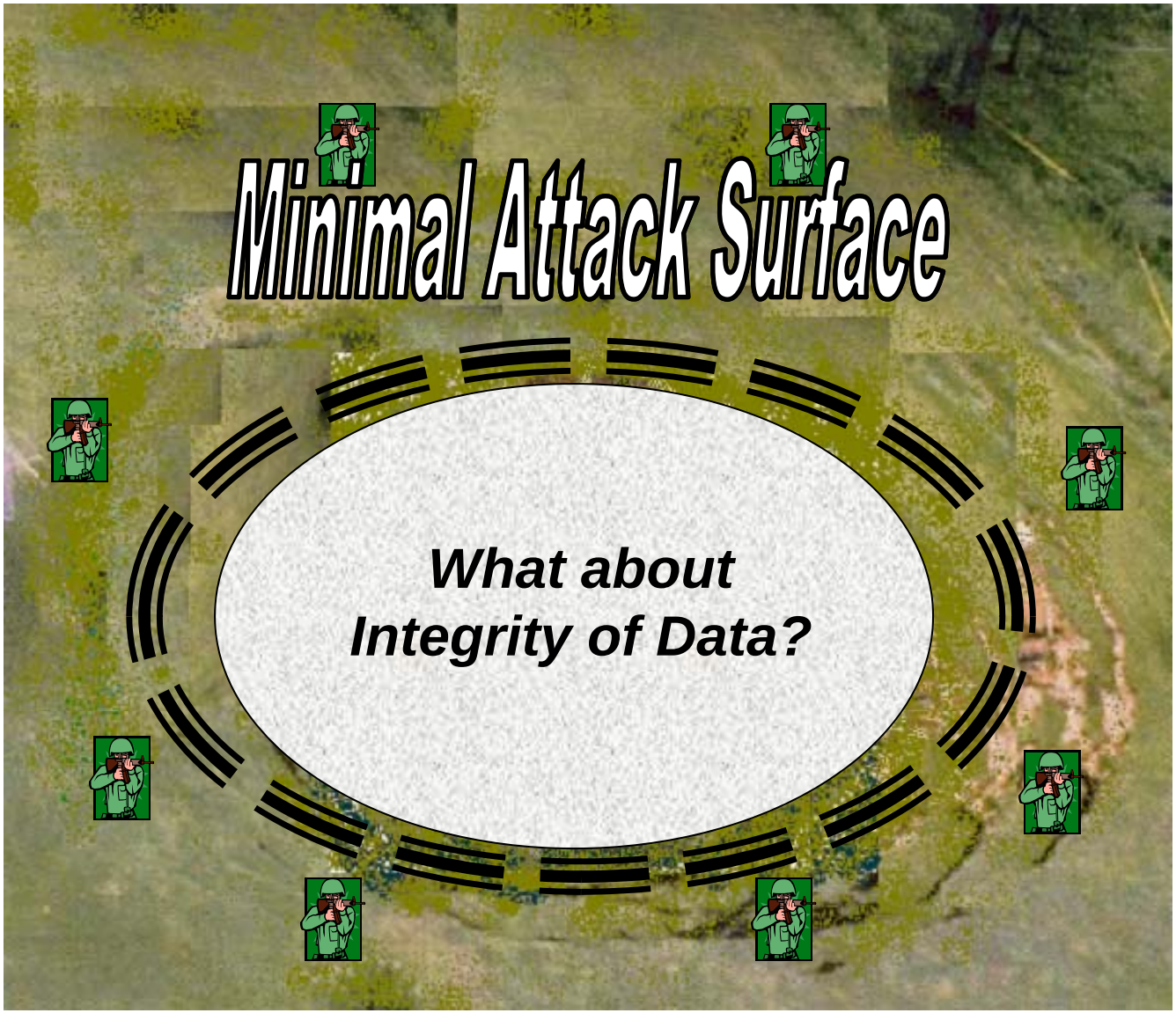
  - ***Least Privilege:*** *The principle that requires that each subject be granted the most restrictive set of privileges needed for the performance of authorized tasks. Limits the damage that can result from unauthorized use.* ***(Confidentiality and Integrity)***

© 2010 LOCKHEED MARTIN CORPORATION

Extremely Robust Confidentiality (Low Risk)

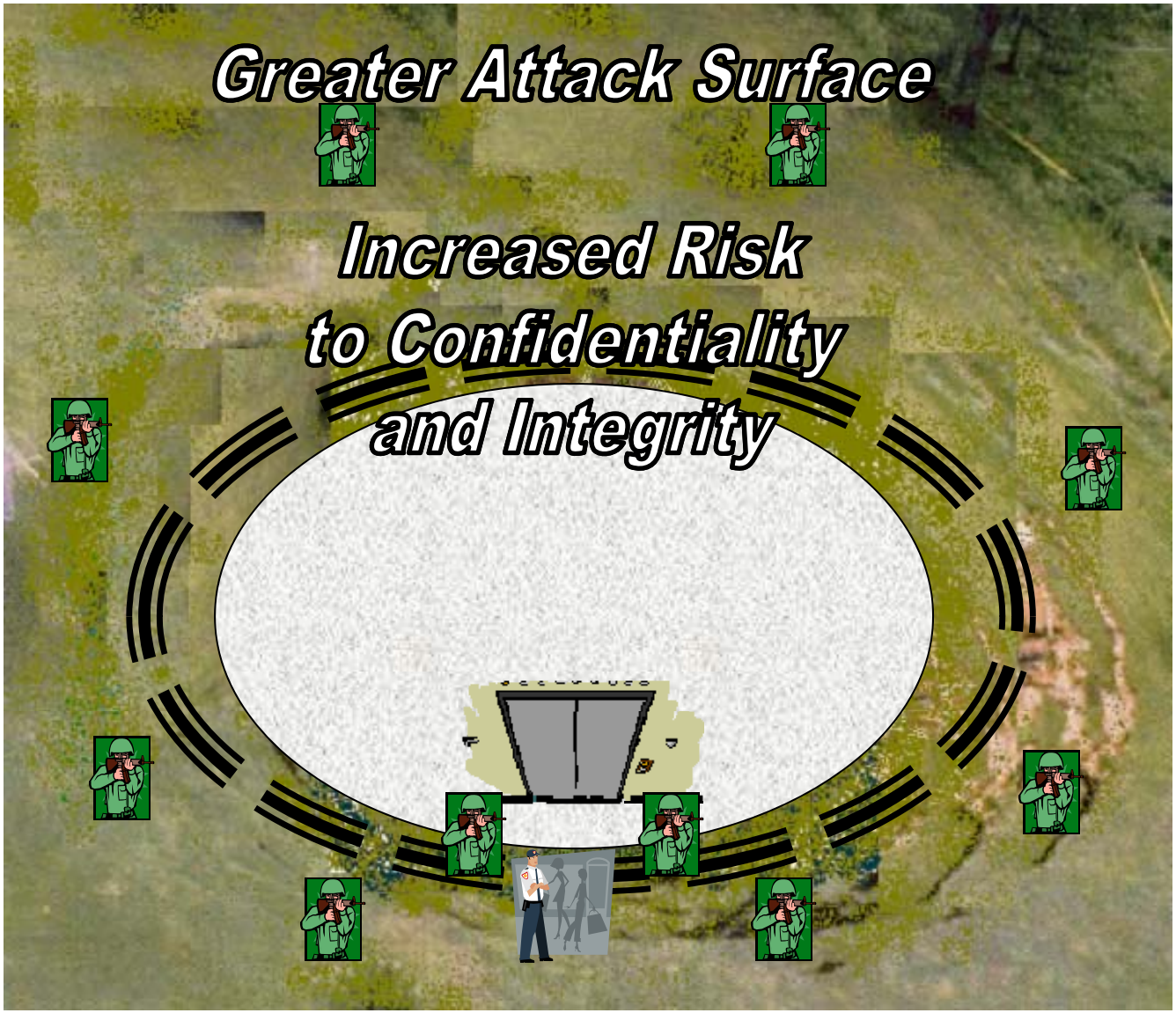
© 2010 LOCKHEED MARTIN CORPORATION

Extremely Robust Confidentiality (Low Risk)



© 2010 LOCKHEED MARTIN CORPORATION

Provide Access (Availability)



© 2010 LOCKHEED MARTIN CORPORATION

Provide Access (Availability)

© 2010 LOCKHEED MARTIN CORPORATION

Physical Security Analogy (More is More Expensive)

$174

UnClass FOUO?

$695 and up Fire / Impact Resistant

FOUO Mission Critical

$1795+ Fire / Impact Class 6 Lock

System High

$2995+ Fire / Impact Class 6 Locks

MSLS

Medium Robustness

High Robustness

© 2010 LOCKHEED MARTIN CORPORATION

Physical Security Analogy (System High vs. MSLS)

Reduce SWaP, Cooling, and Cost

$5985 **vs.** **$2995**

Unevaulated OS

S **S-NATO** **TS**

Three System High Components

Evaulated OS

XTS-400

TS

or

NIAP Evaluated

NATO

Separation Kernel

S

Multi Single Level Component

© 2010 LOCKHEED MARTIN CORPORATION

Data Classification

- **Military**
  - ***Unclassified***
    - Available for public consumption, does not present a negative impact to the organization or nation
  - ***Sensitive but unclassified***
    - Disclosure could cause serious damage
  - ***Confidential***
    - Not used outside of a company or organization
  - ***Secret***
    - Disclosure could cause serious damage to national security
  - ***Top secret***
    - Disclosure could cause grave damage to national security
- **Commercial**
  - ***Public***
    - Disclosure does not negatively impact the business
  - ***Sensitive***
    - Should only be accessible to authorized users
  - ***Private***
    - Disclosure negatively impacts personnel directly
  - ***Confidential***
    - Not used outside of a company or organization
  - ***Board Only***
    - Not to be disclosed outside the Board of Directors

© 2010 LOCKHEED MARTIN CORPORATION

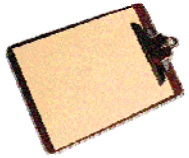
Physical Security Analogy (Multi Level Secure)

✓***Mandatory Access Controls***
✓***Discretionary Access Controls***

MSL Component

The "Downgrader"

The Audit Log



✓***Data Integrity***
✓***Expectations of Integrity***

© 2010 LOCKHEED MARTIN CORPORATION

Security Modes of Operation$_1$

- **Dedicated Mode: Single level – TS or S or S/NATO (or Private or Sensitive, ….)**
  - *All users have clearance, access level, and need-to-know for **all** information on the system or network*
- **System-High Mode: Multiple levels – TS, S, and S/NATO (or Private, Sensitive, ….)**
  - *All users have clearance and access level for **all** information, but need-to-know for **some** information on the system or network.*
- **Compartmented (Partitioned) Mode: Multiple Levels – TS, S, and S/NATO (or Private, Sensitive, ….)**
  - *All users have clearance for **all** information, but **only** access level and need-to-know for **some** information (e.g. user possess TS but only authorized access to Secret)*
- **Multi-level Secure (MLS) Mode: Multiple Levels – TS, S, C, U (or P, Sen, ….)**
  - *Some users have **only** the **clearance** and **need-to-know** for some information on the system or network.*
  - *A multilevel program simultaneously accesses data of differing sensitivity levels and produces data of differing sensitivities*
  - *Multilevel programs are contained only within MLS systems*

- **Every computer system with IA roles must be accredited to operate in any of these modes**
  - *Note user with access to a system is granted authorization for the appropriate data domains (class and need to know)*

© 2010 LOCKHEED MARTIN CORPORATION

Security Modes of Operation$_2$

Mode	Clearance	Access Level (aka. Caveats)	Need to Know
Dedicated	**Yes**	**Yes**	**Yes**
System High	**Yes**	**Yes**	**No**
Compartmented (Partitioned)	**Yes**	**No**	**No**
Multi Level Secure	**No**	**No**	**No**

© 2010 LOCKHEED MARTIN CORPORATION

Security Levels and Labels

	Definition	Example
Security Level	**The combination of a hierarchical classification and a set of nonhierarchical categories that represents the sensitivity of information**	**SENSITIVE, PRIVATE, CONFIDENTIAL**
Compartment	**A class of information that has need-to-know access controls beyond those normally provided for access to SENSITIVE, PRIVATE, CONFIDENTIAL information.**	**Marketing Dept, Finance Dept**
Security Label	**Information representing the sensitivity of a subject or object, such as its hierarchical classification together with any applicable nonhierarchical security categories**	**SENSITIVE/Marketing, SENSITIVE/Finance, PRIVATE/Personnel**

© 2010 LOCKHEED MARTIN CORPORATION

Important to Note: Security, Safety, and Performance

- **Just as there is no perfect Security and Safety, where Security and Safety are required, one has to plan for not having 100% utilization (performance) for the application's functional purpose!**
  - ***Example: Reference Monitor concept implemented in a Security Kernel.***
    - Every Read and Write requires a check to determine if the Subject's Clearance Authorization matches the Objects Security Label!
    - That requires extra CPU cycles.
  - ***Generally a 2-6% Utilization Tax.***
    - Real-time embedded Avionics processors are usually loaded to about 50-60% max to accommodate fault fail-over, so not an issue
    - Ground Stations, Desktops, and Enterprise Servers on the other hand….

© 2010 LOCKHEED MARTIN CORPORATION

Security Policy

- **Security Policy**
  - ***The set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information***
  - ***A System Security Policy***
    - Developed by the CIO and ISSP
    - Involves People, Physical, as well as HW and SW
    - Leads to derived security requirements

© 2010 LOCKHEED MARTIN CORPORATION

Traditional Vetted Security Policies

- **The Bell & LaPadula model is a *Confidentiality policy model* (two axioms)**
  - *simple security – No Read Up*
  - *the *-property – No Write Down*
- **Noninterference (**Goguen and Meseguer**) or Information Flow (an attempt to extend BLP)**
- **The Biba model is an *Integrity policy model* (Access Control)**
  - *simple integrity (No Read Down)*
  - *the *-integrity (No Write Up)*

© 2010 LOCKHEED MARTIN CORPORATION

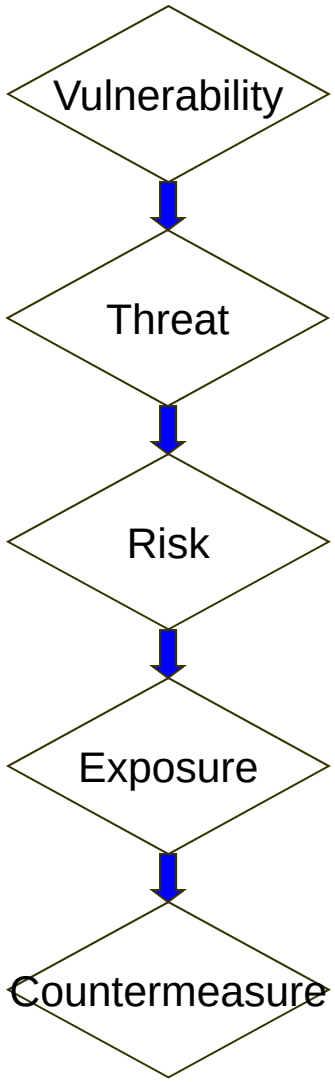
Trusted Computing Base (TCB)

The totality of protection mechanisms within a computer system, including hardware, firmware, and software, the combination of which is responsible for enforcing a security policy.

© 2010 LOCKHEED MARTIN CORPORATION

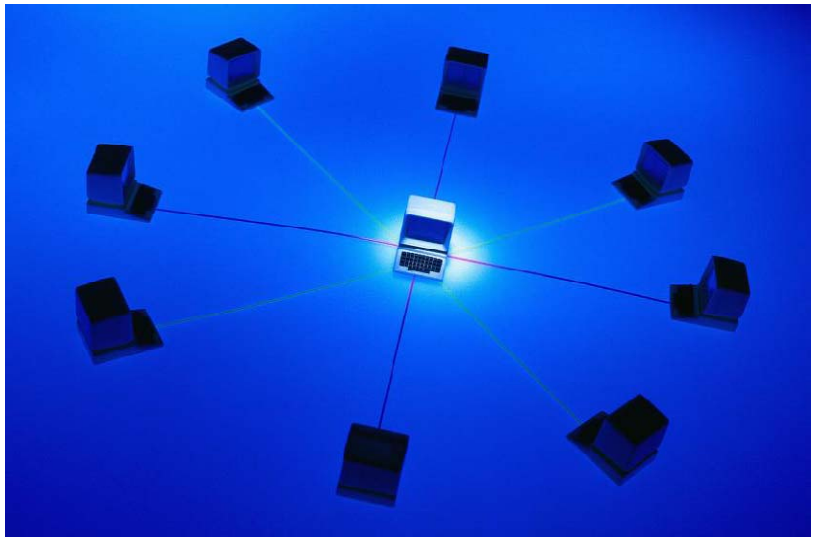
Security Definitions$_1$

- **Reduction of "attack surface" important process within information security**
  - ***Security management uses key definitions to identify areas of concern and how to protect them***
    - Vulnerability
    - Threat
    - Risk
    - Exposure
    - Countermeasure



© 2010 LOCKHEED MARTIN CORPORATION

Security Definitions$_2$



- **Attack surface**
  - ***Area or parts of the system or network that are available to an assailant to compromise an environment***
    - Might include multiple channels of entry
    - Services
    - Software
    - Physical Access
  - ***Fundamental objective in INFOSEC is constantly reducing the "attack surface" to better secure the environment***

© 2010 LOCKHEED MARTIN CORPORATION

Security Definitions$_3$

- **Vulnerability**
  - ***A flaw***
  - ***Attacker access to the flaw, and***
  - ***Attacker capability to exploit the flaw***
    - Examples
      - *Lack of security patches*
      - *Lack of current virus definitions*
      - *Software Bug*
      - *Lax physical security*

© 2010 LOCKHEED MARTIN CORPORATION

Security Definitions$_4$

- **Threats**
  - ***Any potential hazard or harm to the data, systems or environment by leveraging a vulnerability***
  - ***Individual taking advantage of a vulnerability is consider a threat agent***
    - Examples of threats
      - *Intruder using open ports on a firewall to gain access to internal areas of a network*
      - *Uneducated users handling sensitive information*
      - *Elemental factors damaging physical building*

© 2010 LOCKHEED MARTIN CORPORATION

Security Definitions$_5$

- **Risk**
  - ***Risks are the probability of the threats using the vulnerabilities***
  - ***Higher risks come with more vulnerabilities and increased threats***
    - Example of different levels of risk
      - *Allowing usernames/password to be taped to monitors*
      - *Public areas –higher risk levels*
      - *Private areas –lower risk due to less traffic flow of potential threat agents*

© 2010 LOCKHEED MARTIN CORPORATION

Security Definitions$_6$

- **Exposure**
  - ***The damage done through a threat taking advantage of a vulnerability***
    - Examples of exposure
      - *Data deletion or modification, the loss of integrity*
      - *Malicious code deployed in a private network and stealing sensitive customer information*
      - *Unauthorized viewing*
        - » Credit cards, SSN, phone numbers, etc.
        - » Liability incurred by such an action

© 2010 LOCKHEED MARTIN CORPORATION

Security Definitions$_7$

- **Countermeasures (aka Controls)**
  - ***Processes and standards that are used to combat and mitigate the risks***
    - Examples
      - *Keeping up-to-date on service packs, hotfixes*
      - *Maintaining current virus definitions*
      - *Hiring a security staff to monitor the facilities*
      - *Access control systems inside the operating systems*
      - *Biometric devices to provide higher assurance of authentication*
      - *Educating users on managing passwords and/or sensitive information*

© 2010 LOCKHEED MARTIN CORPORATION

Questions?

© 2010 LOCKHEED MARTIN CORPORATION

Security-Relevant Requirements and Trusted Software

© 2010 LOCKHEED MARTIN CORPORATION

Security Relevant vs. Trusted

- **The term *Security Relevant***
  - *Attribute(s) associated with the enforcement of a security policy*
  - *Refers to requirements or functionality*
  - *Security relevant requirements are met with trusted components*

- **The term *Trusted Components***
  - *Protection mechanisms that enforce a security policy*
  - *Refers to software, firmware, or hardware*
  - *Reliably enforces the security policy*

© 2010 LOCKHEED MARTIN CORPORATION

Trusted Software

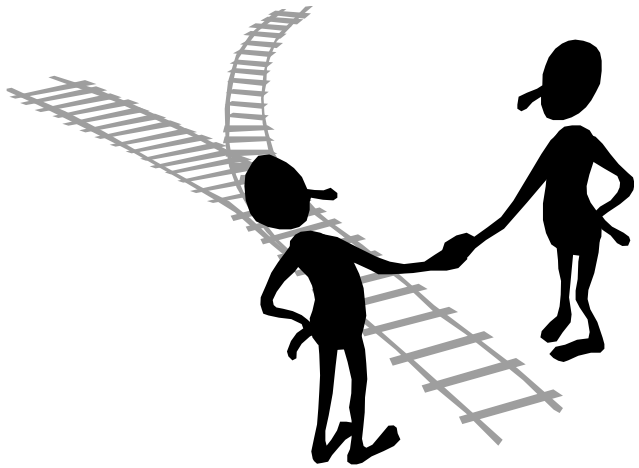
- ***Trusted Software*** is the collection of software elements of the TCB that provide protection mechanisms within a computer system and are responsible for enforcing a security policy.
  - ***Mitigates the risk associated with:***
    - Compromise of classified (DoD) or sensitive/confidential (Commercial) information
    - Denial of service
    - Corruption of system or user data

- **Trusted software can be**
  - *Developed by the product company*
  - *Non-developmental Items (NDI) (i.e., COTS, GOTS, FOSS, Reuse)*

© 2010 LOCKHEED MARTIN CORPORATION

Trusted by Association

- ***Trusted Software by Association*** is software that does not perform any security relevant functions, but is in the same address space as trusted software
  - ***Must be designed to ensure that it does not perform any functions that could compromise the TCB***



© 2010 LOCKHEED MARTIN CORPORATION

Security Relevant Requirements

- **Security <u>Functional</u> Requirements**
  - ***Determine what is implemented in hardware, software, or firmware***
    - Identification and Authentication, Access control, Audit, etc.

- **Security <u>Assurance</u> Requirements**
  - ***Determine how well Functional Requirements are implemented***
    - How reliable are the mechanisms?
    - Can they be circumvented or tampered with?
    - Do mechanisms completely and correctly enforce the policy?
    - How do we know the mechanisms *only* enforce the policy?
  - ***Verify that implemented functional requirements are met***

© 2010 LOCKHEED MARTIN CORPORATION

General Approach

- **Evidences should be produced for each TCB boundary (i.e., all TCB components within the boundary).**
  - *A TCB boundary is the confines within which TCB elements are evaluated.*
  - *When new TCB elements are identified and brought into a boundary, delta updates are made to the C&A packages developed in earlier releases as required to document changes for the new or changed TCB elements.*
- **Within each TCB boundary, the development process produces artifacts that document the security features and characteristics of the product being developed.**
- **Likely has to be updated several times after the initial draft and reviewed by the relevant authority.**
- **That documentation will be used by appropriate authorities to determine "trustedness" of the components in the larger system.**

© 2010 LOCKHEED MARTIN CORPORATION

Procedures

- **Many sources for establishing design / development procedures.**
- **Internationally**
  - ***ISO/IEC 27001:2005 - Information technology -- Security techniques -- Information security management systems -- Requirements***
  - ***ISO/IEC 27002 - Best Practices***
  - ***ISO/IEC 15408 The Common Criteria for Information Technology Security Evaluation***

- **Trusted processes must be intertwined with a company's normal software development processes to insure *"Security is Built-in"***

  https://buildsecurityin.us-cert.gov/swa/resources.html

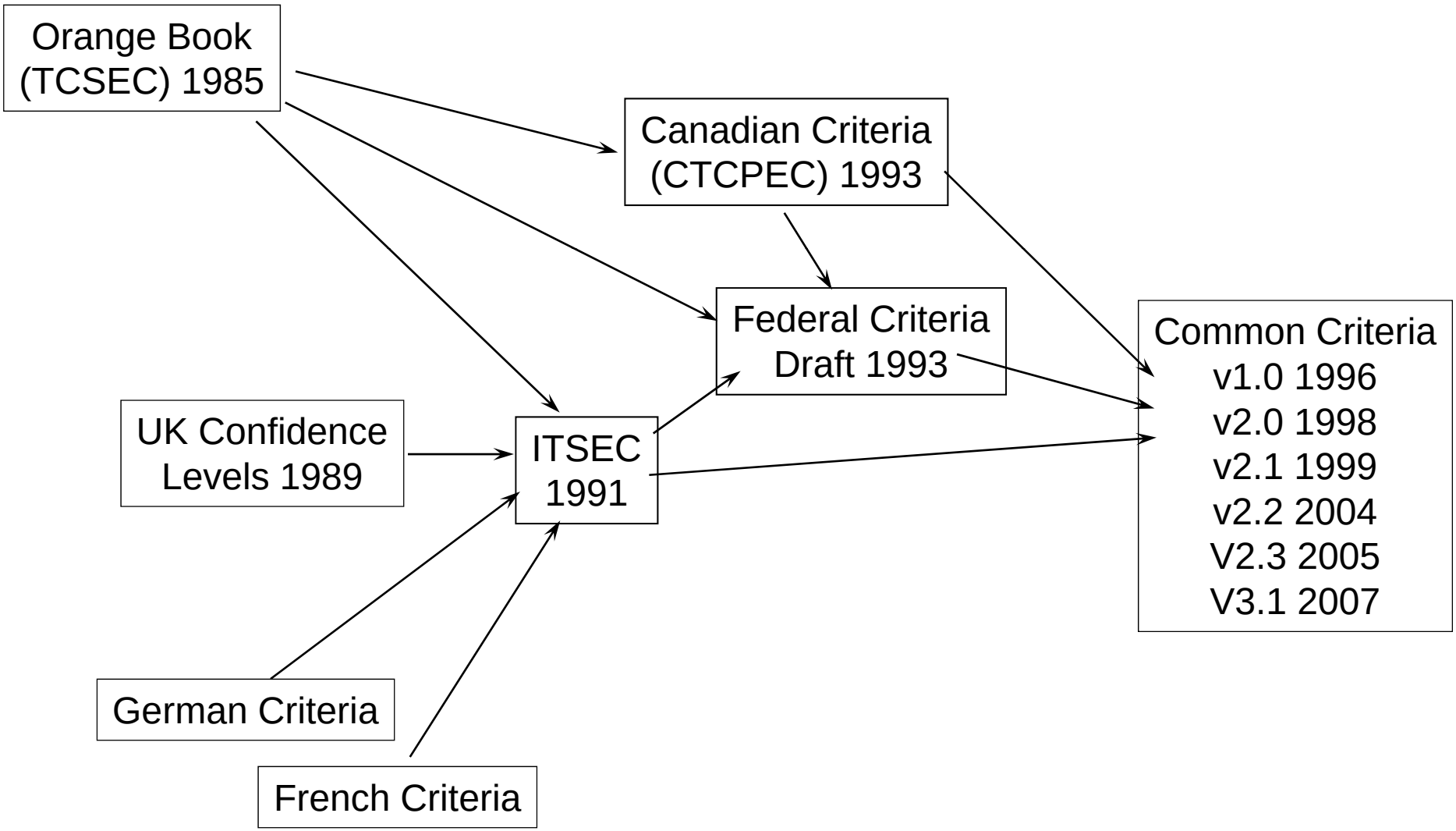
© 2010 LOCKHEED MARTIN CORPORATION

Common Criteria

- **The Common Criteria for Information Technology Security Evaluation (abbreviated Common Criteria or CC) is an international standard (ISO/IEC 15408) for computer security**
  - *Common language and structure for expressing IT security requirements in a consistent, standardized manner*
  - *Used to evaluate and validate that a particular product or system satisfies a defined set of security requirements*
  - *Allows for seven* Evaluation Assurance Levels (EALs) *of both functional and assurance security requirements*
  - *The higher the level, the more confidence you can have that the security functional requirements have been met*
  - *Used to evaluate COTS products, not large systems*

© 2010 LOCKHEED MARTIN CORPORATION

Where did the CC come from?



© 2010 LOCKHEED MARTIN CORPORATION

What is the Common Criteria?

- **Language**
  - ***Functional requirements***
    - Define that which is to be implemented by hardware/software/firmware
  - ***Assurance requirements***
    - Defines the activities conducted and evidence generated by both the product developer and product evaluator
- **Structure**
  - ***Catalogue of standardized, reusable, security requirement components***
  - ***Protection Profile***
  - ***Security Target***

© 2010 LOCKHEED MARTIN CORPORATION

The Protection Profile (PP) & Security Target (ST)

- **Protection Profile**
  - ***A set of requirements that establish an implementation independent solution to a security problem***
  - ***Applies to a type-set of products***
    - OS, Firewall, etc.
  - ***Serves as a high-level requirements spec for development of solutions***
- **Security Target**
  - ***A set of requirements that provide an implementation dependent description of a specific product that addresses the security problem***
  - ***Serves as the basis for conducting the product evaluation***
- **Each contains a consistent thread from 'what' to 'how' supported by end-to-end traceability and rationale**

© 2010 LOCKHEED MARTIN CORPORATION

Common Criteria Terminology

- **Target of Evaluation (TOE)**
  - ***The entity that undergoes evaluation. The TOE is composed of the implementation***
- **Target of Evaluation Security Policy (TSP)**
  - ***A set of rules that regulate how assets are managed, protected and distributed within a TOE***
- **Target of Evaluation Security Functions (TSF)**
  - ***All the hardware, software, and firmware of the TOE that must be relied upon for the correct enforcement of the TSP***

© 2010 LOCKHEED MARTIN CORPORATION

PP/ST Contents & Comparison

Protection Profile

- Introduction
- TOE Description
- Security Environment
  - Assumptions
  - Threats
  - Organizational Security Policies
- Security Objectives
- Security Requirements
  - Functional Req’ts
  - Assurance Req’ts
- Rationale

Security Target

- Introduction
- TOE Description
- Security Environment
  - Assumptions
  - Threats
  - Organizational Security Policies
- Security Objectives
- Security Requirements
  - Functional Req’ts
  - Assurance Req’ts
- *TOE Summary Specification*
- *PP Claims*
- Rationale

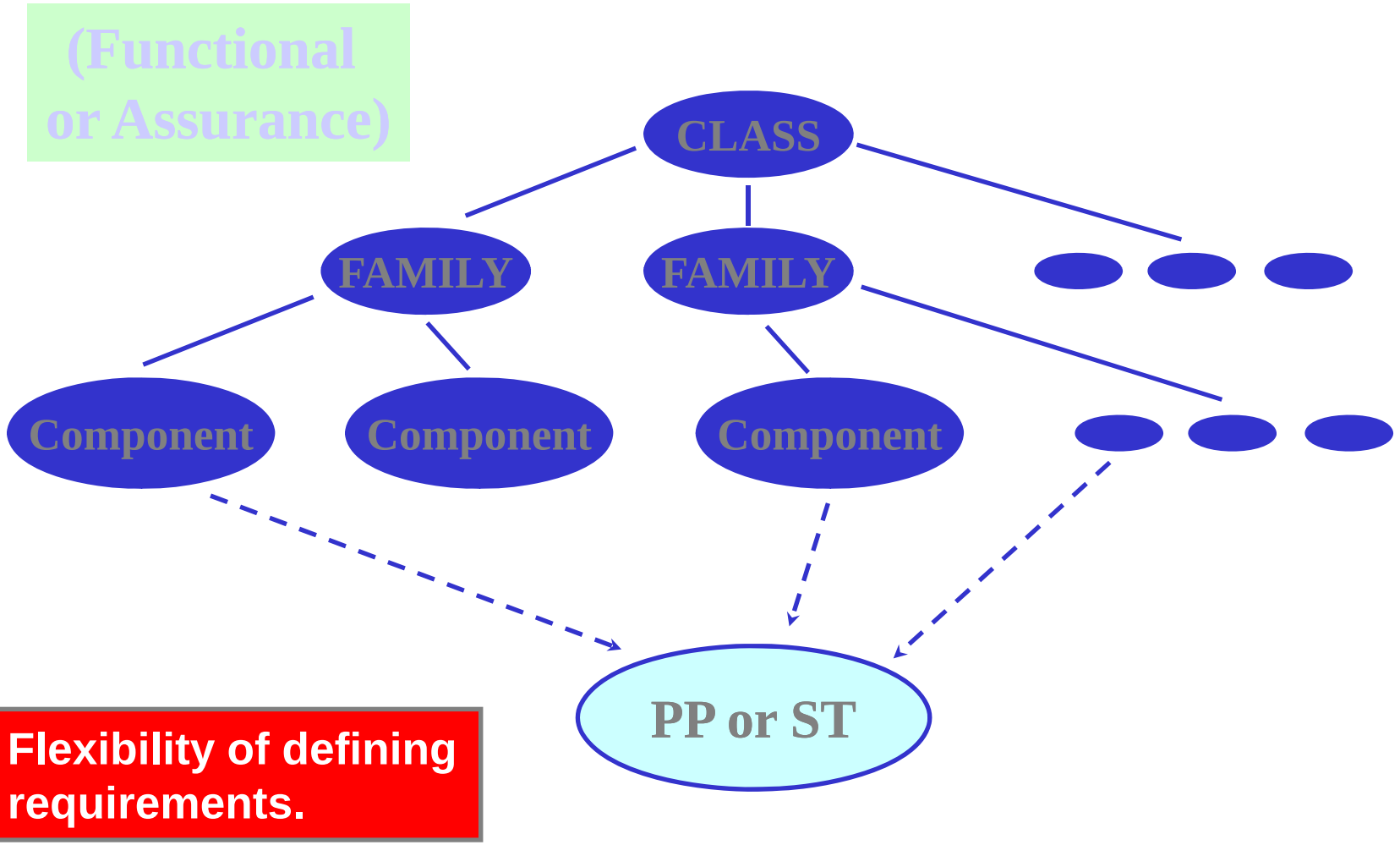
© 2010 LOCKHEED MARTIN CORPORATION

Key CC Concepts: Requirements Hierarchy

- **CC functional / assurance requirements hierarchy:**
  a set of constructs that classify security requirement components into related sets:
  - ***Class (e.g. FDP - User Data Protection):***
    a grouping of families that share a common focus.
  - ***Family (e.g. FDP_ACC - Access Control Policy):***
    a grouping of components that share security objectives but may differ in emphasis or rigor.
  - ***Component (e.g. FDP_ACC.1 - Subset Access Control):***
    the smallest selectable set of elements that may be included in a PP / ST / package.

© 2010 LOCKHEED MARTIN CORPORATION

Example Hierarchy



© 2010 LOCKHEED MARTIN CORPORATION

CC Part 2 -- Catalog

- **Classes of Security Functional Requirements:**

Class	Name
FAU	Audit
FCO	Communications
FCS	Cryptographic Support
FDP	User Data Protection
FIA	Identification & Authentication
FMT	Security Management
FPR	Privacy
FPT	Protection of TOE Security Functions
FRU	Resource Utilization
FTA	TOE Access
FTP	Trusted Path / Channels

© 2010 LOCKHEED MARTIN CORPORATION

CC Functional Requirements Example

5.2 User Data Protection (FDP)

5.2.1 Information Flow Control Policy (FDP_IFC)

5.2.1.1 Complete Information Flow Control (FDP_IFC.2)

FDP_IFC.2.1 **Refinement:** The TSF shall enforce the **Partitioned Information Flow SFP** as a **[selection:** ***Partition Abstraction, Equivalence Set Abstraction, Least Privilege Abstraction*****]** on

- **All partitions**
- **All subjects**
- **All exported resources**

for the following controlled operations, which constitute all possible operations that cause information to flow between subjects and exported resources:

- **Controlled Operations: [assignment: list of all the controlled operations provided by the implementation]. 2**

© 2010 LOCKHEED MARTIN CORPORATION

CC Part 3 -- Catalog

- **Classes of Security Assurance Requirements:**

Class	Name
ADV	Development
AGD	Guidance Documents
ALC	Life Cycle Support
ATE	Tests
AVA	Vulnerability Assessment
APE	Protection Profile Evaluation
ASE	Security Target Evaluation
ACO	Composition

© 2010 LOCKHEED MARTIN CORPORATION

CC Assurance Requirements Example

6.1 Configuration Management (ACM)

6.1.1 CM Automation (ACM_AUT)

6.1.1.1 Complete CM Automation (ACM_AUT.2)

ACM_AUT.2.1D The developer shall use a CM system.

ACM_AUT.2.2D The developer shall provide a CM plan.

ACM_AUT.2.1C The CM system shall provide an automated means by which only authorized changes are made to the TOE implementation representation, and to all other configuration items.

ACM_AUT.2.2C The CM system shall provide an automated means to support the generation of the TOE.

ACM_AUT.2.3C The CM plan shall describe the automated tools used in the CM system.

ACM_AUT.2.4C The CM plan shall describe how the automated tools are used in the CM system.

ACM_AUT.2.5C The CM system shall provide an automated means to ascertain the changes between the TOE and its preceding version.

ACM_AUT.2.6C The CM system shall provide an automated means to identify all other configuration items that are affected by the modification of a given configuration item.

ACM_AUT.2.1E The evaluator shall confirm that the information provided meet all requirements for content and presentation of evidence.

© 2010 LOCKHEED MARTIN CORPORATION

Evaluation Assurance Levels (EALs)

(Basis for Mutual Recognition)

- **Evaluation Assurance Levels & (*rough*) Backward Compatibility Comparison**

EAL	Name	*TCSEC
EAL1	Functionally Tested	
EAL2	Structurally Tested	C1
EAL3	Methodically Tested & Checked	C2
EAL4	Methodically Designed, Tested & Reviewed	B1
EAL5	Semiformally Designed & Tested	B2
EAL6	Semiformally Verified Design & Tested	B3
EAL7	Formally Verified Design & Tested	A1

***TCSEC = "Trusted Computer Security Evaluation Criteria" -- "Orange Book"**

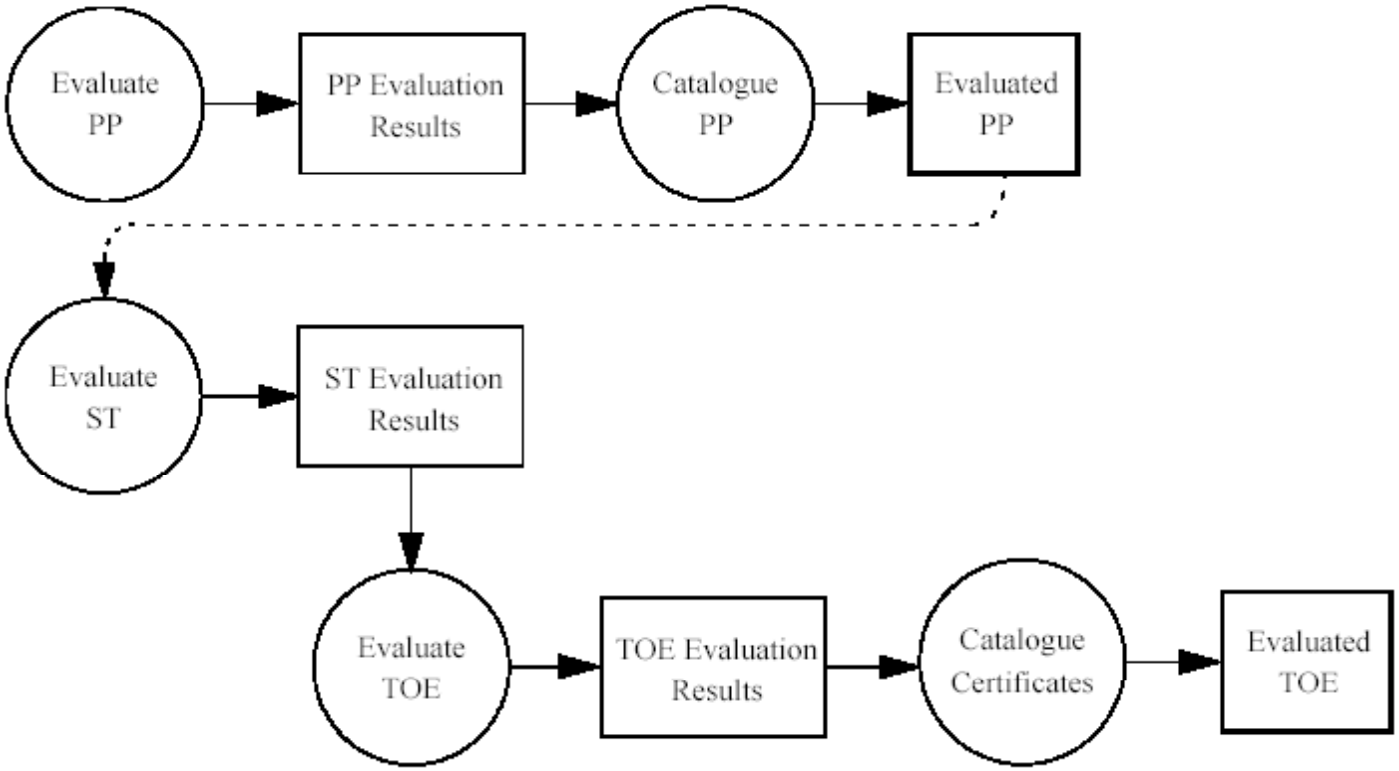
© 2010 LOCKHEED MARTIN CORPORATION

Types of CC Evaluation

- **Protection Profile evaluation**
- **Product evaluation (two phases):**
  - ***Security Target evaluation***
  - ***TOE evaluation (uses evaluated ST as baseline)***

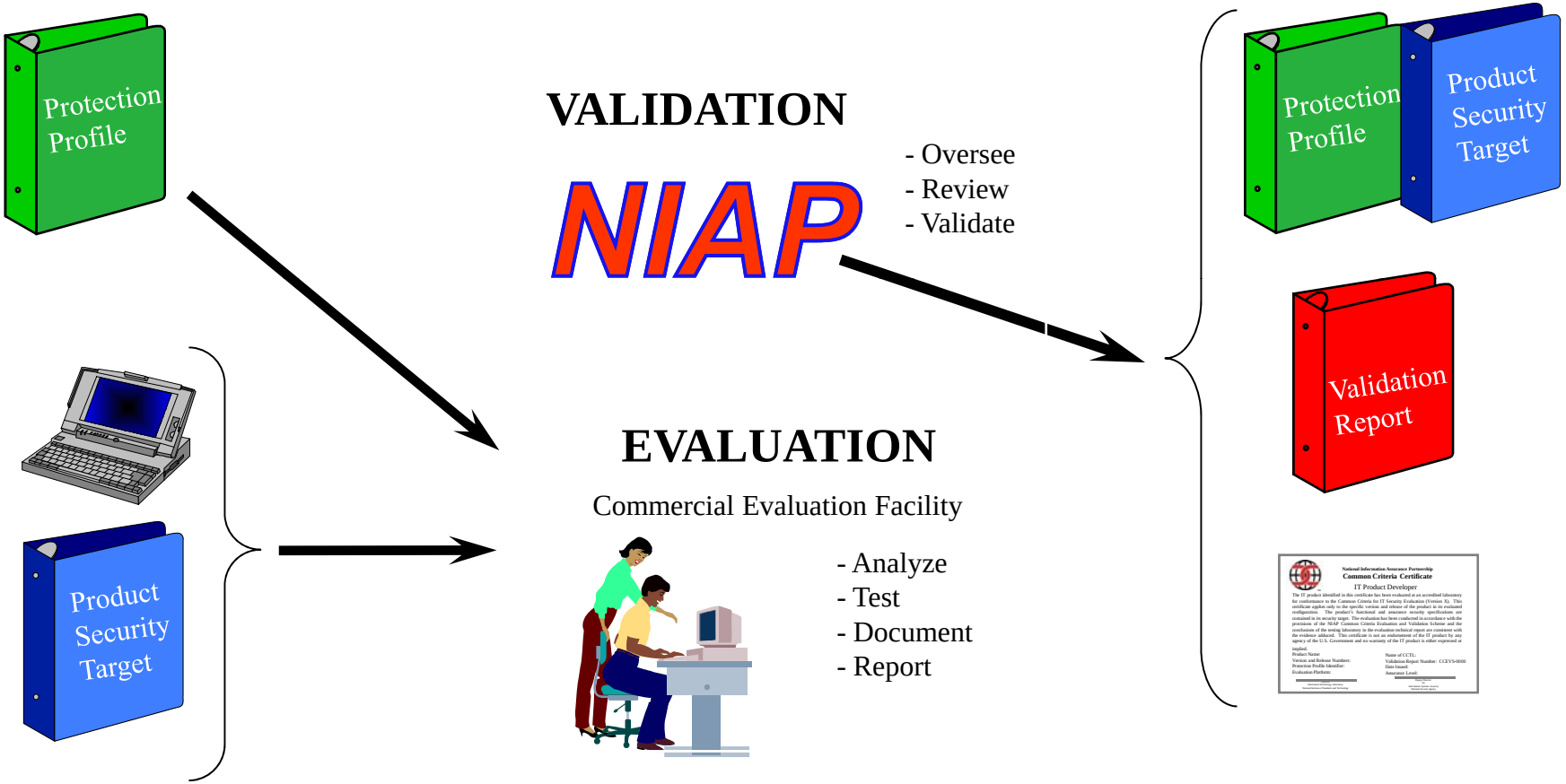
Start Here

or Here



© 2010 LOCKHEED MARTIN CORPORATION

Evaluation Process Overview



© 2010 LOCKHEED MARTIN CORPORATION

Conducting the Evaluation

- **The CC describes *what* is to be accomplished in terms of development and evaluation activities**
- **The evaluation is conducted against a set of assurance criteria typically organized in terms of *Evaluation Assurance Levels (EALS)***
  - ***EALs range from 1 (lowest) to 7 (highest) levels of***
    - Scope, Depth, Rigor
- **The companion "Common Evaluation Methodology" describes *how* the evaluation is to be conducted**
  - ***Methodology defined only for EALs 1-4***

- ***<u>Higher EALs do not necessarily imply "better security", they only mean that the claimed security assurance of the TOE has been more extensively verified.</u>***

© 2010 LOCKHEED MARTIN CORPORATION

Common Criteria's Real Value

- **"The real value of Common Criteria was not that some of our database products got a NIAP Certificate, but rather all our products were greatly improved in quality and security by incorporating the Common Criteria security requirements as part of our product life cycle process."**

  – *Mary Ann Davidson, Chief Security Officer, Oracle*
    - » Three different presentations at
    - » DHS-OSD Software Assurance Forums, 2007 and 2008

© 2010 LOCKHEED MARTIN CORPORATION

The CC Types of Trusted Software

- Security Audit
- Communication
- Crypto Support
- User Data Protect
- Identification & Authentication
- Security Management
- Privacy
- Protect Security Mechanisms
- Resource Utilization
- System/Subsystem Access
- Trusted Path & Channel

© 2010 LOCKHEED MARTIN CORPORATION

Security Audit

- **Trusted programs that are used for recognizing, recording, storing, protecting, and analyzing information related to security relevant activities**
  - *Monitoring user activities*
  - *Detecting real, potential or imminent violations of the TCB*

- **The security audit functions are designed to:**
  - *Help monitor security relevant events*
  - *Determine who is responsible*
  - *Act as a deterrent against security violations*

© 2010 LOCKHEED MARTIN CORPORATION

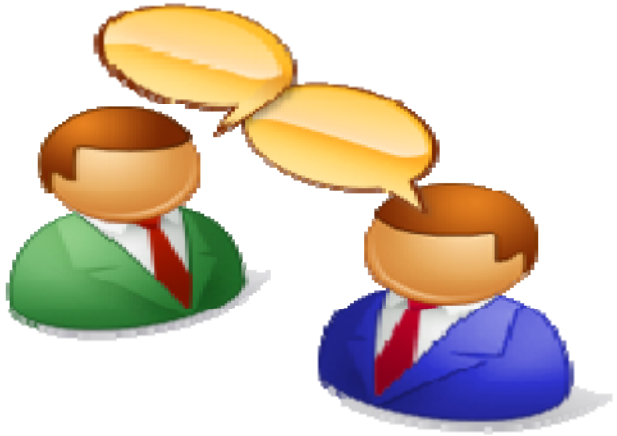
Security Audit

- **Audit functions include:**
  - *Audit data collection*
  - *Audit data protection*
  - *Audit record format*
  - *Audit event selection*
  - *Audit analysis/reduction*
  - *Audit violation alarms*
  - *Real-time audit analysis*

© 2010 LOCKHEED MARTIN CORPORATION

Communication

- **Trusted programs that perform the transport of information**
  - *Between one subsystem and another within the same system*
  - *Between a subsystem and external interfaces*
  - Concerned with Non-repudiation
    - *Assuring the identities of parties participating in a specific data exchange*



© 2010 LOCKHEED MARTIN CORPORATION

Communication

- **Non-repudiation is divided into two major areas:**
  - ***Non-repudiation of Origin***
    - *The recipient or a third party can verify the evidence of origin*
    - *Originator cannot deny sending information*
    - *This evidence must not be forgeable*

  - ***Non-repudiation of Receipt***
    - *Provides evidence that information was received*
    - *The recipient cannot successfully deny having received the information*
    - *The originator or a third party can verify the evidence of receipt*
    - *This evidence must not be forgeable*

© 2010 LOCKHEED MARTIN CORPORATION

Cryptographic Support

- **Trusted hardware, firmware and/or software that implement cryptographic functionality to help satisfy several high-level security objectives**
  - *Protects ID and authentication data*
  - *Provides data separation*
  - *Provides data protection*

- **The category is composed of two areas:**
  - ***Cryptographic key management***
    - Addresses the management aspects of cryptographic keys
  - ***Cryptographic operation***
    - Concerned with the operational use of those cryptographic keys

© 2010 LOCKHEED MARTIN CORPORATION

Cryptographic Support

- **Both Type 1 and Type 2 encryption software are supported by this category:**
  - ***Type 1***
    - Used to encrypt classified information
    - Certified by the National Security Agency (NSA)
  - ***Type 2***
    - Used to encrypt unclassified sensitive information
    - Certified by the National Institute of Standards & Technology (NIST)
      - *3DES*
      - *AES*

© 2010 LOCKHEED MARTIN CORPORATION

User Data Protection

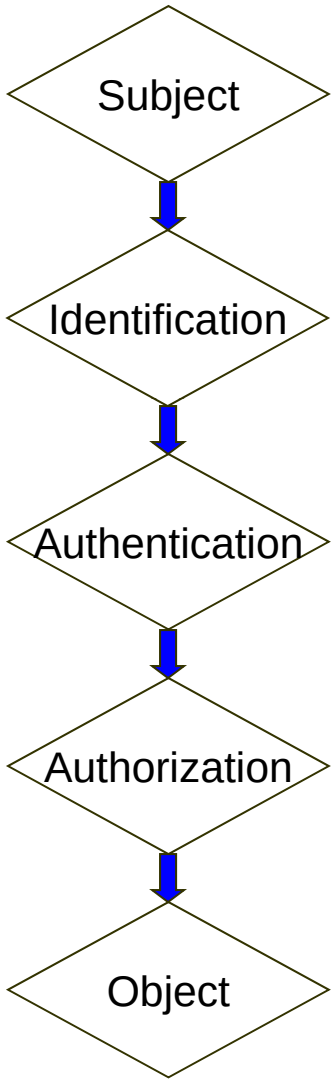
- **Trusted software that is used to protect user data and control access to the user data**
- **User Data**
  - *Data created by or used by the user*
  - *It does not affect the operation of security functions*
  - *Sensitive but not security relevant*

- **Data protection trusted software can perform traditional Discretionary Access Controls (DAC) and/or Mandatory Access Controls (MAC)**

© 2010 LOCKHEED MARTIN CORPORATION

Key Elements of Access Control$_1$

- **Access control elements are broken down into some key, fundamental components**
  - *Include*
    - Subject
    - Object
    - Identification
    - Authentication
    - Authorization
    - Accountability



© 2010 LOCKHEED MARTIN CORPORATION

Key Elements of Access Control$_2$

- Role of a subject
  - *Subjects represent the user, person, service or process that is attempting to access some form of information*
  - *Examples of subjects*
    - Users logging onto the network
    - One mail system communicating with a second system to send or relay messages
    - A user that is attempting to access a shared folder on the network
    - A user that is attempting to update a field within a database
    - A hacker that is attempting to leverage an open port on a firewall to gain access to confidential information

© 2010 LOCKHEED MARTIN CORPORATION

Key Elements of Access Control$_3$

- Role of an object
  - *Objects are the resources, information, data, or processes with which a subject is attempting to interact*
  - *Examples of objects*
    - Files located on a file server
    - Web pages housed on a website
    - Software that might be installed on an end user's machine
    - Printers that are available from the network
    - A table inside of a given database
    - Ports on a firewall
  - *Some access control models are object oriented*

© 2010 LOCKHEED MARTIN CORPORATION

User Data Protection: DAC

- **DAC provides a means of restricting access to objects based on the identity and need-to-know of the users**
  - *Objects have Access Control Lists (ACL's)*

- **DAC includes the following functionality:**
  - *Allows users to specify and control sharing of objects*
  - *Provide controls to limit propagation of access rights*
  - *Protect objects from unauthorized access*
  - *Control/specify modes of access for an object (read, write, execute, no access)*
  - *Capability to create access control lists for an object*

© 2010 LOCKHEED MARTIN CORPORATION

User Data Protection: MAC

- **MAC provides a means of restricting access to objects based on the sensitivity of the information contained in the objects and formal authorization of the users.**

- **The key features of MAC are :**
  - ***Labels are associated with each resource***
    - Explicit labels – All data is explicitly tagged with an attribute that contains a classification level and/or category value (e.g., SECRET, SECRET/NATO)
    - Implicit labels – The data's sensitivity is determined by another attribute (e.g., Message ID)
  - ***Subjects can read down***
  - ***Subjects can write up***

- **What to do when a program needs to access data at a given security level and write at a lower level?**

© 2010 LOCKHEED MARTIN CORPORATION

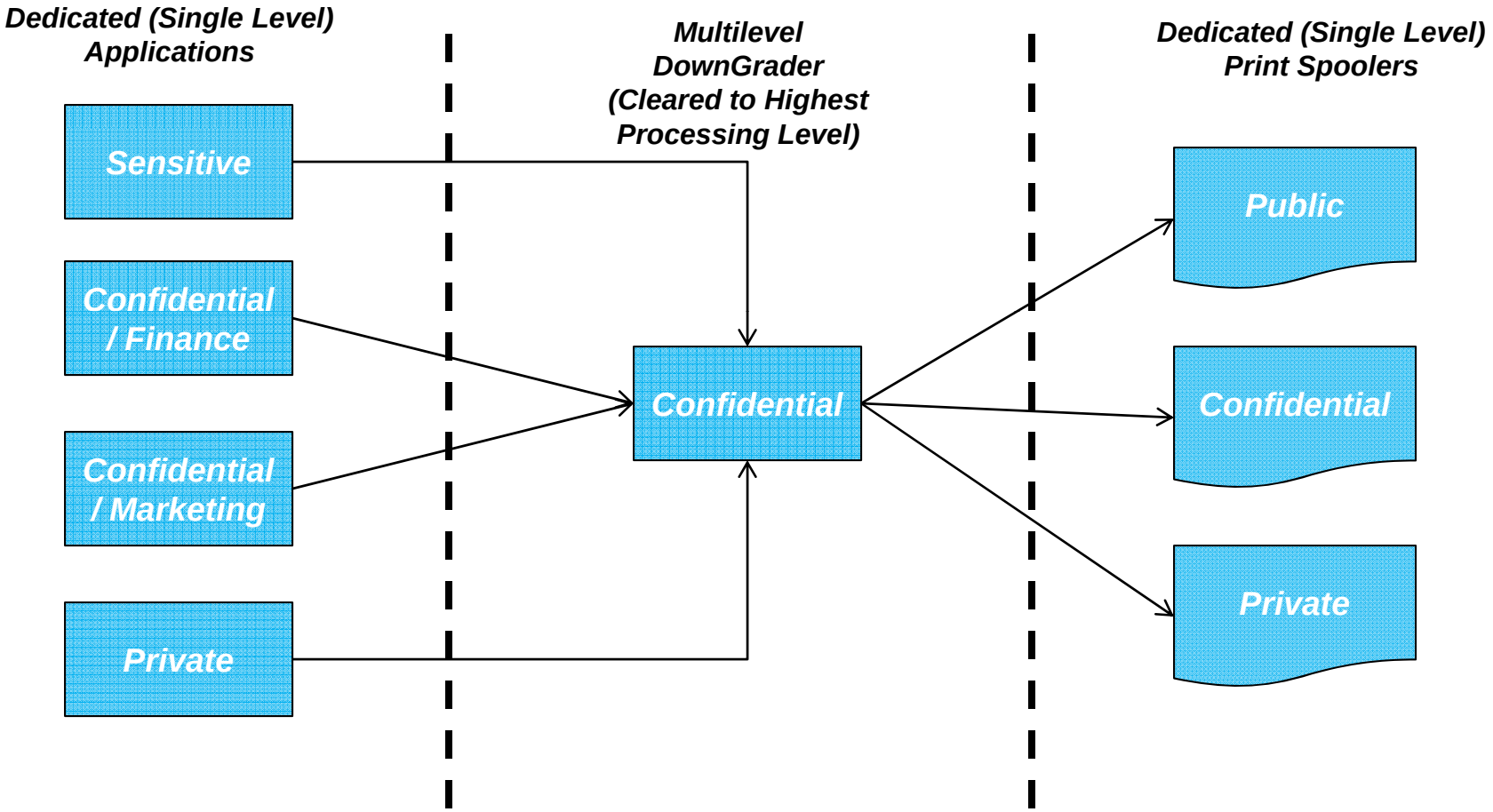
User Data Protection: Write-Down Programs$_1$

- **Write-down programs validate or manipulate classified information at one level to produce a lower classified output**

- **Two types of write-downs**
  - ***Regrading or Guard:***
    - ***Data contained in the object is at a higher classification level than the program/application execution classification level***
    - ***Performs a content validation of the information***
      - *Ensure the content is per the defined interface*
  - ***Sanitizing:***
    - ***Manipulation of the actual data in an object***
      - *Scaling, deleting source information, conversion to percentage, fuzzify, etc.*
    - ***Perform an algorithm on that data***
      - *Algorithms used to manipulate the data are classified*

© 2010 LOCKHEED MARTIN CORPORATION

User Data Protection: Write-Down Programs$_2$



This Architecture will involve more C&A effort!

© 2010 LOCKHEED MARTIN CORPORATION

Identification & Authentication (I&A)

- **Trusted software used to establish and verify a claimed user's identity**
  - *Must definitively identify the person and/or entity performing functions*
  - *Achieved by requiring users to provide information that is known and protected by the TCB*

- **Features supported by I&A:**
  - *Determining and verifying the identity of a user*
  - *Defining and enforcing limits on repeated unsuccessful I&A attempts*
  - *Defining user attributes used in security policy enforcement (e.g., Need-to-Know)*
  - *Association of security attributes for users (e.g., Least Privilege)*

© 2010 LOCKHEED MARTIN CORPORATION

Security Management

- **Trusted software that is used to manage the security attributes, TCB data, TCB functions and separation of capability/duties**
  - *Enable authorized users to set up and control the secure operation of the TCB*

- **Includes management functions related to:**
  - *Access control, accountability and authentication controls enforced by the TCB*
  - *Controls over availability*
  - *General installation and configuration*
  - *Control and maintenance of TCB resources*

© 2010 LOCKHEED MARTIN CORPORATION

Privacy

- **Trusted software that provides user protection against discovery and misuse of identity by other users falls into the category of Class: FPR.**
- **The key features are:**
  - ***Anonymity*** *– ensures a user's access of a resource or service will not disclose their identity.*
  - ***Pseudonymity*** *– same as Anonymity except user is accountable for the access*
  - ***Unlinkability*** *- user may make multiple uses of resources or services without others being able to link these uses together.*
  - ***Unobservability*** *- user may use a resource or service without others, especially third parties, being able to observe that the resource or service is being used.*

© 2010 LOCKHEED MARTIN CORPORATION

Protection of Security Functions

- **Trusted software that is used to support the integrity and management of the TCB and the integrity of TCB data.**

- **The key features are:**
  - *Self test*
  - *Domain separation*
  - *Trusted recovery*
  - *Confidentiality/Integrity/Availability of exported data*
  - *Internal TCB data transfers*
  - *Replay detection*
  - *State synchrony*
  - *Time stamps*
  - *Inter TCB data consistency*

© 2010 LOCKHEED MARTIN CORPORATION

Resource Utilization

- **Trusted software that supports the availability of required resources.**
- **Three key features are:**
  - ***Fault Tolerance*** *- protection against unavailability of capabilities caused by failure of the TCB*
  - ***Priority of Service*** *- ensures proper allocation of resources to time-critical tasks*
  - ***Resource Allocation*** *- limits on the use of available resources, therefore preventing users from monopolizing the resources*

© 2010 LOCKHEED MARTIN CORPORATION

System / Subsystem Access

- **Trusted software that controls the establishment and the termination of a user's session**
  - *Starts with successful identification/authentication*
  - *Ends with de-allocation of session resources*
- **This software places limits on a user's domain based on user's physical location or device**
- **Other attributes that may be limited based on an individual user are:**
  - *Terminal locking based on inactivity*
  - *Advisory warning messages*
  - *Display of unsuccessful login attempts*

© 2010 LOCKHEED MARTIN CORPORATION

Trusted Path & Channel

- **Ensures that the user is communicating directly with the TCB whenever it is invoked and that untrusted applications cannot modify a user's response**

- **The key features are:**
  - *Distributed user authentication database updates*
  - *Mutual authentication between IT products*
  - *Distributed security audit collection*
  - *Web Enabled browser authentication*
  - *Only perform functions that the trusted software has approved requirements for*

© 2010 LOCKHEED MARTIN CORPORATION

Questions?

© 2010 LOCKHEED MARTIN CORPORATION

Secure Software Design Principles

© 2010 LOCKHEED MARTIN CORPORATION

Overview

- **In addition to functionality and performance, it is important to address trustworthiness when designing software**
- **Trusted software design principles guide secure software development practices**
  - ***Recommended regardless of the platform or language of the software***
  - ***Aids in creating software that is self-protecting and performs only intended functions***
  - ***Facilitates evaluation and verification of the trusted software***

© 2010 LOCKHEED MARTIN CORPORATION

Secure Software Design Principles

- **Keep Design Simple**
- **Exercise Defense in Depth**
- **Validate User/Application Data**
- **Privileges**
- **Secure the Weakest Link**
- **Never Assume Secrets are Safe**
- **Be Reluctant to Trust**
- **Fail Securely**
- **Privacy/Protect Sensitive Information**
- **Compartmentalization**
- **Isolation of Security Functions**
- **Secure Network Interfaces**

© 2010 LOCKHEED MARTIN CORPORATION

Keep Design Simple

- **Design trusted software to be small and simple**
  - *Must strike balance between simplicity and additional security features*
  - *Funnel security critical operations via a small number of* choke points *in a system*
  - *Use multiple small, simple, single-function software components that operate at a single security level*
  - *Only call the security function of a component when needed*
- **Minimize the Number of Trusted Functions**
  - *Limit trust to those components absolutely critical to the application's secure operation*
- **Minimize User Interfaces and Outputs**
  - *Ensure user interfaces only contain the required functions*
  - *Design interfaces so that they cannot be bypassed by the user*
- **Minimize the Number of Write-down Programs**
  - *Restructure data units*
  - *Concentrate write-down functions to one program*
  - *Perform careful analysis of program information flows*

© 2010 LOCKHEED MARTIN CORPORATION

Exercise Defense in Depth$_1$

- **Multiple security layers help prevent a full system breach if a single layer is compromised**
- **Use a series of defensive layers that work together to prevent the possibility of full system breach if a single layer is penetrated**

© 2010 LOCKHEED MARTIN CORPORATION

Exercise Defense in Depth$_2$

- **Example: Suppose we have a system that protects data that travel between various server components in enterprise systems…**

Defense Mechanism(s)	Attacker Effort Required to Steal Data
Corporate firewall	**Penetrate corporate firewall**
Corporate firewall + Data encryption	**Penetrate corporate firewall + Break data encryption**
Corporate firewall + Data Encryption + Application specific firewall	**Penetrate corporate firewall + Break data encryption + Penetrate application firewall**

© 2010 LOCKHEED MARTIN CORPORATION

Validation of User/Application Data

- **Understand all the potential areas where un-trusted inputs can enter your software (e.g. parameters or arguments)**
- **Assume all input is malicious**
- **Validate any values passed to the trusted software**
  - ***From trusted software or un-trusted software to ensure the values received are within the allowed data types/formats***

© 2010 LOCKHEED MARTIN CORPORATION

Enforce the Principle of Least Privilege

- **Give out no more privilege than necessary**
  - ***Applications and functions should be given "just enough" privileges to perform task***
  - ***Allowing unnecessary privileges can potentially allow unauthorized entry into sensitive areas***
    - UNIX Example: Given root permission, anything valid that an attacker tries will succeed
- **Extend privilege for shortest possible time**
  - ***Minimize windows of vulnerability***
- **Implement "Roles" concept**
  - ***Roles map to entities in the system that need to provide particular functionality***
  - ***Roles are associated with a set of privileges***

© 2010 LOCKHEED MARTIN CORPORATION

Secure the Weakest Link

- **Software security system is only as secure as its weakest link**
- **Malicious hackers will attack the weakest parts of the software**
  - *Most likely area to be easily broken*
  - *Attackers follow the path of least resistance*
- **Identifying weakest component falls directly out of a good risk analysis**
  - *Prudent to address the most serious risk first, instead of a risk that may be easiest to mitigate*
  - *Deal with problems in order of severity*
  - *Okay to stop addressing risks when all components appear to be within the threshold of acceptable risk*

© 2010 LOCKHEED MARTIN CORPORATION

Never Assume Secrets are Safe

- **Security is often about keeping secrets or confidences**
- **Don't rely on an obscure design or implementation to ensure the security of your system**
- **Always assume that an attacker can obtain enough information about your system to launch an attack**
- **Binaries can be reverse-engineered to obtain sensitive information**
  - *Using Decompilers or Disassemblers*
- **Avoid trusting even a dedicated network if possible**
- **Accidental and malicious attacks**
  - *A disgruntled employee abuses access*

© 2010 LOCKHEED MARTIN CORPORATION

Be Reluctant to Trust$_1$

- **It is risky to assume secrets hidden in code will be safe**
  - *Word documents*
  - *Algorithms*
- **Assumptions can compromise security**
  - *Be reluctant to extend trust*
- **Servers and clients should be designed to mistrust each other**
  - *Both can potentially be hacked*

© 2010 LOCKHEED MARTIN CORPORATION

Be Reluctant to Trust$_2$

- **Can COTS components be trusted to be secure?**
  - *Were the developers security experts?*
  - *Who developed the product?*
  - *What processes were used?*
- **Sadly, hundreds of products from security vendors have gaping security holes**
  - *Many security products introduce more risk than they address*
- **COTS products meeting IA requirements must have National Information Assurance Partnership (NIAP) Lab CCEVS evaluation certificates**
  - *http://niap.bahialab.com/cc-scheme/faqs/nstissp-faqs.cfm*

© 2010 LOCKHEED MARTIN CORPORATION

Free and Open Source Software

- **While FOSS use is appealing from a number of standpoints, it requires the same assurance as COTS does in a trusted software environment.**
- **Evaluation Costs**
  - *Decision for use cannot be made based on it being "free"*
  - *Requires approval from legal to use (copyright and licensing in order)*
  - *Treated the same as COTS for DoD and U.S. Government and therefore may require an evaluation as directed by NSTISSP Policy #11*
- **Security Assurance**
  - ***Additional requirements on the pedigree of the software with respect to security:***
    - Every country's government will have differing acceptance criteria depending in what foreign country the software is designed and built?
    - What is known about the nationality and the security risks of the individuals contributing to the code base?
    - Is there subversive code in the product?

© 2010 LOCKHEED MARTIN CORPORATION

Fail Securely

- **Failure is usually unavoidable…plan for it!**
  - *Security problems related to failure is avoidable*
- **Systems exhibit insecure behavior when they fail**
  - *Attackers only need to cause or wait for the right kind of failure in order to exploit the system*
- **Incorporate mechanism(s) to ensure that the system is in a non-compromising state during and after a failure**
  - ***Default to deny access***
  - ***Undo changes and restore to a secure state***
  - ***Always check return values for failure***

© 2010 LOCKHEED MARTIN CORPORATION

Protect Sensitive Information

- **It is important to prevent access to sensitive information or obscure it to alleviate the risk of information leakage**
- **Delete sensitive information after it is used or no longer needed**
  - *Ensures that if the system is compromised, there is nothing interesting stored there long term*
- **Stored sensitive data**
  - *Encrypt for Confidentiality* **-** *AES 256 (Public Algorithms)*
    - Keep the encryption key in a secure location
  - *Hash for Integrity*
    - Use cyclic redundancy checks (CRC) or secure hash algorithms[1] (SHA)
- **Remove unused code/ features from trusted software**
  - *Pay particular attention to debug software since debug code opens up the trusted software to an external interface*

[1]*Creates Message Digest*

© 2010 LOCKHEED MARTIN CORPORATION

Compartmentalization$_1$

- **Compartmentalizing minimizes the amount of damage that can be inflicted on a system**
- **Break up the system into as few units as possible**
  - *Remember to isolate security relevant functions*
- **If one unit is breached, the other units will not be affected**
  - *Same principle as multiple chamber system on submarines*

Note: Over compartmentalizing can make the system completely unmanageable if there is too much segregation of functionality

"need to know vs. need to share"

© 2010 LOCKHEED MARTIN CORPORATION

Compartmentalization$_2$

- **It is easier to point out poor examples of compartmentalization than it is to find good examples…**
  - ***UNIX privilege model – operations work on an all-or-nothing basis***
  - ***Most operating systems do not compartmentalize, it is not possible to protect one part of the OS from others***

- **Some OS's are compartmentalized such as**
  - ***Trusted Solaris***
  - ***STOP OS:XTS-400***
  - ***NSA Evaluated High Robustness Separation Kernels (GHS, WindRiver, LynuxWorks)***

© 2010 LOCKHEED MARTIN CORPORATION

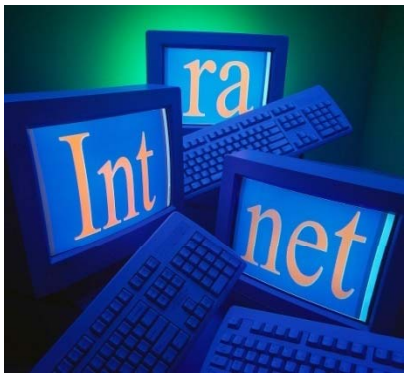
Isolation of Security Relevant Functions

- **Use protection features of operating system to provide program isolation**
  - ***Does your home PC run as Admin or Restricted User?***
- **Provide at least one isolated security domain in which trusted software will execute**
- **Use language features to minimize interaction between program packages**
- **Partition memory and/or computing assets such that trusted software is isolated from un-trusted software.**
- **Ensure un-trusted components have no access to security relevant information, functions or privileges**
- **Ensure non-security relevant functionality cannot tamper with trusted software and adversely affect correct operation of TCB**

© 2010 LOCKHEED MARTIN CORPORATION

Secure Network Interfaces

- **Entry and exit points for the trusted software must be well defined, understood and documented.**
- **Activate the network ports when a request for communication has been received.**
- **Deactivate the network interface once the communication has been completed**
- **Audit the fact that an external communication has occurred and record the information in the security audit log**
- **Remove networking features not required by the trusted software to perform its function**



© 2010 LOCKHEED MARTIN CORPORATION

Questions?

© 2010 LOCKHEED MARTIN CORPORATION

Static Code Analysis

© 2010 LOCKHEED MARTIN CORPORATION

Static Code Analysis Tool$_1$

- **Automated tool for helping analysts find security-related problems in software**
- **Based on the idea that many software vulnerabilities are in reusable library functions**
  - ***Programs can be scanned to check whether they contain calls to those functions***
    - *Equivalent to opening up source code in editor and searching for the name of vulnerable functions such as strcpy() and stat()*
- **Sophisticated analyzers use data and control flow analysis to find subtle bugs and reduce false alarms**
- **Tools are not designed to replace a human analyst!**
  - *Purpose of security analyzers is **not to be smarter** than human experts but **to be faster***

© 2010 LOCKHEED MARTIN CORPORATION

Static Code Analysis Tool$_2$

- **Analyzers are frequently used during**
  - *Source-code audits and walkthroughs*
  - *Development phase (so developers can integrate analyzer into the development environment)*
- **Not intended to detect design or architecture level flaws**
- **Not well suited for finding integration bugs**
  - *No gains by putting off use until late in the software development life cycle*

© 2010 LOCKHEED MARTIN CORPORATION

Static Code Analysis Tool$_3$

- **There are two types of analyzers**
  - ***Software semantics analyzer***
    - Predicts software behavior through static analysis
    - Focus on reducing false alarms
    - Helps identify problems that would be harder for human analyst to see
  - ***Style-checker***
    - Focus on detecting programming practices that should be avoided
    - Must be used from the very beginning of the development phase, otherwise it will be too expensive to bring software into compliance with guidelines the analyzer tries to enforce

© 2010 LOCKHEED MARTIN CORPORATION

Commercial Static Analysis Tools

- **Multi-Language**
  - ***CodeSesure***
  - ***Fortify***
  - ***Ounce Security Analyzer***
  - ***LDRA TestBed***
  - ***GammaTech CodeSonar***
  - ***Klockwork Insight***
  - ***Parasoft***
  - ***Coverity Prevent***
- **C/C++**
  - ***Microsoft PREfast***
  - ***PC-Lint***
  - ***Green Hills Software DoubleCheck***
  - ***HP Code Advisor***
  - ***Abraxas Software CodeCheck***
- **.NET**
  - ***ReSharper***
  - ***NDepend***
  - ***CodeIt.Right***
- **Java**
  - ***checKing***
  - ***IntelliJ IDEA***
  - ***Swat4j***

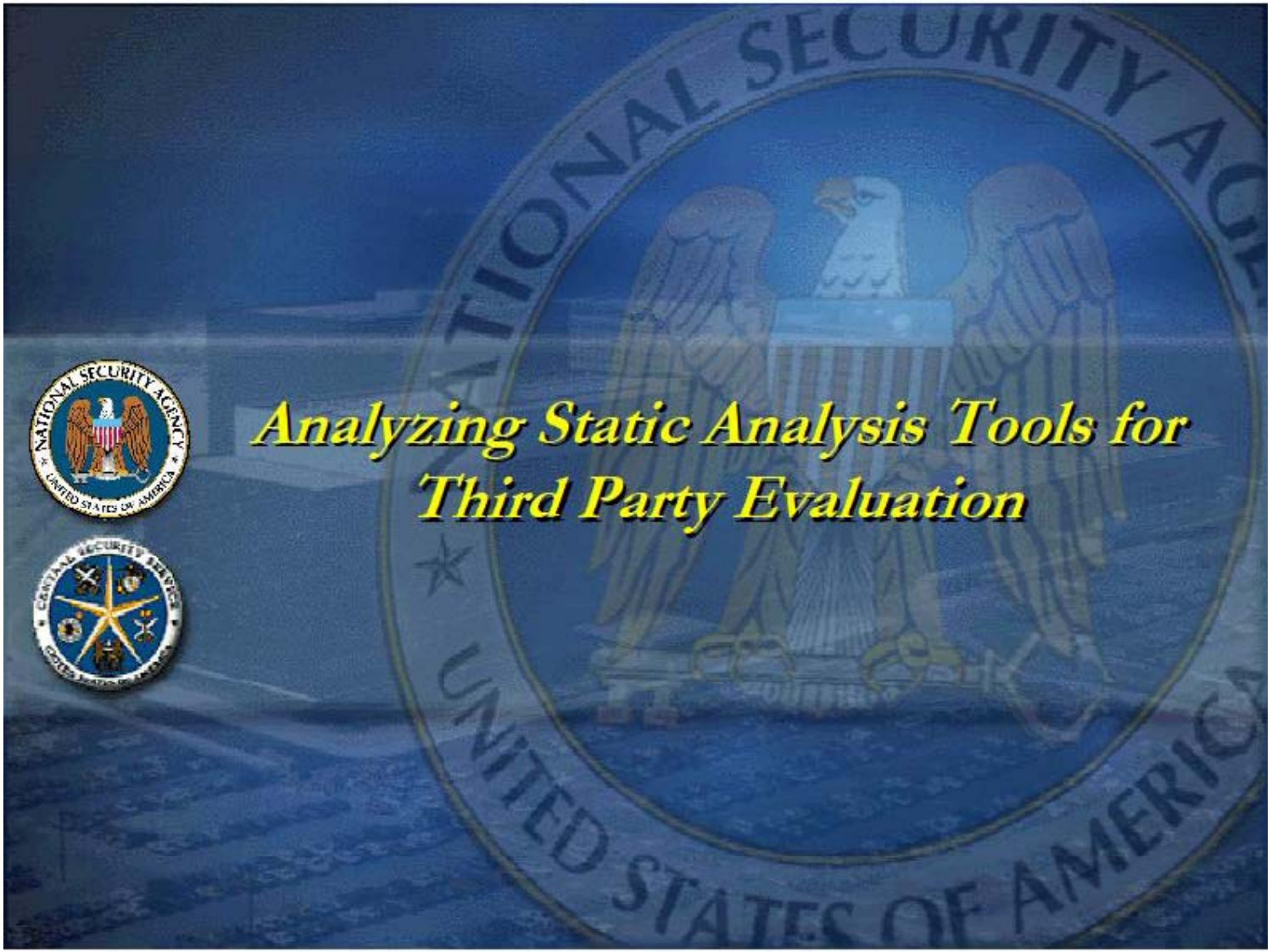
© 2010 LOCKHEED MARTIN CORPORATION

Open Source Static Analysis Tools

- **Multi-Language**
  - ***RATS***
  - ***Yasca***
- **C/C++**
  - ***Sparse***
  - ***Splint***
  - ***Uno***
  - ***BLAST***
  - ***Fragma-C***
  - ***CppCheck***
- **.NET**
  - ***FxCop***
  - ***StyleCop***
- **Java**
  - ***FindBugs***
  - ***Checkstyle***
  - ***Sonar***
  - ***PMD***

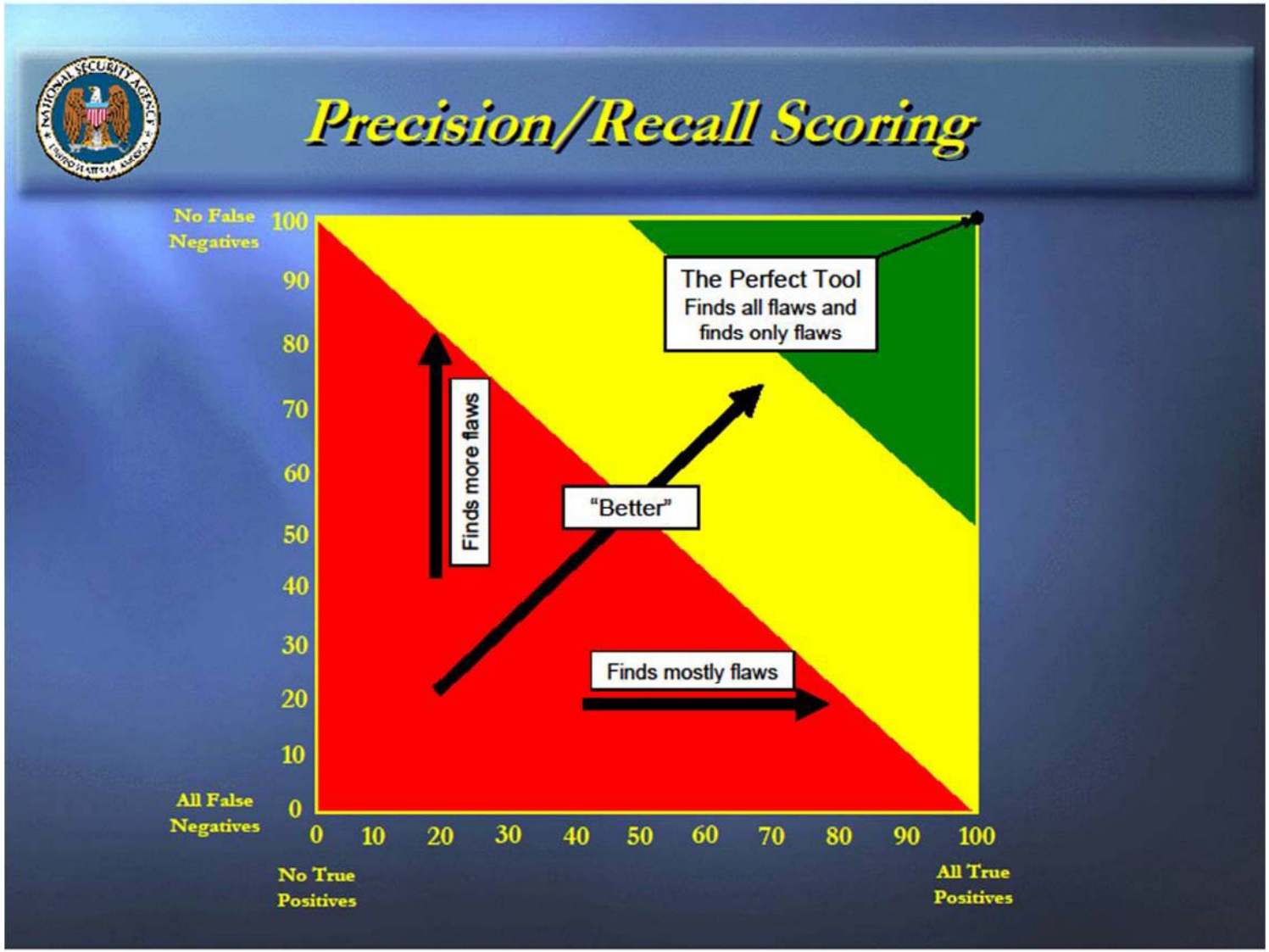
© 2010 LOCKHEED MARTIN CORPORATION

Reported Dec 2006 DHS SwA Forum

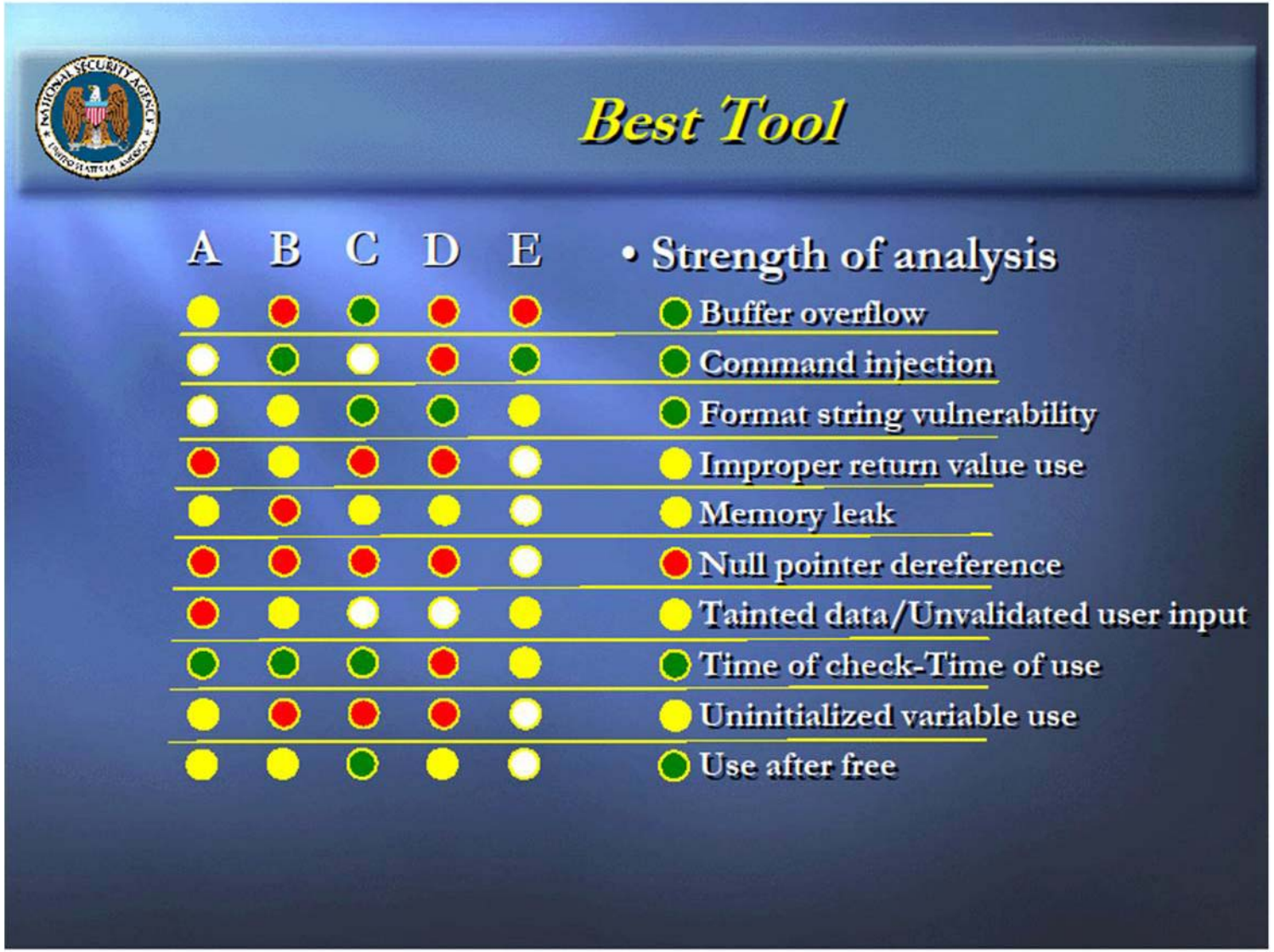


© 2010 LOCKHEED MARTIN CORPORATION

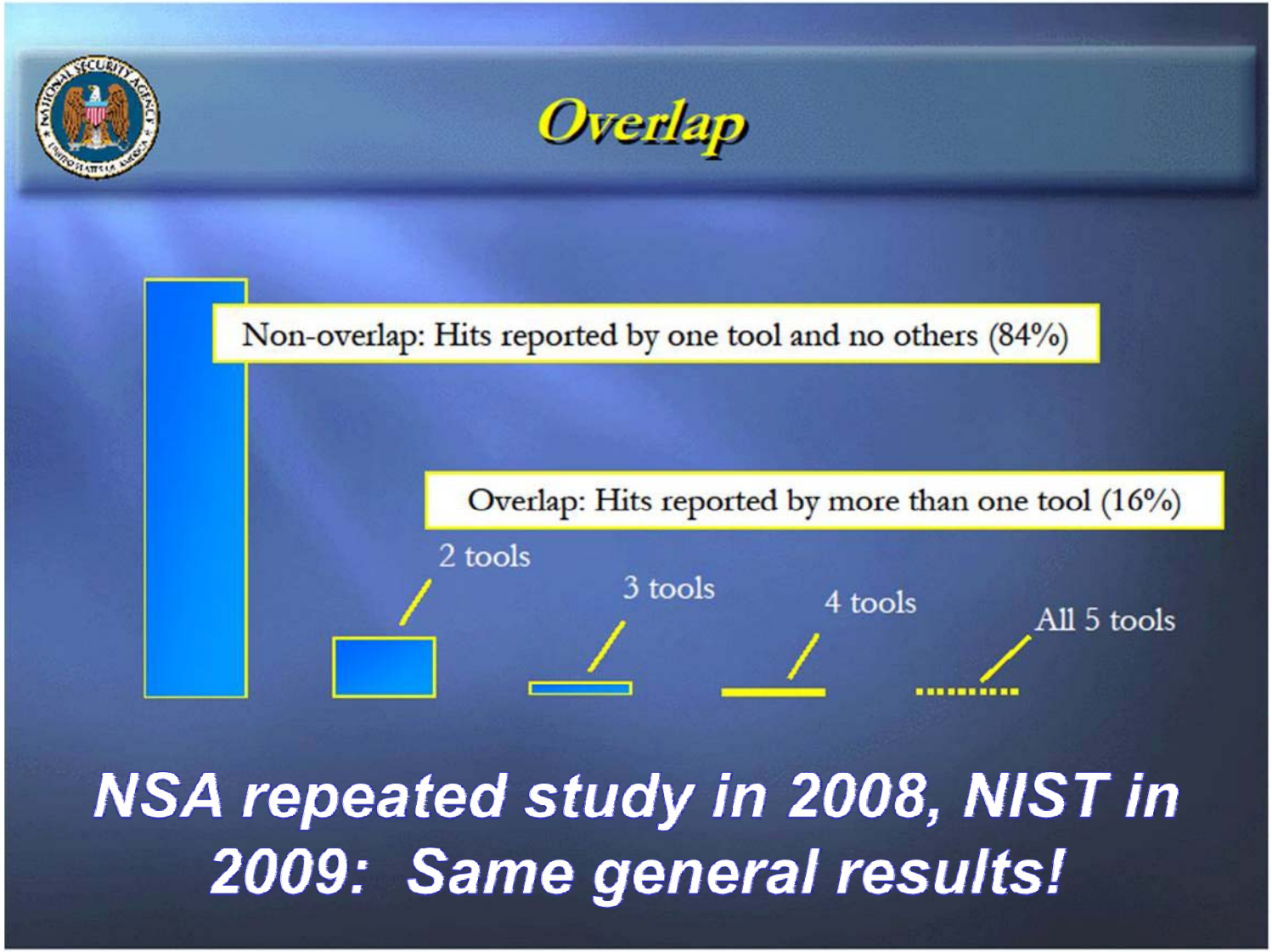
SCA is for assistance, not Ultimate QC!



© 2010 LOCKHEED MARTIN CORPORATION



© 2010 LOCKHEED MARTIN CORPORATION



© 2010 LOCKHEED MARTIN CORPORATION

SCA Guidelines

- **Static code analysis is required for ALL deliverable trusted software**
  - *Using analysis tool will help make source code analysis more efficient*
- **Being cognizant of security vulnerabilities and instituting countermeasures during ALL phases of development is the best and most affordable way to mitigate security risks**

BUILD SECURITY IN!!!

© 2010 LOCKHEED MARTIN CORPORATION

Questions?

© 2010 LOCKHEED MARTIN CORPORATION

Software Vulnerabilities: Addressing Security Errors in Software

© 2010 LOCKHEED MARTIN CORPORATION

Introduction$_1$

- **Many commonly exploited software vulnerabilities are caused by:**
  - ***Common programming flaws***
  - ***Inherent weaknesses in some programming languages***
  - ***Lack of secure coding standards.***

- **Therefore, it is important for an organization to adopt and enforce a uniform set of rules and guidance for security.**

- **Several organizations and efforts are working towards standards and practices that will help improve software security by building it into the development rather than responding to the vulnerabilities after the exploit.**

© 2010 LOCKHEED MARTIN CORPORATION

Introduction$_2$

- **There are a number of sources available that provide information on coding practices and rules, including the following:**
  - ***SEI CERT Secure Coding Standards*** *- a collaborative site that provides rules and recommendations for secure coding practices in the C and C++ programming languages*
  - ***MITRE Common Weaknesses Enumeration (CWE)*** *- a dictionary of known security weaknesses in code, design and architecture*
  - ***Software Assurance Metrics And Tool Evaluation (SAMATE) Reference Dataset (SRD)*** *– a set of programs with known weaknesses in code, design and architecture*
  - ***Build Security In Website*** *–collection of resources describing best practices and guidance for incorporating security into the development process*

- **These sources can be used to create a custom set of prescriptive coding rules that can be used as part of the development of trusted software.**

https://buildsecurityin.us-cert.gov/swa/forums.html

© 2010 LOCKHEED MARTIN CORPORATION

Addressing Security Errors in Software

- **This section will focus on implementation-level vulnerabilities, which are typically most prevalent and easiest to fix if caught during development.**
- **This section aims to answer the following questions:**
  - *What constitutes a software vulnerability?*
  - *Under what circumstances can a potential vulnerability occur?*
  - *What does vulnerable code look like?*
  - *What is the most effective mitigation strategy for a given vulnerability?*

© 2010 LOCKHEED MARTIN CORPORATION

Vulnerabilities

- **Vulnerability**
  - ***Any problem that can be exploited by an attacker***
    - Weakness which allows an attacker to violate the integrity of that system

- **Vulnerability Types**
  - ***Local implementation errors***
    - Use of the gets() function call in C/C++
  - ***Iterprocedural interface errors***
    - A race condition between an access control check and a file operation
  - ***Higher design-level mistakes***
    - Error handling and recovery systems that fail in an insecure fashion, or object-sharing systems that mistakenly include transitive trust issues

© 2010 LOCKHEED MARTIN CORPORATION

Classifying Vulnerabilities

- **To measure risk in a system, vulnerabilities must be identified**
  - *Software vulnerabilities remain uncategorized and unidentified*
- **Vulnerability collections exist, but lack categorizations**
  - *Focus on specific occurrences of bugs and flaws*
    - Bugtraq
    - Mitre's CVE
- **Some efforts underway to solve this problem**
  - *Mitre's Common Weaknesses Enumeration (CWE)*
    - Focuses on generic problems rather than specifics
  - *CERT Secure Coding Standards (C,C++, Java)*
    - Prescriptive coding rules for security

© 2010 LOCKHEED MARTIN CORPORATION

Finding Vulnerabilities

- **Examining the code**
  - *More complex vulnerabilities require more code to be examined*
  - *Static analyzers help to automate this process*
- **Examining the design from a high, but descriptive level**
  - *Requires expertise about the system*
  - *Hard to automate*
- **Examining the execution environment**

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - *Buffer Overflows*
  - *Integer Overflows*
  - *Format String*
  - *Injection Flaws*
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

Unvalidated Input

- **“Never Trust Input”**
- **Unvalidated Input forms the basis for some of the worst and most frequently exploited vulnerabilities**
- **Always validate all input, from all sources:**
  - *Databases*
  - *Command-line*
  - *Environment Variables and System Properties*
  - *Temporary Files*
- **Use strong input validation**
  - ***Only accept well-formed input: whitelist validation***
    - Regular expressions are your friends
  - ***Avoid checking explicitly for bad input: blacklist validation***
  - ***Reject bad data, don’t try to repair it***

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - ***Buffer Overflows***
  - *Integer Overflows*
  - *Format String*
  - *Injection Flaws*
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

Buffer Overflow

- **Single biggest software security threat!**
- **Occurs when a program allows input to write beyond the end of the allocated buffer**
- **Caused by lack of bounds checks on array and pointer references**
- **Side effects of overrunning a buffer can range anywhere from a crash to a hacker gaining complete control of the application**
- ***Stack-smashing* attack targets careless use of data buffers allocated on the program's runtime stack**
  - *Creative hacker can usually run arbitrary code through stack smashing*
    - Hacker places some attack code somewhere
    - Overwrites the stack in such a way that control gets passed to the attack code

© 2010 LOCKHEED MARTIN CORPORATION

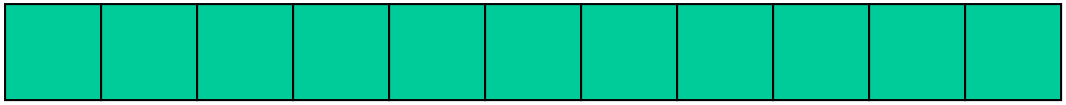
Affected Languages

- **Because C and C++ have no runtime checks that prevent writing past the end of a buffer, it allows programs to overflow buffers at will (or completely by accident)**
  - *Programmers have to perform the check on their own code*

- **Java, C# and VB have native string types, bounds-checked arrays and generally prohibit direct memory access**
  - *Buffer overrun much less likely in these languages, but not impossible*

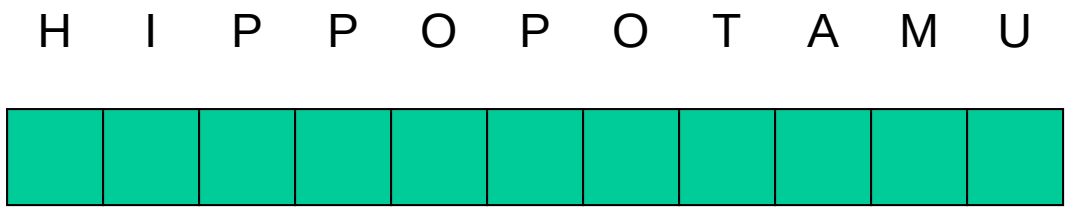
© 2010 LOCKHEED MARTIN CORPORATION

Buffer Overflow



© 2010 LOCKHEED MARTIN CORPORATION

Buffer Overflow



© 2010 LOCKHEED MARTIN CORPORATION

Prevention and Countermeasures

- **Always use a compile time constant to initialize buffers and do bounds checking**
  - *i.e. char* buffer[BUFFER_SIZE];*
- **Make sure all buffer manipulations are bounded by their size so overflow cannot occur**
  - *i.e. memset(buffer, 0, BUFFER_SIZE);*
- **Replace dangerous string handling functions**
  - *i.e. strcpy, strcat, sprintf, etc.*
  - *Instead use strlcpy, strlcat*
- **Replace C string buffers with C++ strings**
  - *Uses the Standard Template Library (STL)*
  - *User isn't responsible for null termination*
  - *Increased strictness of C++ compiler*
- **Use analysis tools to identify security defects**

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - *Buffer Overflows*
  - ***Integer Overflows***
  - *Format String*
  - *Injection Flaws*
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

Integer Overflows$_1$

- **For nearly every binary format available to represent numbers, there are operations that don't give you typical results as you would expect on pencil and paper.**
  - ***Some languages implement range–checked integer types***
    - Reduce problems when used consistently
- **Occurs when an integer is increased beyond its maximum value and wraps-around or "overflows" into its minimum value.**
- **Effects range from crashes and logic errors to escalation of privileges and execution of arbitrary code**
- **Can be triggered by user provided input**

© 2010 LOCKHEED MARTIN CORPORATION

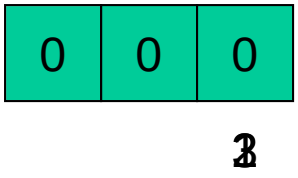
Integer Overflows$_2$

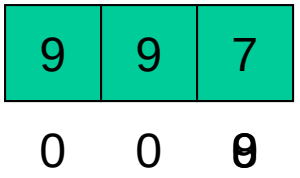
- **The following operations are likely to cause an integer overflow:**
  - *Casting operations*
  - *Operator conversions*
  - *Arithmetic Operations*
  - *Comparison Operations*
  - *Binary Operations*

© 2010 LOCKHEED MARTIN CORPORATION

Affected Languages

- **All languages are affected by integer overflows**
  - ***Prone to denial of service and logic errors***
- **Overflows can be signed or unsigned**
- **C and C++ have true integer types**
- **C# insists on signed integers**
- **Java only supports a subset of the full range of integer types**
  - ***Supports 64 bit integers***
  - ***Only supports the char unsigned type***





© 2010 LOCKHEED MARTIN CORPORATION

Prevention and Countermeasures

- **Check numeric input against a max and min bound before using it, and after any operations which may cause overflow.**
- **Make checks for integer problems straightforward and easy to understand.**
- **Use unsigned integers where possible for array offsets and memory allocation sizes.**
- **Check all calculations used to determine memory allocations or array indexes.**
- **Pay close attention to code that catches integer exceptions.**

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - *Buffer Overflows*
  - *Integer Overflows*
  - ***Format String***
  - *Injection Flaws*
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

Format String Problems

- **Results from trusting user supplied input without validation**
- **Attacker provides input that will later be included as a format string argument**
- **Can be used to write to arbitrary memory locations**
  - *Can happen without tampering with adjoining memory blocks*
  - *Allows hacker to bypass stack protections and possibly modify small portions of memory*
  - *Results in the usual exploits of a buffer overflow attack*
- **More common on UNIX and Linux systems**
- **On Windows, hacker has to rewrite main executable or resource DLL to create format string bugs**
- **Users can be easily misled with format string bugs**

© 2010 LOCKHEED MARTIN CORPORATION

Affected Languages

- **C and C++ are most strongly affected by format string errors**
- **Attack can lead to immediate arbitrary code execution and information leak**
- **Java and C# don't usually allow execution of arbitrary code, but attacks are not impossible**

© 2010 LOCKHEED MARTIN CORPORATION

Prevention and Countermeasures

- **Never pass user input directly to a formatting function**
  - *Do this at every level of handling input*
- **Ensure input strings used by the application are only read from trusted sources**
  - *Use a whitelist to rigorously test user-supplied format strings*
- **Avoid or minimize use of printf family of functions**
  - *Use stream operators instead*
- **Pass formatting specifiers to the application to ensure no hexadecimal characters are returned**

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - *Buffer Overflows*
  - *Integer Overflows*
  - *Format String*
  - ***Injection Flaws***
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

Injection Flaws

- **Involves the insertion of control structures from user input where the program was expecting data.**
- **All major scripting and markup languages are potentially vulnerable.**
- **Attacks can involve unauthorized disclosure of sensitive information, authentication bypass, arbitrary code execution, data loss, and more.**

© 2010 LOCKHEED MARTIN CORPORATION

Injection

(adjective) (name) sat on a (thing).

(adjective) (name) had a great fall.

All the King's (plural things) and all the King's (plural things) couldn't put (adjective) (name) back together again.

© 2010 LOCKHEED MARTIN CORPORATION

Injection

- **adjective = “Humpty”**
- **name = “Dumpty”**
- **thing = “wall.”**
- **thing, plural = “horses”**
- **thing, plural = “men”**

- **Malicious Injection “He reminds me of my last manager”**

© 2010 LOCKHEED MARTIN CORPORATION

Injection

Humpty Dumpty sat on a wall. He reminds me of my last manager.

Humpty Dumpty had a great fall.

All the King's horses and all the King's men couldn't put Humpty Dumpty back together again.

© 2010 LOCKHEED MARTIN CORPORATION

Command Injection

- **Untrusted data passed through and interpreted as a command**
- **Unprivileged users given full control of directory structure or unauthorized data access**
- **Commonly through API calls that directly call the system command interpreter without validation**

© 2010 LOCKHEED MARTIN CORPORATION

Affected Languages

- **Any language where commands and data are placed inline together**
- **Most languages handle this vulnerability by providing good APIs with proper input validation**
- **New APIs can still introduce new command injection errors**

© 2010 LOCKHEED MARTIN CORPORATION

Prevention and Countermeasures

- **Perform input validation before passing to command processor**
- **Fail securely if input validation check fails**
  - *Signal an error - refuse to run command as is*
  - *Log the error and all relevant data*
- **Use a *whitelist* validation approach**
  - *Use regular expressions to ensure that input contains no dangerous meta-characters, such as ";" or "&&"*
- **Write your own secure API wrappers**
  - *Use additional validation techniques*
  - *Ensures that validation is always performed*

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - *Buffer Overflows*
  - *Integer Overflows*
  - *Format String*
  - *Injection Flaws*
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

Data Protection/Storage Failure$_1$

- **When storing data securely, consider the following:**
  - *Permission required to access the data*
  - *Data encryption issues*
  - *Threats to stored secrets*

- **Two major components:**
  - *Weak access control mechanisms*
  - *Hard coding secret data*

© 2010 LOCKHEED MARTIN CORPORATION

Data Protection/Storage Failure$_2$

- **Weak Access Controls to Protect Secret Data**
  - *Allow full access to anyone*
    - Everyone group in Windows, world in UNIX
  - *Create an executable that can be written by ordinary users*
  - *Create a writable executable that is set to run as root or localsystem*
  - *Writable configuration information*
    - Ability to alter process or library paths
  - *Inappropriate information readable by unprivileged users*

© 2010 LOCKHEED MARTIN CORPORATION

Data Protection/Storage Failure$_3$

- **Embedding Secret Data in Code**
  - *Hard coding the password required for database connection in the application code*
    - `if (strcmp(password, “badidea"))`
  - *Embedding encryption key used to communicate with servers*
    - `if (key.equals("68af404b513073584c4b6f22b6c63e6b"))`
    - **If key is decoded, it becomes public and all users of the system and possibly the servers themselves may have to be updated**

© 2010 LOCKHEED MARTIN CORPORATION

Prevention and Countermeasures

- **Threat modeling**
  - *Think about the access controls explicitly placed on objects and those inherited by default*
- **Avoid storing sensitive data on general purpose, production server**
- **Leverage the security capabilities of the operating system**
- **Use appropriate permissions**
  - *Access control lists*
- **Remove "secret" memory once it's been used**
  - *Scrub memory before freeing it*
- **Always use a well known encryption standard (AES, 3DES, RSA) over a homebrewed encryption scheme.**

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - *Buffer Overflows*
  - *Integer Overflows*
  - *Format String*
  - *Injection Flaws*
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

Improper Error Handling$_1$

- **Failure to handle an error condition correctly**
- **Program can be left in an insecure state**
- **Application dies resulting in denial of service issue**
  - *Crashes, aborts, restarts*
- **Common source is sample code that is copied and pasted into a program.**
  - *Usually does not include error checking*

© 2010 LOCKHEED MARTIN CORPORATION

Improper Error Handling$_2$

- **Six variants:**
  - *Yielding too much info*
    - Tell the user what failed, why and how to fix it
  - *Ignoring errors*
    - Disregarding return values
  - *Misinterpreting errors*
  - *Using useless error values*
  - *Handling the wrong exceptions*
  - *Handling all exceptions*

© 2010 LOCKHEED MARTIN CORPORATION

Affected Languages

- **Languages that uses function return values (C\C++)**
- **Languages that rely on exceptions (C# and Java)**

© 2010 LOCKHEED MARTIN CORPORATION

Prevention and Countermeasures

- **Handle the appropriate exceptions**
- **Don’t “gobble” exceptions**
- **Check return values**
  - *Security-related functions*
  - *Every function that changes the user setting or machine wide settings*
- **Make every attempt to recover from error gracefully**
- **Avoid leaking error information to untrusted users**

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - *Buffer Overflows*
  - *Integer Overflows*
  - *Format String*
  - *Injection Flaws*
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

Information Leakage$_1$

- **Unauthorized access to data that leads to a breech of security policy**
- **Two primary ways information is leaked:**
  - *By accident*
    - Valuable information gets out due to a problem in the code or non-obvious channel
    - Comments left in the code, verbose error messages, etc
  - *By intention*
    - Confidential data which are not properly protected
    - Privacy issues
      - *Account numbers, user identifiers and user specific data*

- **Important to understand the techniques and approaches used by hackers and the types of information useful to them**

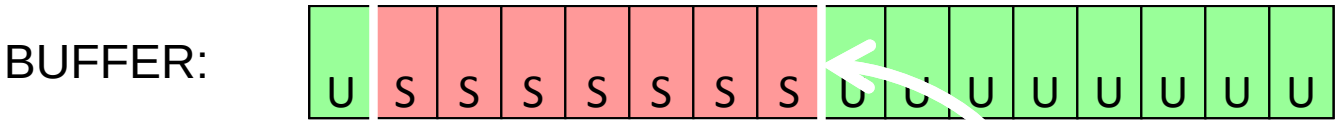
© 2010 LOCKHEED MARTIN CORPORATION

Information Leakage$_2$

- **Ways to accidentally leak information include:**
  - ***Side Channels***
    - *Timing channels* - hacker learns about internal state of the system by measuring how long operations take to run
    - *Storage Channels* - hacker allowed to look at data and extract information from it
  - ***Too Much Information (TMI)***
    - *Exposing whether usernames are correct*
    - *Detailed version information*
    - *Host network information*
    - *Application Information*
    - *Path Information*
    - *Stack Layout Information*

© 2010 LOCKHEED MARTIN CORPORATION

Covert Channel Example (Distributed Communication Buffer)



Step 1: Buffer is empty **BAD!**

Step 2: Object B sends 16 SECRET octets

Step 3: Object A sends UNCLASSIFIED octet and a double

Octet put into the buffer

7 bytes skipped for alignment

Double put into the buffer

© 2010 LOCKHEED MARTIN CORPORATION

Affected Languages

- **This vulnerability is language-independent.**
  - *High-level languages can exacerbate the problem by providing verbose error messages*

© 2010 LOCKHEED MARTIN CORPORATION

Prevention and Countermeasures

- **Define who should have access to what**
- **Use OS defenses such as access control lists and permissions**
- **Use crypto to protect sensitive data**
- **Do not disclose system status information to untrusted users**
- **Provide only the minimal amount of error information as necessary**

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - *Buffer Overflows*
  - *Integer Overflows*
  - *Format String*
  - *Injection Flaws*
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

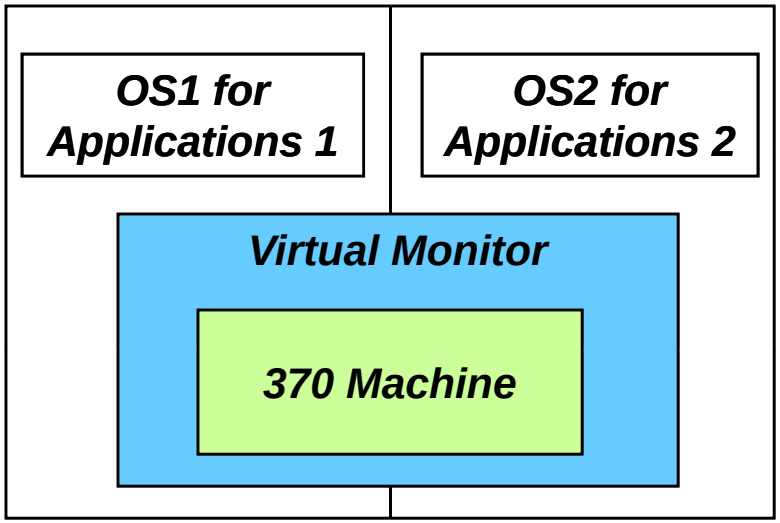
Race Conditions

- **Race condition occurs when an assumption needs to hold true for a period of time, but actually may not**
  - *Window of vulnerability* is the period of time when violating the assumption leads to incorrect behavior
- **Possible in environments in which there are multiple threads or processes occurring simultaneously that may potentially interact**
- **Race conditions can result in resource exhaustion, unexpected state, program crashes, or ability to view or manipulate sensitive data**

© 2010 LOCKHEED MARTIN CORPORATION

History of O/S

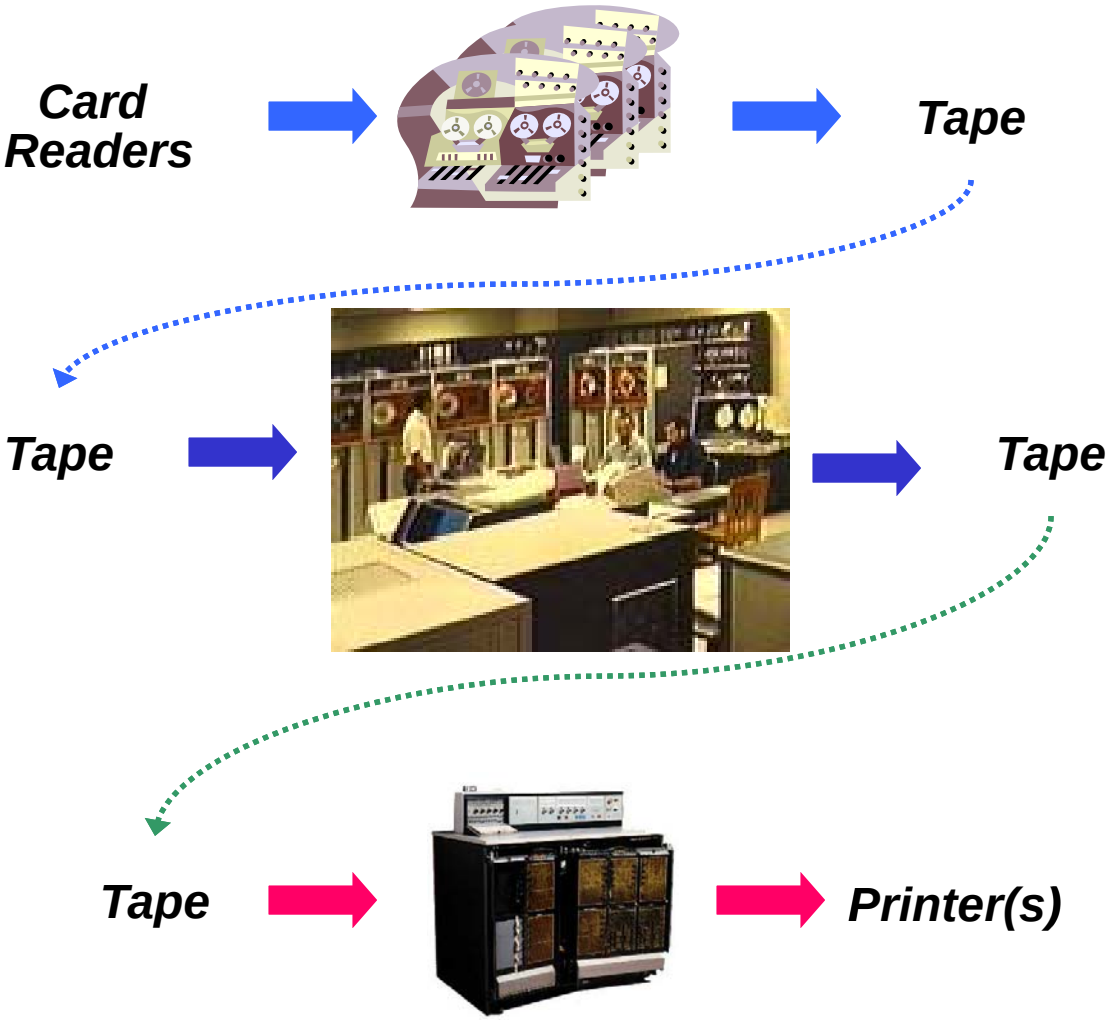
- 1964 System/360 OS: multiuser, different types of applications
- 1968 Multics: Multiplexed Information and Computing services (GE and MIT - educational)
- 1972 370/VM (Virtual Machine): Divide machine in layers. A single machine cannot serve all applications



- 1978 Unix (AT&T) Minis and Micros - 500 different hardware running flavors (portable and modular)
- 1978 MS-DOS
- 1985 Windows 1.0 released. (Announced in 1983, ship date was April 1984.)
- *"And you know the rest of the Microsoft Story"*

© 2010 LOCKHEED MARTIN CORPORATION

Batch Processing



© 2010 LOCKHEED MARTIN CORPORATION

Batch (cont'd.)

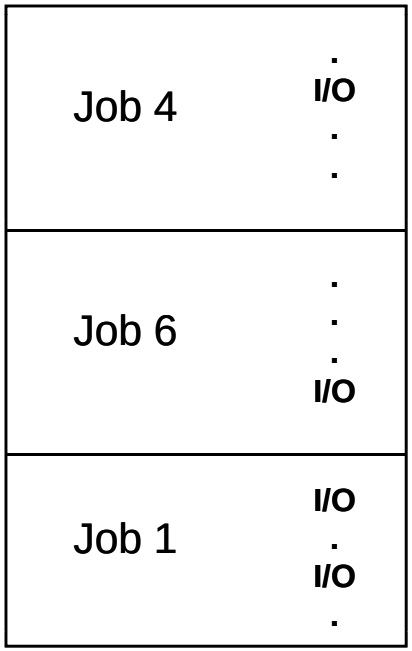
- **Jobs are sequential**
  - ***One program in Memory / CPU at a time.***
- **Each job runs until completion which minimized throughput of jobs.**
  - ***30 min***
  - ***5 min***
  - ***1 min***
- **If human interaction was required, program writer gave instructions to the computer operator.**
- **If Input / Output occurs, CPU waits until I/O completes.**

© 2010 LOCKHEED MARTIN CORPORATION

Multiprogramming O/S

Tasks are: running, waiting to run, or suspended

Random Access Memory



- *Advantages*
  - *CPU Doesn't wait for completion of I/O.*
  - *Memory Protection prevents cross process corruption.*
  - *Computationally intense jobs (i.e. little I/O) have high throughput.*
- *Disadvantages*
  - *Unfortunately, I/O intensive jobs can be starved of CPU cycles.*

© 2010 LOCKHEED MARTIN CORPORATION

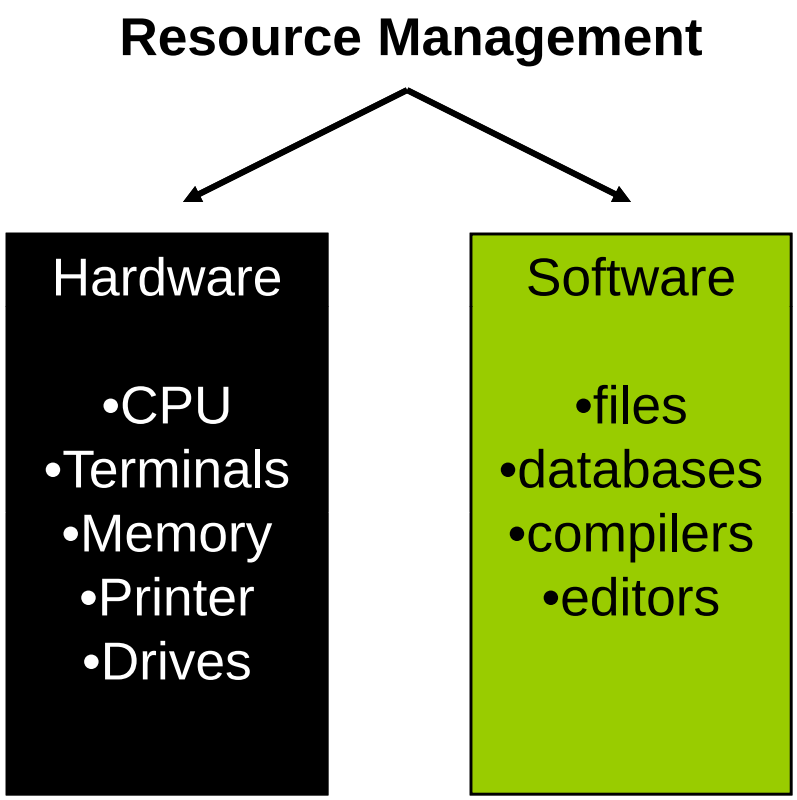
Multitasking (Time Sharing) O/S

- **Maintained from Multiprogramming:**
  - ***the partitioned memory***
  - ***I/O context switch mechanism based on Input/Output of programs***

- **Features Added:**
  - ***Hardware Interrupts signal the O/S to perform a context switch.***
  - ***The Interrupts are very periodic, thus dividing time into equal slices.***
  - ***O/S performs a round robin scheduling of jobs.***

© 2010 LOCKHEED MARTIN CORPORATION

Multiprogramming/Multitasking Issues



The singularity of some devices, such as the printer, require the O/S is designed to prevent concurrent use.

If Job 1 and Job 2 are both printing, switching back and forth between executions, will produce garbage.

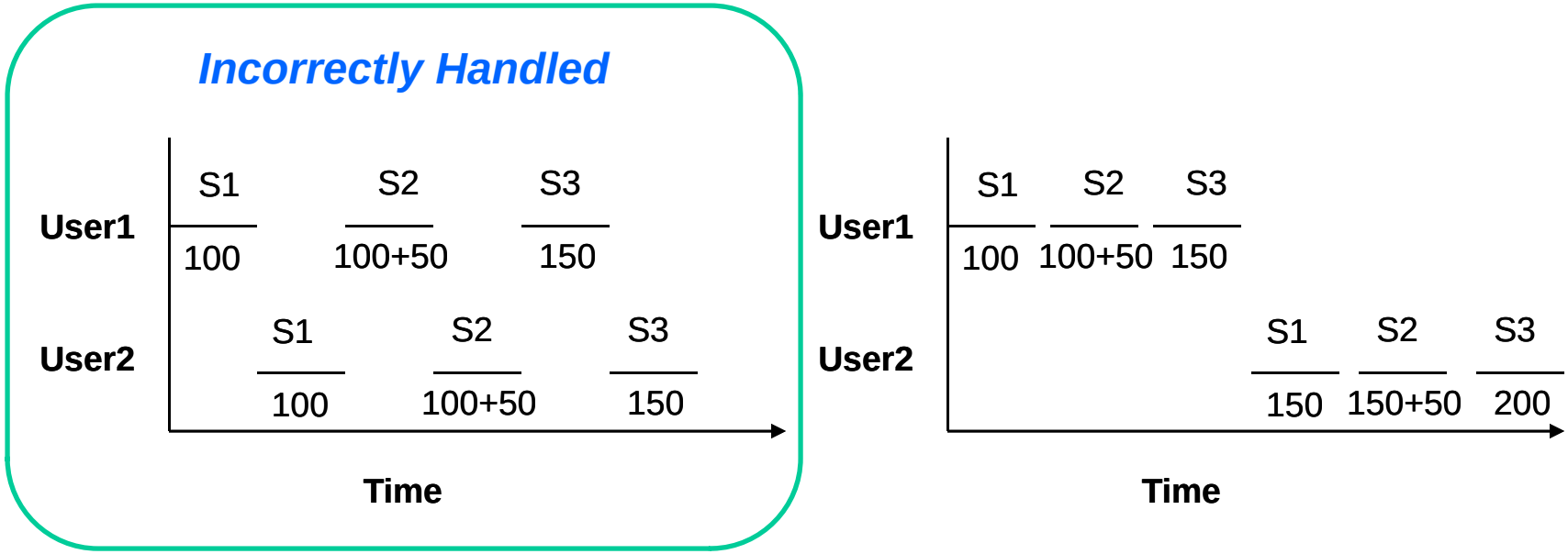
Have to provide mutual exclusion and blocking, but done correctly!!!

© 2010 LOCKHEED MARTIN CORPORATION

Race Condition

- **Used to keep two processes from using the same resource "simultaneously**
- **Given, two individuals with a joint banking account deposit $50 at two different bank branches at the same time.**

S1: read balance S2: add 50 to balance S3: store balance



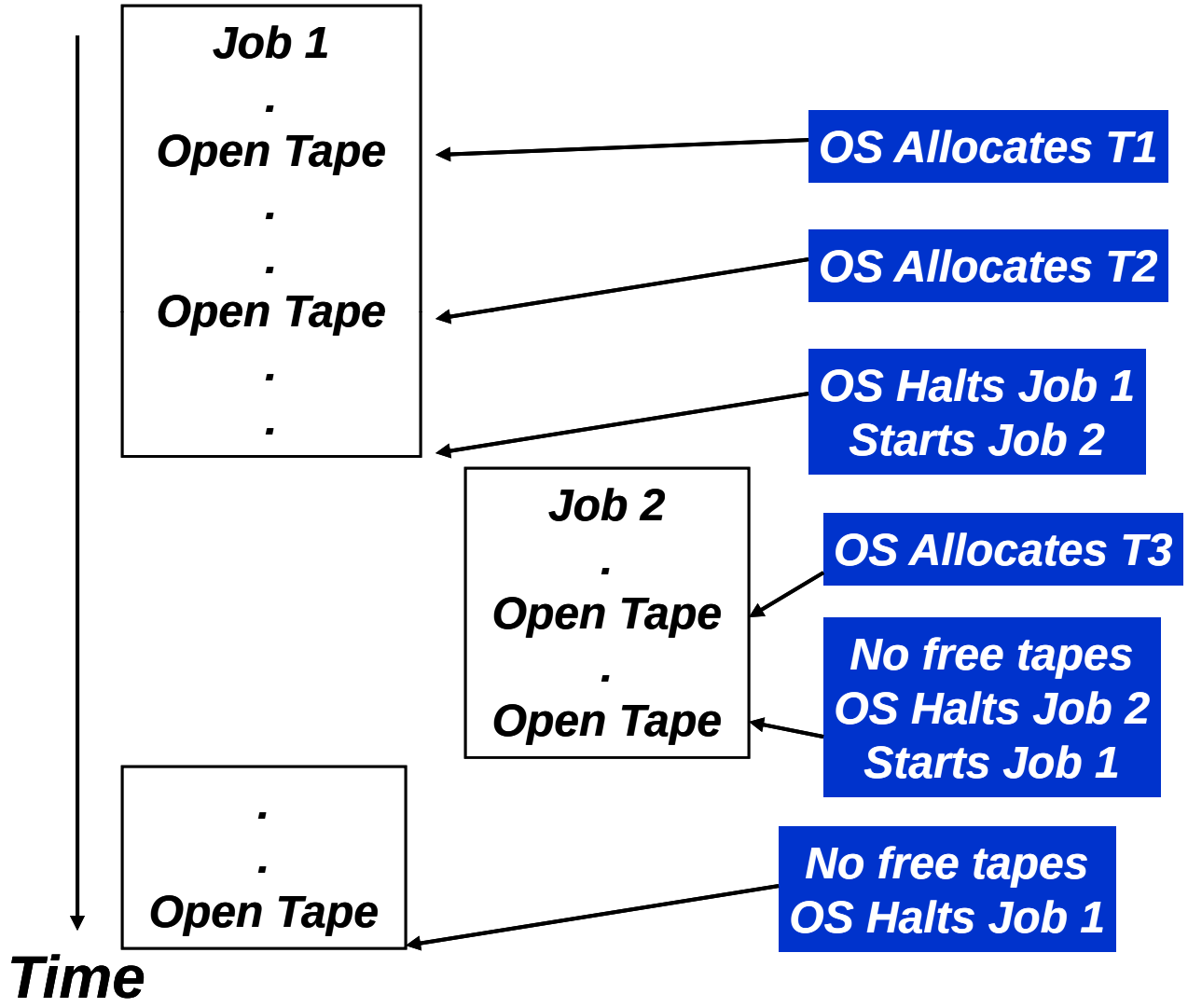
© 2010 LOCKHEED MARTIN CORPORATION

Deadlock

Problem:
Given:
2 Jobs
3 tape drives

Job 1: Merge data from 2 tapes onto a third.

Job 2: Copy data from 1 tape to another.



© 2010 LOCKHEED MARTIN CORPORATION

Prevention and Countermeasures

- **Make sure all assumptions hold for however long they need to hold reducing the window of vulnerability to zero**
  - *Strategy is to make relevant code atomic with respect to relevant data*
- **Ensure the program has exclusive rights (*lock: aka MUTEX*) to something while it's manipulating it**
  - ***File, Device, Object or Variable***
  - ***Be careful when using locks, a common problem is* deadlock**
    - Programs are stuck in a state waiting for each other to release a locked resource
    - Prevent it by requiring all threads to obtain locks in the same order (such as privilege rankings, alphabetically, etc.)
  - ***Another common problem is* livelock**
    - Programs manage to gain and release a lock, but in a way that they can't ever progress

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - *Buffer Overflows*
  - *Integer Overflows*
  - *Format String*
  - *Injection Flaws*
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

Malicious Code

- **Code inserted with the intent to subvert the security of the application**

- **Can be hidden in:**
  - ***Debug Software***
  - ***Dynamic Code***
    - Compilation
    - Construction
    - Execution
    - Loading
  - ***Time/Logic Bomb***
  - ***Trojan***

© 2010 LOCKHEED MARTIN CORPORATION

Prevention and Countermeasures

- **Train QC members on how to find malicious code during inspections and walkthroughs**
- **Train trusted developers in secure coding practices (i.e. no backdoors or weak structures)**
- **Use a security-focused static code analysis tool that searches for malicious code**
- **Security Policy and Procedures to insure single individual on development team cannot insert malicious code. (insider threat)**

© 2010 LOCKHEED MARTIN CORPORATION

Most Common Vulnerabilities

- **Unvalidated Input**
  - *Buffer Overflows*
  - *Integer Overflows*
  - *Format String*
  - *Injection Flaws*
    - Command
    - SQL
    - LDAP
- **Data Protection/Storage Failure**
- **Improper Error Handling**
- **Information Leakage**
- **Race Conditions**
- **Malicious Code**
- **Poor Usability**

© 2010 LOCKHEED MARTIN CORPORATION

Poor Usability$_1$

- **End-users and admins have different needs.**
  - ***Most non-technical users will not read a lot of security information.***
    - Accept/I Agree Syndrome
    - Security is (almost) never an end user's priority.
- **Designers are not users.**

© 2010 LOCKHEED MARTIN CORPORATION

Poor Usability$_2$

- **Too little appropriate information**
  - *Admin needs enough information to make a good security decision*
- **Too much information**
  - *User does not need too much security information; it is confusing*
- **Too many messages**
  - *Users will not read them*
- **Inaccurate or Generic information/ Errors with only Error Codes**
  - *Does not tell the user anything*

© 2010 LOCKHEED MARTIN CORPORATION

Prevention and Countermeasures

- **Make the User Interface Simple and Clear**
- **Make Security Decisions for Users**
- **Default to a Secure Configuration whenever possible**
- **Provide a Simple and Easy to Understand Message, and allow for progressive disclosure if needed by more sophisticated users or admins.**
- **Do not dump geek-speak in a dialog box. No user will read it.**

© 2010 LOCKHEED MARTIN CORPORATION

Questions?

© 2010 LOCKHEED MARTIN CORPORATION

References

[Lavenhar, 2005] Lavenhar, Steven R. *Source Code Analysis Tools – Business Case*. Cigital, 2005

[Viega, 2002] Viega, and Gary McGraw. *Building Secure Software*. 3rd ed. Boston, MA: Addison-Wesley, 2002.

BuildSecurityIn.net Coding Practices

(https://buildsecurityin.us-cert.gov/portal/article/knowledge/Coding_Practices)

BuildSecurityIn.net Coding Rules

(https://buildsecurityin.us-cert.gov/portal/article/knowledge/Coding_Rules)

BuildSecurityIn.net Source Code Analysis Tools

(https://buildsecurityin.us-cert.gov/portal/article/tools/code_analysis)

CERT – Secure Coding

(https://www.cert.org/secure-coding)

BuildSecurityIn.net The Common Criteria

https://buildsecurityin.us-cert.gov/daisy/bsi/articles/best-practices/requirements/239.html#dsy239_refs

© 2010 LOCKHEED MARTIN CORPORATION

Contact Information

- **Dr. Ben Calloni, P.E., CISSP, OCRES-AP**
  - ***Lockheed Martin Fellow, Software Security***
  - ***817/935-4482***
  - ***ben.a.calloni@lmco.com***

  - ***SMU Adjunct Professor, System Security Engineering***
  - ***Object Management Group Board of Directors***
    - OMG System Assurance Task Force Co-Chair
  - ***Former Member The Open Group Board of Directors (Vice-Chair)***
    - President of the Customer Council
  - ***Member of DHS-OSD Software Assurance Forum / WG's***
  - ***Network Centric Operations Industrial Consortium***
    - Information Assurance Working Group

© 2010 LOCKHEED MARTIN CORPORATION