



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **Definitions and concepts of security - Part 1**

## **RL 1.1.1**

## About Subject

---

“Secure software engineering focuses on creating software that functions correctly and survive any Cyberattack. The main topics of this course include requirements engineering for secure software, secure software architecture and design, considerations for secure coding and testing, common software vulnerabilities, risk analysis, misuse cases, secure programming techniques, analysis of software based attacks (and defenses), code reviews, and security testing.”

# Modules

No	Title of the Module
M1	Overview of security
M2	Security in SDLC
M3	Threat Modelling
M4	Security Requirements Engineering
M5	Secure Architecture & Design
M6	Testing for security
M7	Vulnerabilities in code
M8	Database security
M9	Web Application Security
M10	Security Mechanisms
M11	Managing for security

# Course Objective / Learning Outcomes

CO1	Understand software engineering principles for designing secure systems
CO2	Learn lifecycle models for software security.
CO3	Understand software attacks and techniques of building software that can withstand attacks

No	Learning Outcomes
LO1	Understand causes of security issues in software systems
LO2	Learn practices that enhance security in software development lifecycle.
LO3	Understand techniques of addressing security issues in software

# Course Handout

No	Name	Type	Duration	Weight	Day, Date, Session, Time
EC-1	Quiz-1		*	5%	September 1-10, 2023
	Quiz-2		*	5%	October 1-10, 2023
	Assignment		*	10%	November 1-10, 2023
EC-2	Mid-Semester Test	Open Book	2 hours	30%	Saturday, 23/09/2023 (Evening)
EC-3	Comprehensive Exam	Open Book	2 ½ hours	50%	Saturday, 25/11/2023 (Evening)

# Schedule

Date	Saturday Lecture <b>(2 Hour 16 Lecture)</b>	Event / Remark
22 07 2023	CS-1	Introduction Class
29 07 2023	CS-2	Assignment -1 ( 10 marks)
05 08 2023 / 12 08 2023	CS-3 and CS-4	
19 08 2023	CS-5	Quiz – 1 ( 5 marks)
26 08 2023/ 02 09 2023/09 09 2023/16 09  2023	CS-6 / 7/8/9	
22,23,24 SEP 2023	30 marks	<b>Mid-Semester Test Regular ( 30 Marks)</b>
30 09 2023	CS-10	
06,07,08 OCT 2023		<b>Mid-Semester Test Make-up</b>
14 10 2023/21 10 2023	CS-11/12	
28 10 2023	CS-13	Quiz – 2 ( 5 marks)
04 11 2023/11 11 2023/18 11 2023	CS-14/15/16	
24,25,26 NOV 2023		<b>Comprehensive Examination Regular ( 50 marks)</b>
01,02,03 DEC 2023		<b>Comprehensive Exam Make-up</b>



# Any Query ?

# Security Problem

---

Organizations store, process, transmit their most sensitive information using software-intensive systems.

Private citizens depend on software to shop, bank, invest, and carry out most personal and social activities

Global connectivity makes the sensitive information and software systems vulnerable to unintentional and unauthorized use.

# Security Problem

---

As per some experts, we are in era of

- Information warfare
- Cyber terrorism
- Computer crime

Terrorists, Organized criminals, other criminals are targeting software-intensive systems and are able to gain entry.

- There are many systems which can not resist attacks

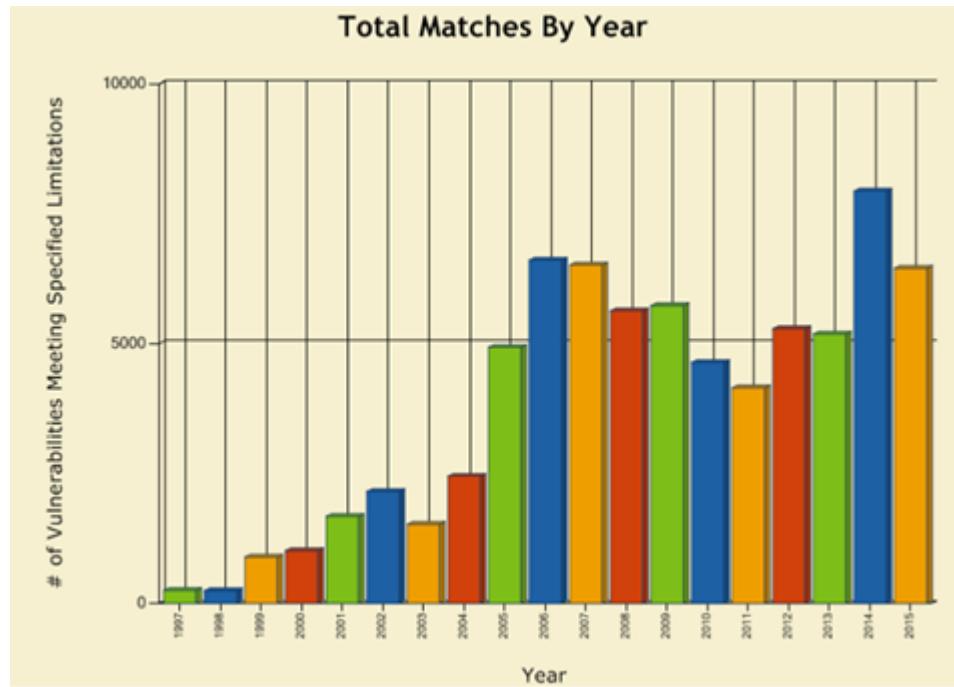
# Cyber Security Report to US President

---

Software development is not yet a science or a rigorous discipline, and the development process by and large is not controlled to minimize the vulnerabilities that attackers exploit. Today, as with cancer, vulnerable software can be invaded and modified to cause damage to previously healthy software, and infected software can replicate itself and be carried across networks to cause damage in other systems. Like cancer, these damaging processes may be invisible to the lay person even though experts recognize that their threat is growing.

- President's Information Technology Advisory Committee[2005]

# Trends of reported vulnerabilities



NVD is the U.S. government repository of standards based vulnerability management data. This data enables automation of vulnerability management, security measurement, and compliance

Source : National Vulnerability Database of NIST (National Institute of Standards and Technology)

URL: [https://web.nvd.nist.gov/view/vuln/statistics-results?adv\\_search=true&cvss=on&pub\\_date\\_start\\_month=0&pub\\_date\\_start\\_year=1997&pub\\_date\\_end\\_month=11&pub\\_date\\_end\\_year=2015&cvss\\_version=3](https://web.nvd.nist.gov/view/vuln/statistics-results?adv_search=true&cvss=on&pub_date_start_month=0&pub_date_start_year=1997&pub_date_end_month=11&pub_date_end_year=2015&cvss_version=3)

# Growth of Threats

---

Growing internet connectivity of computers and networks and dependence on network-enabled services(e.g. email, online transactions) means

- Increased number and sophistication of attack methods

Growing trend in which system accepts updates and extensions

- Each new extension adds new capabilities, new interfaces, and thus new risks

Unbridled growth in the size and complexity of software systems

- More lines of code produce more bugs and vulnerabilities

# Security Principles

---

Saltzer and Schroeder defined security as “techniques that control who may use or modify the computer or the information contained in it”

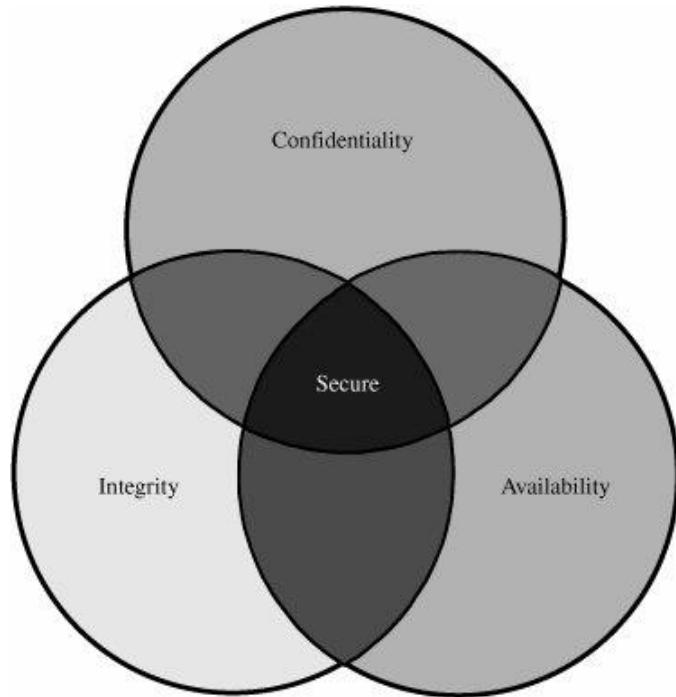
Described the three main categories of concern:

- Confidentiality
- Integrity
- Availability

Saltzer and Schroeder, “The Protection of Information in Computer Systems.” *Communications of the ACM*, 1974

# Security Principles

---



## **Integrity**

Information be protected from improper modification

## **Availability**

Available to user or program with legitimate right

## **Confidentiality**

Protection of data from unauthorized disclosure

## **Relationship Between Confidentiality, Integrity, and Availability**

# Security Principles

---

Two additional properties commonly associated with human users are required in software entities that act as users, e.g. proxy agents, web services etc.

- Accountability : All security-relevant actions of the software-as-user must be recorded and tracked with attribution, both while and after the recorded action occurs
- Non-repudiation : Ability to prevent the software-as-user from disproving or denying responsibilities for actions it has performed



# **Definitions and concepts of security - Part 2**

## **RL 1.1.2**

# Software Assurance

---

The Committee on National Security Systems (CNSS) defines software assurance as follows:

“Software assurance (SwA) is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner.”

Software assurance includes software reliability, software safety, and software security

- Software assurance becomes important since critical infrastructure (viz. power, communication etc.) depend on software-intensive systems

# Processes for Secure Software

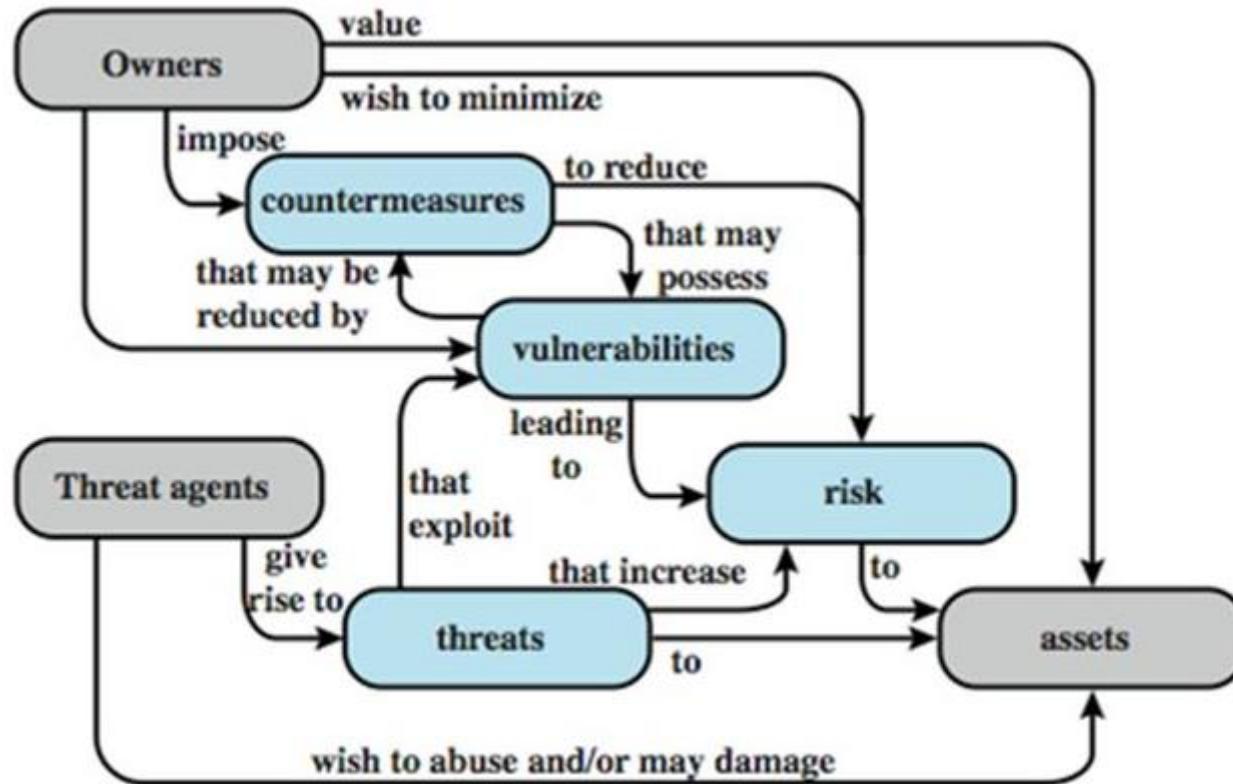
The most critical difference between secure and insecure software lies in the nature of the processes and practices used to specify, design, and develop the software

- Gortzel[2006]

Software vulnerabilities can originate from

- Decisions made by software engineers
- Flaws introduced in specification & design
- Faults from developed code
- Choice of programming language, development tools, operational environment etc.

# Security Concepts and Relationships



# Security Concepts and Relationships

---

## Threat

- Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, individuals, other organizations, or the Nation through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service. [CNSS 2010] 2. Any event that will cause an undesirable impact or loss to an organization if it occurs.

## Vulnerability

- Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited by a threat source. [CNSS 2010] 2. The absence or weakness of a safeguard. It can also be described as a weakness in an asset or the methods of ensuring that the asset is survivable.

## Risk

- A measure of the extent to which an entity is threatened by a potential circumstance or event, and typically a function of 1) the adverse impacts that would arise if the circumstance or event occurs; and 2) the likelihood of occurrence.

## Countermeasure

- Actions, devices, procedures, or techniques that meet or oppose(i.e., counters) a threat, a vulnerability, or an attack by eliminating or preventing it, by minimizing the harm it can cause, or by discovering and reporting it so that corrective action can be taken. NIST SP 800-53: Actions, devices, procedures, techniques, or other measures that reduce the vulnerability of an information system. Synonymous with security controls and safeguards. [CNSS 2010]

# Security Concepts and Relationships

The **attack surface** of a software environment is the sum of the different points (the "attack vectors") where an unauthorized user (the "attacker") can try to enter data to or extract data from an environment.

An **attack vector** is a path or means by which a hacker (or cracker) can gain access to a computer or network server in order to deliver a payload or malicious outcome.

If an attack vector is thought of as a guided missile (e.g. email), its payload can be compared to the warhead (e.g. malicious attachment) in the tip of the missile.

# Security Concepts and Relationships

The principle of ***defense-in-depth*** is that layered security mechanisms increase security of the system as a whole. If an attack causes one security mechanism to fail, other mechanisms may still provide the necessary security to protect the system.

***Social engineering*** attack is based on deceiving end users or administrators at a target site. Such attacks are typically carried out by email or by contacting users by phone and impersonating an authorized user, in an attempt to gain unauthorized access to a system or application

# Security Concepts and Relationships

In computer security, a **sandbox** is a security mechanism for separating running programs. It is often used to execute untested code, or untrusted programs from unverified third parties, suppliers, untrusted users and untrusted websites. A **sandbox** typically provides a tightly controlled set of resources for guest programs to run in, such as disk and memory, network access, the ability to inspect the host system or read from input devices (disallowed or heavily restricted).

**Sandboxes** may be seen as a specific example of virtualization. **Sandboxing** is frequently used to test unverified programs that may contain a virus or other malicious code, without allowing the software to harm the host device



# **Threats to Software/Assets – Part 1**

## **RL 1.2.1**

# The Asymmetric Problem of Security

No matter how much effort we spend, we will never get code 100 percent correct

This is an asymmetric problem

- **We** must be 100 percent correct, 100 percent of time, on a schedule, with limited resources, only knowing what we know today
  - And the product has to be reliable, supportable, compatible, manageable, affordable, accessible, usable, global, doable, deployable.
- **They** can spend as long as they like to find one bug, with the benefit of future research

– [www.microsoft.com](http://www.microsoft.com)

# The Asymmetric Problem of Security

---

Tools make it easy to build exploit code

Reverse engineering tools

- Situation explained
  - Structural Comparison of Executable Objects by Halvar Flake (available at [https://static.googleusercontent.com/media/www.zynamics.com/en//downloads/dimva\\_pape\\_r2.pdf](https://static.googleusercontent.com/media/www.zynamics.com/en//downloads/dimva_pape_r2.pdf))
  - PCT Bug – “Detecting and understanding the vulnerability took less than 30 minutes”
  - H.323 ASN.1 Bug: “The total analysis took less than 3 hours time”
- Exploitation
  - Penetration testing tools e.g. [www.metasploit.com](http://www.metasploit.com)

# The Asymmetric Problem of Security

---

In general

- Cost for attacker to build attack is very low
- Cost to your users is very high
- Cost of reacting is higher than cost of defending

# Hacking a Politician email

---

Hacker chooses to hack politician's mail account and possibly impact elections(2008)

The approach was

- Find the mail id
- Use Forgot Password feature
- The mail provider asks standard personal questions
- The politician biography is well known

# Security on Cloud (Example)

Code Spaces kept up their security measures, ensured that their server security was tight, and relied on Amazon for the bulk of their infrastructure -- like thousands of other companies.

The attack that brought Code Spaces under was as simple as gaining access to its AWS control panel.

Code Spaces was built mostly on AWS, using storage and server instances to provide its services. Those server instances weren't hacked, nor was Code Spaces' database compromised or stolen.

Attacker gained access to the company's AWS control panel & deleted resources on the cloud.

**The demise of Code Spaces at the hands of an attacker shows that, in the cloud, off-site backups and separation of services could be key to survival**

In the space of one hour, my entire digital life was destroyed. First my Google account was taken over, then deleted. Next my Twitter account was compromised, and used as a platform to broadcast racist and homophobic messages. And worst of all, my AppleID account was broken into, and my hackers used it to remotely erase all of the data on my iPhone, iPad, and MacBook.

My accounts were daisy-chained together. Getting into Amazon let my hackers get into my Apple ID account, which helped them get into Gmail, which gave them access to Twitter.

– Mat Honan, Sr. Staff Writer, wired.com

<http://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/all/>

# Panama Papers

---

- As per Forbes, data security experts noted that the company had not been encrypting its emails and furthermore seemed to have been running a three-year-old version of Drupal with several known vulnerabilities. Some reports also suggest that some parts of the site may have been running WordPress with an out of date version of Revolution Slider, a plugin that has suffered from vulnerabilities in the past.
- Drupal has over 100 reported vulnerabilities in CVE database.
  - e.g. CVE-2016-3171 : Drupal 6.x before 6.38, when used with PHP before 5.4.45, 5.5.x before 5.5.29, or 5.6.x before 5.6.13, might allow remote attackers to execute arbitrary code via vectors related to session data truncation.



# **Threats to Software/Assets – Part 2**

## **RL 1.2.2**

# Attack surfaces

---

Attack surface: the reachable and exploitable vulnerabilities in a system

- Open ports
- Services outside a firewall
- An employee with access to sensitive info
- ...

Three categories

- **Network attack surface** (i.e., network vulnerability)
- **Software attack surface** (i.e., software vulnerabilities)
- **Human attack surface** (e.g., social engineering)

Attack analysis: assessing the scale and severity of threats

# Automotive Attack Surface

---

Modern cars are controlled by complex distributed computer systems comprising millions of lines of code executing on tens of heterogeneous processors with rich connectivity provided by internal networks (e.g., Controller Area Network CAN).

This structure has offers significant benefits to efficiency, safety and cost, but also creates the opportunity for new attacks.

An attacker connected to a car's *internal network* can circumvent *all* computer control systems, including safety critical elements such as the brakes and engine.

The long-range wireless attack surface is that exposed by the remote telematics systems (e.g., Ford's Sync, GM's OnStar, Toyota's SafetyConnect, Lexus' Enform, BMW's BMW Assist, and Mercedes-Benz' mbrace) that provide continuous connectivity via cellular voice and data networks for supporting safety (crash reporting), diagnostics (early alert of mechanical issues), anti-theft (remote track and disable), and convenience (hands-free data access such as driving directions or weather).

Comprehensive Experimental Analyses of Automotive Attack Surfaces Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson,  
Hovav Shacham, and Stefan Savage University of California, San Diego  
Karl Koscher, Alexei Czeskis, Franziska Roesner, and Tadayoshi Kohno University of Washington USENIX Security, August 10–12, 2011

# Examples of threats

	Availability	Confidentiality	Integrity
Hardware	Equipment is stolen or disabled, thus denying service.	An unencrypted CD-ROM or DVD is stolen.	
Software	Programs are deleted, denying access to users.	An unauthorized copy of software is made.	A working program is modified, either to cause it to fail during execution or to cause it to do some unintended task.
Data	Files are deleted, denying access to users.	An unauthorized read of data is performed. An analysis of statistical data reveals underlying data.	Existing files are modified or new files are fabricated.
Communication Lines and Networks	Messages are destroyed or deleted. Communication lines or networks are rendered unavailable.	Messages are read. The traffic pattern of messages is observed.	Messages are modified, delayed, reordered, or duplicated. False messages are fabricated.

# Data in Transit

---

Attacks on data in networks can be

Passive attacks : eavesdropping on, or monitoring of transmissions

- Release of message contents
- Traffic analysis : Encrypted message can not be read. Location, identity of host, frequency, and length of messages can help opponents make guess

Active attacks

- Replay : passive capture of data & subsequent retransmission to produce an unauthorized effect
- Masquerade : one entity pretends to be another entity.
- Modification of messages : some portion of a legitimate message is altered.
- Denial of service : inhibit normal use of facilities

# Attack trees

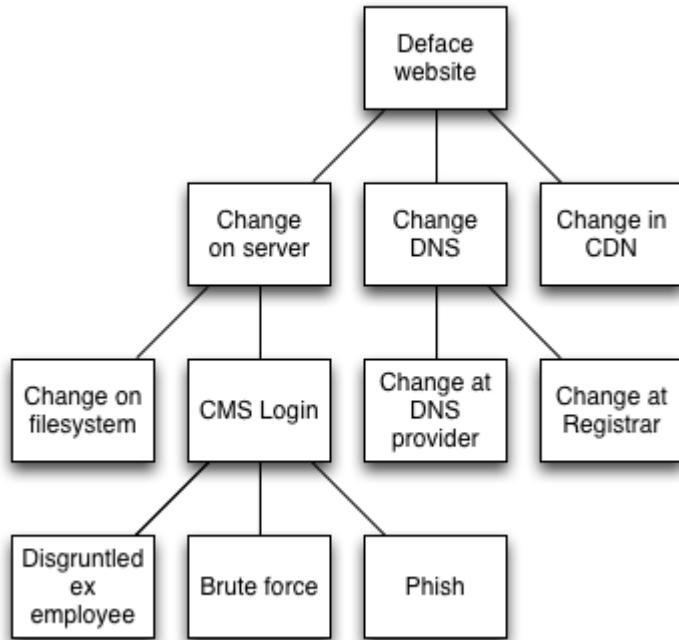
---

A branching, hierarchical data structure that represents a set of potential vulnerabilities

Objective: to effectively exploit the info available on attack patterns

- published on CERT or similar forums
- Security analysts can use the tree to guide design and strengthen countermeasures

# An attack tree (to deface a web site)



Content delivery network (CDN) :  
System of distributed servers  
(network) that deliver webpages  
and other Web content to user  
based on the geographic locations

Domain Name System (DNS) :  
Hierarchical decentralized naming  
system for computers, services, or  
any resource connected to the  
Internet or a private network

CMS : Content Management  
System



# Malware Nomenclature – Part 1

## RL 1.3.1

# Malware

---

“A program that is inserted into a system, usually covertly, with the intent of compromising the confidentiality, integrity, or availability of the victim’s data, applications, or operating system or otherwise annoying or disrupting the victim.”

# Malicious software

---

Programs exploiting system vulnerabilities

Known as malicious software or malware

- program fragments that need a host program
  - e.g. viruses, logic bombs, and backdoors
- independent self-contained programs
  - e.g. worms, bots
- replicating or not

Sophisticated threat to computer systems

# Malware Terminology

---

Virus: *attaches itself to a program*

Worm: *propagates copies of itself to other computers*

Logic bomb: *“explodes” when a condition occurs*

Trojan horse: *fakes/contains additional functionality*

Backdoor (trapdoor): *allows unauthorized access to functionality*

Mobile code: *moves unchanged to heterogeneous platforms*

Auto-rooter Kit (virus generator): *malicious code (virus) generators*

Spammer and flooder programs: *large volume of unwanted “pkts”*

Keyloggers: *capture keystrokes*

Rootkit: *sophisticated hacker tools to gain root-level access*

Zombie: *software on infected computers that launch attack on others (aka bot)*

# More terms

---

Payload: actions of the malware

Crimeware: kits for building malware; include propagation and payload mechanisms

- Zeus, Sakura, Blackhole, Phoenix

APT (advanced persistent threats)

- Advanced: sophisticated
- Persistent: attack over an extended period of time
- Threat: selected targets (capable, well-funded attackers)

# Viruses

---

Piece of software that infects programs

- modifying them to include a copy of the virus
- so it executes secretly when host program is run

Specific to operating system and hardware

- taking advantage of their details and weaknesses

A typical virus goes through phases of:

- dormant: *idle*
- propagation: *copies itself to other program*
- triggering: *activated to perform functions*
- execution: *the function is performed*

Biological Virus : Tiny scraps of genetic code – DNA or RNA – that can take over a living cell and trick it into making replicas of the original virus

# Virus structure

---

## Components:

- infection mechanism: enables replication
- trigger: event that makes payload activate
- payload: what it does, malicious or benign

Prepended/postpended/embedded

When infected program invoked, executes virus code then original program code

Can block initial infection (difficult) or propagation (with access controls)

# Virus classification

---

## By target

- boot sector: *infect a master boot record*
- file infector: *infects executable OS files*
- macro virus: *infects files to be used by an app*
- multipartite: infects multiple ways

## By concealment

- encrypted virus: *encrypted; key stored in virus*
- stealth virus: *hides itself (e.g., compression)*
- polymorphic virus: *recreates with diff “signature”*
- metamorphic virus: *recreates with diff signature and behavior*

# Virus Variants

---

## Macro and scripting viruses

- Became very common in mid-1990s since
  - platform independent
  - infect documents
  - easily spread
- Exploit macro capability of Office apps
  - executable program embedded in office doc
  - often a form of Basic
- More recent releases include protection
- Recognized by many anti-virus programs

## E-Mail Viruses

- More recent development
- Example : Melissa
  - exploits MS Word macro in attached doc
  - if attachment opened, macro activates
  - sends email to all on users address list and does local damage



# **Malware Nomenclature – Part 2**

## **RL 1.3.2**

# Worms

---

Replicating program that propagates over net

- using email, remote exec, remote login

Has phases like a virus:

- dormant, propagation, triggering, execution
- propagation phase: searches for other systems, connects to it, copies self to it and runs

May disguise itself as a system process

Concept seen in Brunner's "Shockwave Rider"

Implemented by Xerox Palo Alto labs in 1980's

# Morris worm

---

One of best known worms

Released by Robert Morris in 1988

- Affected 6,000 computers; cost \$10-\$100 M

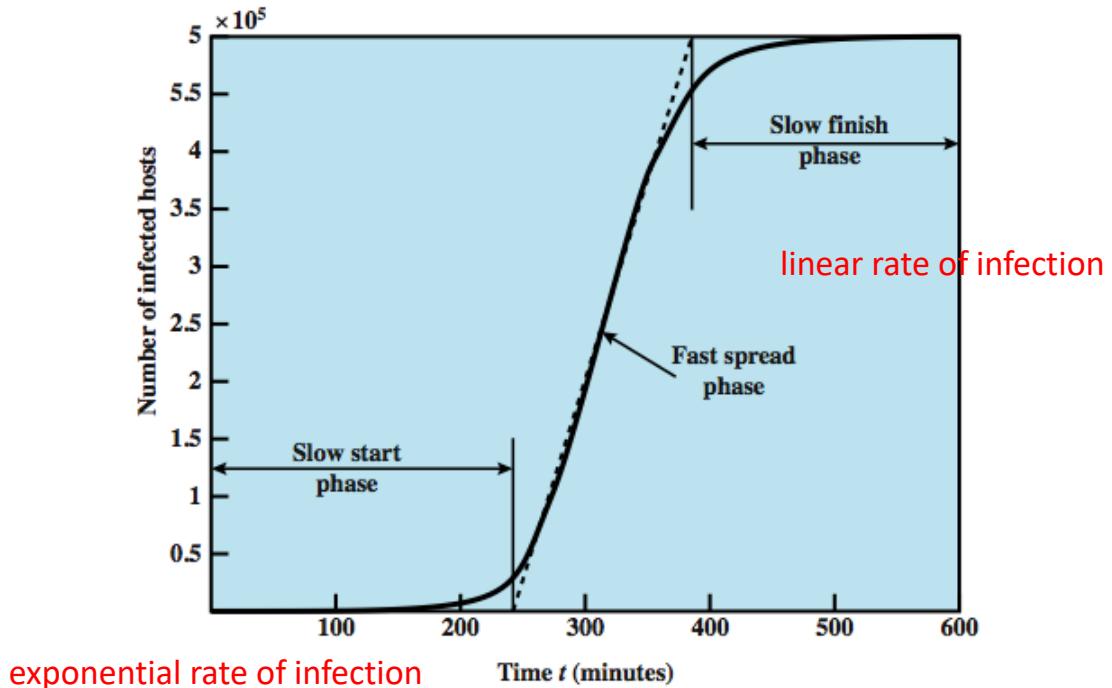
Various attacks on UNIX systems

- cracking password file to use login/password to logon to other systems
- exploiting a bug in the finger protocol
- exploiting a bug in sendmail

If succeed have remote shell access

- sent bootstrap program to copy worm over

# Worm Propagation Model (based on recent attacks)



# Some worm attacks

---

## Code Red

- July 2001 exploiting MS IIS bug
- probes random IP address, does DDoS attack
- consumes significant net capacity when active
- **360,000 servers in 14 hours**

Code Red II variant includes backdoor: hacker controls the worm

## SQL Slammer (*exploited buffer-overflow vulnerability*)

- early 2003, attacks MS SQL Server
- compact and very rapid spread

## Mydoom (*100 M infected messages in 36 hours*)

- mass-mailing e-mail worm that appeared in 2004
- installed remote access backdoor in infected systems

# State of worm technology

---

Multiplatform: not limited to Windows

Multi-exploit: Web servers, emails, file sharing ...

Ultrafast spreading: do a scan to find vulnerable hosts

Polymorphic: each copy has a new code

Metamorphic: change appearance/behavior

Transport vehicles (e.g., for DDoS)

Zero-day exploit of unknown vulnerability (to achieve max surprise/distribution)

# Worm countermeasures

---

Overlaps with anti-virus techniques

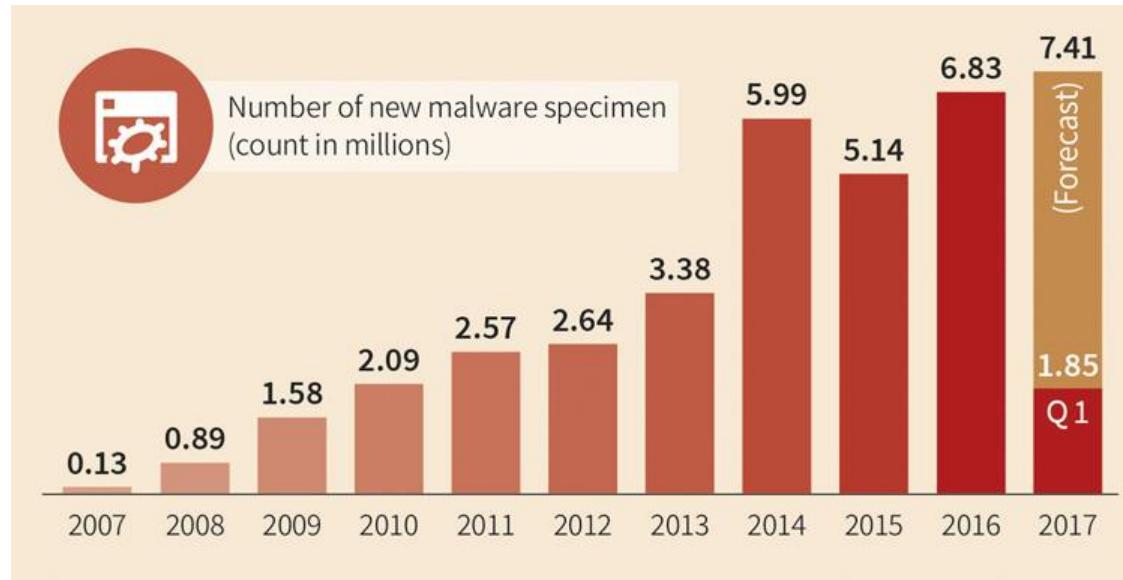
Once worm on system A/V can detect

Worms also cause significant net activity

Worm defense approaches include:

- signature-based worm scan filtering: define signatures
- filter-based worm containment (focus on contents)
- payload-classification-based worm containment (examine packets for anomalies)
- threshold random walk scan detection (limit the rate of scan-like traffic)
- rate limiting and rate halting (limit outgoing traffic when a threshold is met)

# Malware Trends



# Malware Trends

---

The vast majority of malware is categorized as Trojan Horse and comprises typical malicious activities like downloading and dropping files, spyware, keyloggers and password stealers, integration into botnets and conducting distributed denial of service attacks (DDoS).

Position two is held by adware.

A sharp rise could also be seen in ransomware(from a small base). Its number increased more than ninefold from the first half of 2016 to the second. Moreover, the number of the latter half of 2016 was almost achieved in the first quarter of 2017. Few ransomware families caused quite some stir.

# Malware Trends

---

The predominant platform for malware is still Windows. It covers 99.1% of the malware specimen. Trailing behind are scripts, Java applets, macros and other operating systems like OSX, Android, and Unix/Linux.

The number of new malware is still rising.

No big changes in terms of malicious activities of Trojan horses

The number of adware is increasing

The share of ransomware is growing substantially.

Software Security Engineering, Julia H. Allen, et al, Pearson, 2008.

Computer Security: Principles and Practice by William Stallings, and Lawrie Brown Pearson, 2008.

Security in Computing by Charles P. Pfleeger, Shari L. Pfleeger, and Deven Shah Pearson Education 2009

Threat Modelling by Adam Shostack, John Wiley 2014

[www.gdatasoftware.com](http://www.gdatasoftware.com)

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao

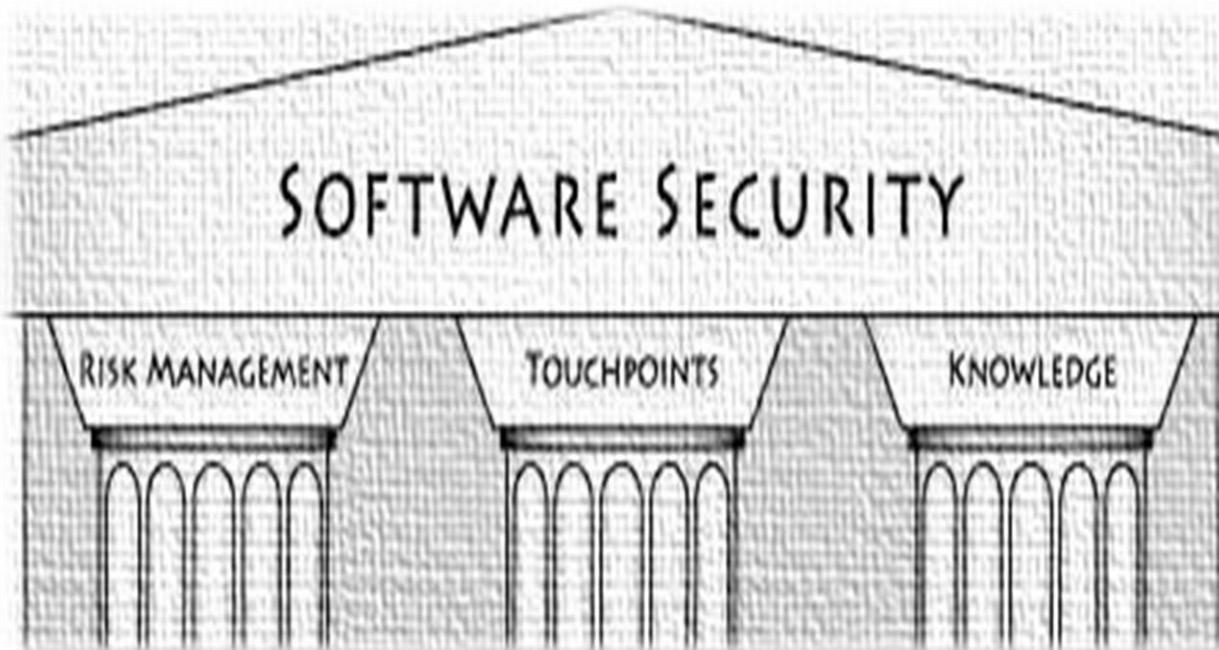


# **Phases in software development - Part 1**

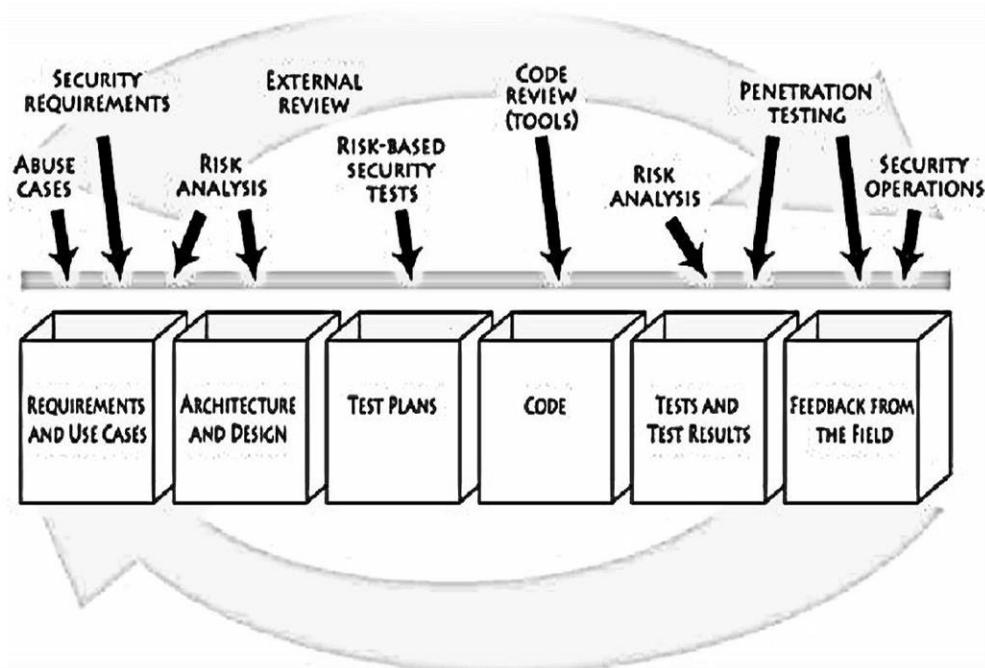
## **RL 2.1.1**

# Solving the Problem

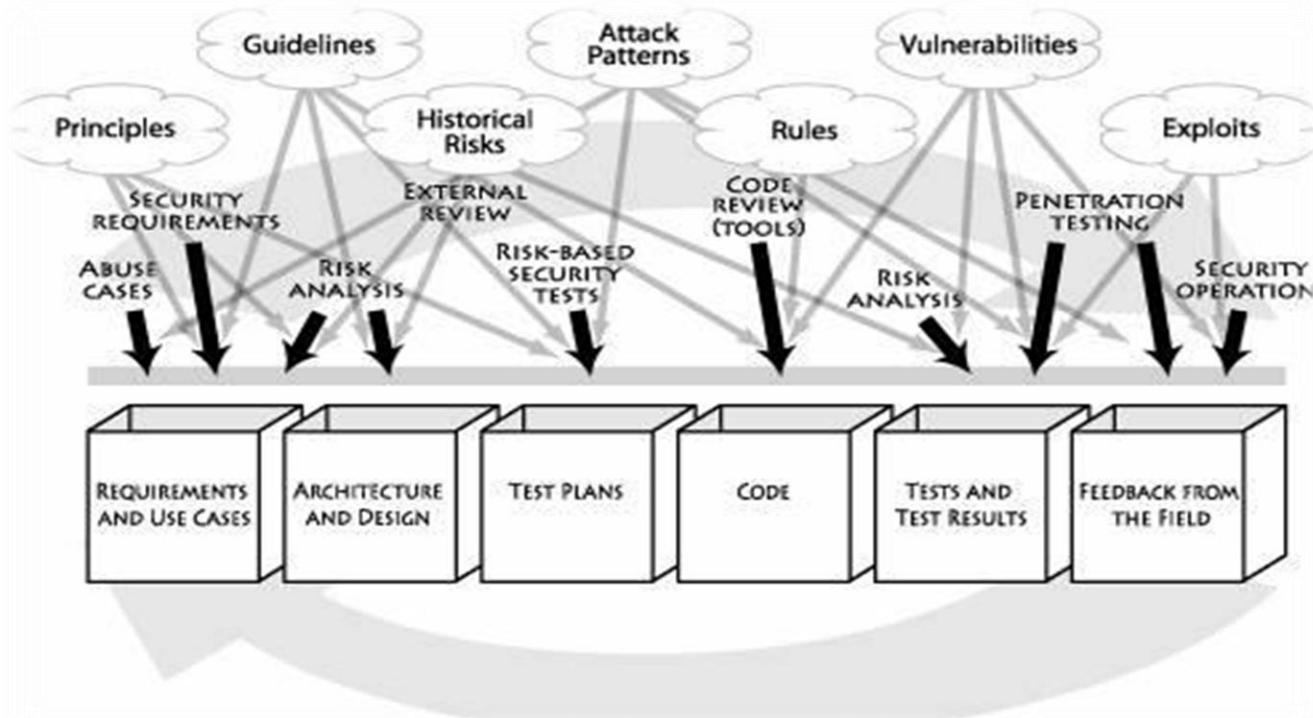
## The Three Pillars of Software Security



# Some Software security best practices



# Some Software security best practices



# The Rise of Security Engineering

- Connectivity and distributed computation is so pervasive that the only way to begin to secure our computing infrastructure is to enlist everyone.
  - Builders must practice security engineering, ensuring that the systems we build are defensible and not riddled with holes (especially when it comes to the software).
  - Operations people must continue to architect reasonable networks, defend them, and keep them up.
  - Administrators must understand the distributed nature of modern systems and begin to practice the principle of least privilege.
  - Users must understand that software *can* be secure so that they can take their business to software providers who share their values. (Witness the rise of Firefox.) Users must also

# Software Engineering - Definitions

---

- (1969 – Fritz Bauer) Software engineering is the establishment and use of *sound engineering principles* in order to obtain *economically* software that is *reliable* and works *efficiently* on *real machines*
- (IEEE) The application of a *systematic, disciplined, quantifiable* approach to the *development, operation, and maintenance* of software; that is, the application of engineering to software

# **Knowledge Areas in v3(2014) of SWEBOk**

<b>Requirements</b>	<b>Configuration Management</b>
<b>Design</b>	<b>Quality</b>
<b>Construction</b>	<b>Processes</b>
<b>Testing</b>	<b>Models &amp; Methods</b>
<b>Maintenance</b>	<b>Engineering Management</b>
	<b>Project Management</b>
	<b>Economics</b>

# Software Engineering Process KA

A Process defines who is doing what, when, and how to reach a certain goal

-Ivar Jacobson, Grady Booch, and James Rumbaugh

A software process is a set of interrelated activities and tasks that transform input work products into output work products. At minimum, the description of a software process includes required inputs, transforming work activities, and outputs generated.

- SWEBOK

A software process infrastructure can provide process definitions, policies for interpreting and applying the processes, and descriptions of the procedures to be used to implement the processes.

A software development life cycle (SDLC) includes the software processes used to specify and transform software requirements into a deliverable software product.

# **Prescriptive and agile processes**

---

Prescriptive processes are processes where all of the process activities are planned in advance and progress is measured against this plan.

In agile processes, planning is incremental and it is easier to change the process to reflect changing customer requirements.

In practice, most practical processes may include elements of both plan-driven and agile approaches.

***There are NO right or wrong software processes.***

# How Process Models Differ?

---

While all Process Models take same primary and supporting activities, they differ with regard to

- Overall flow of activities, actions, and tasks and the interdependencies among them
- Degree to which actions and tasks are defined within each framework activity
- Degree to which work products are identified and required
- Manner in which quality assurance activities are applied
- Manner in which project tracking and control activities are applied
- Overall degree of detail and rigor with which the process is described
- Degree to which customer and other stakeholders are involved in the project
- Level of autonomy given to the software team
- Degree to which team organization and roles are prescribed

# Software Requirements

---

The Software Requirements knowledge area (KA) is concerned with the elicitation, analysis, specification, and validation of software requirements as well as the management of requirements during the whole life cycle.

Software projects are critically vulnerable when the requirements related activities are poorly performed

# Functional & Nonfunctional Requirements

*Functional* requirements describe the functions that the software is to execute; for example, formatting some text or modulating a signal. They are sometimes known as capabilities or features. A functional requirement can also be described as one for which a finite set of test steps can be written to validate its behavior.

*Nonfunctional* requirements are the ones that act to constrain the solution. Nonfunctional requirements are aka constraints or quality requirements. They can classified according to whether they are performance requirements, maintainability requirements, safety requirements, reliability requirements, security requirements, interoperability requirements

# Emergent properties of software

Some requirements represent *emergent properties* of software—that is, requirements that cannot be addressed by a single component but that depend on how all the software components interoperate. The throughput requirement for a call center would, for example, depend on how the telephone system, information system, and the operators all interacted under actual operating conditions. Emergent properties are crucially dependent on the system architecture.

Experts consider **Security** as an *emergent property* of the software.

# Software Design

---

Software design consists of two activities that fit between software requirements analysis and software construction:

- Software architectural design (sometimes called high-level design): develops top-level structure and organization of the software and identifies the various components.
- Software detailed design: specifies each component in sufficient detail to facilitate its construction.

# Software Design

---

- Software design principles include abstraction; coupling and cohesion; decomposition and modularization; encapsulation/information hiding; separation of interface and implementation; sufficiency, completeness, and primitiveness; and separation of concerns.
- Design for security is concerned with how to prevent unauthorized disclosure, creation, change, deletion, or denial of access to information and other resources. It is also concerned with how to tolerate security-related attacks or violations by limiting damage, continuing service, speeding repair and recovery, and failing and recovering securely.

# Software Construction

---

Software construction refers to the detailed creation of working software through a combination of coding, verification, unit testing, integration testing, and debugging.

Throughout construction, software engineers both unit test and integration test their work. Thus, the Software Construction KA is closely linked to the Software Testing KA.

Code is the ultimate deliverable of a software project, and thus the Software Quality KA is closely linked to the Software Construction KA.

# Software Testing

---

Software testing consists of the *dynamic* verification that a program provides *expected* behaviors on a *finite* set of test cases, suitably *selected* from the usually infinite execution domain

- *Dynamic*: The input value alone is not always sufficient to specify a test, since a system might react to the same input with different behaviors, depending on the system state.
- *Finite*: Even in simple programs, so many test cases are theoretically possible that exhaustive testing is infeasible.
- *Selected*: How to identify the most suitable test set under given conditions is a complex problem; in practice, risk analysis techniques and software engineering expertise are applied.
- *Expected*: It must be possible, although not always easy, to decide whether the observed outcomes of program testing are acceptable or not.



# **Phases in software development - Part 2**

## **RL 2.1.2**

# Software quality

---

Software quality may refer:

- to desirable characteristics of software products,
- to the extent to which a particular software product possess those characteristics, and
- to processes, tools, and techniques used to achieve those characteristics

Experts defined variously

“conformance to requirements” - Phil Crosby

“achieving excellent levels of fitness for use” - Watt Humphrey

“market-driven quality” where the “customer is the final arbiter” - IBM

# Software quality

---

A healthy software engineering culture includes the understanding that tradeoffs among cost, schedule, and quality are a basic tenant of the engineering of any product.

The tradeoff is best decided by understanding four cost of quality categories: prevention, appraisal, internal failure, and external failure.

Prevention costs include investments in software process improvement efforts, quality infrastructure, quality tools, training, audits, and management reviews

Appraisal costs arise from project activities that find defects.

Costs of internal failures are those that are incurred to fix defects found during appraisal activities and discovered prior to delivery of the software product to the customer

External failure costs include activities to respond to software problems discovered after delivery to the customer.

# Configuration management

---

Configuration management (CM) is the discipline of identifying the configuration of a system at distinct points in time for the purpose of systematically controlling changes to the configuration and maintaining the integrity and traceability of the configuration throughout the system life cycle.

A system can be defined as the combination of interacting elements organized to achieve one or more stated purposes

Configuration of a software system is collection of specific versions of hardware, firmware, or software items combined according to specific build procedures for a purpose.

# Software Maintenance

---

A software product must change or evolve over time. Once a software is in operation, defects are uncovered, operating environments change, and new user requirements surface.

Software maintenance is an integral part of a software life cycle. However, software development used to receive more importance than software maintenance in most organizations. It is changing

- Due to large capital spending needed for software development, organizations are trying to maximize lifespan of existing software
- Due to large scale availability of open source components, organizations increased focus on maintenance

# Maintainer's activities

---

Five key characteristics comprise the maintainer's activities:

- maintaining control over the software's day-to-day functions;
- maintaining control over software modification;
- perfecting existing functions;
- identifying security threats and fixing security vulnerabilities; and
- preventing software performance from degrading to unacceptable levels.

# Software Engineering Management

Software engineering management can be defined as the application of management activities — planning, coordinating, measuring, monitoring, controlling, and reporting — to achieve quality etc.

There are aspects specific to software projects and software life cycle that complicate effective management, including

- Clients often don't know what is needed or what is feasible
- As a result of changing requirements, software is often built using an iterative process
- The degree of novelty and complexity is often high
- There is often a rapid rate of change in the underlying technology

# Software Engineering Economics

Software engineering economics is about relating the attributes of software and software processes to economic measures

- involves balancing risk and profitability, while maximizing benefits and wealth of the organization.
- identify organizational goals, time horizons, risk factors, and financial constraints
- identify and implement the appropriate portfolio and investment decisions to manage cash flow, and funding;
- measure financial performance, such as cash flow and ROI

# Software Engineering Process

software process is a set of interrelated activities and tasks that transform input work products into output work products

a software process includes required inputs, transforming work activities, and outputs generated

a software process may also include its entry and exit criteria and decomposition of the work activities into tasks, which are the smallest units of work

Value individuals & interactions over processes & tools – Agile manifesto

Agile models are designed to facilitate evolution of the software requirements during the project

# Software Engineering Professional Practice

---

Software Engineering Professional Practice includes knowledge, skills, and attitudes that software engineers must possess to practice software engineering in a professional, responsible, and ethical manner

The concept of professional practice becomes applicable within the professions that have a generally accepted body of knowledge

A code of ethics and professional conduct for software engineering was approved by the ACM Council and the IEEE CS Board of Governors in 1999

–Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the eight principles concerning the public, client and employer, product, judgment, management, profession, colleagues, and self, respectively



# **Work products during SDLC**

## **RL 2.2**

# Requirement Models

---

**Scenario-based modeling** – represents the system from the user's point of view

**Flow-oriented modeling** – provides an indication of how data objects are transformed by a set of processing functions

**Class-based modeling** – defines objects, attributes, and relationships

**Behavioral modeling** – depicts the states of the classes and the impact of events on these states

# Use-Cases

---

A collection of user scenarios that describe the thread of usage of a system

Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way

Each scenario answers the following questions:

- Who is the primary actor, the secondary actor (s)?
- What are the actor’s goals?
- What preconditions should exist before the story begins?
- What main tasks or functions are performed by the actor?
- What extensions might be considered as the story is described?
- What variations in the actor’s interaction are possible?
- What system information will the actor acquire, produce, or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

# Use case—detailed example (Pressman)

---

Example: “SAFEHOME” system (Pressman)

**Use case name:** *InitiateMonitoring*

**Participating actors:** homeowner, technicians, sensors

**Flow of events (homeowner):**

- Homeowner wants to set the system when the homeowner leaves house or remains in house
- Homeowner observes control panel
- Homeowner enters password
- Homeowner selects “stay” or “away”
- Homeowner observes that red alarm light has come on, indicating the system is armed

# Use case—detailed example (Pressman)

---

## Pre condition(s)

Homeowner decides to set control panel

## Post condition(s)

- Control panel is not ready; homeowner must check all sensors and reset them if necessary
- Control panel indicates incorrect password (one beep)—homeowner enters correct password
- Password not recognized—must contact monitoring and response subsystem to reprogram password
- Stay selected: control panel beeps twice and lights stay light; perimeter sensors are activated
- Away selected: control panel beeps three times and lights away light; all sensors are activated

# Use case—detailed example (Pressman)

---

## Quality requirements:

Control panel may display additional text messages

time the homeowner has to enter the password from the time the first key is pressed

Ability to activate the system without the use of a password or with an abbreviated password

Ability to deactivate the system before it actually activates

# Misuse case (techtarget.com)

---

**Name:** Attack on "forgot password" functionality

**Summary:** A malicious user tries to attack the "forgot password" functionality in order to gain access to the Web application or guess a valid e-mail address

**Author:** Anurag Agarwal

**Date:** April 15, 2006

## Possible Attacks:

- SQL injection attack
- Brute force attack to guess a valid user
- Sniffing attack on e-mail sent with password on an insecure transmission channel

**Trigger Point:** Can happen anytime

**Preconditions:** None

## Assumptions:

- The attacker can perform this attack remotely over the Internet
- The attacker can be an anonymous user

# Misuse case (techtarget.com)

---

## Worst case threat (post condition) :

- Attacker gains entry into the company database and steals sensitive information
- Attacker is able to modify an existing e-mail address to its own e-mail address and mails the password to himself to gain unauthorized entry into the system

## Related business rule:

- The system should e-mail the password to a valid e-mail address entered

## Capture guarantee (post condition) :

- Attacker cannot gain access to the database to steal or modify information
- Attacker cannot identify the e-mail address of a valid user
- Attacker cannot view the password sent in an e-mail to an e-mail address of a valid user

# Misuse case (techtarget.com)

---

## Potential misuser profile:

- Script kiddie
- Skilled attacker

**Threat level:** High

## Mitigation steps:

- SQL injection attack
  - List all the mitigation steps to avoid a SQL injection attack.
- Brute force attack
  - Accept first name, last name along with e-mail address
  - Have proper error handling so as not to reveal information to the attacker
  - Delay 3 to 5 seconds before re-entering the e-mail address
  - Lock out page for that IP address after 10 attempts
- Sniffing attack
  - Send password e-mail on a secure transmission channel with strong encryption

## Data Flow Diagram

- Depicts how input is transformed into output as data objects move through a system

## Process Specification

- Describes data flow processing at the lowest level of refinement in the data flow diagrams

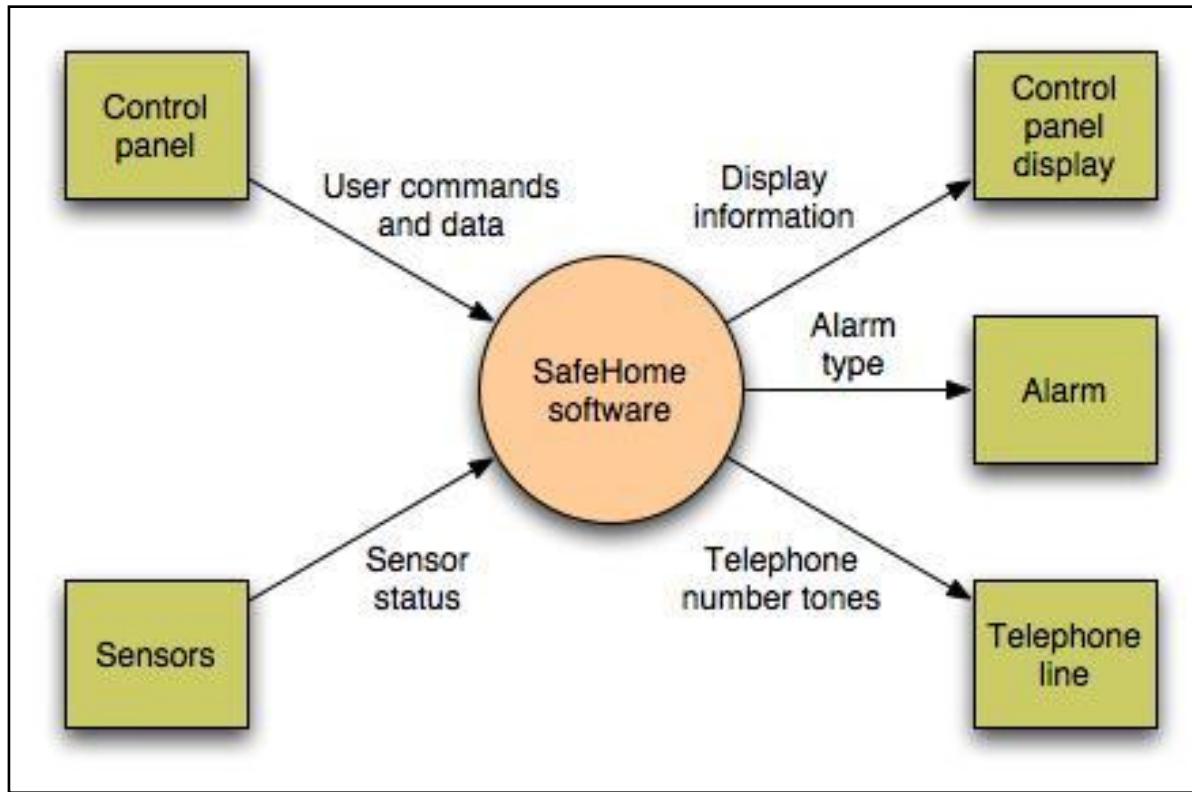
# Data Flow Modeling

---

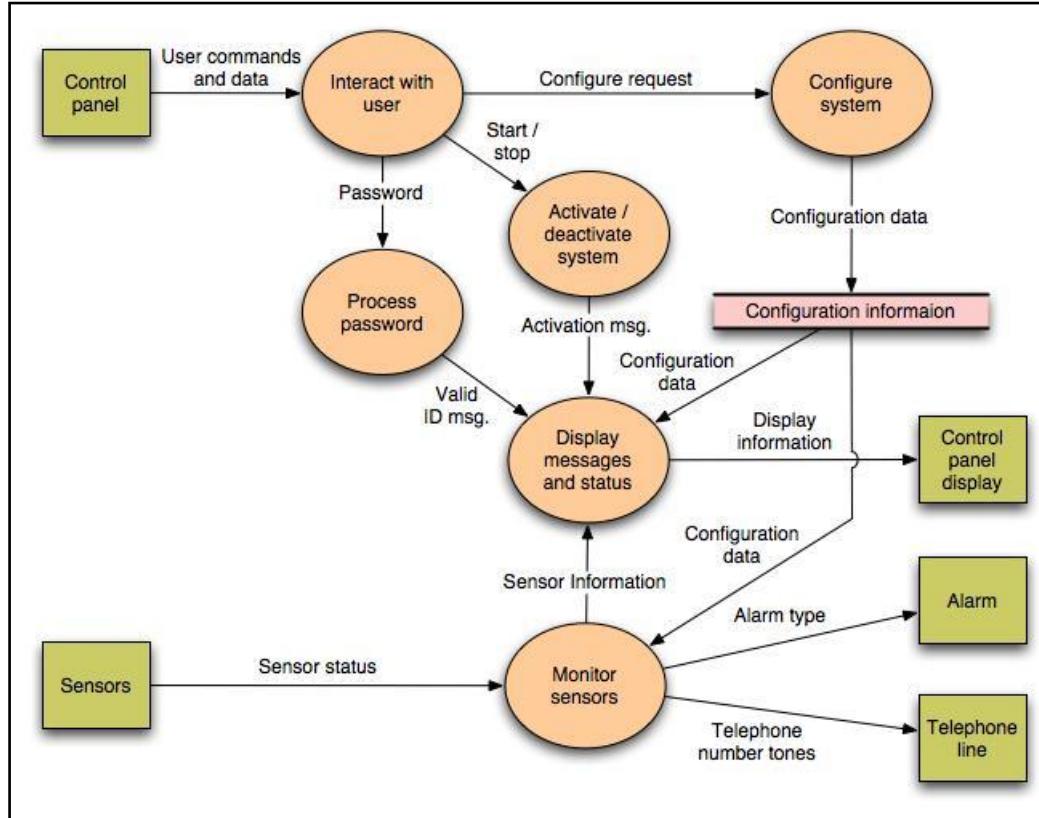
## Guidelines

- Depict the system as single bubble in level 0.
- Carefully note primary input and output.
- Refine by isolating candidate processes and their associated data objects and data stores.
- Label all elements with meaningful names.
- Maintain information conformity between levels.
- Refine one bubble at a time.

# Data Flow Diagram



# Data Flow Diagram (Next Level)





# **Security implications to SDLC – Part 1**

## **RL 2.3.1**

## OWASP ( Open Web Application Security Project) SAMM

SAMM (Software Assurance Maturity Model) is the OWASP framework to help organizations assess, formulate, and implement a strategy for software security, that can be integrated into their existing Software Development Lifecycle (SDLC)

SAMM is based around a set of 12 security practices, which are grouped into four business functions

Every security practice contains a set of activities, structured into three maturity levels (1-3).

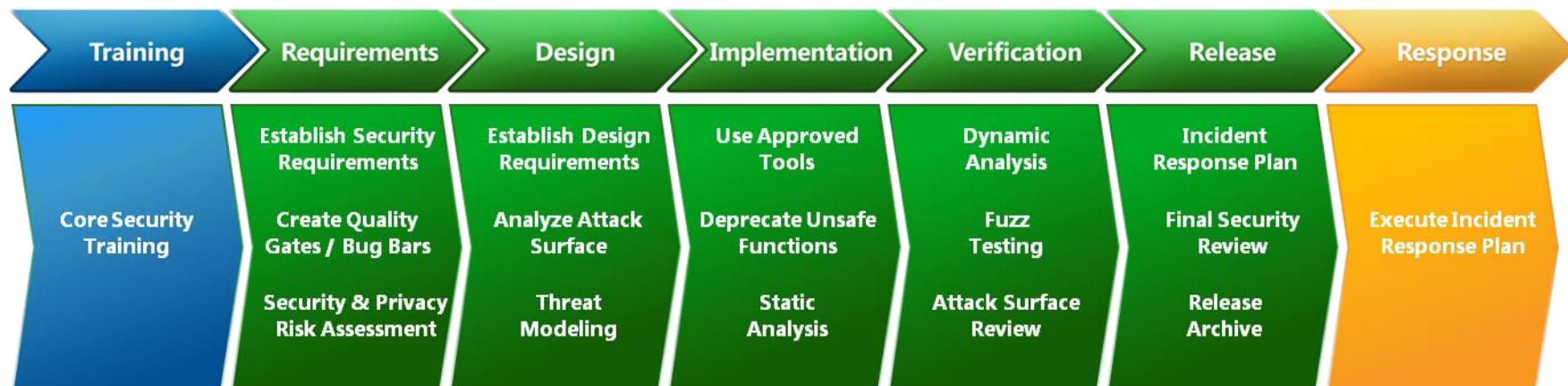
# Security Development Lifecycle

The Security Development Lifecycle (SDL) is a security assurance process developed by Microsoft as a company-wide initiative and a mandatory policy to reduce the number and severity of vulnerabilities in software products.

The Microsoft SDL is based on three core concepts —

- *education,*
- *continuous process improvement, and*
- *accountability*

# Security Development Lifecycle



# SDL - Core Security Training

---

Basic software security training should cover foundational concepts such as:

- Secure design, including the following topics: Attack surface reduction, Defense in depth, principle of least privilege, Secure defaults
- Threat modeling, including the following topics: Overview of threat modeling, Design implications of a threat model, Coding constraints based on a threat model
- Secure coding, including the following topics: Buffer overruns (for applications using C and C++), Integer arithmetic errors (for applications using C and C++), Cross-site scripting (for managed code and Web applications), SQL injection (for managed code and Web applications), Weak cryptography
- Security testing, including the following topics: Differences between security testing and functional testing, Risk assessment, Security testing methods
- Privacy, including the following topics: Types of privacy-sensitive data, Privacy design best practices, Risk assessment, Privacy development best practices, Privacy testing best practices

# SDL Roles

---

SDL roles are designed to provide project security and privacy oversight and have the authority to accept or reject security and privacy plans from a project team.

*Security Advisor/Privacy Advisor.* This role is filled by subject-matter experts (SMEs) from outside the project team. The person chosen for this task must fill two sub-roles:

- Auditor: monitors each phase of the software development process and attest to successful completion of each security requirement
- Expert: must possess verifiable subject-matter expertise in security.

*Team Champion.* should be filled by SMEs from the project team. Responsible for the negotiation, acceptance, and tracking of minimum security and privacy requirements

# Some SDL Practices

---

## Threat Modeling

- used in environments where there is meaningful security risk
- allows development teams to consider, document, and discuss the security implications of designs in the context of their planned operational environment
- Threat modeling is a team exercise, encompassing program/project managers, developers, and testers, performed during the software design stage

## Attack Surface Reduction

- a means of reducing risk by giving attackers less opportunity to exploit a potential weak spot or vulnerability
- encompasses shutting off or restricting access to system services, applying the principle of least privilege, and employing layered defenses wherever possible

# Some SDL Practices

---

## Static Analysis

- The team should be aware of the strengths and weaknesses of static analysis tools and be prepared to augment static analysis tools with other tools or human review as appropriate

## Dynamic Program Analysis

- specify tools that monitor application behavior for memory corruption, user privilege issues, and other critical security problems

## Fuzz Testing

- a specialized form of dynamic analysis used to induce program failure by deliberately introducing malformed or random data to an application

# Some SDL Practices

---

## Final Security Review

- a deliberate examination of all the security activities performed on a software application prior to release
- performed by the security advisor with assistance from the regular development staff and the security and privacy team leads
- includes an examination of threat models, exception requests, tool output, and performance against the previously determined quality gates or bug bars
- If a team does not meet all SDL requirements and the security advisor cannot approve the project, the project cannot be released
- Teams must either address whatever SDL requirements that they can prior to launch or escalate to executive management for a decision

Software Engineering: A Practitioner's Approach, 7/e Roger Pressman

Software Engineering, Pearson Education, 9th Ed., 2010. Ian Sommerville

SWEBOK by IEEE/ACM

[www.owasp.com](http://www.owasp.com)

[www.microsoft.com](http://www.microsoft.com)

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **Threat Modelling Concepts**

## **RL 3.1**

# Securing a Computer Based System

A computer-based system has three separate but valuable components (Assets) :

- Hardware
- Software
- Data

## Vulnerabilities

- Weaknesses in a system that may be able to be *exploited* in order to cause loss or harm
  - e.g., a file server that doesn't authenticate its users

## Threats

- A loss or harm that might befall a system
  - e.g., users' personal files may be revealed to the public

# Characteristics of Computer Intrusion

By computing system, we include

- Hardware
- Software
- Storage media
- Data and
- People

A system is most vulnerable at its weakest point

- A robber will not attempt to penetrate a 2-inch-thick metal door if a window gives easy access

# Principle of Easiest Penetration

An intruder must be expected to use any available means of penetration. The penetration may not necessarily be by the most obvious means, nor is it necessarily the one against which the most solid defense has been installed. And it certainly does not have to be the way we want the attacker to behave.

# Threat Model

---

When designing a system, we need to state the threat model

## Threat Model

- Set of threats we are undertaking to defend against
- Whom do we want to prevent from doing what?

## Attack

- An action which exploits a vulnerability to execute a threat
- e.g., telling the file server you are a different user in an attempt to read or modify their files

# Threats to Assets

---

According to Pfleeger, the threats are

- **Interruption** – an asset is destroyed, unavailable or unusable (*availability*)
- **Interception** – unauthorized party gains access to an asset (*confidentiality*)
- **Modification** – unauthorized party tampers with asset (*integrity*)
- **Fabrication** – unauthorized party inserts counterfeit object into the system (*authenticity*)

# Threat Consequences (IETF RFC 4949)

Threat Consequence	Threat Action (Attack)
<b>Unauthorized Disclosure</b> A circumstance or event whereby an entity gains access to data for which the entity is not authorized	<b>Exposure:</b> Sensitive data are directly released to an unauthorized entity. <b>Interception:</b> An unauthorized entity directly accesses sensitive data traveling between authorized sources and destinations. <b>Inference:</b> A threat action whereby an unauthorized entity indirectly accesses sensitive data (but not necessarily the data contained in the communication) by reasoning from characteristics or by-products of communications. <b>Intrusion:</b> An unauthorized entity gains access to sensitive data by circumventing a system's security protections.
<b>Deception</b> A circumstance or event that may result in an authorized entity receiving false data and believing it to be true	<b>Masquerade:</b> An unauthorized entity gains access to a system or performs a malicious act by posing as an authorized entity. <b>Falsification:</b> False data deceive an authorized entity. <b>Repudiation:</b> An entity deceives another by falsely denying responsibility for an act
<b>Disruption</b> A circumstance or event that interrupts or prevents the correct operation of system services and functions	<b>Incapacitation:</b> Prevents or interrupts system operation by disabling a system component. <b>Corruption:</b> Undesirably alters system operation by adversely modifying system functions or data. <b>Obstruction:</b> A threat action that interrupts delivery of system services by hindering system operation
<b>Usurpation</b> A circumstance or event that results in control of system services or functions by an unauthorized entity	<b>Misappropriation:</b> An entity assumes unauthorized logical or physical control of a system resource. <b>Misuse:</b> Causes a system component to perform a function or service that is detrimental to system security

# Who are Attackers

---

One approach to prevention is to understand who carries out attacks and why

## Amateurs

- Ordinary computer professionals or users who, while doing their jobs, discover they have access to something valuable. Amateurs may be disgruntled employees who vow to get even with management

## Crackers or Malicious Hackers

- Attempt to access computing facilities for which they have not been authorized (often students who see it as victimless crime). Some carry out for curiosity, personal gain or self-satisfaction

# Who are Attackers

---

## Career Criminals

- Understands the targets of computer crime; often begin as computer professionals, then shift to crime finding payoff. “They don’t want to write a worm that destroys your hardware. They want to assimilate your computers and use them to make money”

## Terrorists

- They use computers in three ways
  - Targets of attack – denial of service, web site defacement attacks are popular to attract attention to the cause and bring undesired negative attention to the targets of attack
  - Propaganda vehicles – inexpensive way to get a message to many
  - Methods of attack – use computers to launch attacks

# Method, Opportunity, Motive

---

A malicious attacker must have three things

- Method : the skills, knowledge, tools, and other things with which to be able to pull off the attack
- Opportunity : the time and access to accomplish the attack
- Motive : a reason to want to perform this attack against this system

Deny any of those three things and the attack will not occur.

# Methods of defense

---

How can we defend against a threat?

- Prevent it: prevent the attack
- Deter it: make the attack harder or more expensive
- Deflect it: make yourself less attractive to attacker
- Detect it: notice that attack is occurring (or has occurred)
- Recover from it: mitigate the effects of the attack

# Methods of defense

---

Threat: your car may get stolen

How to defend?

- Prevent: Immobilizer? Is it possible to absolutely prevent?
- Deter: Store your car in a secure parking facility
- Deflect: Have sticker mentioning car alarm, keep valuables out of sight
- Detect: Car alarms
- Recover: Insurance

# Structured Approach to Threat Modeling

According to Adam Shostack, you begin threat modeling by focusing on four key questions

- What are you building?
- What can go wrong?
- What should you do about those things that can go wrong?
- Did you do a decent job of analysis (retrospect)

# Structured Approach to Threat Modeling

People often use an approach centered on

- Models of their assets (Valuable things they have),
- Models of attackers (People who might go after assets), or
- Models of their software (Common way to attack is via the deployed software)

Centering on one of these is preferable to using approaches that combine them because the combinations tend to be confusing

According to Adam Shostack, first two sets of models help in engaging with non-technical people and third type of models are important for software development

# What Threat Modelling is (not)

What threat modelling is	What threat modelling is not
A team activity	An activity performed by a single team member in isolation
An activity that helps identify security vulnerabilities in a variety of software applications	Just for large software projects
An activity that should be performed for every iteration or sprint during agile development	An activity done once during the lifecycle of the project

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **OWASP Threat Modelling Process – Part 1**

## **RL 3.2.1**

# OWASP Threat Modeling

---

According to OWASP, the threat modeling process for software application can be decomposed into 3 high level steps

- Decompose the Application
- Determine and rank threats
- Determine countermeasures and mitigation

# Decompose the Application (step 1)

Understanding of the application and how it interacts with external entities.

- involves creating use-cases to understand how the application is used,
- identifying entry points to see where a potential attacker could interact with the application,
- identifying assets i.e. items/areas that the attacker would be interested in, and
- identifying trust levels which represent the access rights that the application will grant to external entities.

Produce data flow diagrams (DFDs) for the application. The DFDs show the different paths through the system, highlighting the privilege boundaries.

## Determine and rank threats (step 2)

---

A threat categorization such as STRIDE can be used,

Spoofing

Tampering

Repudiation

Information disclosure

Denial of service

Elevation of privilege

The STRIDE categorization helps to identify threats from the attacker perspective.

## Determine and rank threats (step 2)

---

A threat categorization ASF (Application Security Framework) defines threat categories such as

- Auditing & Logging,
- Authentication,
- Authorization,
- Configuration Management,
- Data Protection in Storage and Transit,
- Data Validation,
- Exception Management.

The ASF categorization helps to identify threats from the defensive perspective.

## Determine and rank threats (step 2)

---

DFDs produced in step 1 help to identify the potential threat targets from the attacker's perspective, viz.

- data sources,
- processes,
- data flows, and
- interactions with users.

Use and abuse cases can illustrate how existing protective measures could be bypassed, or where a lack of such protection exists.

## Determine and rank threats (step 2)

These threats can be identified further as the roots for threat trees;

- one tree for each threat goal.

From the defensive perspective, ASF categorization helps to identify the threats as weaknesses of security controls for such threats.

The determination of the security risk for each threat can be determined using a value-based risk model such as DREAD

# **Determine countermeasures and mitigation**

---

A lack of protection against a threat might indicate a vulnerability whose risk exposure could be mitigated with the implementation of a countermeasure.

Countermeasures can be identified using threat-countermeasure mapping lists.

Based on risk ranking assigned to the threats, it is possible to sort threats from the highest to the lowest risk, and prioritize the mitigation effort, such as by responding to such threats by applying the identified countermeasures

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **OWASP Threat Modelling Process – Part 2**

## **RL 3.2.2**

# OWASP Threat Modeling Example

## **Application Description:**

The college library website is the first implementation of a website to provide librarians and library patrons (students and college staff) with online services. As this is the first implementation of the website, the functionality will be limited. There will be three users of the application:

1. Students
2. Staff
3. Librarians

Staff and students will be able to log in and search for books, and staff members can request books. Librarians will be able to log in, add books, add users, and search for books.

# External Dependencies

External dependencies are items external to the code of the application that may pose a threat to the application. These items are typically still within the control of the organization, but possibly not within the control of the development team

ID	Description
1	The database server will be MySQL and it will run on a Linux server. This server will be hardened as per the college's server hardening standard. This will include the application of the latest operating system and application security patches.
2	The connection between the Web Server and the database server will be over a private network.

# Trust Levels

---

Trust levels represent the access rights that the application will grant to external entities.

The trust levels are cross referenced with the entry points and assets.

This allows us to define the access rights or privileges required at each entry point, and those required to interact with each asset

# Trust Levels

ID	Name	Description
1	Anonymous Web User	A user who has connected to the college library website but has not provided valid credentials.
2	User with Valid Login Credentials	A user who has connected to the college library website and has logged in using valid login credentials.
3	User with Invalid Login Credentials	A user who has connected to the college library website and is attempting to log in using invalid login credentials.
4	Librarian	The librarian can create users on the library website and view their personal information.
5	Database Server Administrator	The database server administrator has read and write access to the database that is used by the college library website.
6	Website Administrator	The Website administrator can configure the college library website.
7	Web Server User Process	This is the process/user that the web server executes code as and authenticates itself against the database server as.
8	Database Read User	The database user account used to access the database for read access.
9	Database Read/Write User	The database user account used to access the database for read and write access.

# Entry Points

Entry points define the interfaces through which potential attackers can interact with the application or supply it with data. In order for a potential attacker to attack an application, entry points must exist.

ID	Name	Description	Trust Levels
1.1	Library Main Page	The splash page for the college library website is the entry point for all users.	(1) Anonymous Web User (2) User with Valid Login Credentials (3) User with Invalid Login Credentials (4) Librarian
1.3	Search Entry Page	The page used to enter a search query.	(2) User with Valid Login Credentials (4) Librarian

# Assets

---

Attacker is interested in the system because it has Assets

Assets can be

Physical – Private Data, List of customers etc.

Privilege – System has ability to update/process data

Abstract – Reputation of the organization

Assets are documented in the threat model as follows:

ID,

Name,

Description

Trust Level (required for access)

# Assets

ID	Name	Description	Trust Levels
1	Library Users and Librarian	Assets relating to students, faculty members, and librarians.	
1.1	User Login Details	The login credentials that a student or a faculty member will use to log into the College Library website.	(2) User with Valid Login Credentials, (4) Librarian, (5) Database Server Administrator, (7) Web Server User Process, (8) Database Read User, (9) Database Read/Write User
1.3	Personal Data	The College Library website will store personal information relating to the students, faculty members, and librarians.	(4) Librarian, (5) Database Server Administrator, (6) Website Administrator, (7) Web Server User Process, (8) Database Read User, (9) Database Read/Write User
2	System	Assets relating to the underlying system.	
2.2	Ability to Execute Code as a Web Server User	This is the ability to execute source code on the web server as a web server user.	(6) Website Administrator, (7) Web Server User Process
2.4	Ability to Execute SQL as a Database Read/Write User	This is the ability to execute SQL. Select, insert, and update queries on the database and thus have read and write access to any information stored within the College Library database.	(5) Database Server Administrator, (9) Database Read/Write User
3	Website	Assets relating to the College Library website.	
3.1	Login Session	This is the login session of a user to the College Library website. This user could be a student, a member of the college faculty, or a Librarian.	(2) User with Valid Login Credentials, (4) Librarian
3.3	Ability to Create Users	The ability to create users would allow an individual to create new users on the system. These could be student users, faculty member users, and librarian users.	(4) Librarian, (6) Website Administrator

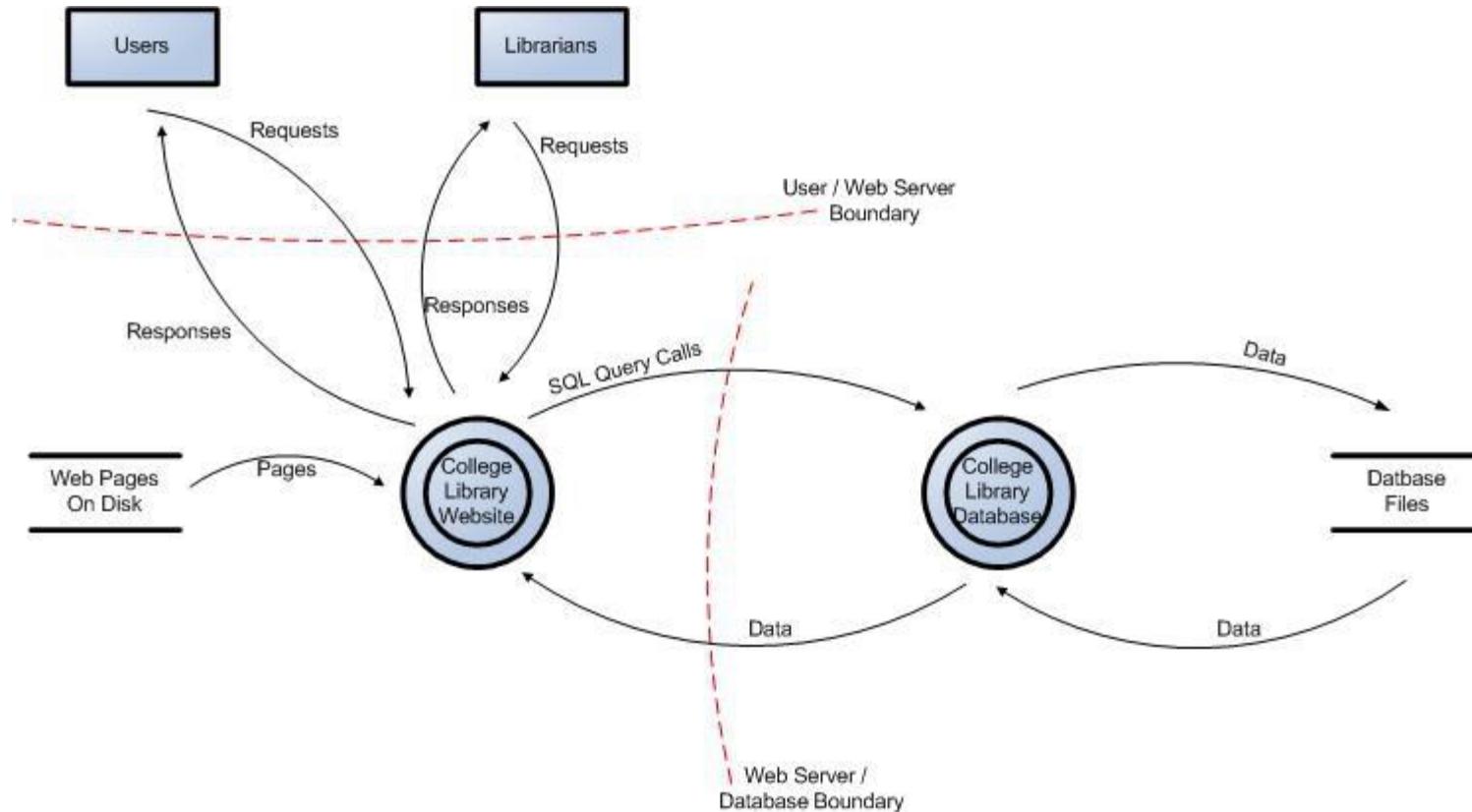
# Data Flow Diagrams

---

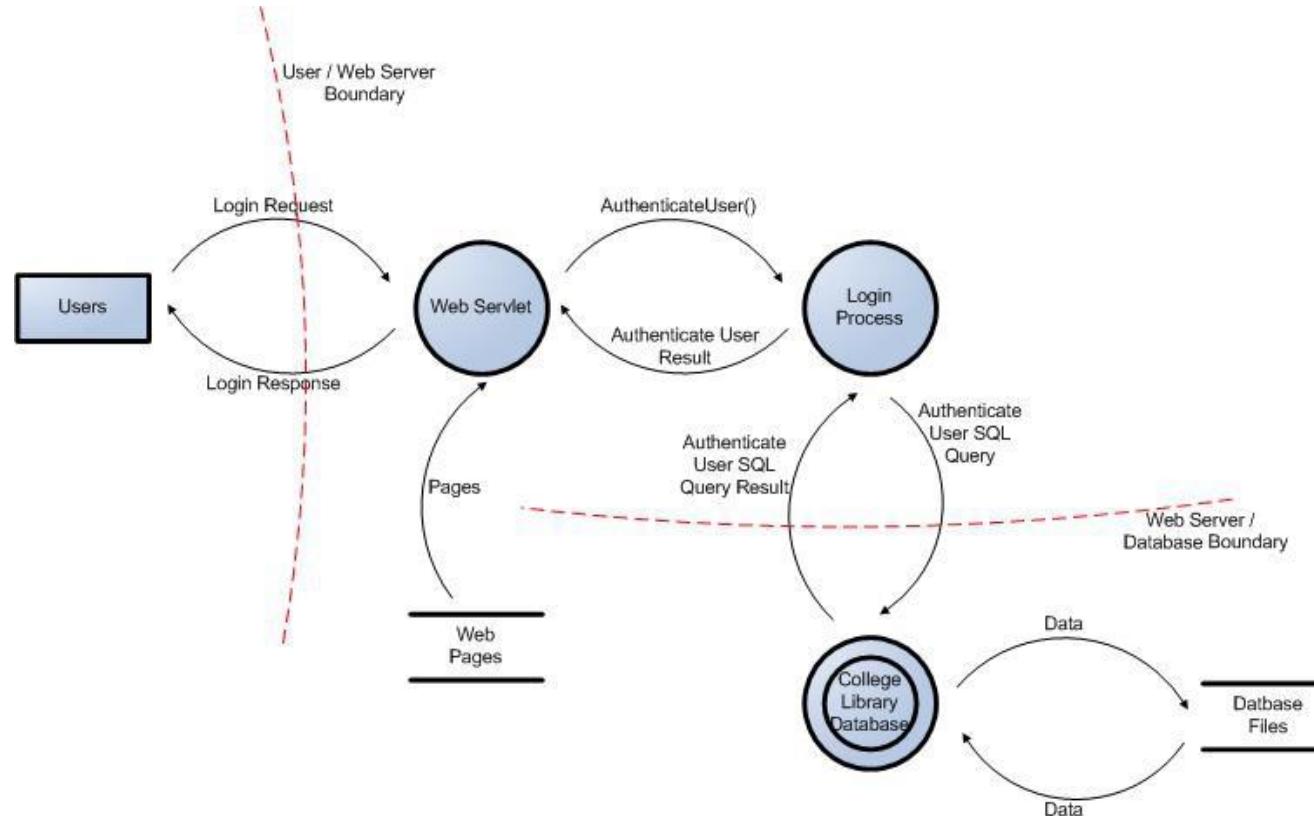
Knowledge of Assets, Entry points, etc. help in creating DFDs.

- The DFDs will allow us to gain a better understanding of the application by providing a visual representation of how the application processes data
- DFDs focus on how data moves through the application and what happens to the data as it moves
- DFDs are hierarchical in structure, so they can be used to decompose the application into subsystems and lower-level subsystems

# DFD for the example



# Partially Expanded DFD



# Trust Boundaries in DFD

---

Add trust boundaries that intersect data flows

- Points/surfaces where an attacker can interject
  - Machine boundaries, privilege boundaries, integrity boundaries are examples of trust boundaries
  - Threads in a native process are often inside a trust boundary, because they share the same privileges, rights, identifiers and access
- Processes talking across a network always have a trust boundary
  - They may create a secure channel, but they're still distinct entities
  - Encrypting network traffic doesn't address tampering or spoofing

Iterate over processes, data stores, and see where they need to be broken down

# Threats (STRIDE)

Type	Examples
Spoofing	Threat action aimed to illegally access and use another user's credentials, such as username and password.
Tampering	Threat action aimed to maliciously change/modify persistent data, such as persistent data in a database, and the alteration of data in transit between two computers over an open network, such as the Internet.
Repudiation	Threat action aimed to perform illegal operations in a system that lacks the ability to trace the prohibited operations.
Information disclosure	Threat action to read a file that one was not granted access to, or to read data in transit.
Denial of service	Threat aimed to deny access to valid users, such as by making a web server temporarily unavailable or unusable.
Elevation of privilege	Threat aimed to gain privileged access to resources for gaining unauthorized access to information or to compromise a system.

# Example Countermeasures (ASF)

Threat Type	Countermeasure
Authentication	<ul style="list-style-type: none"><li>1.Credentials and authentication tokens are protected with encryption in storage and transit</li><li>2.Protocols are resistant to brute force, dictionary, and replay attacks</li></ul>
Authorization	<ul style="list-style-type: none"><li>1.Strong ACLs are used for enforcing authorized access to resources</li><li>2.Role-based access controls are used to restrict access to specific operations</li></ul>
Configuration Management	<ul style="list-style-type: none"><li>1.Least privileged processes are used and service accounts with no administration capability</li><li>2.Auditing and logging of all administration activities is enabled</li></ul>
Data Protection in Storage and Transit	<ul style="list-style-type: none"><li>1.Standard encryption algorithms and correct key sizes are being used</li><li>2.Hashed message authentication codes (HMACs) are used to protect data integrity</li></ul>
Data Validation / Parameter Validation	<ul style="list-style-type: none"><li>1.Data type, format, length, and range checks are enforced</li><li>2.No security decision is based upon parameters (e.g. URL parameters) that can be manipulated</li></ul>
Error Handling and Exception Management	<ul style="list-style-type: none"><li>1.All exceptions are handled in a structured manner</li><li>2.Error messages are scrubbed so that no sensitive information is revealed to the attacker</li></ul>
User and Session Management	<ul style="list-style-type: none"><li>1.No sensitive information is stored in clear text in the cookie</li><li>2.Cookies are configured to expire</li></ul>
Auditing and Logging	<ul style="list-style-type: none"><li>1.Sensitive information (e.g. passwords, PII) is not logged</li><li>2.Integrity controls (e.g. signatures) are enforced on log files to provide non-repudiation</li></ul>

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **SDL Threat Modeling – Part 1**

## **RL 3.3.1**

# Objectives

---

Produce software that's secure by design

- Improve designs the same way we've improved code

Because attackers think differently

- Creator blindness/new perspective

Allow you to predictably and effectively find security problems early in the process

# Responsibilities

---

## Building a threat model (at Microsoft)

- Program Manager (PM) owns overall process
- Testers
  - Identify threats in analyze phase
  - Use threat models to drive test plans
- Developers create diagrams

# Customers / Work Products

---

## Customers for threat models

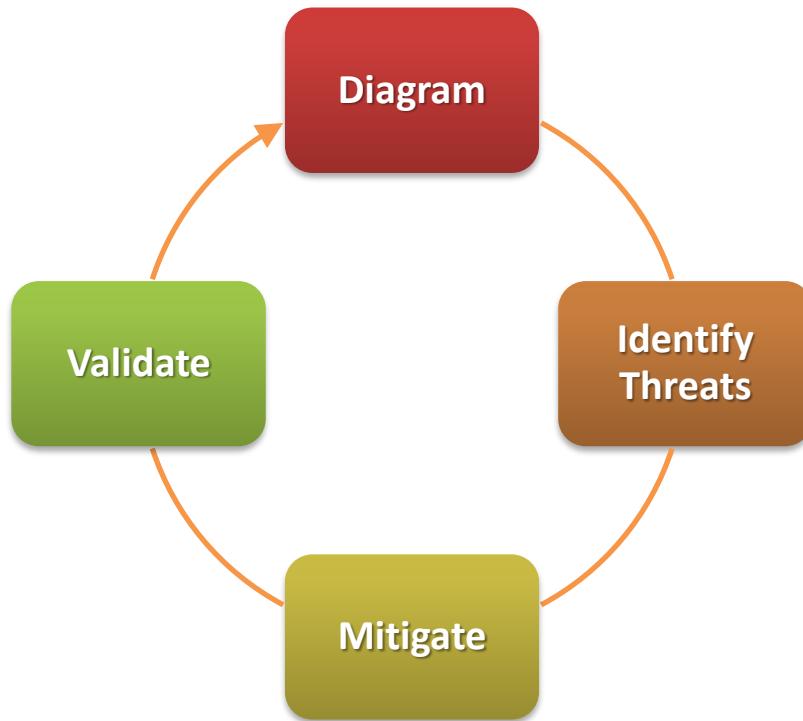
- Your team
- Other features, product teams
- Customers, via user education
- “External” quality assurance resources, such as pen testers

## Threat model documentation

- The product as a whole
- The security-relevant features
- The attack surfaces

# The Process in a Nutshell

---



# Diagramming

---

Use DFDs (Data Flow Diagrams)

- Include processes, data stores, data flows
- Include *trust boundaries*
- Diagrams per scenario may be helpful

Update diagrams as product changes

Enumerate assumptions, dependencies

Number everything (if manual)

# Effective Threat Modeling Meetings

Develop draft threat model before the meeting

- Use the meeting to discuss

Start with a DFD walkthrough

Identify most interesting elements

- Assets (if you identify any)
- Entry points/trust boundaries

Walk through STRIDE against those elements

Threats that cross elements/recur

- Consider library, redesigns

# Validating Threat Models

---

Validate the whole threat model

- Does diagram match final code?
- Are threats enumerated?
- Minimum: STRIDE per element that touches a trust boundary
- Has Test / QA reviewed the model?
  - Tester approach often finds issues with threat model or details
- Is each threat mitigated?
- Are mitigations done right?

Did you check these before Final Security Review?

- Shipping will be more predictable

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **SDL Threat Modeling – Part 2**

## **RL 3.3.2**

# Threat: Spoofing

---

Threat	<b>Spoofing</b>
Property	Authentication
Definition	Impersonating something or someone else
Example	Pretending to be any of billg, microsoft.com, or ntdll.dll

# Threat: Tampering

---

Threat	<b>Tampering</b>
Property	Integrity
Definition	Modifying data or code
Example	Modifying a DLL on disk or DVD, or a packet as it traverses the LAN

# Threat: Repudiation

---

Threat	<b>Repudiation</b>
Property	Non-Repudiation
Definition	Claiming to have not performed an action
Example	“I didn’t send that email,” “I didn’t modify that file,” “I didn’t visit that web site”

# Threat: Information Disclosure

---

Threat	Information Disclosure
Property	Confidentiality
Definition	Exposing information to someone not authorized to see it
Example	Allowing someone to read the Windows source code; publishing a list of customers to a Web site

# Threat: Denial of Service

---

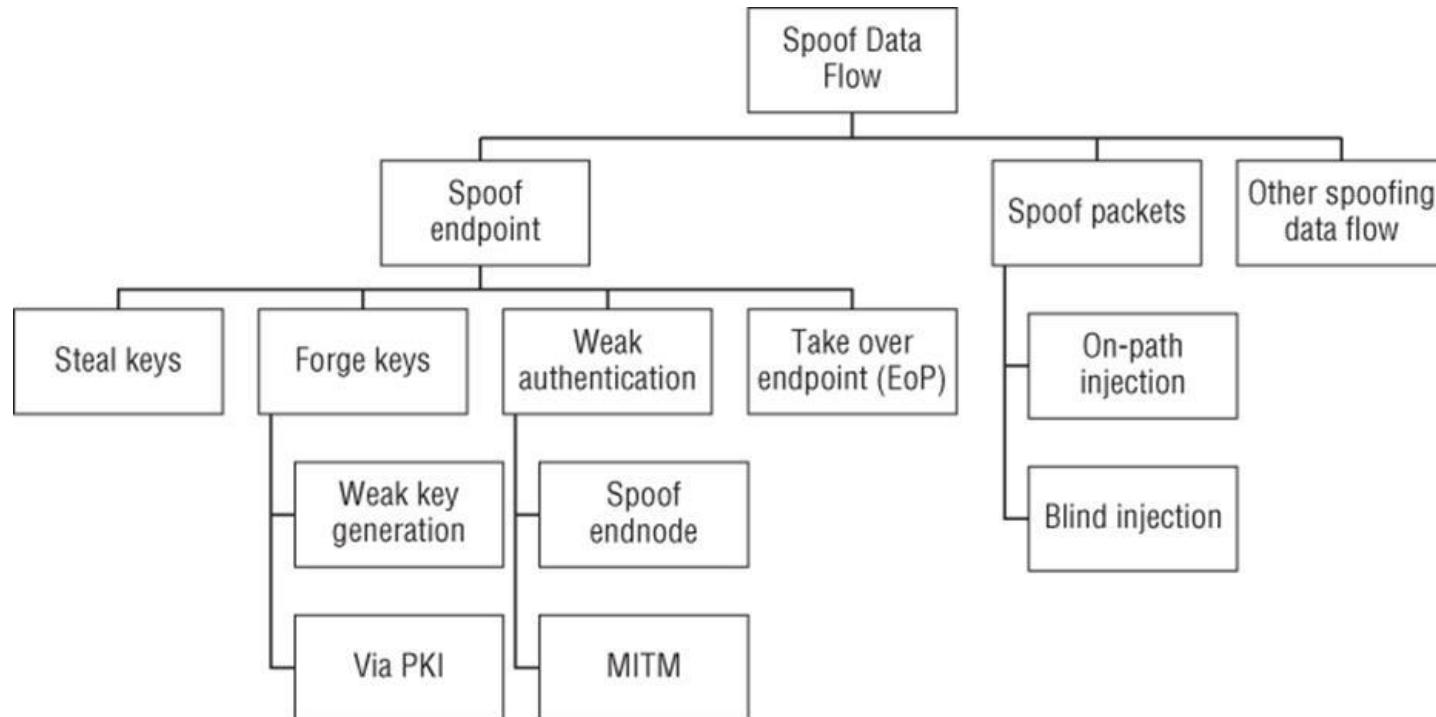
Threat	<b>D</b> enial of Service
Property	Availability
Definition	Deny or degrade service to users
Example	Crashing Windows or a Web site, sending a packet and absorbing seconds of CPU time, or routing packets into a black hole

# Threat: Elevation of Privilege

---

Threat	Elevation of Privilege (EoP)
Property	Authorization
Definition	Gain capabilities without proper authorization
Example	Allowing a remote Internet user to run commands is the classic example, but going from a "Limited User" to "Admin" is also EoP

# Threat Tree (Spoofing data flow)



# How to Mitigate

---

Address each threat

Four ways to address threats

1. Redesign to eliminate
2. Apply standard mitigations
  - What have similar software packages done and how has that worked out for them?
3. Invent new mitigations (riskier)
4. Accept vulnerability in design

# Standard Mitigations

<b>Spoofing</b>	<b>Authentication</b>	To authenticate principals: <ul style="list-style-type: none"><li>• Cookie authentication</li><li>• Kerberos authentication</li></ul> To authenticate code or data: <ul style="list-style-type: none"><li>• Digital signatures</li></ul>
<b>Tampering</b>	<b>Integrity</b>	<ul style="list-style-type: none"><li>• ACLs</li><li>• Digital signatures</li></ul>
<b>Repudiation</b>	<b>Non Repudiation</b>	<ul style="list-style-type: none"><li>• Secure logging and auditing</li><li>• Digital Signatures</li></ul>
<b>Information Disclosure</b>	<b>Confidentiality</b>	<ul style="list-style-type: none"><li>• Encryption</li><li>• ACLS</li></ul>
<b>Denial of Service</b>	<b>Availability</b>	<ul style="list-style-type: none"><li>• ACLs</li><li>• Filtering</li><li>• Quotas</li></ul>
<b>Elevation of Privilege</b>	<b>Authorization</b>	<ul style="list-style-type: none"><li>• ACLs</li><li>• Group or role membership</li><li>• Privilege ownership</li><li>• Input validation</li></ul>

# DREAD

---

In the Microsoft DREAD threat-risk ranking model, the technical risk factors for impact are Damage and Affected Users, while the ease of exploitation factors are Reproducibility, Exploitability and Discoverability. This risk factorization allows the assignment of values to the different influencing factors of a threat. To determine the ranking of a threat, the threat analyst has to answer basic questions for each factor of risk

- For Damage: How big would the damage be if the attack succeeded?
- For Reproducibility: How easy is it to reproduce an attack to work?
- For Exploitability: How much time, effort, and expertise is needed to exploit the threat?
- For Affected Users: If a threat were exploited, what percentage of users would be affected?
- For Discoverability: How easy is it for an attacker to discover this threat?

# DREAD Example

---

The college library website use case:

***Threat: Malicious user views confidential information of students, faculty members and librarians.***

- **Damage potential:** Threat to reputation as well as financial and legal liability:8
- **Reproducibility:** Fully reproducible:10
- **Exploitability:** Require to be on the same subnet or have compromised a router:7
- **Affected users:** Affects all users:10
- **Discoverability:** Can be found out easily:10

Overall DREAD score:  $(8+10+7+10+10) / 5 = 9$

In this case having 9 on a 10 point scale is certainly a high risk threat

Threat Modelling by Adam Shostack, John Wiley 2014

Security in Computing by Charles P. Pfleeger, Shari L. Pfleeger, and Deven Shah  
Pearson Education 2009

Computer Security: Principles and Practice by William Stallings, and Lawrie Brown  
Pearson, 2008.

[www.owasp.com](http://www.owasp.com)

[www.microsoft.com](http://www.microsoft.com)

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **Security Requirements Engineering – Part 1**

## **RL 4.1.1**

# Importance of Requirements Engineering

Some studies have shown that requirements engineering defects cost 10 to 200 times as much to correct once the system has become operational than if they were detected during requirements development.

According to Charette[2005], Requirements problems are among the top causes of the following undesirable phenomena

- Projects are significantly over budget, go past schedule, have significantly reduced scope, or are cancelled
- Development teams deliver poor-quality applications
- Products are not significantly used once delivered

# Requirements Engineering Challenges

Requirements Engineering on individual projects often suffers from the following problems:

- Requirements identification typically does not include all relevant stakeholders and does not use the most modern or efficient techniques.
- Requirements are often statements describing architectural constraints or implementation mechanisms rather than statements describing what the system must do.
- Requirements are often directly specified without any analysis or modeling. When analysis is done, it is usually restricted to functional end-user requirements, ignoring
  - 1) quality requirements such as security,
  - 2) other functional and nonfunctional requirements, and
  - 3) architecture, design, implementation, and testing constraints.

# Requirements Engineering Challenges

Requirements specification is typically haphazard, with specified requirements being

- ambiguous,
- incomplete (e.g., nonfunctional requirements are often missing),
- inconsistent,
- not cohesive,
- infeasible,
- obsolete,
- neither testable nor capable of being validated, and
- not usable by all of their intended audiences.

Requirements management is typically weak, with ineffective forms of data capture

- e.g., in one or more documents (rather than in a database or tool) and missing attributes.
- often limited to tracing, scheduling, and prioritization, without change tracking or other configuration management.

# Quality Requirements

---

Project teams often neglect *quality* requirements, such as performance, safety, security, reliability, and maintainability.

- Developers of certain kinds of mission-critical systems and systems in which human life is involved, such as the space shuttle, have long recognized the importance of quality requirements and have accounted for them in software development.
- In many other systems, however, quality requirements are treated in an inadequate way. Hence we see the failure of software associated with power systems, telephone systems, unmanned spacecraft, and so on.

This inattention to quality requirements is exacerbated by the desire to keep costs down and meet aggressive schedules.

# Security Requirements Engineering

According to BSI[09], if security requirements are not effectively defined, the resulting system cannot be **evaluated** for success or failure prior to its implementation

Operational environments and business goals often change **dynamically**, with the result that security requirements development is not a one-time activity.

Requirements engineering research and practice pay a lot of attention to the functionality of the system from the user's perspective, but little attention is devoted to what the system should **not** do [Bishop 2002]

# Security Requirements Engineering

Users have implicit assumptions for the software applications and systems to be secure and are surprised when they are not. These user assumptions need to be translated into security requirements for the software systems when they are under development.

It is important for requirements engineers to think about the attacker's perspective and not just the functionality of the system from the end-user's perspective.

- An attacker is not particularly interested in functional features of the system, unless they provide an avenue for attack. Instead, the attacker typically looks for defects and other conditions outside the norm that will allow a successful intrusion to take place.

# Security Is Not a Set of Features

Security features such as password protection, firewalls, virus detection tools etc. are, in fact, not security requirements.

They are rather implementation mechanisms that are intended to satisfy unstated requirements, such as authenticated access

A systematic approach to security requirements engineering will help avoid the problem of generic lists of features and take into account the attacker's perspective.

No convenient security pull-down menu that will let you select “security” and do the needful.



# **Security Requirements Engineering - Part 2**

## **RL 4.1.2**

# Security Is Not a Set of Features

Security is an **emergent** property of a system, not a feature

Because security is not a feature, it cannot be bolted on after other software features are codified, nor can it be **patched** in after attacks have occurred in the field. Instead, security must be built into the product from the ground up

Most cost-effective approach to software security incorporates thinking beyond normative features and maintains that thinking throughout the development process

Every time a new requirement, feature, or use case is created, the developer or security specialist should spend some time thinking about how that feature might be unintentionally misused or intentionally abused

# Thinking Beyond Normal

---

When we design and analyze a system, we're in a great position to know our systems better than potential attackers do.

We can leverage this knowledge to the benefit of security and reliability, by asking and answering the critical questions:

- Which assumptions are implicit in our system?
- Which kinds of things make our assumptions false?
- Which kinds of attack patterns will an attacker bring to bear?

# Thinking like an attacker

---

System's creators are not the best security analysts of that system.

'Thinking like an attacker' is extremely difficult for those who have built up a set of implicit assumptions

System Creators make excellent subject matter experts (SMEs).

Together, SMEs and security analysts can ferret out base assumptions in a system under analysis and think through the ways an attacker will approach the software

# Creating Useful Misuse Cases

Misuse cases is to decide and document *a priori* how software should react to illegitimate use

Unlike the functional requirements, designers/developers play the role of user and explain design and underlying assumptions to security expert documents

To guide brainstorming, software security experts ask many questions of a system's designers to help identify the places where the system is likely to have weaknesses. This activity mirrors the way attackers think.

The brainstorming covers user interfaces, environmental factors, and events that developers assume a person can't or won't do

- “Users can't enter more than 50 characters because the JavaScript code won't let them”
- “Users don't understand the format of the cached data, so they can't modify it.”

# Misuse vs. Abuse

---

According to Chun Wei,

Misuse cases are defined as “behavior that the system/entity owner does not want to occur”

- An interaction results in a session key being revealed to an actor who should not see the session key

Abuse cases are defined as “... where the results of the interaction are harmful to the system ...”

- the actor posts the session key on a public website, then an abuse case takes place

But some authors do not distinguish

# Specifying Abuse Cases

---

The process of specifying abuse cases makes a designer very clearly differentiate appropriate use from inappropriate use.

The security expert/designer must ask the right questions:

- How can the system distinguish between good input and bad input?
- Can it tell whether a request is coming from a legitimate application or from a rogue application replaying traffic?
- where might a bad guy be positioned? On the wire? At a workstation? In the back office?
- Any communication line between two endpoints or two components is a place where an attacker might try to interpose himself or herself
- what can this attacker do in the system? Watch communications traffic? Modify and replay such traffic? Read files stored on the workstation? Change registry keys or configuration files? Be the DLL?

Trying to answer such questions helps software designers explicitly question design and architecture assumptions, and it puts the designer squarely ahead of the attacker by identifying and fixing a problem before it's ever created.



# **CMU SQUARE Process Model**

## **RL 4.2.1**

# SQUARE Process Model

---

Security Quality Requirements Engineering (SQUARE) is a process model that was developed at Carnegie Mellon University [Mead 2005].

SQUARE provides a means for eliciting, categorizing, and prioritizing security requirements for information technology systems and applications.

The focus of the model is to build security concepts into the early stages of the SDLC. It can also be used for documenting and analyzing the security aspects of systems once they are implemented in the field and for steering future improvements and modifications to those systems.

The SQUARE work is supported by the Army Research Office through grant (“Perpetually Available and Secure Information Systems”) to Carnegie Mellon University’s CyLab.

# SQUARE Process Steps

---

1. Agree on definitions
2. Identify security goals
3. Develop artifacts to support security requirements definition
4. Perform (security) risk assessment
5. Select elicitation techniques
6. Elicit security requirements
7. Categorize requirements as to level (e.g., system, software) and whether they are requirements or other kinds of constraints
8. Prioritize requirements
9. Inspect requirements

# SQUARE Process Steps

No	Step	Input	Techniques	Participants	Output
1	Agree on definitions	Candidate definitions from IEEE and other standards	Structured interviews, focus group	Stakeholders, requirements engineers	Agreed-to definitions
2	Identify security goals	Definitions, candidate goals, business drivers, policies and procedures, examples	Facilitated work session, surveys, interviews	Stakeholders, requirements engineers	Goals
3	Develop artifacts to support security requirements definition	Potential artifacts list (e.g., scenarios, misuse cases, templates, forms)	Work session	Requirements engineers	Needed artifacts: scenarios, misuse cases, models, templates, forms

# SQUARE Process Steps

No	Step	Input	Techniques	Participants	Output
4	Perform (security) risk assessment	Misuse cases, scenarios, security goals	Risk assessment method, analysis of anticipated risk against organizational risk tolerance, including threat analysis	Requirements engineers, risk expert, stakeholders	Risk assessment results
5	Select elicitation techniques	Goals, definitions, candidate techniques, expertise of stakeholders, organizational style, culture, level of security needed, cost-benefit analysis	Work session	Requirements engineers	Selected elicitation techniques

# SQUARE Process Steps

No	Step	Input	Techniques	Participants	Output
6	Elicit security requirements	Artifacts, risk assessment results, selected techniques	QFD, Joint Application Development, interviews, surveys, model based analysis, checklists, lists of reusable requirements types, document reviews	Stakeholders facilitated by requirements engineers	Initial cut at security requirements
7	Categorize requirements as to level (e.g., system, s/w ) and whether they are requirements or other kinds of constraints	Initial requirements, architecture	Work session using a standard set of categories	Requirements engineers, other specialists as needed	Categorized requirements

# SQUARE Process Steps

No	Step	Input	Techniques	Participants	Output
8	Prioritize requirements	Categorized requirements and risk assessment results	Prioritization methods such as Analytical Hierarchy Process (AHP - structured technique for organizing information) etc.	Stakeholders facilitated by requirement s engineers	Prioritized requirements
9	Inspect requirements	Prioritized requirements, candidate formal inspection technique	Inspection method such as Fagan (formal approach with exit criteria) and peer reviews	Inspection team	Initial selected requirements, documentation of decision-making process and rationale

# Sample: Identify Security Goals

---

Work with the client to identify security goals that mapped to the company's overall business goals.

Consider Asset Management System (AMS) of Acme Co.

*Business goal of AMS:* To provide an application that supports asset management and planning.

*Security goals:* Three high-level security goals were derived for the system:

- Management shall exercise effective control over the system's configuration and use.
- The confidentiality, accuracy, and integrity of the AMS shall be maintained.
- The AMS shall be available for use when needed.

# Attack Patterns

---

Attack patterns are descriptions of common methods for exploiting software. Act as a mechanism to capture and communicate the attacker's perspective.

They derive from the concept of design patterns [Gamma 95] applied in a destructive rather than constructive context and are generated from in-depth analysis of specific real-world exploit examples

The following typical information is captured for each attack pattern:

- Pattern name and classification
- Attack prerequisites
- Description
- Targeted vulnerabilities or weaknesses
- Method of attack
- Attacker goal
- Attacker skill level required
- Resources required
- Blocking solutions
- Context description
- References

# Attack Patterns

---

- **Pattern Name and Classification:** A unique, descriptive identifier for the pattern.
- **Attack Prerequisites:** What conditions must exist or what functionality and what characteristics must the target software have, or what behavior must it exhibit, for this attack to succeed?
- **Description:** A description of the attack including the chain of actions taken.
- **Related Vulnerabilities or Weaknesses:** What specific vulnerabilities or weaknesses does this attack leverage?
- **Method of Attack:** What is the vector of attack used (e.g., malicious data entry, maliciously crafted file, protocol corruption etc.)?

# Attack Patterns

---

- **Attack Motivation-Consequences:** What is the attacker trying to achieve by using this attack?
- **Attacker Skill or Knowledge Required:** What level of skill or specific knowledge must the attacker have to execute such an attack?
- **Resources Required:** What resources (e.g., CPU cycles, IP addresses, tools, time) are required to execute the attack?
- **Solutions and Mitigations:** What actions or approaches are recommended to mitigate this attack, either through resistance or through resiliency?
- **Context Description:** In what technical contexts (e.g., platform, OS, language, architectural paradigm) is this pattern relevant? This information is useful for selecting a set of attack patterns that are appropriate for a given context.
- **References:** What further sources of information are available to describe this attack?

# Attack Pattern Example

---

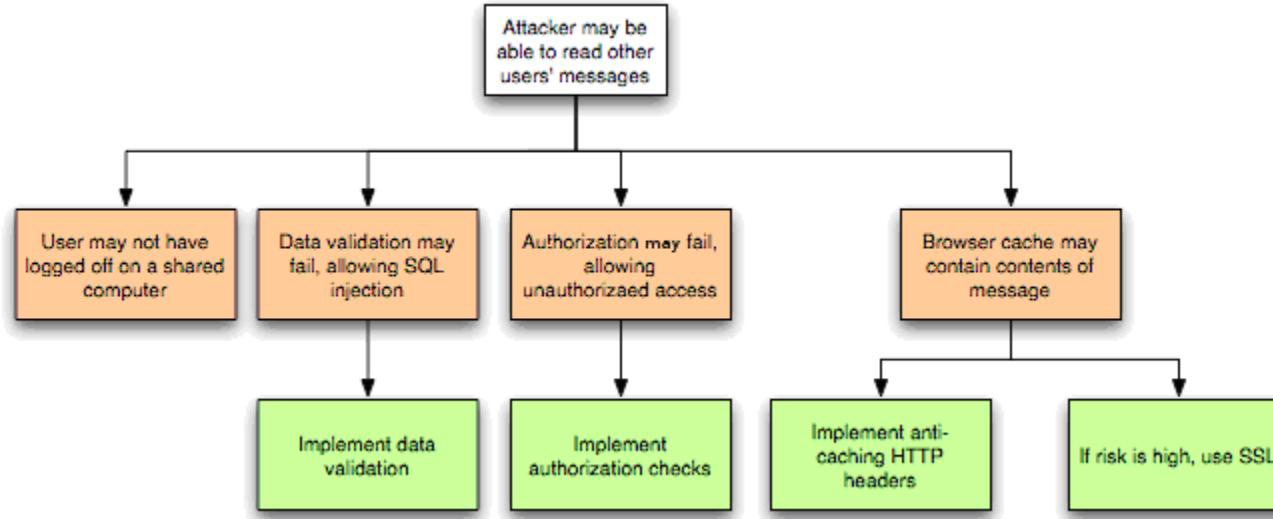
- **Pattern name and classification:** Shell Command Injection—Command Delimiters.
- **Attack Prerequisites:** The application must pass user input directly into a shell command.
- **Description:** Using the semicolon or other off-nominal characters, multiple commands can be strung together. Unsuspecting target programs will execute all the commands. An example may be when authenticating a user using a web form, where the username is passed directly to the shell as in: `exec( "cat data_log_" + userInput + ".dat")`.
  - The "+" sign denotes concatenation. The developer expects that the user will only provide a username. However, a malicious user could supply `"username.dat; rm -rf / ;"` as the input to execute the malicious commands on the machine running the target software. In the above case, the actual commands passed to the shell will be: `cat data_log_username.dat; rm -rf /; .dat`
  - The first command may or may not succeed; the second command will delete everything on the file system to which the application has access, and success/failure of the last command is irrelevant.

# Attack Pattern Example

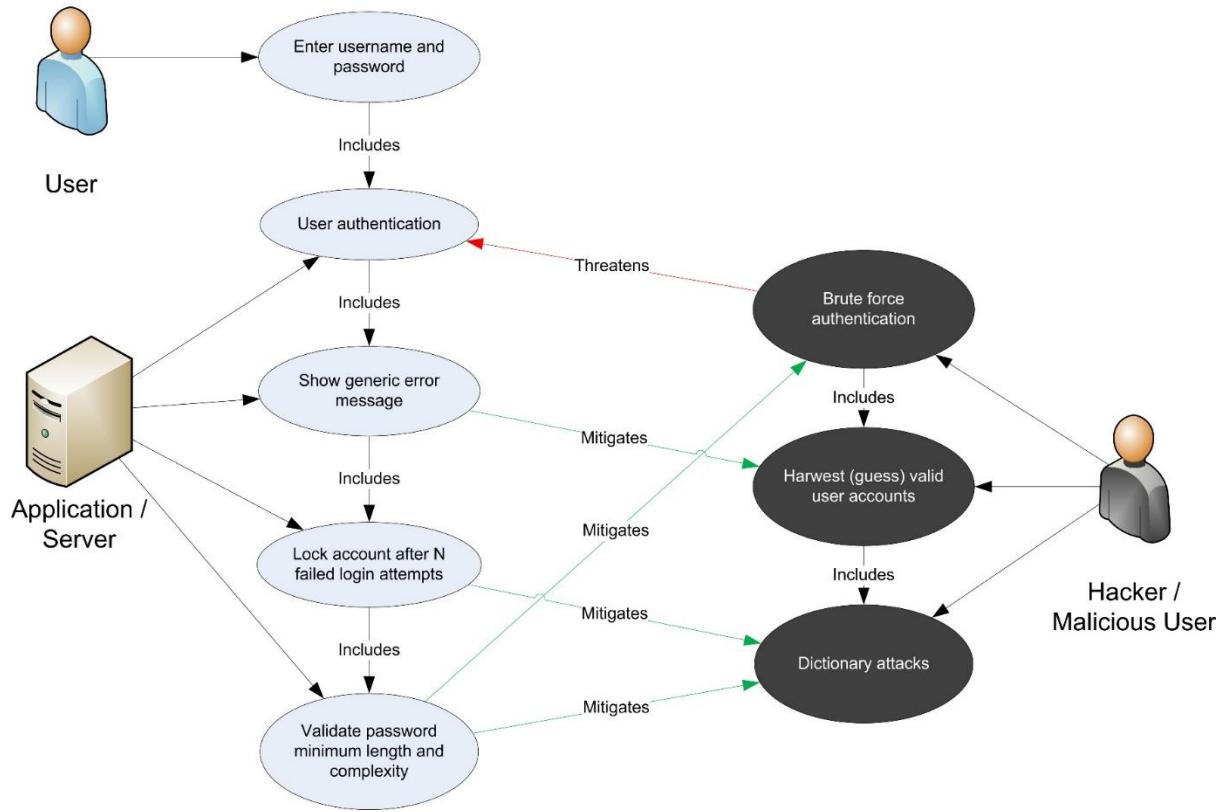
---

- **Related Vulnerabilities or Weaknesses:** : CWE-OS Command Injection, CVE-1999-0043, CVE-1999-0067, CVE-1999-0097, CVE-1999-0152, CVE-1999-0210, CVE-1999-0260, 1999-0262, CVE-1999-0279, CVE-1999-0365, etc.
- **Method of Attack:** By injecting other shell commands into other data that are passed directly into a shell command.
- **Attack Motivation-Consequences:** Execution of arbitrary code.
- **Attacker Skill or Knowledge Required:** Finding and exploiting this vulnerability does not require much skill.
- **Resources Required:** No special or extensive resources are required for this attack.
- **Solutions and Mitigations:** Define valid inputs to all fields and ensure that the user input is always valid. Also perform white-list and/or black-list filtering as a backup to filter out known command delimiters.
- **Context Description:** OS: UNIX.
- **References:** Exploiting Software [Hoglund 04].

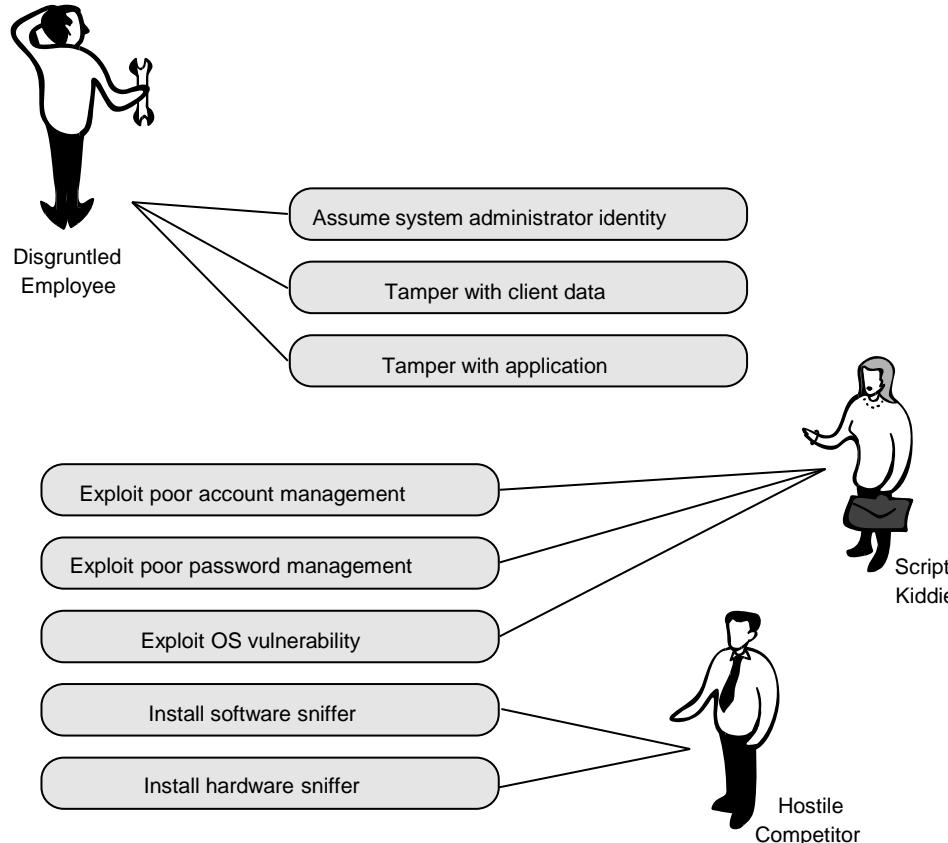
# A Threat Tree Example



# Use Case with Abuse Case



# Abuse case example



# Sample: Perform Risk Assessment

There are two essential assets in this system.

The first is the Windows Server computer, which houses the majority of the production system's intellectual assets (that is, the code that runs the system). This computer acts as a server that allows remote users to access the Asset Management System.

The second essential asset is the information inside the Windows Server computer—specifically, the files stored in the Microsoft IIS server and the information stored in the Sybase database and MapGuide database are critical for making informed decisions. If this information is lost or compromised, the ability to make accurate decisions is lost.

# Elicitation Methods

---

- Misuse/Abuse cases:
  - Misuse/abuse cases apply the concept of a negative scenario—that is, a situation that the system's owner does *not* want to occur—in a use-case context. Business leaders, military planners, and game players are familiar with the strategy of analyzing their opponents' best moves as identifiable threats
- QFD
  - As per Dr. Yoji Akao, who originally developed Quality Function Deployment (QFD) in Japan in 1966, it is a “method to transform qualitative user demands into quantitative parameters, to deploy the functions forming quality, and to deploy methods for achieving the design quality into subsystems and component parts, and ultimately to specific elements of the process”
- Joint Application Development (JAD)
  - The JAD methodology [Wood 1995] involves all stakeholders via highly structured and focused meetings. In the preliminary phases of JAD, the requirements engineering team is charged with fact-finding and information-gathering tasks. Typically, the outputs of this phase, as applied to security requirements elicitation, are security goals and artifacts. The actual JAD session is then used to validate this information by establishing an agreed-on set of security requirements for the product.

## Sample: Elicit and Categorize Security Requirements

Security requirements are identified and then organized to map to the high-level security goals (from Step 2).

Examples include :

- Requirement 1: The system is required to have strong authentication measures in place at all system gateways and entrance points (maps to Goals 1 and 2).
- Requirement 2: The system is required to have sufficient means to govern which system elements (e.g., data, functionality) users can view, modify, and/or interact with (maps to Goals 1 and 2).
- Requirement 3: A continuity of operations plan (COOP) is required to assure system availability (maps to Goal 3).

# Incorporating SQUARE in SDLC

---

- All nine steps of SQUARE fall under the requirements analysis and specification phase.
- The software requirements specification (SRS) should accommodate the outcome of the first eight SQUARE steps
  - The SRS must clearly specify the security definitions agreed on (Step 1). It is necessary to document security goals (Step 2) along with the project goals and constraints. Develop artifacts (Step 3) such as misuse cases and scenarios to support security requirements definition
  - Categorize the security requirements (Step 7) and prioritize them (Step 8) along with documented functional requirements. (For clarity, it is preferable to separate security requirements from functional requirements.)



# OWASP Recommendations

## RL 4.3.1

# OWASP SAMM

---

SAMM (Security Assessment Maturity Model) divides activities associated with software development into 4 business functions for incorporating security

- Governance
  - Includes strategy, metrics, policy, compliance, education and guidance
- Construction
  - Includes threat assessment, security requirements, and security architecture
- Verification
  - Includes design review, implementation review, and security testing
- Operations
  - Includes issue management, environment hardening, operational enablement

# SAMM Security Requirements

---

SAMM expects the following as part of security requirements:

- Consider security explicitly during the software requirements process
- Increase granularity of security requirements derived from business logic and known risks.
- Mandate security requirements process for all software projects and third-party dependencies.

# Consider security explicitly

(during the software requirements process)

Objective	Activities	Assessment	Results
Consider security explicitly during the software requirements process.	<ul style="list-style-type: none"><li>Derive security requirements from business functionality</li><li>Evaluate security and compliance guidance for requirements</li></ul>	<ul style="list-style-type: none"><li>Do project teams specify security requirements during development?</li><li>Do project teams pull requirements from best practices and compliance guidance?</li></ul>	<ul style="list-style-type: none"><li>High-level alignment of development effort with business risks</li><li>Ad hoc capturing of industry best-practices for security as explicit requirements</li><li>Awareness amongst stakeholders of measures being taken to mitigate risk from software</li></ul>

# Increase granularity

(of security requirements derived from business logic and known risks)

Objective	Activities	Assessment	Results
Increase granularity of security requirements derived from business logic and known risks.	<ul style="list-style-type: none"><li>• Build an access control matrix for resources and capabilities (e.g. For data resources, it will be in terms of creation, read, update, and deletion)</li><li>• Specify security requirements based on known risks</li></ul>	<ul style="list-style-type: none"><li>• Do stakeholders review access control matrices for relevant projects?</li><li>• Do project teams specify requirements based on feedback from other security activities?</li></ul>	<ul style="list-style-type: none"><li>• Detailed understanding of attack scenarios against business logic</li><li>• Prioritized development effort for security features based on likely attacks</li><li>• More educated decision-making for tradeoffs between features and security efforts</li><li>• Stakeholders that can better avoid functional requirements that inherently have security flaws</li></ul>

# Mandate security requirements process

(for all software projects and third-party dependencies)

Objective	Activities	Assessment	Results
Mandate security requirements process for all software projects and third-party dependencies.	<ul style="list-style-type: none"><li>• A. Build security requirements into supplier agreements</li><li>• B. Expand audit program for security requirements</li></ul>	<ul style="list-style-type: none"><li>• Do stakeholders review vendor agreements for security requirements?</li><li>• Are audits performed against the security requirements specified by project teams?</li></ul>	<ul style="list-style-type: none"><li>• Formally set baseline for security expectations from external code</li><li>• Centralized information on security effort undertaken by each project team</li><li>• Ability to align resources to projects based on application risk and desired security requirements</li></ul>

# Assessment Matrix for Security Requirements

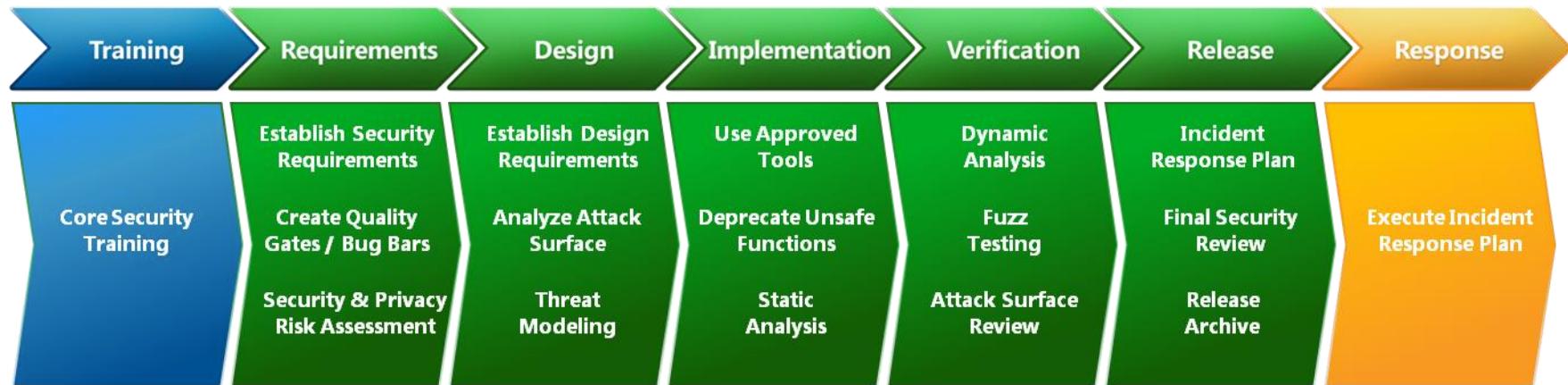
Score ->	0.0	0.2	0.5	1.0
Do project teams specify security requirements during development?	No	Some	Half	Most
Do project teams pull requirements from best practices and compliance guidance?	No	Per Team	Org wide	Integrated Process
Do stakeholders review access control matrices for relevant projects?	No	Some	Half	Most
Do project teams specify requirements based on feedback from other security activities?	No	Some	Half	Most
Do stakeholders review vendor agreements for security requirements?	No	Some	Half	Most
Are audits performed against the security requirements specified by project teams?	No	Once	Every 2-3 years	Annual



# **SDL Recommendations**

## **RL 4.3.2**

# Security Development Lifecycle (SDL)



# SDL Requirements Practices

---

The major SDL practices during requirements analysis are

- Establish Security & Privacy Requirements
- Create Quality Gates/Bug Bars
- Create Security/Privacy Risk Assessments

# Establish Security & Privacy Requirements

Define and integrate security and privacy requirements early

- identify key milestones and deliverables and minimize disruptions to plans and schedules.

Security and privacy analysis including

- assigning security experts,
- defining minimum security and privacy criteria for an application, and
- deploying a security vulnerability/work item tracking system.

When should this practice be implemented?

- Traditional Software development: Requirements Phase
- Agile development: One Time

# Create Quality Gates/Bug Bars

---

Defining minimum acceptable levels of security and privacy quality

- the team understand risks associated with security issues,
- Team identifies and fixes security bugs during development, and
- Team apply the standards throughout the entire project.

Set a bug bar to clearly define the severity thresholds of security vulnerabilities

- (for example, no known vulnerabilities in the application with a “critical” or “important” rating at time of release)

When should this practice be implemented?

- Traditional Software development: Requirements Phase
- Agile development: One Time

# Security/Privacy Risk Assessments

Identify portions of a project requiring threat modeling and security design reviews before release

Determine the Privacy Impact Rating of a feature, product, or service

When should this practice be implemented?

- Traditional Software development: Requirements Phase
- Agile development: One Time

Software Security Engineering, Julia H. Allen, et al, Pearson, 2008.

Security in Computing by Charles P. Pfleeger, Shari L. Pfleeger, and Deven Shah  
Pearson Education 2009

Computer Security: Principles and Practice by William Stallings, and Lawrie Brown  
Pearson, 2008.

[www.owasp.com](http://www.owasp.com)

[www.microsoft.com](http://www.microsoft.com)

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **Secure Architecture & Design - Introduction**

## **RL 5.1.1**

# Nomenclature (SWEBOK)

---

Software design is the activity that uses software requirements to produce a description of the software's internal structure that will serve as the basis for its construction

Software design consists of two activities :

- Software architectural design (sometimes called high-level design): develops top-level structure and organization of the software and identifies the various components.
- Software detailed design: specifies each component in sufficient detail to facilitate its construction.

# Understanding Software Architecture

The software architecture of a program or computing system is the structure or structures of the system which comprise

- The software components
- The externally visible properties of those components
- The relationships among the components

Software architectural design represents the structure of the data and program components that are required to build a computer-based system

An architectural design model is transferable

- It can be applied to the design of other systems
- It represents a set of abstractions that enable software engineers to describe architecture in predictable ways

# Importance of Software Architecture

Representations of software architecture are an enabler for communication between all stakeholders interested in the development of a computer-based system

The software architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity

The software architecture constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together

# Uses of software architecture descriptions

**Reuse:** Architecture descriptions can help software reuse. The software engineering world has, for a long time, been working towards a discipline where software can be assembled from parts that are developed by different people and are available for others to use.

**Construction and Evolution:** As architecture partitions the system into parts, some architecture provided partitioning can naturally be used for constructing the system, which also requires that the system be broken into parts such that different teams (or individuals) can separately work on different parts.

**Analysis:** It is highly desirable if some important properties about the behaviour of the system can be determined before the system is actually built. This will allow the designers to consider alternatives and select the one that will best suit the needs.

# General Objectives of Software Architecture and Design

---

## Completeness

- Supports the full scope of the defined requirements

## Stability

- Consistently performs as intended within its defined operational context

## Flexibility

- Can adapt to changing conditions
- Decompose such that selected components can be replaced going forward with minimal impact to the software

## Extensibility

- Leverages industry standards
- Long-lived and resistant to obsolescence

## Scalability

- Operates effectively at any size and load

# Security-Specific Objectives of Software Architecture and Design

---

## Comprehensive functional security architecture

- Security features and capabilities are fully enabled

## Attack resistance

- Contains minimal security weaknesses that could be exploited

## Attack tolerance

- While resisting attack, software function and capability are not unduly affected

## Attack resilience

- In the face of successful attack, the effects on the software are minimized
- Operates effectively at any size and load

# How to Design

---

A designer must practice diversification and convergence -[Belady]

- The designer selects from design components, component solutions, and knowledge available through catalogs, textbooks, and experience
- The designer then chooses the elements from this collection that meet the requirements defined by requirements engineering and analysis modeling
- Convergence occurs as alternatives are considered and rejected until one particular configuration of components is chosen

Software design is an iterative process

- As design iteration occurs, refinements lead to design representations at lower levels of abstraction

# Architectural Issues

---

Security architecture (the architecture of security components, e.g. firewall, encryption mechanism) is not same as secure architecture (i.e. resilient and resistant to attacks)

Secure architecture not only must address known weaknesses and attacks, but must be flexible and resilient under changing security conditions

Architects (and designers) must focus on minimizing the risk profile. It requires complex and diverse knowledge (both on threats and technologies).

# Architectural Risk Analysis

---

- The risk assessment methodology (Build Security In) encompasses six fundamental activity stages:
  - application characterization
  - architectural vulnerability assessment
  - threat analysis
  - risk likelihood determination
  - risk impact determination
  - risk mitigation

<https://buildsecurityin.us-cert.gov/articles/best-practices/architectural-risk-analysis/architectural-risk-analysis>

# Application Characterization

---

- Assessing the architectural risks for a software system is easier when the boundaries of the software system are identified, along with the resources, integration points, and information that constitute the system
- The artifacts required for review:
  - software business case
  - functional and non-functional requirements
  - enterprise architecture requirements
  - use case documents
  - misuse and abuse case documents
  - software architecture documents describing logical, physical, and process views
  - data architecture documents
  - detailed design documents such as UML diagrams that show behavioral and structural aspects of the system
  - software development plan
  - transactions
  - security architecture documents
  - identity services and management architecture documents
  - quality assurance plan
  - test plan/acceptance plan
  - risk list / risk management plan
  - problem resolution plan
  - issues list
  - project metrics
  - programming guidelines
  - configuration and change management plan
  - project management plan
  - disaster recovery plan
  - system logs
  - operational guides

# Architectural Risk Analysis

---

- Architectural risk analysis examines the preconditions that must be present for vulnerabilities to be exploited and assesses the states that the system may enter upon exploitation.
  - assess vulnerabilities not just at a component or function level, but also at interaction points
  - risk analysis testing can only prove the presence, not the absence, of flaws
- Three activities can guide architectural risk analysis:
  - known vulnerability analysis,
  - ambiguity analysis, and
  - underlying platform vulnerability analysis

# Known Vulnerability Analysis

---

- Consider the architecture against a body of known bad practices or known good principles for confidentiality, integrity, and availability
  - e.g., the good principle of "*least privilege*" prescribes that all software operations should be performed with the least possible privilege.
  - Diagram the system's major modules, classes, or subsystems and circle areas of high privilege versus areas of low privilege. Consider the boundaries between these areas and the kinds of communications across those boundaries.

# Ambiguity Analysis

---

- Ambiguity can be a source of vulnerabilities when it exists between requirements or specifications and development.
  - Note places where the requirements are ambiguously stated and the implementation and architecture either disagree or fail to resolve the ambiguity.
    - e.g. , a requirement for a web application might state that an administrator can lock an account and the user can no longer log in while the account remains locked. What about sessions for that user that are actively in use at the time the administrator locks the account? Is the user suddenly and forcibly logged out, or is the active session still valid until the user logs out?

# Underlying Platform Vulnerability Analysis

- Carry out analysis of the vulnerabilities associated with the application's execution environment including operating system vulnerabilities, network vulnerabilities, platform vulnerabilities, and interaction vulnerabilities resulting from the interaction of components.
- There are web sites that aggregate vulnerability information. These sites and lists should be consulted regularly to keep the vulnerability list current for a given architecture.



# **Principles for Secure Design**

## **RL 5.2**

# Fundamental Design Concepts

---

Abstraction—data, procedure, control

Patterns—"conveys the essence" of a proven design solution

Separation of concerns—any complex problem can be more easily handled if it is subdivided into pieces

Modularity—compartmentalization of data and function

Information Hiding—controlled interfaces

Functional independence—single-minded function and low coupling

Refinement—elaboration of detail for all abstractions

Aspects—a mechanism for understanding how global requirements affect design

Refactoring—a reorganization technique that simplifies the design

*The beginning of wisdom (for a software engineer) is to recognize the difference between getting program to work, and getting it right*  
— M A Jackson

# Design Principles for Software Security

---

- Securing the Weakest Link
- Defense in Depth
- Failing Securely
- Least Privilege
- Separation of Privilege
- Economy of Mechanism
- Least Common Mechanism
- Reluctance to Trust
- Never Assuming that your Secrets are Safe
- Complete Mediation
- Psychological Acceptability
- Promoting Privacy

<https://buildsecurityin.us-cert.gov/articles/knowledge/principles/design-principles>

# Securing the Weakest Link

---

- A software security system is only as secure as its weakest component
- Some cryptographic algorithms can take many years to break, but the endpoints of communication (e.g., servers) may be much easier to attack.
- Attackers don't attack a firewall unless there's a well-known vulnerability in the firewall itself (something all too common, unfortunately). they'll try to break the applications that are visible through the firewall, since these applications tend to be much easier targets
- Sometimes it's not the software that is the weakest link in your system; e.g., consider social engineering, an attack in which a bad guy uses social manipulation to break into a system

# Defense in depth

---

- Layered security mechanisms increase security of the system as a whole
- If an attack causes one security mechanism to fail, other mechanisms may still provide the necessary security to protect the system
- Implementing a defense-in-depth strategy can add to the complexity of an application, that might bring new risks with it
  - e.g., increasing the required password length from eight characters to 15 characters may result in users writing their passwords down, thus decreasing the overall security to the system
  - however, adding a smart-card requirement to authenticate to the application would add a complementary layer to the authentication process & can be beneficial.

# Failing Securely

---

When a system fails, it should do so securely.

- e.g. on failure undo changes and restore to a secure state; always check return values for failure; and in conditional code/filters make sure that there is a default case that does the right thing. The confidentiality and integrity of a system should remain even though availability has been lost.

```
DWORD dwRet = IsAccessAllowed(...);  
if (dwRet == ERROR_ACCESS_DENIED) {  
    // Security check failed.  
    // Inform user that access is denied.  
} else {  
    // Security check OK.  
}
```

```
DWORD dwRet = IsAccessAllowed(...);  
if (dwRet == NO_ERROR) {  
    // Secure check OK.  
    // Perform task.  
} else {  
    // Security check failed.  
    // Inform user that access is denied.  
}
```

# Least Privilege

---

- Only the minimum necessary rights should be assigned to a subject that requests access to a resource and should be in effect for the shortest duration necessary (remember to relinquish privileges).
- According to Saltzer and Schroeder [Saltzer 75] in "Basic Principles of Information Protection," Every program and every user of the system should operate using the least set of privileges necessary to complete the job. if a question arises related to misuse of a privilege, the number of things that must be audited is minimized.
  - a programmer who may need to read some sort of data object, but assigns higher privilege, since "Someday I might need to write to this object, and it would suck to have to go back and change this request."

# Separation of Privilege

---

- A system should ensure that multiple conditions are met before granting permissions to an object. If an attacker is able to obtain one privilege but not a second, he or she may not be able to launch a successful attack.
- a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key
  - This principle is often used in bank safe-deposit boxes. It is also at work in the defense system that fires a nuclear weapon only if two different people both give the correct command.

# Economy of Mechanism

---

- If the design, implementation, or security mechanisms are highly complex, then the likelihood of security vulnerabilities increases. Subtle problems in complex systems may be difficult to find, especially in copious amounts of code.
- Simplifying design or code is not always easy, but developers should strive for implementing simpler systems when possible.
- The checking and testing process is less complex, because fewer components and cases need to be tested.
- Complex mechanisms often make assumptions about the system and environment in which they run. If these assumptions are incorrect, security problems may result.

# Least Common Mechanism

---

- Avoid having multiple subjects sharing mechanisms to grant access to a resource. For e.g., serving an application on the Internet allows both attackers and users to gain access to the application.
- Every shared mechanism (especially one involving shared variables) represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security.
- Example : A web site provides electronic commerce services for a major company. Attackers flood the site with messages, and tie up the electronic commerce services. Legitimate customers are unable to access the web site and, as a result, take their business elsewhere.
  - Here, the sharing of the Internet with the attackers' sites caused the attack to succeed. The appropriate countermeasure would be include proxy servers or traffic throttling. The former targets suspect connections; the latter reduces load on the relevant segment of the network indiscriminately.

# Reluctance to Trust

---

- Developers should assume that the environment in which their system resides is insecure
- Trust in external systems, code, people, etc., should always be closely held and never loosely given.
- software engineers should anticipate malformed input from unknown users
- users are susceptible to social engineering attacks, making them potential threats to a system
- no system is one hundred percent secure, so the interface between two systems should be secured.

# Reluctance to Trust

---

- Point to remember is that trust is transitive. Once you dole out some trust, you often implicitly extend it to anyone the trusted entity may trust
- Hiding secrets in client code is risky. talented end users will be able to abuse the client and steal all its secrets
- According to Viega and McGraw, there are hundreds of products from security vendors with gaping security holes; Many security products introduce more risk than they address
  - Beware of vendors who resort to technobabble using newly invented terms or trademarked terms without actually explaining how the system works
  - Avoid software which uses secret algorithms. ``hackers'' can reverse-engineer the program to see how it works anyway
- According to Bishop, an entity is trustworthy if there is sufficient credible evidence leading one to believe that the system will meet a set of given requirements. Trust is a measure of trustworthiness, relying on the evidence provided. These definitions emphasize that calling something "trusted" or "trustworthy" does not make it so

# Never Assuming That Your Secrets Are Safe

Relying on an obscure design or implementation does not guarantee that a system is secured. You should always assume that an attacker can obtain enough information about your system to launch an attack.

- Tools such as decompilers and disassemblers allow attackers to obtain sensitive information that may be stored in binary files.
- According to Viega and McGraw, for years, there was an arms race and an associated escalation in techniques of vendors and hackers; vendors would try harder to keep people from finding the secrets to "unlock" software, and the software crackers would try harder to break the software. For the most part, the crackers won.
- According to Viega and McGraw, the most common threat to companies is the insider attack; but many companies say "That won't happen to us; we trust our employees." The infamous FBI spy Richard P. Hanssen carried out the ultimate insider attack against U.S. classified networks for over 15 years.

# Complete Mediation

---

A software system that requires access checks to an object each time a subject requests access, especially for security-critical objects, decreases the chances of mistakenly giving elevated permissions to that subject.

- A system that checks the subject's permissions to an object only once can invite attackers to exploit that system.
- According to Bishop, When a UNIX process tries to read a file, the operating system determines if the process is allowed to read the file. If so, the process receives a file descriptor encoding the allowed access. Whenever the process wants to read the file, it presents the file descriptor to the kernel. The kernel then allows the access. If the owner of the file disallows the process permission to read the file after the file descriptor is issued, the kernel still allows access. This scheme violates the principle of complete mediation, because the second access is not checked. The cached value is used, resulting in the denial of access being ineffective.

# Psychological Acceptability

---

- Accessibility to resources should not be inhibited by security mechanisms. If security mechanisms hinder the usability or accessibility of resources, then users may opt to turn off those mechanisms.
  - Where possible, security mechanisms should be transparent to the users of the system or at most introduce minimal obstruction. Security mechanisms should be user friendly to facilitate their use and understanding in a software application.
- Configuring and executing a program should be as easy and as intuitive as possible, and any output should be clear, direct, and useful.
  - If security-related software is too complicated to configure, system administrators may unintentionally set up the software in a non-secure manner.
- Similarly, security-related user programs must be easy to use and output understandable messages.

# Promoting Privacy

---

- Protecting software systems from attackers that may obtain private information is an important part of software security.
- Many users consider privacy a security concern. Try not to do anything that might compromise the privacy of the user.



# Secure Architectural Patterns

## RL 5.3.1

# Secure Patterns

---

- A software design pattern is a general repeatable solution to a recurring software engineering problem.
- Secure design patterns a general solution to a security problem that can be applied in many different situations.
- Secure design patterns are not restricted to object-oriented design approaches but may also be applied, in many cases, to procedural languages.

# Secure Architectural-level Patterns

Architectural-level patterns focus on the high-level allocation of responsibilities between different components of the system and define the interaction between those high-level components.

- Architectural-level Patterns
  - Distrustful Decomposition
  - Privilege Separation (PrivSep)
  - Defer to Kernel

# Distrustful Decomposition

---

Separate the functionality of your software into *mutually untrusting chunks*, so as to shrink the attack windows into each chunk

- Design each chunk under the assumption that other software chunks with which it interacts have been attacked, and it is attacker software rather than normal application software that is running in those interacting chunks.
- Do not expose your data to other chunks via shared memory.

As a result of mutually untrusting chunking, your entire system will not be given into the hands of an attacker if any one of its chunks has been compromised

# Distrustful Decomposition (cont..)

**Motivation** : Many attacks target vulnerable applications running with elevated permissions

Some examples of this class of attack are

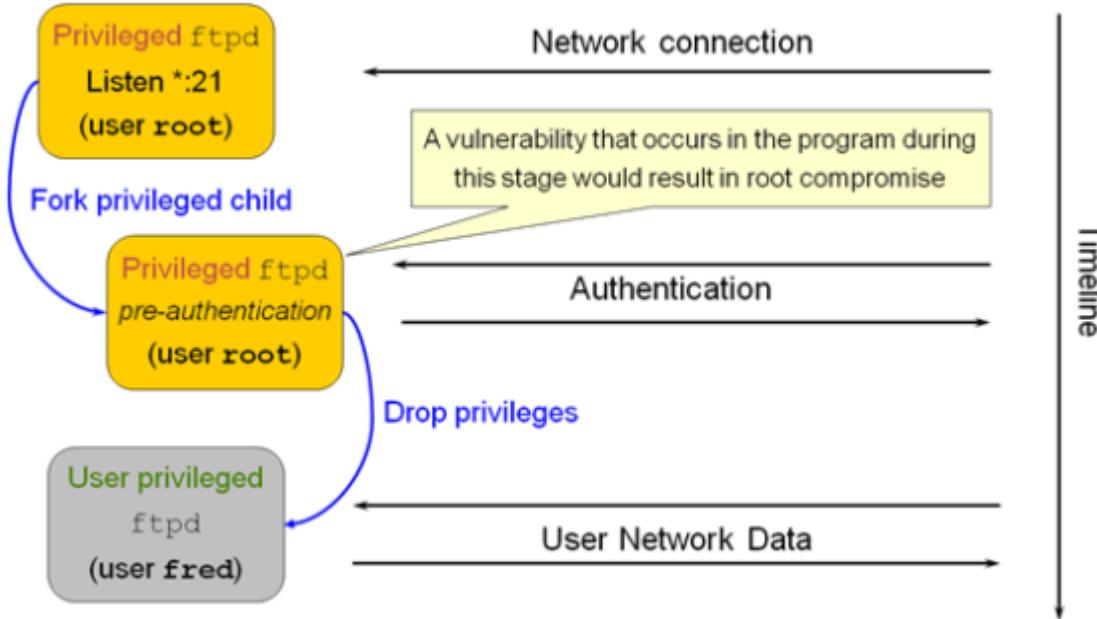
- Attacks in which versions of IE browser running in an account with administrator privileges is compromised
- Security flaws in Norton AntiVirus 2005 that allowed attackers to run arbitrary VBS scripts when running with administrator privileges
- A buffer overflow vulnerability in BSD-derived telnet daemons that allows an attacker to run arbitrary code as root

**Consequences** : Prevents an attacker from compromising an entire system in the event that a single component program is successfully exploited because no other program trusts the results from the compromised one

# Privilege Separation (PrivSep)

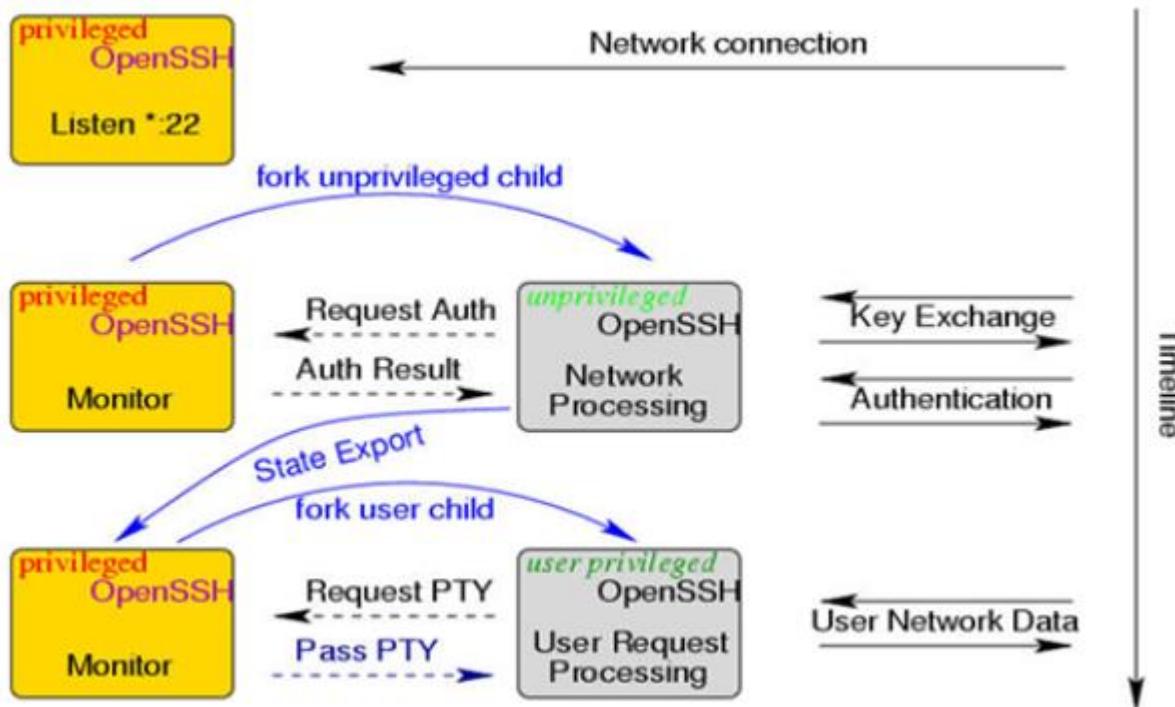
- The PrivSep pattern is a more specific instance of the Distrustful Decomposition pattern.
- Keep to a minimum the part of your code that executes with special privilege.
- If an attacker succeeds in breaking into software that's running at a high level of privilege, the attacker will be operating at a high level of privilege too. That'll give him an extra-wide open "attack window" into your system
- The pattern is applicable if the system performs a set of functions do *not* require elevated privileges, but have relatively large attack surfaces (e.g. communication with untrusted sources, potentially error-prone algorithms)

# Privilege Separation (cont..)



Here is a vulnerable implementation where a privileged process is trying to authenticate an unauthenticated user

# Privilege Separation (cont..)



- The implementation as per PrivSep pattern.
- The interactions with user and authentication are moved into an unprivileged process.

# Defer to Kernel

---

The intent of this pattern is to clearly separate “functionality that requires elevated privileges” from “functionality that does not require elevated privileges and to take advantage of existing user verification functionality available at the kernel level.

Designers tend to take control of authorization functionality into their hands. The pattern discourages the tendency

The pattern is applicable to systems:

- That run by users who do not have elevated privileges;
- Where some (possibly all) of the functionality of the system requires elevated privileges; or
- Where the system must verify that the current user is authorized to execute any functionality that requires elevated privileges



# Secure Design Patterns

## RL 5.3.2

# Classes of Patterns

---

**Design-level patterns.** Design-level patterns describe how to design and implement pieces of a high-level system component, that is, they address problems in the internal design of a single high-level component, not the definition and interaction of high-level components themselves.

- Secure Factory
- Secure Strategy Factory
- Secure Builder Factory
- Secure Chain of Responsibility
- Secure State Machine
- Secure Visitor

# Classes of Patterns

---

**Implementation-level patterns.** Implementation-level patterns address low-level security issues. Patterns in this class are usually applicable to the implementation of specific functions or methods in the system. Implementation-level patterns address the same problem set addressed by the CERT Secure Coding Standards

- Secure Logger
- Clear Sensitive Information
- Secure Directory
- Input Validation

# Secure Factory

---

- Secure Factory secure design pattern is a security specific extension of the Abstract Factory pattern
- The Secure Factory secure design pattern is applicable if
  - The system constructs different versions of an object based on the security credentials of a user/operating environment.
  - The available security credentials contain all of the information needed to select and construct the correct object.

# Secure Strategy Pattern

---

- The strategy pattern enables an algorithm's behavior to be selected at runtime
- Strategy pattern provides a means to define a family of algorithms, encapsulate each one as an object, and make them interchangeable during runtime
- a class that performs validation on incoming data may use a strategy pattern to select a validation algorithm based on the type of data, the source of the data, user choice, or other discriminating factors
- The secure strategy object performs a task based on the security credentials of a user or environment

# Secure Builder Factory

---

- Secure Builder Factory design pattern is to separate the security dependent rules, involved in creating a complex object, from the basic steps involved in the actual creation of the object.
  - Identify a complex object whose construction depends on the level of trust associated with a user or operating environment. Define the general builder interface using the Builder pattern for building complex objects of this type.
  - Implement the concrete builder classes that implement the various trust level specific construction rules for the complex object.

# Secure Chain of Responsibility

The intent of the Secure Chain of Responsibility pattern is to decouple the logic that determines user/environment-trust dependent functionality from the portion of the application make it relatively easy to dynamically change the user/environment-trust dependent functionality.

## **Motivation**

In an application using a role-based access control mechanism, the behavior of various system functions depends on the role of the current user

## **Consequence**

The security-credential dependent selection of the appropriate specific behavior for a general system function logic is hidden from the portions of the system that make use of the general system function.

# Secure State Machine

---

The intent of the Secure State Machine pattern is to allow a clear separation between security mechanisms and user-level functionality by implementing the security and user-level functionality as two separate state machines.

## Motivation

Intermixing security functionality and typical user-level functionality in the implementation of a secure system can increase the complexity of both. The increased complexity makes it more difficult to test, review, and verify the security properties of the implementation.

## Consequences

- Can test and verify the security mechanisms separately from the user-level functionality
- New security implementation could be implemented with lesser effort

# Secure Visitor

---

Secure systems may need to perform various operations on hierarchically structured data where each node in the data hierarchy may have different access restrictions. The pattern idea is to incorporate security mechanism in data node rather than visitor code.

## Motivation

Secure Visitor pattern allocates all of the security considerations to the nodes in the data hierarchy, leaving developers free to write visitors that only concern themselves with user-level functionality

## Consequences

The use of this pattern requires that the nodes in the data hierarchy, not the visitors themselves, implement security

# Secure Logger

---

- The intent of the Secure Logger pattern is to prevent an attacker from gathering potentially useful information about the system from system logs and to prevent an attacker from hiding their actions by editing system logs.
- The Secure Logger pattern is applicable if
  - The system logs information to a log file or some other form of logging subsystem.
  - The information contained in the system log could be used by an attacker to devise attacks on the system.
  - System logs are used to detect and diagnose attacks on the system.

# Clear Sensitive Information

---

It is possible that sensitive information stored in a reusable resource may be accessed by an unauthorized user or adversary if the sensitive information is not cleared before freeing the reusable resource. The use of this pattern ensures that sensitive information is cleared from reusable resources before the resource may be reused.

Reusable resources include things such as the following:

- dynamically allocated memory
- statically allocated memory
- automatically allocated (stack) memory
- memory caches
- disk
- disk caches

# Secure Directory

---

The intent of the Secure Directory pattern is to ensure that an attacker cannot manipulate the files used by a program *during* the execution of the program.

The Secure Directory pattern is applicable for use in a program if

- The program will be run in an insecure environment; that is, an environment where malicious users could gain access to the file system used by the program.
- The program reads and/or writes files.
- Program execution could be negatively affected if the files read or written by the program were modified by an outside user while the program was running.

The program should check that a directory offered to it is secure, and refuse to use it otherwise. Implementation of the Secure Directory pattern involves the following steps:

- Find the canonical pathname of the directory of the file to be read or written.
- Check to see if the directory, as referenced by the canonical pathname, is secure.
  - If the directory is secure, read or write the file.
  - If the directory is not secure, issue an error and do not read or write the file.

# Input Validation

---

Input validation requires that a developer correctly identify and validate all external inputs from untrusted data sources

## Motivation

- The use of unvalidated user input is the root cause of many serious security exploits, such as buffer overflow attacks, SQL injection attacks, and cross-site scripting attacks.
- In a client-server architecture, it is problematic if only client-side validation is performed. It is easy to spoof a web page submission and bypass any scripting on the original page

## References

Software Security Engineering, Julia H. Allen, et al, Pearson, 2008.

Secure Design Patterns by Chad Dougherty, Kirk Sayre, Robert C. Seacord, David Svoboda, Kazuya Togashi (JPCERT/CC) <https://www.sei.cmu.edu/reports/09tr010.pdf>

[www.swebok.com](http://www.swebok.com)

[www.us-cert.gov/bsi](http://www.us-cert.gov/bsi)

Pressman, R.S., Software Engineering: A Practitioner's Approach, MGHISE, 7th Ed., 2010

Pankaj Jalote , An Integrated Approach to Software Engineering, 3rd Edition , Springer, 2005

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **Security Testing Concepts– Part 1**

## **RL 6.1.1**

# Testing Strategy

---

The strategy provides a road map that describes the steps to be taken, when, and how much effort, time, and resources will be required

A strategy for software testing integrates the design of software test cases into a well-planned series of steps that result in successful development of the software

The strategy incorporates test planning, test case design, test execution, and test result collection and evaluation

Testing begins at the component level and work outward toward the integration of the entire computer-based system

Different testing techniques are appropriate at different points in time

# Software Testing Axioms

---

- It is impossible to test a program completely.
- Software testing is a risk-based exercise.
- Testing cannot show the absence of bugs.
  - The Pesticide Paradox
    - In 1990, Boris Beizer, coined the term *pesticide paradox* to describe the phenomenon that the more you test software, the more immune it becomes to your tests
- The more bugs you find, the more bugs there are.
- Not all bugs found will be fixed.
- It is difficult to say when a bug is indeed a bug.
- Specifications are never final.
- Software testers are not the most popular members of a project.
- Software testing is a disciplined and technical profession

# Software Testing Key Issues (SWEBOk)

*Dynamic:* The input value alone is not always sufficient to specify a test, since a complex, nondeterministic system might react to the same input with different behaviors, depending on the system state.

*Finite:* Even in simple programs, so many test cases are theoretically possible that exhaustive testing could require months or years to execute.

*Selected:* How to identify the most suitable test set under given conditions is a complex problem; in practice, risk analysis techniques and software engineering expertise are applied.

*Expected:* It must be possible, although not always easy, to decide whether the observed outcomes of program testing are acceptable or not; otherwise, the testing effort is useless.

# Security Testing (SWEBOK)

---

- Security testing is focused on the verification that the software is protected from external attacks.
- Security testing verifies the confidentiality, integrity, and availability of the systems and its data.
- Security testing includes verification against misuse and abuse of the software or system (negative testing).

# Security Testing Myth & Reality

---

Myth :

In the security industry people frequently test against a set of mental criteria that are neither well defined nor complete. As a result of this, many outsiders regard security testing as a black art.

Reality :

It is possible for people without in-depth security knowledge to make impactful security testing.

# Penetrate and Patch

---

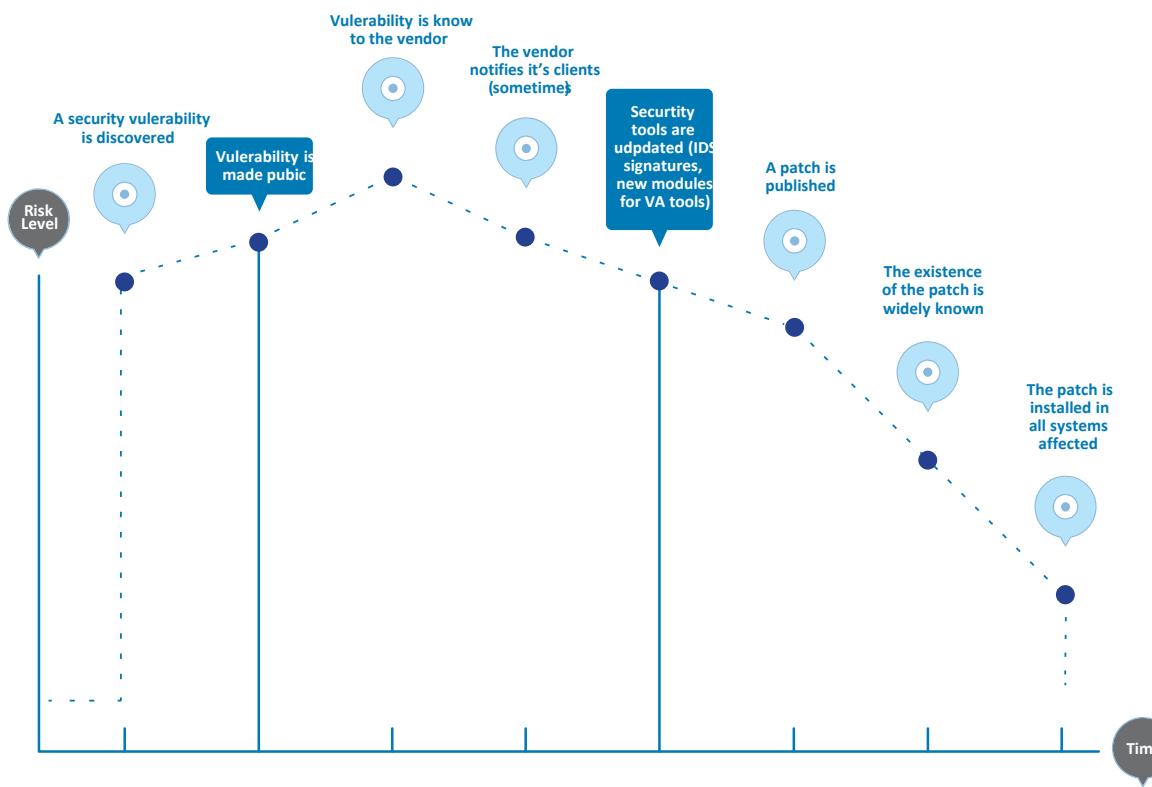
A number of software tools exist to support *post facto* security analysis of installed computer systems. Examples include

- Network scanning tool is SATAN.
- Internet Security Scanner (ISS) scans network ports and attempts to find and exploit known vulnerabilities.
- The Computer Oracle and Password System (COPS) is a collection of programs to detect different problem areas in Unix security.

Such tools are reactive.

- Security analysis needs to be performed as part of the software development process, before software is released.
- Crackers often know about vulnerabilities before system administrators
- System administrators have neither the time nor the inclination to patch software if they have not noticed any security breaches.
- While a patch may close one security hole, it may simultaneously open up others.

# Window of Vulnerability





# **Source Code Analyzers**

## **RL 6.1.2**

# Source Code Analysis Tools

---

Source Code Analysis Tools (security analyzers) are automated tools for helping analysts find security-related problems in software

- use data- and control-flow analysis to find subtler bugs and to reduce false alarms

Some vulnerabilities (e.g. use of strcpy()) can be detected with high accuracy, others are harder to detect, and, in fact, one can always devise vulnerabilities that are undetectable altogether.

Tools have tradeoff between false alarms (also known as false positives) and missed vulnerabilities (also known as false negatives)

- can be configured to make a tool more sensitive (decreasing false negatives while increasing false positives) or make it less sensitive (increasing false negatives while decreasing false positives)

# Capabilities of Security Analyzers

- Examining Calls to Potentially Insecure Library Functions
- Detecting Bounds-Checking Errors and Scalar Type Confusion
- Detecting Type Confusion Among References or Pointers
- Detecting Memory Allocation Errors
- Detecting Vulnerabilities that Involve Sequences of Operations (Control-Flow Analysis)
- Data-Flow Analysis
- Pointer-Aliasing Analysis
- ...

## Check Calls to Potentially Insecure Library Functions

This security-scanning capability can encompass several components:

- **A database of vulnerable library calls** is the heart of security scanning technology. The vulnerability database must be up to date, and would have to be constantly updated as well to remain relevant.
- **The ability to preprocess source code** is important for C/C++ analyzers, because it lets the analyzer see the same code that will be seen by the compiler. Without this capability there are numerous ways to deceive the analyzer. Many analyzers use heuristics to approximate the functionality of a preprocessor.
- **Lexical analysis** is the process of breaking a program into tokens prior to parsing. Lexical analysis is necessary to reliably distinguish variables from functions and to identify function arguments. These functions can also be performed with heuristics—at the cost of some reliability, however.

# Detecting Bounds-Checking Errors and Scalar Type Confusion

---

Vulnerabilities occur when scalar assignments transparently *change* the value being assigned e.g.

- integer overflow: an integer variable overflows and becomes negative
- integer truncation: an integer value is truncated while being cast to a data type with fewer digits
- unsigned underflow: an unsigned integer value underflows and becomes large

one of these issues results in a vulnerability typically because the affected variable gives the size of a buffer

# Detecting Type Confusion Among References or Pointers

---

Type confusion with pointers or references is a common source of bugs and may result in vulnerabilities unless the type confusion is detected at runtime

- In some cases, static type checking can identify reference type confusion
- Usually fail with a cast between incompatible types having a common superclass allowing (for example) methods written for one data type to be applied to a different data type leading to vulnerabilities

# Detecting Memory Allocation Errors

---

- Memory corruption vulnerabilities can vary from one operating system to the next because the operating systems use different techniques for heap maintenance
- Heap corruption can arise if an attacker is able to overwrite information used to maintain the heap
- A number of circumstances can allow an attacker to corrupt this information. e.g.
  - a buffer overflow in an allocated chunk of memory
  - a double free.
  - a write to freed memory.

# Vulnerabilities Involving Sequences of Operations (Control-Flow Analysis)

---

File accesses by a program can create vulnerabilities if done incorrectly; operations have to be carried out in the right order

- e.g., a C program first obtains a file handle to check certain properties of the file before it can access the file contents
- the mask governing permissions of newly created files must be set explicitly if a new file may be created
- integer ranges have to be checked before being used without any modification taking place between the time of check and time of use.

Security analyzers often look for specific library function calls and print a warning regardless of whether the operation in question is being carried out correctly, which causes noise

Control-flow analysis be used when some potentially dangerous operation must be preceded by precautionary measures, such as closing and reopening standard file descriptors in C before writing to them, or setting default file permissions before creating a new file

- e.g. a linux file descriptor hangs in system folder if not explicitly closed & stays vulnerable

# Data-Flow Analysis

---

- Security analyzers use data-flow analysis primarily to reduce false positives and false negatives, e.g., many buffer overflows in real code are not exploitable because the attacker cannot control the data that overflows the buffer.
- The data-flow analysis that is often used in security-related applications is *taint analysis*.
  - A variable is tainted if its value can be influenced by a potential attacker
  - If a tainted variable is used to compute the value of a second variable, then the second variable also becomes tainted

# Pointer-Aliasing Analysis

---

Pointer aliasing occurs when two pointers point to the same data

- The data that would be found by dereferencing one of the pointers can change even though the source code contains no mention of that pointer.

*Pointer-aliasing analysis* refers to any static technique that tries to solve this problem by tracking which pointers point to what locations

- Related to data flow analysis

Many programming languages allow them to be manipulated in arbitrary ways, for e.g., a loop that increments the value of a pointer until it points to a space character

- Remains a challenging part in static code analysis

# Major weaknesses

---

- Many types of security vulnerabilities are very difficult to find automatically, such as authentication problems, access control issues, insecure use of cryptography, etc. However, tools of this type are getting better.
- High numbers of false positives.
- Frequently can't find configuration issues, since they are not represented in the code.
- Difficult to 'prove' that an identified security issue is an actual vulnerability.
- Many of these tools have difficulty analyzing code that can't be compiled. Analysts frequently can't compile code because they don't have the right libraries, all the compilation instructions, all the code, etc.



# **White Box Security Testing**

## **RL 6.2.1**

# White Box Testing

---

White box testing consists testing with access to the source code

- it is a good practice to perform white box testing during the unit testing phase

White box testing requires knowing what makes software secure or insecure, how to think like an attacker, and how to use different testing tools and techniques.

- First tester comprehends and analyzes available design documentation, source code, and other relevant development artifacts, so knowing what makes software secure is a fundamental requirement.
- Second, tester creates tests that exploit software, a tester must think like an attacker.
- Third, to perform testing effectively, testers need to know the different tools and techniques available for white box testing

# White Box Testing for Security

**Data-Flow Analysis**

**Code-Based Fault Injection**

**Abuse Cases**

**Trust Boundaries Mapping**

**Code Coverage Analysis**

# Data-Flow Analysis

---

- The data-flow testing technique is based on investigating the ways values are associated with variables and the ways that these associations affect the execution of the program
  - focuses on occurrences of variables, following paths from the definition (or initialization) of a variable to its uses
- A data-flow analysis for an entire program involving all variables and traversing all usage paths may require immense computational resources; however, this technique can be applied for select variables
- The path and the usage of the data can help in identifying suspicious code blocks and in developing test cases to validate the runtime behavior of the software.

# Code-Based Fault Injection

---

- The fault injection technique perturbs program states by injecting software source code to force changes into the state of the program as it executes.
  - Consists of non-intrusively inserting code into the software that is being analyzed and then compiling and executing the modified (or instrumented) software
- This technique forces non-normative behavior of the software, and the resulting understanding can help determine whether a program has vulnerabilities that can lead to security violations.
  - can be used to force error conditions to exercise the error handling code,
  - change execution paths,
  - input unexpected (or abnormal) data,
  - change return values, etc.

# Abuse Cases

---

- Abuse cases help security testers view the software under test in the same light as attackers do.
  - can be used to develop innovative and effective test cases mirroring the way attackers would view the system
- The abuse case can also be applied to interactions between components within the system to capture abnormal behavior, should a component misbehave.
- The practical method for creating abuse cases is usually through a process of brainstorming, involving security, reliability, and subject matter expertise
- Known attack patterns help developing abuse cases.

# Trust Boundaries Mapping

---

- Defining zones of varying trust in an application helps identify vulnerable areas of communication and possible attack paths for security violations.
- For systems that have n-tier architecture or that rely on several third-party components, the potential for missing trust validation checks is high, so drawing trust boundaries becomes critical for such systems
- Combining trust zone mapping with data-flow analysis helps identify data that move from one trust zone to another and whether data checkpoints are sufficient to prevent trust elevation possibilities

# Code Coverage Analysis

---

- Code coverage is a way of determining which code statements or paths have been exercised during testing. It is a test effectiveness measurement
- Help in identifying redundant test cases that do not increase coverage and also help in identifying redundant test cases that do not increase coverage
- There are various measures for coverage, such as path coverage, path testing, statement coverage, multiple condition coverage, and function coverage
- Covering all the code paths or statements does not guarantee that the software does not have faults; however, the missed code paths or statements should definitely be inspected.
- Unexercised code has serious bugs that can be leveraged into a successful attack



# **Black Box Security Testing**

## **RL 6.2.2**

# Black Box Testing for Security

Black box tests can help

- identify implementation errors that were not discovered during code reviews, unit tests, or security white box tests
- discover potential security issues resulting from boundary conditions that were difficult to identify and understand during the design and implementation phases
- uncover security issues resulting from incorrect product builds (e.g., old or missing modules/files)
- detect security issues that arise as a result of interaction with underlying environment (e.g., improper configuration files, unhardened OS and applications)

# Black Box Testing Tools

---

Black box test activities almost universally involve the use of tools to help testers identify potential security vulnerabilities

There are tools that focus on specific areas, including

- network security,
- database security,
- security subsystems, and
- web application security

# Network Security Tools

---

- *Network security* based test tools focus on identifying vulnerabilities on externally accessible network-connected devices e.g. firewalls, servers, and routers
- Network security tools generally begin by using a port scanner to identify all active devices connected to the network, services operating on the hosts, and applications running on each identified service
- Some network scanning tools identify specific security vulnerabilities associated with the scanned host based on information contained within a vulnerability database.
- Tools are closely associated with penetration testing

# Database Security Test Tools

---

Tools identify vulnerabilities in a systems database

- incorrect configuration of the database security parameters or
- improper implementation of the business logic used to access the database

Identify vulnerabilities that result in the disclosure or modification of sensitive data in the database

# Security Subsystem Tools

---

- identify security vulnerabilities in specific subsystems
- used to test whether security-critical subsystems have been designed and implemented properly
  - correct operation of random number generators
  - cryptographic processors, and
  - other security-critical components.

# Web Application Security Tool

- highlight security issues within applications accessed via the Internet
- these tools generally focus on identifying vulnerabilities and abnormal behavior within applications available over ports 80 (HTTP) and 443 (HTTPS)
  - These ports are allowed through a firewall to support web servers.
  - these tools may also test Web Services based application technologies over the same ports



# Fuzz Testing RL 6.3.1

# Fuzzing

---

- The term fuzzing is derived from the fuzz utility (of Wisconsin), which is a random character generator for testing applications by injecting random data at their interfaces
- The idea is to look for interesting program behavior that results from noise injection and may indicate the presence of a vulnerability or other software fault.
  - completely random fuzzing is a comparatively ineffective way to uncover problems in an application.
- Fuzzing technology (along with the definition of fuzzing) has evolved to include more intelligent techniques. (Microsoft refers to this as “smart fuzzing”)
  - e.g., fuzzing tools are aware of commonly used Internet protocols, so that testers can selectively choose which parts of the data will be fuzzed.

# Kinds of fuzzing

---

- **Black box**
  - The tool knows nothing about the program or even its inputs. Limited benefits
- **Grammar based**
  - The tool generates input based on known grammar
- **White box**
  - The tool generates new inputs based on the code of the program.  
Computationally complex

# Network-based fuzzing

---

Act as one of the communicating parties

- Inputs could be produced
  - from scratch (e.g., from a protocol grammar)
  - replay of the previously recorded interaction
  - alterations of recorded interactions

Act as a “man in the middle”

- mutate messages exchanged between parties (using the knowledge of grammar)

SPIKE The fuzzer creation kit

- (<http://resources.infosecinstitute.com/intro-to-fuzzing/> )

# File format fuzzing

---

- Instead of just trying really long inputs, a more advanced way to fuzz is to try corner cases in some input format or malformed inputs just outside this input format
  - *CVE-2007-0243 Java JRE GIF Image Processing Buffer Overflow Vulnerability*
    - Critical: Highly critical   Impact: System access   Where: From remote
    - ... caused by an error when processing GIF images and can be exploited to cause a heap-based buffer overflow via a specially crafted GIF image with an image width of 0
  - *Microsoft Security Bulletin MS04-028 Buffer Overrun in JPEG Processing (GDI+)*
    - Could Allow Code Execution
    - Impact: Remote Code Execution   Maximum: Critical
    - ... cause by a zero sized comment field, without content.

# Fuzzing Variations

---

1. Simple (original) fuzzing
    - try out ridiculously long inputs
    - try really long inputs for string arguments to trigger segmentation faults and hence find buffer overflows
      - e.g. register with Facebook with a 1Mbyte long username
  2. Protocol/format/language fuzzing
    - try out strange inputs, given some format/language
  3. State-based fuzzing
    - try out strange sequences of input
- 2 & 3 are essentially forms of model-based testing

# State-based Protocol Fuzzing

---

- Instead of fuzzing the content of individual messages, we can also fuzz the order of messages.
- This is interesting for protocols that have different types of messages, which are expected to come in a particular order:
- This can reveal flaws in the application logic (more specifically, flaws in the implementation of the protocol state machine)
- Essentially this is a form of model-based testing, where we automatically test if an implementation conforms to model (in the form of a finite state machine aka finite automaton), by random test sequences

# Dealing with crashes in Fuzz Testing

- One of the most interesting outputs of fuzz testing come from analysis of crashes
  - What is the **root cause** (so it can be fixed)?
  - Is there a way to **make the input smaller**, so it is more understandable?
  - Are **two or more crashes signaling the same bug**?
  - Does the crash signal an **exploitable vulnerability**?

# Finding errors before crash

---

- **Compile** the program with **Address Sanitizer** (ASAN)
  - <https://github.com/google/sanitizers/wiki/AddressSanitizer>
- ASAN instruments accesses to arrays to check for overflows, and use-after-free errors. (Alarm to be raised in case of deviation/violation)
  - Carry out Fuzz testing
  - Did the execution result in ASAN-signaled error?
  - If so, check for exploitability
- Similarly, it is possible to *compile with other sorts of error checkers* for the purposes of testing

# CERT Basic Fuzzing Framework (BFF)

- The CERT Basic Fuzzing Framework (BFF) is a software testing tool that finds defects in applications that run on the Linux and Mac OS X platforms
  - uses mutational (taking well-formed input data and corrupting it in various ways) fuzzing on software that consumes file input.
  - automatically collects test cases that cause software to crash in unique ways, and associated debugging information
  - helps efficiently discover and analyze security vulnerabilities found via fuzzing
  - Uses machine learning techniques to minimize the manual effort for the fuzzing.
- CERT used the BFF to find a number of critical vulnerabilities in products such as Adobe Reader and Flash Player; Foxit Reader; Apple QuickTime, Preview, and Mac OS X; Xpdf; Poppler; etc.



# **Penetration Testing**

## **RL 6.3.2**

# Background of Penetration Testing

---

- By the 1970s, the US government was regularly using teams to assess the security of computer systems by trying to penetrate them. These teams were referred to as red teams, or tiger teams.
- The penetration testing had been largely focused on the environments:
  - Attempt to compromise the security of the Computer operating systems, along with their access control mechanisms by penetration testing.
  - In a networked computer context, operating system configurations, including their respective network services, are usual targets for penetration tests. These operating system components have offered countless opportunities for penetration testing over the years
- Recently push was for applying penetration testing techniques “up” the software abstraction levels, beyond the operating system and network services, towards the application software itself

# Security is not compositional

---

- According to Leslie Lamport, a Turing Award winner, Security is not compositional.
- Two components that are secure on their own are not necessarily secure when used in combination
  - A change to one component might not break that component, but could break the whole system due to the lack of compositionality
- Upon change to the software, the configuration, the network topology, and so on, potentially new vulnerabilities are created

# An art or a science?

---

- Pen testers must be creative. They think about how a system is put together, and where assumptions made by designers represent weaknesses
- Pen testers will cleverly adapt the weaknesses to gain a foothold in one place. They may use that foothold to exploit a weakness somewhere else.
- Systems could be incorrectly built or misconfigured in the some ways.
- Tools are built to systematically look for weakness patterns, and exploit them.

Thus Pen testing is both an art and a science.

# Penetration Testing Skills

---

- A pen tester needs to know a lot about the target domain.
- For example, if the pen tester is attacking web applications, then the pen tester needs to know how the web works. Need to know how systems are built in that domain.
  - What protocols allow applications to communicate. For the web, that's HTTP and TCP, and IP.
  - The languages that are used to build applications like PHP, Java, or Ruby for talking about the web.
  - Frameworks that used to build applications or application components like, for the web, Ruby on Rails, DreamWeaver, Drupal, and so on.
- Pen tester also need to know common weaknesses from that domain.
  - For example, the bugs that are common to web applications like SQL injections or cross-site scripting, or cross-site request forgery. Or common misconfigurations or bad designs, like the use of default passwords or hidden files.

# Categories of Penetration Testing Tools

- **Host-Based Tools**
- **Network-Based Tools**
- **Application Testing Proxies**
- **Application Scanning Tools**
- Tools integrated with other IT security technologies, viz. firewalls, intrusion detection and prevention systems

# Host-Based Tools

---

- Host-based testing tools test the local operating system to assess its technical strengths and weaknesses, e.g. Dan Farmer's COPS program, which evaluated the security posture of a UNIX computer
  - e.g., evaluate file access control mechanisms for opportunities for attackers to affect the security of the host
  - examine every file, configuration data (including registry keys on Windows systems), installed patch inventory, and so on from the perspective of every ID on the system
  - Check common operating system configuration mistakes and omissions such as dangerous setuid files, unnecessary network services enabled, and excessive privileges for user accounts

# Network-Based Tool

---

- Network-based testing tools assess the security configuration of a computer operating system from afar—across a network, e.g. Chris Klaus's Internet Security Scanner (ISS) program
- Examine a target computer(s) for weaknesses that may be exploitable from a remote networked location using a database of vulnerabilities
  - Advantage- Scale: A huge number of computers can be evaluated across a network;
  - Limitation- Coverage: Network-based testing can only evaluate the externally accessible interfaces

# Nmap for network probing

---

Nmap stands for “network mapper”. Free, open source (commercial versions too)

<http://nmap.org/>

Figures out

- what **hosts** are available on the network,
- what **services** (application name and version) those hosts are offering,
- what **operating systems** (and OS versions) they are running,
- what type of **packet filters/firewalls** are in use
- ... etc.

Works by **sending raw IP packets** into the network and **observing the effects**

- Standard “ping” protocol, Looks for HTTPS(port 443) or HTTP(port 80) servers, - Probes to other TCP ports
  - Probes that elicit different responses on different OSes (“fingerprinting”)

**Can be stealthy!**

- Control the rate of scanning to “work under the radar”

# Application Testing Proxies

---

## Application Testing Proxies

- enable the security tester to look behind the graphical user interface when testing a web application or web service
- Requests to and responses from the server are intercepted, observed, and optionally manipulated

## Web applications are common pen testing targets

- Web proxies sit *between* the browser and server
- Displaying exchanged packets
- Modifying them as directed by the tester

# Application Scanning Tools

---

- These tools do penetration testing scans of general purpose web-based software applications
- connect to web applications and attempt a series of well defined tests for each data field, cookie, etc.
- Such tools initiate a “learning mode” in which they observe the normal operation of a web application. Based on learning, they attempt to exploit common web application defects such as data overruns, SQL injection, and cross-site scripting (XSS)

# Ethical Hacking

---

- Penetration testing tools are meant to reveal security vulnerabilities so that they can be fixed, not so that they can be exploited for the purposes of crime or harm.
- But it is true that people will use these penetration testing tools for nefarious purposes.
- In that way, they are sort of two way tools.
  - Just as guns can be used to defend, guns can be used to attack. Ethical hacking is not to be someone who uses pen testing tools to attack.

Software Security Engineering, Julia H. Allen, et al, Pearson, 2008.

[www.us-cert.gov/bsi](http://www.us-cert.gov/bsi)

[www.swebok.com](http://www.swebok.com)

[www.owasp.com](http://www.owasp.com)

[www.digital.com](http://www.digital.com)

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **Penetration Testing**

## **RL 6.3.2**

# Background of Penetration Testing

- By the 1970s, the US government was regularly using teams to assess the security of computer systems by trying to penetrate them. These teams were referred to as red teams, or tiger teams.
- The penetration testing had been largely focused on the environments:
  - Attempt to compromise the security of the Computer operating systems, along with their access control mechanisms by penetration testing.
  - In a networked computer context, operating system configurations, including their respective network services, are usual targets for penetration tests. These operating system components have offered countless opportunities for penetration testing over the years
- Recently push was for applying penetration testing techniques “up” the software abstraction levels, beyond the operating system and network services, towards the application software itself

# Security is not compositional

---

- According to Leslie Lamport, a Turing Award winner, Security is not compositional.
- Two components that are secure on their own are not necessarily secure when used in combination
  - A change to one component might not break that component, but could break the whole system due to the lack of compositionality
- Upon change to the software, the configuration, the network topology, and so on, potentially new vulnerabilities are created

# An art or a science?

---

- Pen testers must be creative. They think about how a system is put together, and where assumptions made by designers represent weaknesses
- Pen testers will cleverly adapt the weaknesses to gain a foothold in one place. They may use that foothold to exploit a weakness somewhere else.
- Systems could be incorrectly built or misconfigured in the some ways.
- Tools are built to systematically look for weakness patterns, and exploit them.

Thus Pen testing is both an art and a science.

# Penetration Testing Skills

---

- A pen tester needs to know a lot about the target domain.
- For example, if the pen tester is attacking web applications, then the pen tester needs to know how the web works. Need to know how systems are built in that domain.
  - What protocols allow applications to communicate. For the web, that's HTTP and TCP, and IP.
  - The languages that are used to build applications like PHP, Java, or Ruby for talking about the web.
  - Frameworks that used to build applications or application components like, for the web, Ruby on Rails, DreamWeaver, Drupal, and so on.
- Pen tester also need to know common weaknesses from that domain.
  - For example, the bugs that are common to web applications like SQL injections or cross-site scripting, or cross-site request forgery. Or common misconfigurations or bad designs, like the use of default passwords or hidden files.

# Categories of Penetration Testing Tools

- **Host-Based Tools**
- **Network-Based Tools**
- **Application Testing Proxies**
- **Application Scanning Tools**
- Tools integrated with other IT security technologies, viz. firewalls, intrusion detection and prevention systems

# Host-Based Tools

---

- Host-based testing tools test the local operating system to assess its technical strengths and weaknesses, e.g. Dan Farmer's COPS program, which evaluated the security posture of a UNIX computer
  - e.g., evaluate file access control mechanisms for opportunities for attackers to affect the security of the host
  - examine every file, configuration data (including registry keys on Windows systems), installed patch inventory, and so on from the perspective of every ID on the system
  - Check common operating system configuration mistakes and omissions such as dangerous setuid files, unnecessary network services enabled, and excessive privileges for user accounts

# Network-Based Tool

---

- Network-based testing tools assess the security configuration of a computer operating system from afar—across a network, e.g. Chris Klaus's Internet Security Scanner (ISS) program
- Examine a target computer(s) for weaknesses that may be exploitable from a remote networked location using a database of vulnerabilities
  - Advantage- Scale: A huge number of computers can be evaluated across a network;
  - Limitation- Coverage: Network-based testing can only evaluate the externally accessible interfaces

# Nmap for network probing

---

Nmap stands for “network mapper”. Free, open source (commercial versions too)

<http://nmap.org/>

Figures out

- what **hosts** are available on the network,
- what **services** (application name and version) those hosts are offering,
- what **operating systems** (and OS versions) they are running,
- what type of **packet filters/firewalls** are in use
- ... etc.

Works by **sending raw IP packets** into the network and **observing the effects**

- Standard “ping” protocol, Looks for HTTPS(port 443) or HTTP(port 80) servers, - Probes to other TCP ports  
Probes that elicit different responses on different OSes (“fingerprinting”)

**Can be stealthy!**

- Control the rate of scanning to “work under the radar”

# Application Testing Proxies

---

## Application Testing Proxies

- enable the security tester to look behind the graphical user interface when testing a web application or web service
- Requests to and responses from the server are intercepted, observed, and optionally manipulated

## Web applications are common pen testing targets

- Web proxies sit *between* the browser and server
- Displaying exchanged packets
- Modifying them as directed by the tester

# Application Scanning Tools

---

- These tools do penetration testing scans of general purpose web-based software applications
- connect to web applications and attempt a series of well defined tests for each data field, cookie, etc.
- Such tools initiate a “learning mode” in which they observe the normal operation of a web application. Based on learning, they attempt to exploit common web application defects such as data overruns, SQL injection, and cross-site scripting (XSS)

# Ethical Hacking

---

- Penetration testing tools are meant to reveal security vulnerabilities so that they can be fixed, not so that they can be exploited for the purposes of crime or harm.
- But it is true that people will use these penetration testing tools for nefarious purposes.
- In that way, they are sort of two way tools.
  - Just as guns can be used to defend, guns can be used to attack. Ethical hacking is not to be someone who uses pen testing tools to attack.



# BUFFER OVERFLOW

# Buffer Overflow: A Well-Known Problem

---

- A very common attack mechanism
  - from 1988 Morris Worm to Code Red, Slammer, Sasser and many others
- Prevention techniques known
- Still of major concern due to
  - legacy of widely deployed buggy code
  - continued careless programming techniques

# Morris worm

---

- One of best known worms
  - Affected 6,000 computers in 1988; cost \$10-\$100 M
- Released by Robert Morris
  - Graduate student at Cornell, son of NSA chief scientist
  - Convicted under Computer Fraud and Abuse Act, sentenced to 3 years of probation and 400 hours of community service
  - Now a computer science professor at MIT
- Worm was intended to propagate slowly and harmlessly measure the size of the Internet. Due to a coding error, it created new copies as fast as it could and overloaded infected machines
- The worm propagated thru buffer overflow attack against a vulnerable version of fingerd on VAX system

# Buffer Overflow Basics

---

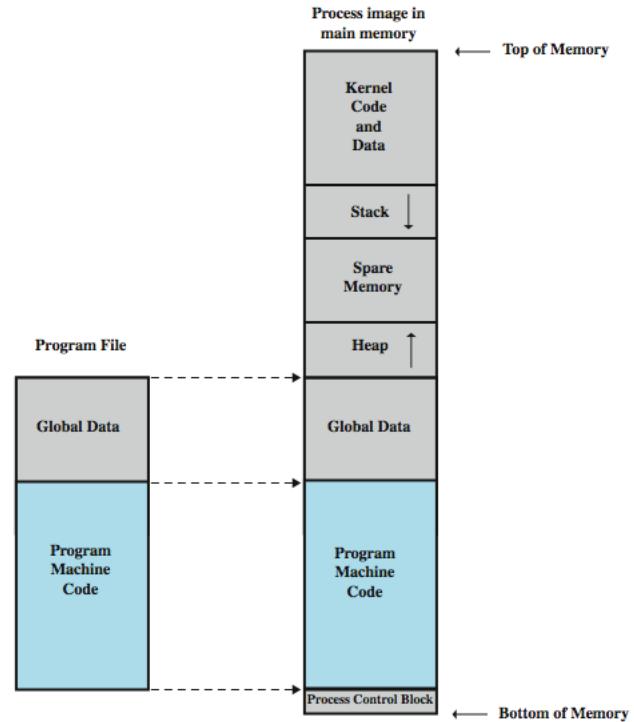
Caused by programming error

Allows more data to be stored than capacity available in a fixed sized buffer

- buffer can be on stack, heap, global data
- Overwriting adjacent memory locations
  - corruption of program data
  - unexpected transfer of control
  - memory access violation
  - execution of code chosen by attacker

# Process in Memory

- Processes are divided into three regions: Text, Data, and Stack.
- The text region includes code (instructions) and read-only data. This region is normally marked read-only and any attempt to write to it will result in a segmentation violation
- The data region contains initialized and uninitialized data. Static variables are stored in this region.
- A procedure call alters the flow of control, when finished performing its task, a function returns control to the statement or instruction following the call. This high-level abstraction is implemented with the help of the stack.
  - The stack is used to dynamically allocate the local variables used in functions, to pass parameters to the functions, and to return values from the function.





# **Buffer Overflow Defenses**

## **RL 7.1.2**

# Buffer Overflow Defenses

---

Buffer overflows are widely exploited

Large amount of vulnerable code in use

- despite cause and countermeasures known

Two broad defense approaches

- compile-time - harden new programs
- run-time - handle attacks on existing programs

# Compile-Time Defenses

---

- Aim to prevent or detect buffer overflows
- Possibilities include
  - Choose a high-level language that does not permit buffer overflows
  - Encourage safe coding standards
  - Use safe standard libraries
  - Include additional code to detect corruption of the stack frame

# Compile-Time Defenses: Programming Language

- Use a modern high-level languages with strong typing
  - not vulnerable to buffer overflow
  - compiler enforces range checks and permissible operations on variables
- Flexibility & Safety come with cost in resource use
  - at compile time
  - additional checks at run time
- Add restrictions on access to hardware
  - still need some code(e.g. device drivers) in C like languages

# Compile-Time Defenses: Stack Protection

- Stackguard: add function entry and exit code to check stack for signs of corruption
  - Use random canary
  - e.g. Stackguard, Win/GS, GCC
  - check for overwrite between local variables and saved frame pointer and return address
  - abort program if change found
  - issues: recompilation, debugger support
- Or save/check safe copy of return address (in a safe, non-corruptible memory area), e.g. Stackshield, RAD (Return Address Defender)

## Run-Time Defenses: Non Executable Address Space

- Many BO attacks copy machine code into buffer and xfer ctrl to it
- Use virtual memory support to make some regions of memory non-executable (to avoid exec of attacker's code)
  - e.g. stack, heap, global data
  - need h/w support in MMU
  - long existed on SPARC/Solaris systems
  - recent on x86 Linux/Unix/Windows systems
- Issues: support for executable stack code

## Run-Time Defenses: Address Space Randomization

---

Manipulate location of key data structures

- stack, heap, global data: change address by 1 MB
- using random shift for each process
- have large address range on modern systems means wasting some has negligible impact

Randomize location of heap buffers and location of standard library functions

# Source Code Analysis Tools

---

Source Code Analysis Tools (security analyzers) are automated tools for helping analysts find security-related problems in software

- use data- and control-flow analysis to find subtler bugs and to reduce false alarms

Some vulnerabilities (e.g. use of strcpy()) can be detected with high accuracy, others are harder to detect, and, in fact, one can always devise vulnerabilities that are undetectable altogether.

Tools have tradeoff between false alarms (also known as false positives) and missed vulnerabilities (also known as false negatives)

- can be configured to make a tool more sensitive (decreasing false negatives while increasing false positives) or make it less sensitive (increasing false negatives while decreasing false positives)



# **Heap & Integer Vulnerabilities**

## **RL 7.2.1**

# Heap Overflow

---

- Possible to attack buffer located in heap
  - typically located above program code and global data and grows up in memory (while stack grows down towards it)
  - memory requested by programs to use in dynamic data structures, e.g. linked lists
- No return address
  - hence no easy transfer of control
- May have function pointers that can be exploited
  - Typically for custom processing of data, e.g. decoding a compressed image
  - or manipulate management data structures
- Defenses: non executable or random heap

# Integer Security

---

Integers represent a source of vulnerabilities in C and C++ programs.

Integer range checking has not been systematically done in many C and C++ software

- Security flaws involving integers exist
- Some of these are likely to be vulnerabilities

Integers in C and C++ are either signed or unsigned.

- For each signed type there is an equivalent unsigned type.
- Signed integers are used to represent positive and negative values.
  - On a computer using two's complement arithmetic, a signed integer ranges from  $-2^{n-1}$  through  $2^{n-1}-1$ .
- Unsigned integer values range from zero to a maximum
  - This maximum value can be calculated as  $2^{n-1}$ , where n is the number of bits used to represent the unsigned type.

Computer Security: Principles and Practice by William Stallings, and Lawrie Brown  
Pearson, 2008.

[www.us-cert.gov/bsi](http://www.us-cert.gov/bsi)

[sei.cmu.edu/cert](http://sei.cmu.edu/cert)

[www.owasp.com](http://www.owasp.com)

[www.digital.com](http://www.digital.com)

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **Database Security Overview**

## **RL 8.1**

# DB Security

What are the ways of securing a database?

- They include database hardening, always encrypted data, separate authentication, advanced threat protection, and the principle of least privilege, which asserts that users and applications should be granted access only to the data and operations they require to perform their jobs.



- Database security is the processes, tools, and controls that secure and protect databases against accidental and intentional threats. The objective of database security is to secure sensitive data and maintain the confidentiality, availability, and integrity of the database. In addition to protecting the data within the database, database security protects the database management system and associated applications, systems, physical and virtual servers, and network infrastructure.
- To answer the question "what is database security," it's important to acknowledge that there are several types of security risks. Database security must guard against human error, excessive employee database privileges, hacker and insider attacks, malware, backup storage media exposure, physical damage to database servers, and vulnerable databases such as unpatched databases or those with too much data in buffers.

# Types of database security

---

## Network security

**Firewalls** serve as the first line of defense in DiD database security. Logically, a firewall is a separator or restrictor of network traffic, which can be configured to enforce your organization's data security policy. If you use a firewall, you will increase security at the operating system level by providing a chokepoint where your security measures can be focused.

## Access management

**Authentication** is the process of proving the user is who he or she claims to be by entering the correct user ID and password. Some security solutions allow administrators to centrally manage the identities and permissions of database users in one central location. This includes the minimization of password storage and enables centralized password rotation policies.

**Authorization** allows each user to access certain data objects and perform certain database operations like read but not modify data, modify but not delete data, or delete data.

**Access control** is managed by the system administrator who assigns permissions to a user within a database. Permissions are ideally managed by adding user accounts to database roles and assigning database-level permissions to those roles. the risk of accidental data disclosure.

# Types of database security

---

## Threat protection

**Auditing** tracks database activities and helps maintain compliance with security standards by recording database events to an audit log. This allows you to monitor ongoing database activities, as well as analyze and investigate historical activity to identify potential threats or suspected abuse and security violations.

**Threat detection** uncovers anomalous database activities that indicate a potential security threat to the database and can surface information about suspicious events directly to the administrator.

# Types of database security

---

## Information protection

**Data encryption** secures sensitive data by converting it into an alternative format so only the intended parties can decipher it back to its original form and access it. Although encryption doesn't solve access control problems, it enhances security by limiting data loss when access controls are bypassed. For example, if the database host computer is misconfigured and a malicious user obtains sensitive data, such as credit card numbers, that stolen information might be useless if it's encrypted.

**Database backup data and recovery** is critical to protecting information. This process involves making backup copies of the database and log files on a regular basis and storing the copies in a secure location. The backup copy and file are available to restore the database in the event of a security breach or failure.

**Physical security** strictly limits access to the physical server and hardware components. Many organizations with on-premises databases use locked rooms with restricted access for the database server hardware and networking devices. It's also important to limit access to backup media by storing it at a secure offsite location.

# Database Security Issues

---

## Types of Data Security issues

- Legal and ethical issues
  - Some information may be deemed to be private and cannot be accessed legally by unauthorized persons. In the United States, there are numerous laws governing privacy of information
- Policy issues
  - Policy issues at the governmental, institutional, or corporate level as to what kinds of information should not be made publicly available—for example, credit ratings and personal medical records
- System-related issues
  - system levels at which various security functions should be enforced—for example, whether a security function should be handled at the physical hardware level, the operating system level, or the DBMS level
- The need to identify multiple security levels
  - e.g., top secret, secret, confidential, and unclassified. The security policy of the organization with respect to permitting access to various classifications of data must be enforced.

# Database Security

---

## Threats to databases

- Loss of **integrity**
  - Information be protected from improper modification
- Loss of **availability**
  - Available to user or program with legitimate right
- Loss of **confidentiality**
  - Protection of data from unauthorized disclosure

To protect databases against these types of threats four kinds of countermeasures can be implemented:

- **Access control**
- **Inference control**
- **Flow control**
- **Encryption**

# Introduction to Database Security

---

- The security mechanism of a DBMS must include provisions for restricting access to the database as a whole
  - This function is called **access control** and is handled by creating user accounts and passwords to control login process by the DBMS.
- The security problem associated with databases is that of controlling the access to a **statistical database**, which is used to provide statistical information or summaries of values based on various criteria.
  - The countermeasures to **statistical database security** problem is called **inference control measures**.
- Another security is that of **flow control**, which prevents information from flowing in such a way that it reaches unauthorized users.
  - Channels that are pathways for information to flow implicitly in ways that violate the security policy of an organization are called **covert channels**.
- A final security issue is **data encryption**, which is used to protect sensitive data (such as credit card numbers) that is being transmitted via some type communication network.
  - The data is **encoded** using some **encoding algorithm**.
  - An unauthorized user who access encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher data.

# Sensitive Data and Types of Disclosures

Sensitivity of data is a measure of the importance assigned to the data by its owner, for the purpose of denoting its need for protection

Several factors can cause data to be classified as sensitive:

- Inherently sensitive. The value of the data itself may be so revealing or confidential that it becomes sensitive—for example, a person's salary or that a patient has HIV/AIDS.
- From a sensitive source. The source of the data may indicate a need for secrecy—for example, an informer whose identity must be kept secret.
- Declared sensitive. The owner of the data may have explicitly declared it as sensitive.
- A sensitive attribute or sensitive record. The particular attribute or record may have been declared sensitive—for example, the salary attribute of an employee or the salary history record in a personnel database.
- Sensitive in relation to previously disclosed data. Some data may not be sensitive by itself but will become sensitive in the presence of some other data—for example, the exact latitude and longitude information for a location where some previously recorded event happened that was later deemed sensitive.

# Data availability

---

Several factors need to be considered before deciding whether it is safe to reveal the data. The three most important factors are

- Data availability.** If a user is updating a field, then this field becomes inaccessible and other users should not be able to view this data. This blocking is only temporary and only to ensure that no user sees any inaccurate data
- Access acceptability.** A user request that does not directly access a sensitive data item, may be denied on the grounds that the requested data may reveal information about the sensitive data that the user is not authorized to have.
- Authenticity assurance.** There may be additional considerations, e.g., a user may only be permitted access during working hours. The system may track previous queries to ensure that a combination of queries does not reveal sensitive data.

# Information Security vs Information Privacy

---

- Questions of who has what rights to information about individuals for which purposes become more important in this information age
- There is a considerable overlap between issues related to access to resources (security) and issues related to appropriate use of information (privacy).
- Security in information technology refers to many aspects of protecting a system from unauthorized use, including authentication of users, information encryption, access control, firewall policies, and intrusion detection.
- The concept of privacy goes beyond security. Privacy examines how well the use of personal information that the system acquires about a user conforms to the explicit or implicit assumptions regarding that use.
  - Two different perspectives for privacy: preventing storage of personal information versus ensuring appropriate use of personal information.

# Database Security and DBA

---

The DBA (database administrator)'s responsibilities include safeguarding database security. The DBA holds privileged account & among others activities, performs

- Account creation. This action creates a new account and password for a user or a group of users to enable access to the DBMS.
- Privilege granting. This action permits the DBA to grant certain privileges to certain accounts.
- Privilege revocation. This action permits the DBA to revoke (cancel) certain privileges that were previously given to certain accounts.
- Security level assignment. This action consists of assigning user accounts to the appropriate security clearance level.

# Database Logs and Database Audit

---

The database system must also keep **track of all operations** on the database that are applied by a certain user throughout **each login session**.

- To keep a record of all updates applied to the database and of the particular user who applied each update, we can modify **system log**, which includes an entry for each operation applied to the database that may be required for recovery from a transaction failure or system crash.

If any tampering with the database is suspected, a database audit is performed

- A database audit consists of reviewing the log to examine all accesses and operations applied to the database during a certain time period.

A database log that is used mainly for security purposes is sometimes called an audit trail



# **Discretionary & Mandatory access control**

## **RL 8.2.1**

# Database Security Mechanisms

---

- A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security portions of a database against unauthorized access.
- Two types of database security mechanisms:
  - Discretionary security mechanisms
    - used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update)
  - Mandatory security mechanisms
    - used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization

# Discretionary Access Control

The typical method of enforcing **discretionary access control** in a database system is based on the **granting** and **revoking privileges**.

## Types of Discretionary Privileges

- The account level:
  - At this level, the DBA specifies the particular privileges that each account holds independently of the relations in the database.
- The relation level (or table level):
  - At this level, the DBA can control the privilege to access each individual relation or view in the database.

# Types of Discretionary Privileges

---

- The privileges at the **account level** apply to the capabilities provided to the account itself and can include
  - the **CREATE SCHEMA** or **CREATE TABLE** privilege, to create a schema or base relation;
  - the **CREATE VIEW** privilege;
  - the **ALTER** privilege, to apply schema changes such adding or removing attributes from relations;
  - the **DROP** privilege, to delete relations or views;
  - the **MODIFY** privilege, to insert, delete, or update tuples;
  - and the **SELECT** privilege, to retrieve information from the database by using a **SELECT** query.

# Types of Discretionary Privileges

- The granting and revoking of privileges generally follow an authorization model for discretionary privileges known as the access matrix model where
  - The **rows** of a matrix M represents **subjects** (users, accounts, programs)
  - The columns represent objects (relations, records, columns, views, operations).
  - Each position  $M(i,j)$  in the matrix represents the types of privileges (read, write, update) that subject i holds on object j.
- The discretionary access control has traditionally been the main security mechanism for DBMS.
  - This is an all-or-nothing method

# Mandatory Access Control

---

- The discretionary access control is an all-or-nothing method, e.g. a user has privilege to access customer data or Not.
  - Business may not be treating all customers equal. Discretionary access control has no direct mechanism to handle such situation.
- Many applications require additional security policy that classifies data and users based on security classes. This is called Mandatory Access Control.
  - This approach as mandatory access control, would typically be combined with the discretionary access control mechanisms

# Multilevel Security

---

- To incorporate multilevel security notions into the relational database model, it is common to consider attribute values and tuples as data objects.
- Hence, each attribute A is associated with a classification attribute C in the schema, and each attribute value in a tuple is associated with a corresponding security classification.
- In addition, in some models, a tuple classification attribute TC is added to the relation attributes to provide a classification for each tuple as a whole.
- Hence, a multilevel relation schema R with n attributes would be represented as

$R(A_1, C_1, A_2, C_2, \dots, A_n, C_n, TC)$

where each  $C_i$  represents the classification attribute associated with attribute  $A_i$ .

# Security Classes

---

- Typical security classes are top secret (TS), secret (S), confidential (C), and unclassified (U), where TS is the highest level and U the lowest:  $TS \geq S \geq C \geq U$
- The commonly used model for multilevel security, known as the Bell-LaPadula model, classifies each subject (user, account, program) and object (relation, tuple, column, view, operation) into one of the security classifications, T, S, C, or U:
  - Represent clearance (classification) of a subject S as class(S) and to the classification of an object O as class(O).

## Mandatory Access Control and Multilevel Security

---

- A multilevel relation will appear to contain different data to subjects (users) with different clearance levels.
  - In some cases, it is possible to store a single tuple in the relation at a higher classification level and produce the corresponding tuples at a lower-level classification through a process known as **filtering**.
  - In other cases, it is necessary to store two or more tuples at different classification levels with the same value for the **apparent key**.
    - The apparent key of a multilevel relation is the set of attributes that would have formed the primary key in a regular (single-level) relation
- This leads to the concept of **polyinstantiation** where several tuples can have the same apparent key value but have different attribute values for users at different classification levels.

# Mandatory Access Control and Multilevel Security

- In general, the **entity integrity** rule for multilevel relations states that all attributes that are members of the apparent key must not be null and must have the same security classification within each individual tuple.
- In addition, all other attribute values in the tuple must have a security classification greater than or equal to that of the apparent key.
  - This **constraint** ensures that a user can see the key if the user is permitted to see any part of the tuple at all.
- Other integrity rules, called **null integrity** and **interinstance integrity**, informally ensure that if a tuple value at some security level can be filtered (derived) from a higher-classified tuple, then it is sufficient to store the higher-classified tuple in the multilevel relation.



# **Bell LaPadula Model RL 8.2.2**

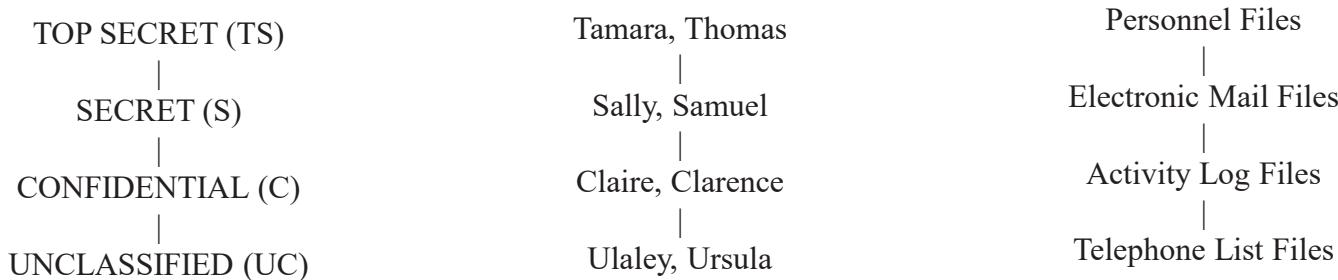
# Confidentiality - Bell-LaPadula Model

- A confidentiality policy prevents the unauthorized disclosure of information.
  - e.g., the navy must keep confidential the date on which a troop ship will sail.
- Bell-LaPadula Model is a multi-level model proposed by Bell and LaPadula of MITRE for enforcing access control in government and military applications
- It uses military-style classifications. Subjects and Objects are often partitioned into different security levels
- A subject can only access objects at certain levels determined by his/her/its security level

# The Model

---

- The confidentiality classification is a set of security clearances arranged in a linear (total) ordering.
- Clearances represent the security levels. The higher the level, the more sensitive the info.
  - A subject has a security clearance. An object has a security classification. While referring to both subject clearances and object classifications, the term “classification” is used



The basic confidentiality classification system. The four security levels are arranged with the most sensitive at the top and the least sensitive at the bottom.



# Inference Control RL 8.3.1

# Statistical Database Security

---

- Statistical databases are used mainly to produce statistics on various populations.
- The database may contain confidential data on individuals, which should be protected from user access.
- Users are permitted to retrieve statistical information on the populations, such as averages, sums, counts, maximums, minimums, and standard deviations.
- A population is a set of tuples of a relation (table) that satisfy some selection condition.
- Statistical queries involve applying statistical functions to a population of tuples.

# Statistical Queries

---

- Statistical database security techniques must prohibit the retrieval of individual data.
- This can be achieved by prohibiting queries that retrieve attribute values and by allowing only queries that involve statistical aggregate functions such as COUNT, SUM, MIN, MAX, AVERAGE, and STANDARD DEVIATION.
  - Such queries are sometimes called statistical queries.

# Inference Control

---

- It is DBMS's responsibility to ensure confidentiality of information about individuals, while still providing useful statistical summaries of data about those individuals to users. Provision of privacy protection of users in a statistical database is paramount.
- In some cases it is possible to infer the values of individual tuples from a sequence of statistical queries.
  - This is particularly true when the conditions result in a population consisting of a small number of tuples.



# Flow Control

## RL 8.3.2

# Introduction to Flow Control

---

- **Flow control** regulates the distribution or flow of information among accessible objects.
- A **flow** between object X and object Y occurs when a program reads values from X and writes values into Y.
  - Flow controls check that information contained in some objects does not flow explicitly or implicitly into less protected objects.
- A **flow policy** specifies the channels along which information is allowed to move.
  - The simplest flow policy with just two classes of information:
    - confidential (C) and nonconfidential (N)
  - allows all flows except those from class C to class N.

# Flow Control & Access Control

---

- Access control mechanisms help in Flow control.
- Flow controls can be enforced by an extended access control mechanism, which involves assigning a security class (usually called the clearance) to each running program.
  - The program is allowed to read a particular memory segment only if its security class is as high as that of the segment.
  - It is allowed to write in a segment only if its class is as low as that of the segment.
- This automatically ensures that no information transmitted by the person can move from a higher to a lower class.
  - For example, a military program with a secret clearance can only read from objects that are unclassified and confidential and can only write into objects that are secret or top secret.

# Covert Channels

---

- A **covert channel** allows a transfer of information that violates the security or the policy.
- A **covert channel allows** information to pass from a higher classification level to a lower classification level through **improper means**.
- **Covert channels** can be classified into two broad categories:
  - **Storage channels** do not require any temporal synchronization, in that information is conveyed by accessing system information or what is otherwise inaccessible to the user.
  - **Timing channel** allow the information to be conveyed by the timing of events or processes.

# Covert Channels - Challenges

---

- Difficult to detect
- Can operate for a long time and leak a substantial amount of classified data
- Can compromise an otherwise secure system, including one that has been formally verified.

# Covert Channels - Examples

---

- A programmer for a bank has no need to access the names or balances in depositors' accounts.
- Programmers for brokerage firms do not need to know what buy and sell orders exist for clients.
- During program testing, access to a form of real data or some sample test data may be justifiable, but not after the program has been accepted for regular use.
- This represents information leakage thru covert channel.

# SQL Injection - Introduction

---

- Web applications (that access a database) often have 2-way communication with database
  - can send commands and data to the database
  - display data retrieved from the database through the Web browser.
- In an SQL Injection attack, the attacker injects a (malicious)string input through the application, which changes or manipulates the SQL statement to the attacker's advantage.
- An SQL Injection attack can harm the database in various ways, such as
  - unauthorized manipulation of the database, or
  - retrieval of sensitive data.
  - It can also be used to execute system level commands

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **Web Security Overview**

## **RL 9.1**

# Web Security

---

Web has two distinct components

- Server
  - Has resources to serve. Content(resources) delivered to the client based on URL etc.
  - Generally managed by an administrator.
  - Self-service models reduce control
- Client/Browser
  - The component that is likely to approach a server for content.
  - Meant for end-user(non-technical)

## Security

- Need to consider both sides for protecting users

# Web Security

---

During early stages of web, entire security focus was on servers.

Today browsers have acquired lot of capabilities:

- JavaScript: Allows a page to execute client--side code.
- DOM model Provides a JavaScript interface to the page's HTML, allowing the page to add/remove tags, change their styling, etc.
- XMLHttpRequests (AJAX): Asynchronous HTTP requests.
- Web sockets: Full--duplex client--server communication over TCP.
- Web workers: Multi--threading support.
- Multimedia support: (video), web cams, screen--sharing.
- Geolocation: Browser can determine your location by examining GPS units. Firefox can also locate you by passing your WiFi information to the Google Location Service.
- <canvas> and WebGL: Bitmap manipulation and interactive 2D/3D graphics.
- Nacl: Allows browsers to run native code!

# Browser Vulnerabilities

---

Whenever a browser communicates with a website,

- the website, as part of that communication, collects some information about the browser (in order to process the formatting of the page to be delivered, if nothing else).
- If malicious code has been inserted into the website's content, then vulnerabilities specific to a particular browser can allow this malicious code to run processes within the browser application in unintended ways (one of the bits of information that a website collects from a browser communication is the browser's identity- allowing specific vulnerabilities to be exploited).
- Once an attacker is able to run processes on the visitor's machine, then exploiting known security vulnerabilities can allow the attacker to gain privileged access to the victim's machine or network

# Securing Web Browser

---

Web browsers can be breached in one or more of the following ways:

- Operating system is breached and malware is reading/modifying the browser memory space in privilege mode
- Main browser executable can be hacked
- Browser components may be hacked
- Browser plugins can be hacked
- Browser network communications could be intercepted outside the machine

# Diversity of Browsers

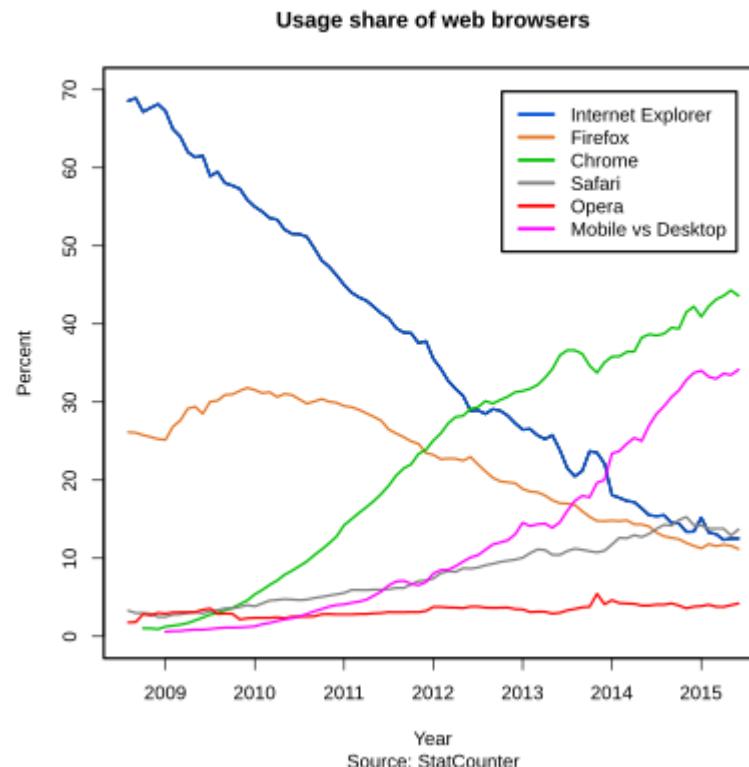
Complications due to

- Presence of multiple browsers
- Usage of numerous versions

Most popular browser today is Chrome. As per vulnerability database, in 2017 alone, 8 vulnerabilities have been reported for the browser.

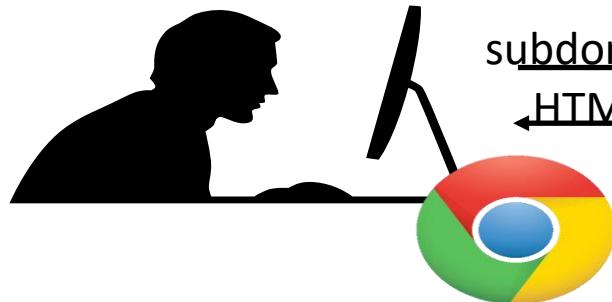
IE has 54 vulnerabilities reported in 2017.

All major browsers seem to have some reported vulnerabilities

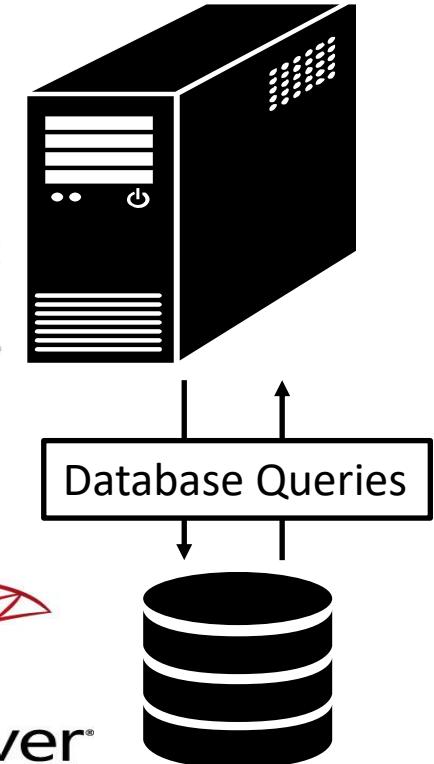


Source: <https://www.cvedetails.com/index.php>

# Web Application Overview



subdomain.mysite.com/folder/page?id=5  
HTML Page, JS file, CSS file, image, etc.



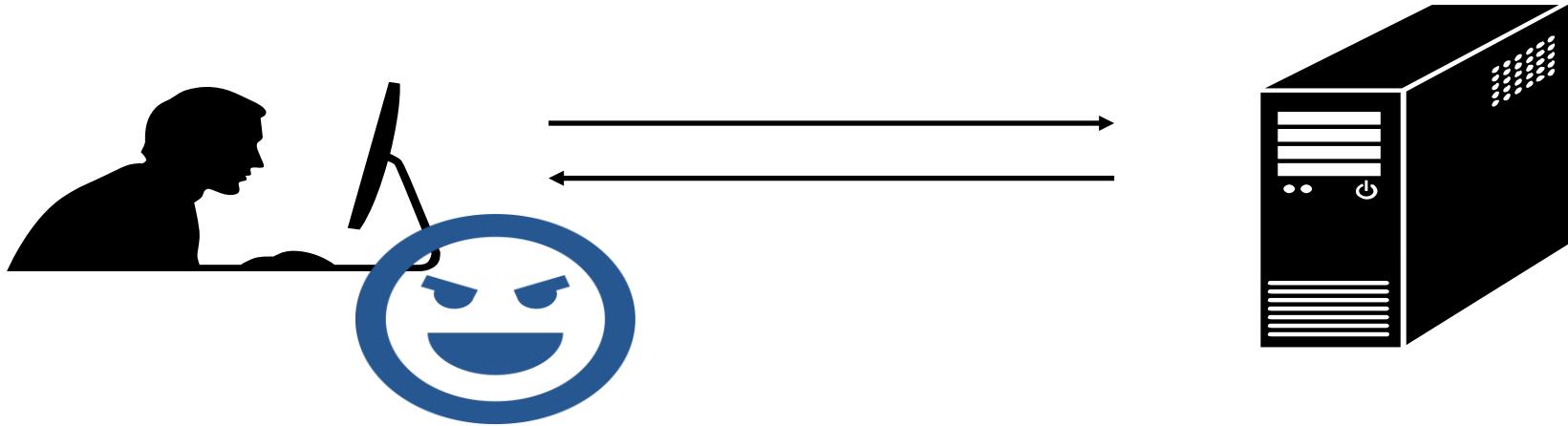
GET Requests: Used for requests for pages, resources, etc.

POST Requests: Used for form submissions, logins, etc.



# Web Security Overview

(By Threat Model)



## Malicious Client Attacking Server

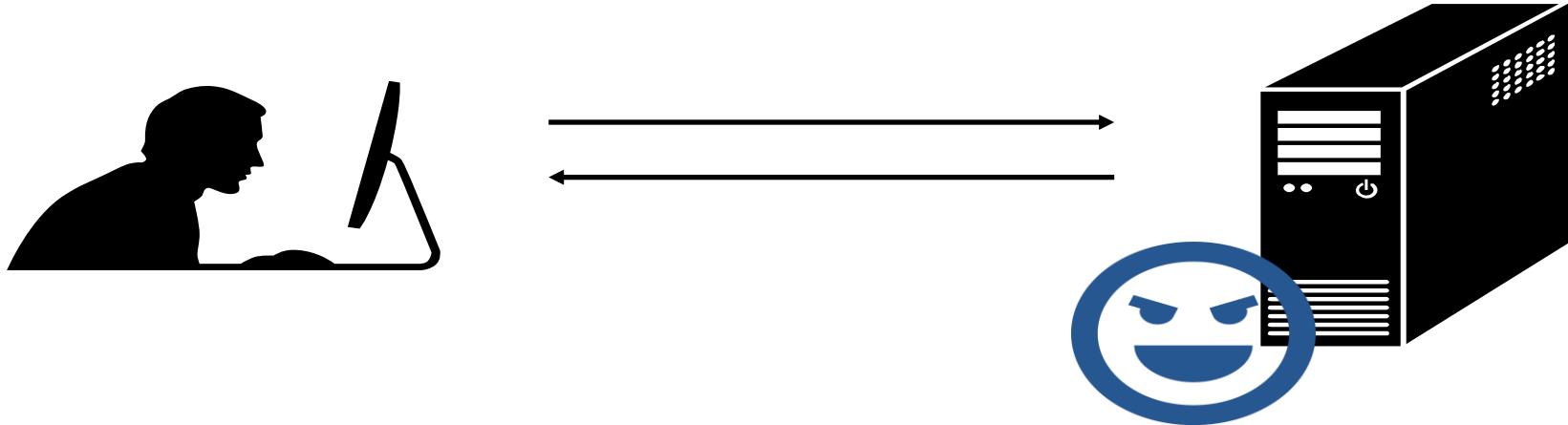
SQL Injection

File System Traversal

Broken Access Control

# Web Security Overview

(By Threat Model)



## Malicious Server Attacking Client

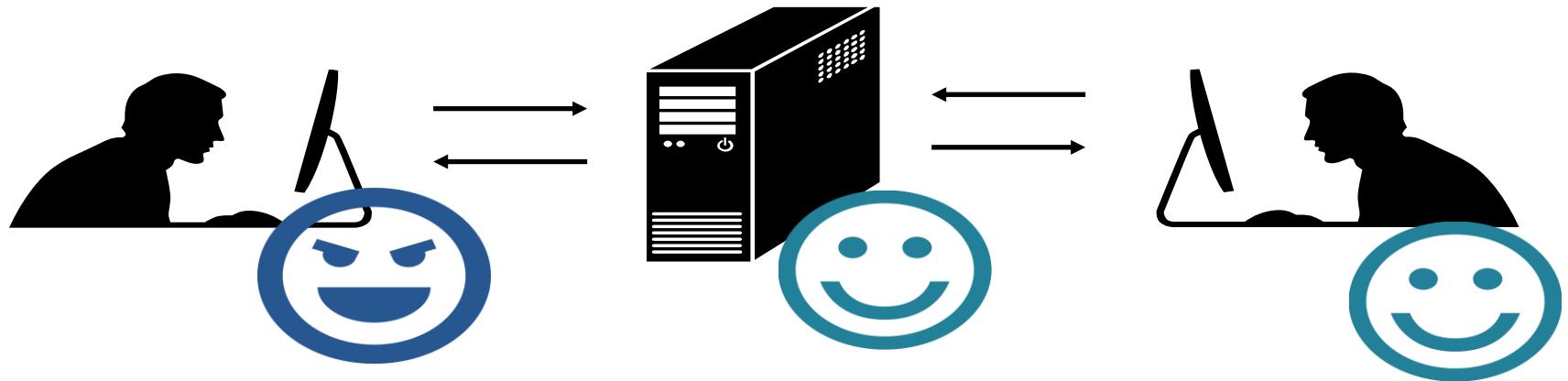
Clickjacking

History Probing

Phishing

# Web Security Overview

(By Threat Model)



## Malicious User Attacking Other Users

Cross-Site Scripting

Cross-Site Request Forgery

Remote Script Inclusion

# Web Security Overview

(By Threat Model)



## Malicious Server in “Mashup” Web Application

Clickjacking

Information Stealing

# Web Security Overview

(By Threat Model)



## Malicious User in Multi-Server Application

**Single sign-on (Facebook, Twitter, etc.):** Sign in as someone else

**Multi-Party Payment (Paypal, Amazon Payments):** Buy things for free

# Large Attack Surface

---

- Web has become a complex platform for distributed computation
- Developed a Huge Attack Surface
- A single web application spans multiple programming languages, Operating Systems, hardware platforms, throwing up emergent vulnerabilities.
  - E.g. might be running Chrome on Windows interacting with a Linux server running Apache and interfacing with MySQL
  - Difficult (almost impossible) to verify end--to--end correctness
- The web specs are very long, very complex, sometimes contradictory, and constantly evolving (quirks at quirksmode.org and several security information sites)

# Anatomy of Web Attack

---

- Typically attacker breaks into a legitimate Web site and posts malware
  - malware is not exclusive to malicious web sites. Often mainstream web sites are made to act as parasitic hosts that serve up malware to their unsuspecting visitors. Due to the complexity of modern web sites there are several techniques by which they are compromised.
- Attacking enduser machines
  - malware on a web site makes its way down on to a user's machine when that user visits the host Web site.
  - Some of the techniques enable this to happen with no user interaction – ‘drive-by-download’.
  - Some techniques which do require some input from the user.
- Leveraging end user machines for malicious activity
  - The most malicious activities begin once new malware has established a presence on a user's machine.



# **Cross-site Scripting**

## **RL 9.2.1**

# Cross-site Scripting attack

---

An XSS attack needs three actors — **the website, the victim and the attacker**

- The attacker injects a payload in the website's database by submitting a vulnerable form with some malicious JavaScript
- The victim requests the web page from the website
- The website serves the victim's browser the page with the attacker's payload(malicious javascript) as part of the HTML body.
- The victim's browser will execute the malicious script inside the HTML body. In this case it would send the victim's cookie to the attacker's server. The attacker now simply needs to extract the victim's cookie when the HTTP request arrives to the server, after which the attacker can use the victim's stolen cookie for impersonation.

# What's the worst an attacker can do with JavaScript?

JavaScript has access to the following

- Malicious JavaScript has access to all the same objects the rest of the web page has, including access to cookies. Cookies are often used to store session tokens, if an attacker can obtain a user's session cookie, they can impersonate that user
- JavaScript can read and make arbitrary modifications to the browser's DOM (within the page that JavaScript is running).
- JavaScript can use XMLHttpRequest to send HTTP requests with arbitrary content to arbitrary destinations.
- JavaScript in modern browsers can leverage HTML5 APIs such as accessing a user's geolocation, webcam, microphone and even the specific files from the user's file system. While most of these APIs require user opt-in, XSS in conjunction with some clever social engineering can bring an attacker a long way.

# The consequences of malicious JavaScript

Cookie theft:

- The attacker can access the victim's cookies associated with the website using `document.cookie`, send them to his own server, and use them to extract sensitive information like session IDs.

Keylogging:

- The attacker can register a keyboard event listener using `addEventListener` and then send all of the user's keystrokes to his own server, potentially recording sensitive information such as passwords and credit card numbers.

Phishing:

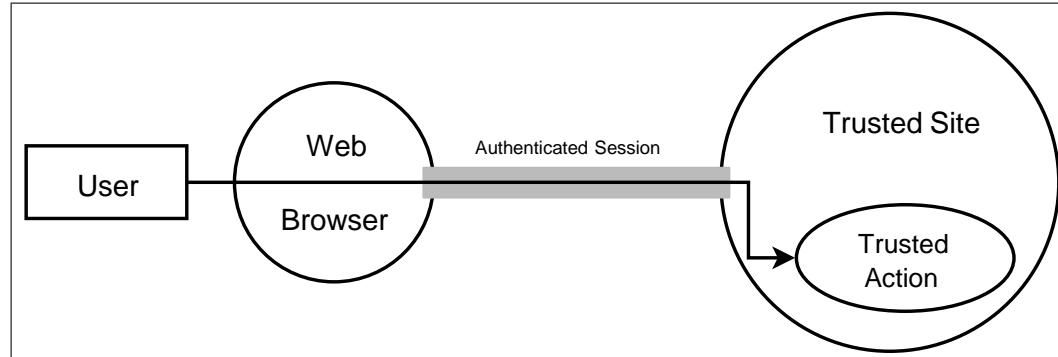
- The attacker can insert a fake login form into the page using DOM manipulation, set the form's action attribute to target his own server, and then trick the user into submitting sensitive information.

# Cross-Site Request Forgery

---

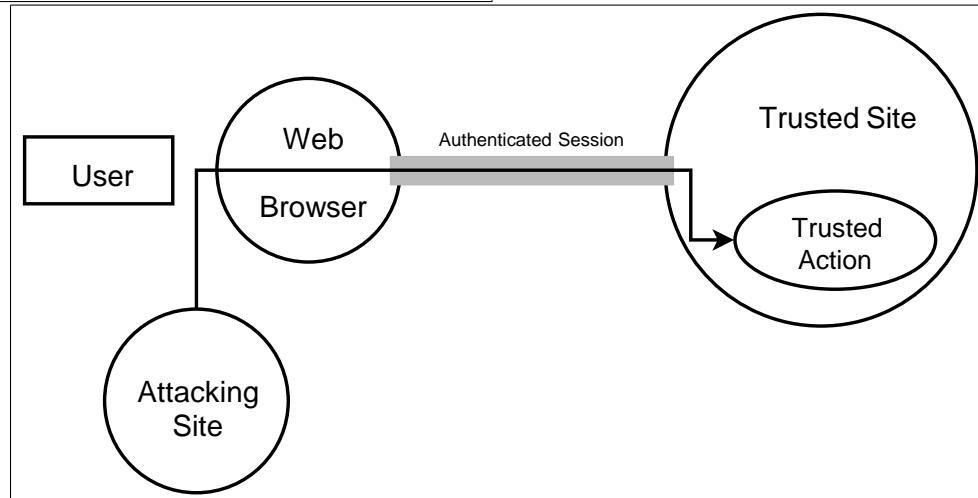
- Cross-Site Request Forgery (CSRF) is an attack that forces a victim to execute unwanted actions on a web application on which he is currently authenticated.
- CSRF attacks specifically target state-changing requests, (no direct theft of data), since the response to the forged request goes to victim.
- With social engineering (such as sending a link via email or chat), an attacker may trick the users of a web application into executing actions of the attacker's choosing (including fund transfers etc.)

# Cross-Site Request Forgery (CSRF)



A valid request.

A CSRF attack



# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **Security Mechanisms**

## **RL 10**



# **Encryption for Security**

## **RL 10.1**

# Secure Communication

---

There are numerous reasons why communication has to be secure

- National defense
- Business transactions
- Privacy needs

Requirements of secure communication

1. Secrecy - Only intended receiver understands the message
2. Authentication - Sender and receiver need to confirm each others identity
3. Message Integrity - Ensure that their communication has not been altered, either maliciously or by accident during transmission

# Cryptography Basics

---

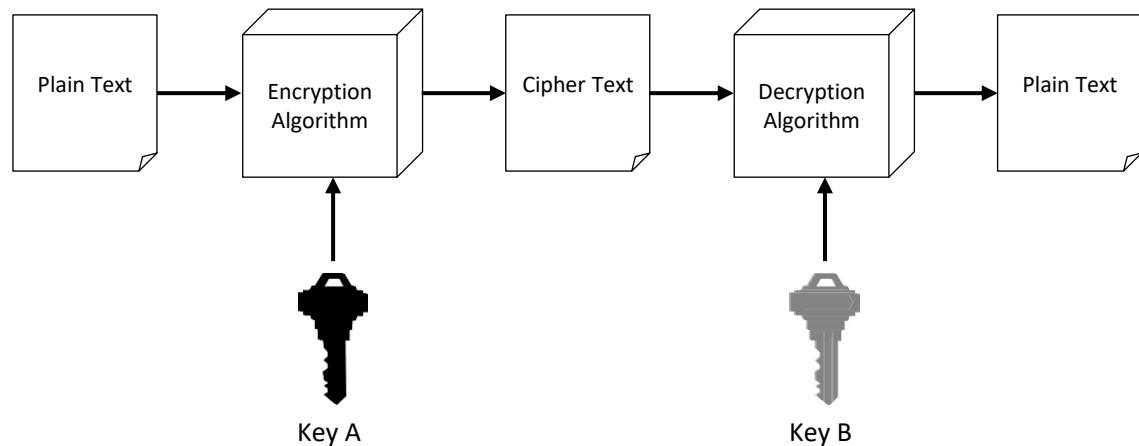
- Cryptography is the science of secret, or secure communication, in the presence of adversaries. It is a discipline combining Mathematics with CS
- It has two main Components:
  1. Encryption
    - Practice of hiding messages so that they can not be read by anyone other than the intended recipient
  2. Authentication & Integrity
    - Ensuring that users of data/resources are the persons they claim to be and that a message has not been surreptitiously altered

# Cryptography - Cipher

Cipher is a method/algorithm for encrypting/decrypting messages

Encryption algorithms are standardized & published

- The key which is an input to the algorithm is secret
  - Key is a string of numbers or characters
  - If same key is used for encryption & decryption the algorithm is called symmetric
  - If different keys are used for encryption & decryption the algorithm is called asymmetric



# Encryption - Symmetric Algorithms

---

Algorithms in which the key for encryption and decryption are the same are Symmetric

- Example: 3DES, AES, RC4 etc.

Types:

## 1. Block Ciphers

- Encrypt data one block at a time (typically 64 bits, or 128 bits)
- Used for a single message

## 2. Stream Ciphers

- Encrypt data one bit or one byte at a time
- Used if data is a constant stream of information

# Symmetric Encryption - Key Strength

- Strength of algorithm is determined by the size of the key
  - The longer the key the more difficult it is to crack
- Key length is expressed in bits
  - Typical key sizes vary between 48 bits and 448 bits
- Set of possible keys for a cipher is called key space
  - For 40-bit key there are  $2^{40}$  possible keys
  - For 128-bit key there are  $2^{128}$  possible keys
  - Each additional bit added to the key length doubles the security
- To crack the key the hacker has to use brute-force
  - (i.e. try all the possible keys till a key that works is found)
  - Super Computer can crack a 56-bit key in 24 hours
  - It will take  $2^{72}$  times longer to crack a 128-bit key (billions of years)

# Characteristics of “Good” Ciphers

---

According to Claude Shannon

- The amount of secrecy needed should determine the amount of labor appropriate for the encryption and decryption.
- The set of keys and the enciphering algorithm should be free from complexity.
- The implementation of the process should be as simple as possible.
- Errors in ciphering should not propagate and cause corruption of further information in the message.
- The size of the enciphered text should be no larger than the text of the original message.

Claude Elwood Shannon was an MIT/Princeton mathematician, electrical engineer known for information theory and cryptography.

# Properties of Trustworthy Encryption Systems

---

Based on sound mathematics.

- Good cryptographic algorithms are derived from solid principles.

Analyzed by competent experts and found to be sound.

- Since it is hard for the writer to envisage all possible attacks on the algorithm

Stood the “test of time.”

- Over time people continue to review both mathematical foundations of an algorithm and the way it builds upon those foundations.
- The flaws in most algorithms are discovered soon after their release.

# Symmetric Encryption

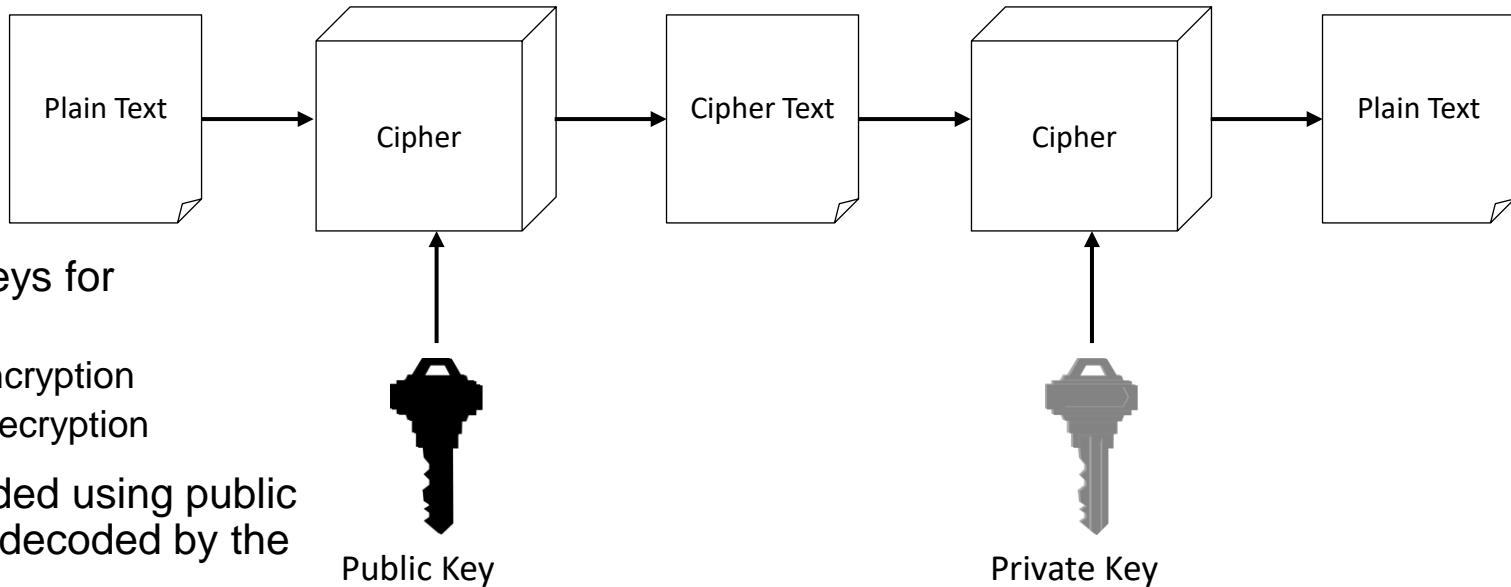
Algorithm	Type	Key Size	Features
DES	Block Cipher	56 bits	Most Common, Not strong enough
TripleDES	Block Cipher	168 bits (112 effective)	Modification of DES, Adequate Security
Blowfish	Block Cipher	Variable (Up to 448 bits)	Excellent Security
AES	Block Cipher	Variable (128, 192, or 256 bits)	Replacement for DES, Excellent Security
RC4	Stream Cipher	Variable (40 or 128 bits)	Fast Stream Cipher, Used in most SSL implementations

## Symmetric Encryption – Limitations

- Any exposure to the secret key compromises secrecy of ciphertext
- A key needs to be delivered to the recipient of the coded message for it to be deciphered
  - Potential for eavesdropping attack during transmission of key

# Asymmetric Encryption

- Uses a pair of keys for encryption
  - Public key for encryption
  - Private key for decryption
- Messages encoded using public key can only be decoded by the private key
  - Secret transmission of key for decryption is not required
  - Every entity can generate a key pair and release its public key



# Asymmetric Encryption Types

---

Two most popular algorithms are RSA & El Gamal

- RSA
  - Developed by Ron Rivest, Adi Shamir, Len Adelman
  - Both public and private key are interchangeable
  - Variable Key Size (512, 1024, or 2048 bits)
  - Most popular public key algorithm
- El Gamal
  - Developed by Taher ElGamal
  - Variable key size (512 or 1024 bits)
  - Less common than RSA, used in protocols like PGP

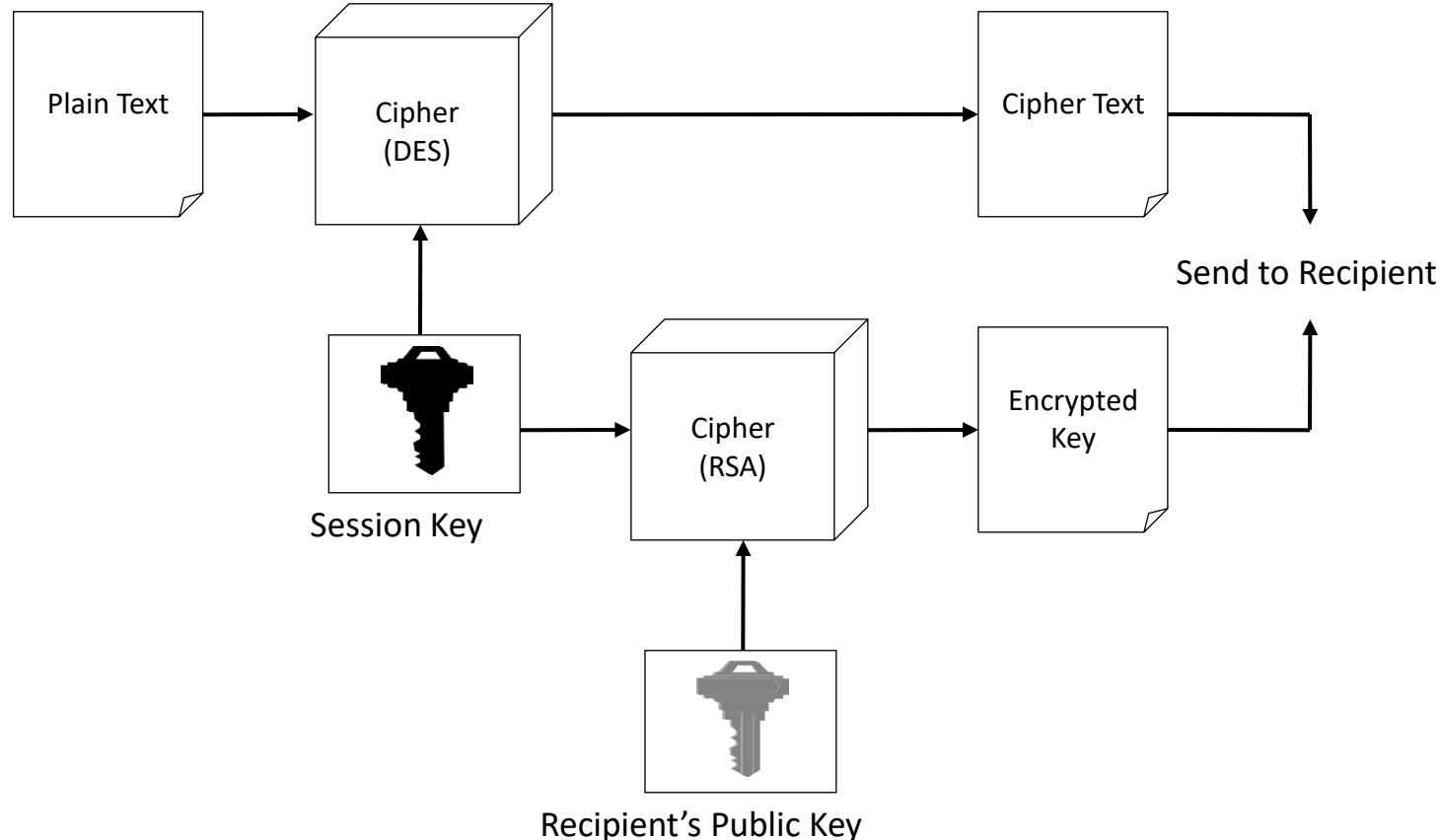
# Asymmetric Encryption Weaknesses

---

- Efficiency is lower than Symmetric Algorithms
  - A 1024-bit asymmetric key is equivalent to 128-bit symmetric key
- Potential for man-in-the middle attack
- It is problematic to get the key pair generated for the encryption

# Session-Key Encryption (Asymmetric Encryption)

- Used to improve efficiency
  - Symmetric key is used for encrypting data
  - Asymmetric key is used for encrypting the symmetric key





**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

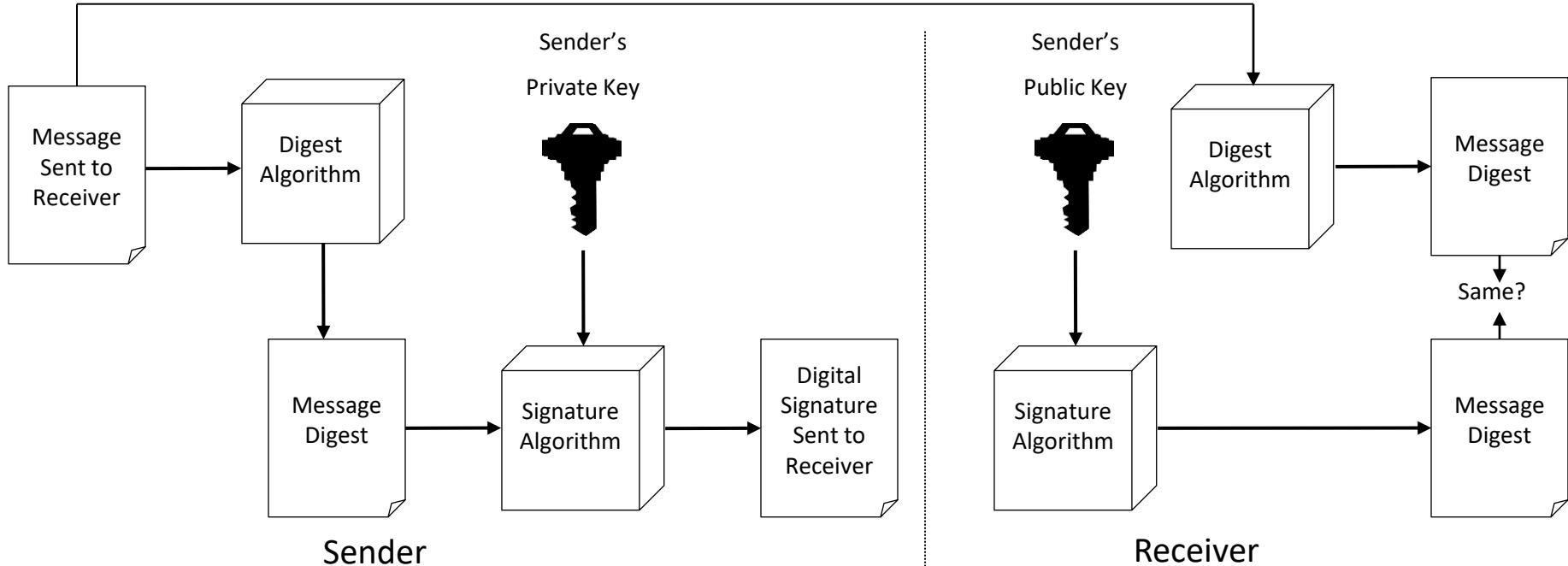
T V Rao



# Digital Signatures

## RL 10.2

# Authentication using Digital Signatures



- A digital signature is a data item which accompanies or is logically associated with a digitally encoded message.
  - A guarantee of the source of the data
  - Proof that the data has not been tampered with

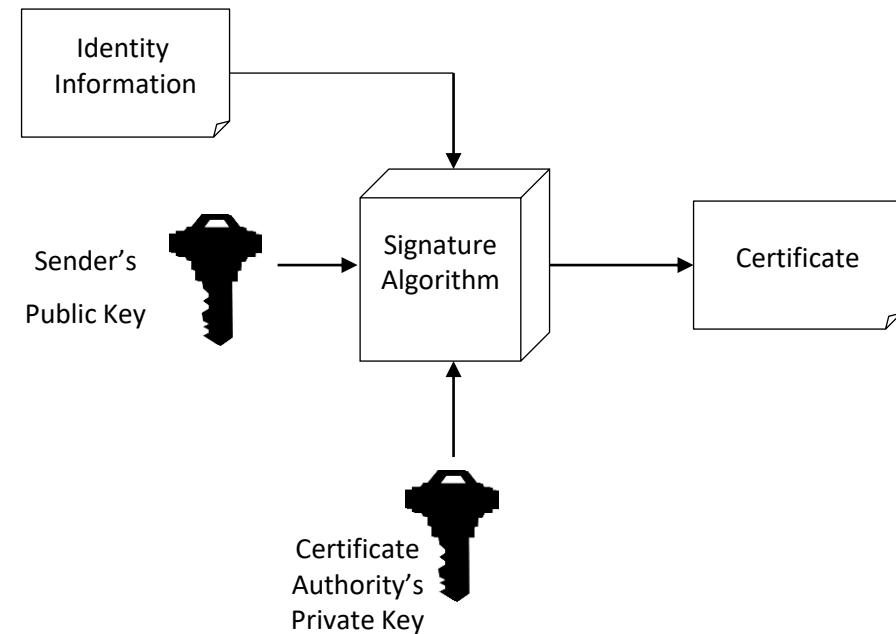
# Authentication – Digital Signature

---

- Based on asymmetric keys and digital signature algorithm
- Authenticators produced are **digital signatures**
- Very useful – **anyone** can verify authenticity of a message
- In a digital-signature algorithm, computationally infeasible to derive  $k_s$  from  $k_v$ 
  - $V$  is a one-way function
  - Thus,  $k_v$  is the public key and  $k_s$  is the private key
- Consider the RSA digital-signature algorithm
  - Similar to the RSA encryption algorithm, but the key use is reversed
  - Private key for encryption and public key for decryption(verification)

# Authentication - Digital Certificates

- A digital certificate is a signed statement by a trusted party that another party's public key belongs to them.
  - This allows one certificate authority to be authorized by a different authority (root CA)
- Top level certificate must be self signed (certificate authority)
  - Name recognition is key to some one recognizing a certificate authority
  - Verisign is industry standard certificate authority





# Intrusion Detection RL 10.3

# Security Intrusion & Detection

As per RFC 2828 - Internet Security Glossary,

- **Security intrusion:** a security event, or combination of multiple security events, that constitutes a security incident in which an intruder ***gains, or attempts to gain***, access to a system (or system resource) without having authorization to do so.
- **Intrusion detection:** a security service that monitors and analyzes system events for the purpose of finding, and providing real-time or near real-time warning of attempts to access system resources in an unauthorized manner.

# Intrusion Detection Systems

- **Host-based IDS:**

- monitor single host activity

- **Network-based IDS:**

- monitor network traffic

- **Distributed or hybrid:**

- Combines information from a number of sensors, often both host and network based, in a central analyzer that is able to better identify and respond to intrusion activity

**Comprises three logical components:**

- **Sensors: collect data**
- **Analyzers: determine if intrusion has occurred**
- **User interface: view output or control system behavior**

# IDS requirements

---

- Run continually with minimum (human) supervision
- Be fault tolerant (recover from crashes)
- Resist subversion (monitor itself from change by intruder)
- Impose a minimal overhead on system
- Configurable according to system security policies
- Adapt to changes in systems and users
- Scale to monitor a large number of systems
- Provide a graceful degradation of service (if one component fails others should continue to work)
- Allow dynamic reconfiguration

# Detection techniques

---

## Anomaly (behavior) detection

- Involves the collection of data relating to the behavior of legitimate users over a period of time
- Current observed behavior is analyzed to determine whether this behavior is that of a legitimate user or that of an intruder

## Signature/heuristic detection

- Uses a set of known malicious data patterns or attack rules that are compared with current behavior
- Can only identify known attacks for which it has patterns or rules (signature)

# Anomaly detection

## Threshold detection

- checks excessive event occurrences over time
- alone a crude and ineffective intruder detector
- must determine both thresholds and time intervals
- lots of false positive/false negative may be possible

## Profile based

- characterize past behavior of *users/groups*
- then detect significant deviations
- based on analysis of audit records: ***gather metrics***

## Example of metrics

**Counters:** e.g., number of logins during an hour, number of times a cmd executed

**Gauge:** e.g., the number of outgoing messages [pkts]

**Interval time:** the length of time between two events, e.g., two successive logins

**Resource utilization:** quantity of resources used (e.g., number of pages printed)

Mean and standard deviations

# Example rules in a signature detection IDS

- Users should not be logged in more than one session
- Users do not make copies of system, password files
- Users should not read in other users' directories
- Users must not write other users' files
- Users who log after hours often access the same files they used earlier
- Users do not generally open disk devices but rely on high-level OS utilities



# Intrusion Prevention RL 10.4

# Firewalls and Intrusion Prevention Systems

- Effective means of protecting LANs
- Internet connectivity essential
  - For organization and individuals
  - But creates a threat
- Could secure workstations and servers
- Also use firewall as perimeter defence
  - Single choke point to impose security

# Firewall Capabilities & Limits

---

- Capabilities
  - Defines a single choke point
  - Provides a location for monitoring security events
  - Convenient platform for some Internet functions such as NAT, usage monitoring, IPSEC, VPNs
- Limitations
  - Cannot protect against attacks bypassing firewall (internal systems can dial-out to an ISP)
  - May not protect fully against internal threats
  - Improperly secured wireless LAN
  - Laptop, PDA, portable storage device infected outside then used inside

# Intrusion Prevention Systems (IPS)

---

- Security products (bringing in IDS & Firewall) which
  - Inline network-/host-based IDS that can block traffic
  - Functional addition to firewall that adds IDS capabilities
- Using IDS algorithms but can block or reject packets like a firewall
- May be network or host based

# Host-Based IPS

---

- Identifies attacks using both:
  - Signature techniques
    - malicious application packets
  - Anomaly detection techniques
    - behavior patterns that indicate malware
- Example of malicious behavior: buffer overflow, access to email contacts, directory traversal
- Can be tailored to the specific platform
  - e.g. general purpose, web/database server specific
- Can also sandbox applets to monitor behavior
- May give desktop file, registry, I/O protection

# Network-Based IPS

---

- Inline NIDS that can discard packets or terminate TCP connections
- Uses signature and anomaly detection
- May provide flow data protection
  - monitoring full application flow content
- Can identify malicious packets using:
  - pattern matching (for specific byte seq)
  - stateful matching (to stop attack streams rather than a single pkts)
  - protocol anomaly (deviations from stds)
  - traffic anomaly (unusual traffic like a UDP floods)

# Thank You!



**BITS** Pilani

Pilani | Dubai | Goa | Hyderabad

SS ZG 566

# Secure Software Engineering

T V Rao



# **Definitions and concepts of security - Part 1**

## **RL 1.1.1**

# Security Problem

---

Organizations store, process, transmit their most sensitive information using software-intensive systems.

Private citizens depend on software to shop, bank, invest, and carry out most personal and social activities

Global connectivity makes the sensitive information and software systems vulnerable to unintentional and unauthorized use.

# Security Problem

---

As per some experts, we are in era of

- Information warfare
- Cyber terrorism
- Computer crime

Terrorists, Organized criminals, other criminals are targeting software-intensive systems and are able to gain entry.

- There are many systems which can not resist attacks

# Growth of Threats

---

Growing internet connectivity of computers and networks and dependence on network-enabled services(e.g. email, online transactions) means

- Increased number and sophistication of attack methods

Growing trend in which system accepts updates and extensions

- Each new extension adds new capabilities, new interfaces, and thus new risks

Unbridled growth in the size and complexity of software systems

- More lines of code produce more bugs and vulnerabilities

# Security Principles

---

Saltzer and Schroeder defined security as “techniques that control who may use or modify the computer or the information contained in it”

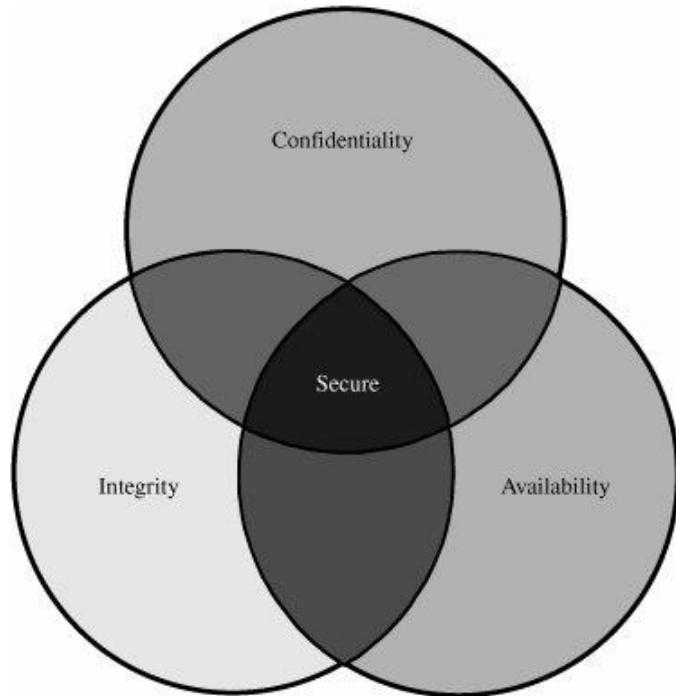
Described the three main categories of concern:

- Confidentiality
- Integrity
- Availability

Saltzer and Schroeder, “The Protection of Information in Computer Systems.” *Communications of the ACM*, 1974

# Security Principles

---



## **Integrity**

Information be protected from improper modification

## **Availability**

Available to user or program with legitimate right

## **Confidentiality**

Protection of data from unauthorized disclosure

### **Relationship Between Confidentiality, Integrity, and Availability**

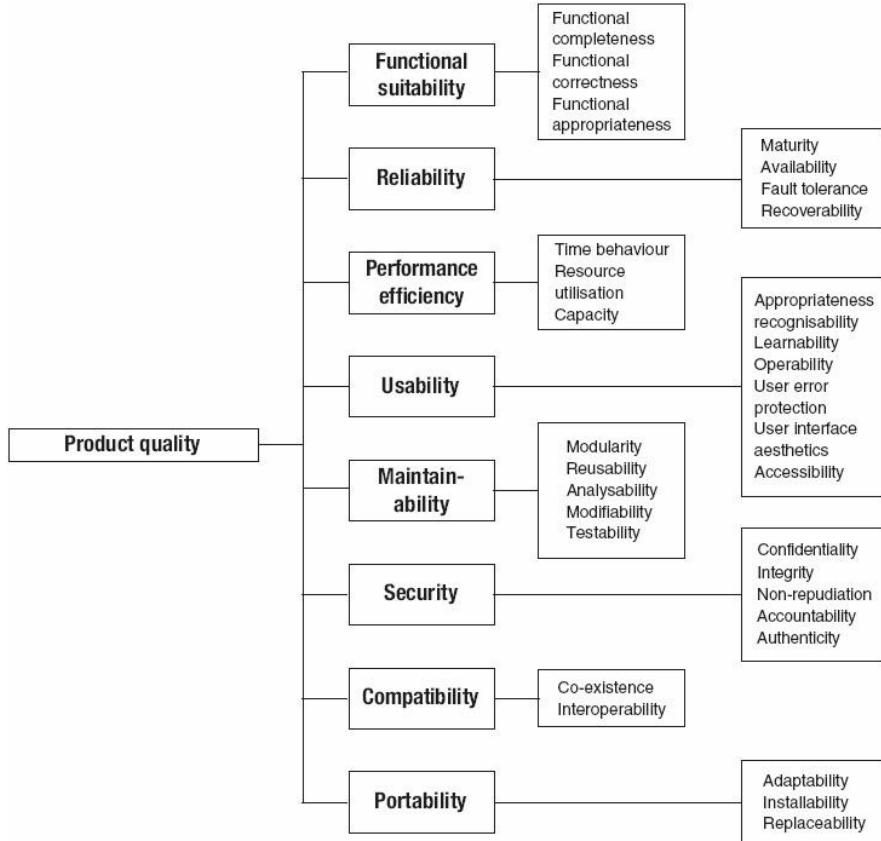
# Security Principles

---

Two additional properties commonly associated with human users are required in software entities that act as users, e.g. proxy agents, web services etc.

- Accountability : All security-relevant actions of the software-as-user must be recorded and tracked with attribution, both while and after the recorded action occurs
- Non-repudiation : Ability to prevent the software-as-user from disproving or denying responsibilities for actions it has performed

# Product quality model of ISO/IEC 25010



# Software Assurance

---

The Committee on National Security Systems (CNSS) defines software assurance as follows:

“Software assurance (SwA) is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at any time during its life cycle, and that the software functions in the intended manner.”

Software assurance includes software reliability, software safety, and software security

- Software assurance becomes important since critical infrastructure (viz. power, communication etc.) depend on software-intensive systems

# Processes for Secure Software

The most critical difference between secure and insecure software lies in the nature of the processes and practices used to specify, design, and develop the software

- Gortzel[2006]

Software vulnerabilities can originate from

- Decisions made by software engineers
- Flaws introduced in specification & design
- Faults from developed code
- Choice of programming language, development tools, operational environment etc.

# Security Concepts and Relationships

The **attack surface** of a software environment is the sum of the different points (the "attack vectors") where an unauthorized user (the "attacker") can try to enter data to or extract data from an environment.

An **attack vector** is a path or means by which a hacker (or cracker) can gain access to a computer or network server in order to deliver a payload or malicious outcome.

If an attack vector is thought of as a guided missile (e.g. email), its payload can be compared to the warhead (e.g. malicious attachment) in the tip of the missile.

# The Asymmetric Problem of Security

No matter how much effort we spend, we will never get code 100 percent correct

This is an asymmetric problem

- **We** must be 100 percent correct, 100 percent of time, on a schedule, with limited resources, only knowing what we know today
  - And the product has to be reliable, supportable, compatible, manageable, affordable, accessible, usable, global, doable, deployable.
- **They** can spend as long as they like to find one bug, with the benefit of future research

– [www.microsoft.com](http://www.microsoft.com)

# The Asymmetric Problem of Security

---

In general

- Cost for attacker to build attack is very low
- Cost to your users is very high
- Cost of reacting is higher than cost of defending

# Security on Cloud (Example)

---

Code Spaces kept up their security measures, ensured that their server security was tight, and relied on Amazon for the bulk of their infrastructure -- like thousands of other companies.

The attack that brought Code Spaces under was as simple as gaining access to its AWS control panel.

Code Spaces was built mostly on AWS, using storage and server instances to provide its services. Those server instances weren't hacked, nor was Code Spaces' database compromised or stolen.

Attacker gained access to the company's AWS control panel & deleted resources on the cloud.

**The demise of Code Spaces at the hands of an attacker shows that, in the cloud, off-site backups and separation of services could be key to survival**

# Examples of threats

	Availability	Confidentiality	Integrity
Hardware	Equipment is stolen or disabled, thus denying service.	An unencrypted CD-ROM or DVD is stolen.	
Software	Programs are deleted, denying access to users.	An unauthorized copy of software is made.	A working program is modified, either to cause it to fail during execution or to cause it to do some unintended task.
Data	Files are deleted, denying access to users.	An unauthorized read of data is performed. An analysis of statistical data reveals underlying data.	Existing files are modified or new files are fabricated.
Communication Lines and Networks	Messages are destroyed or deleted. Communication lines or networks are rendered unavailable.	Messages are read. The traffic pattern of messages is observed.	Messages are modified, delayed, reordered, or duplicated. False messages are fabricated.

# Attack trees

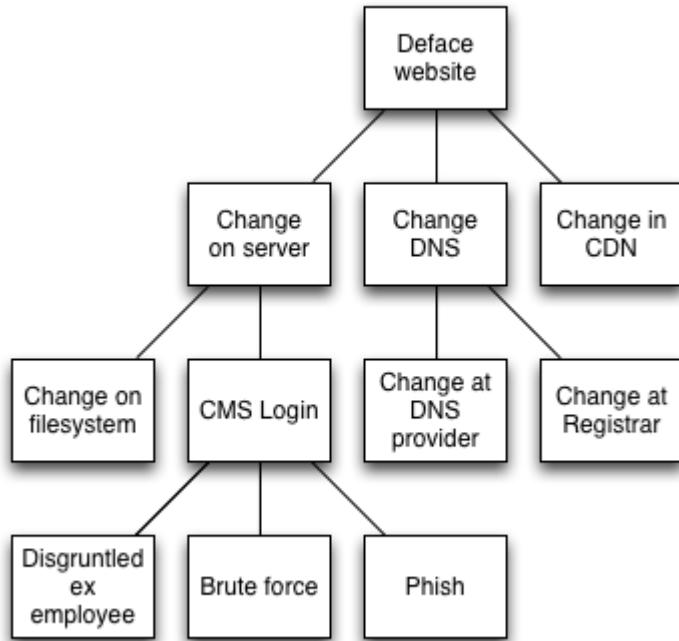
---

A branching, hierarchical data structure that represents a set of potential vulnerabilities

Objective: to effectively exploit the info available on attack patterns

- published on CERT or similar forums
- Security analysts can use the tree to guide design and strengthen countermeasures

# An attack tree (to deface a web site)



Content delivery network (CDN) :  
System of distributed servers  
(network) that deliver webpages  
and other Web content to user  
based on the geographic locations

Domain Name System (DNS) :  
Hierarchical decentralized naming  
system for computers, services, or  
any resource connected to the  
Internet or a private network

CMS : Content Management  
System

# Malicious software

---

Programs exploiting system vulnerabilities

Known as malicious software or malware

- program fragments that need a host program
  - e.g. viruses, logic bombs, and backdoors
- independent self-contained programs
  - e.g. worms, bots
- replicating or not

Sophisticated threat to computer systems

# Malware Terminology

---

Virus: *attaches itself to a program*

Worm: *propagates copies of itself to other computers*

Logic bomb: *“explodes” when a condition occurs*

Trojan horse: *fakes/contains additional functionality*

Backdoor (trapdoor): *allows unauthorized access to functionality*

Mobile code: *moves unchanged to heterogeneous platforms*

Auto-rooter Kit (virus generator): *malicious code (virus) generators*

Spammer and flooder programs: *large volume of unwanted “pkts”*

Keyloggers: *capture keystrokes*

Rootkit: *sophisticated hacker tools to gain root-level access*

Zombie: *software on infected computers that launch attack on others (aka bot)*

# Worms

---

Replicating program that propagates over net

- using email, remote exec, remote login

Has phases like a virus:

- dormant, propagation, triggering, execution
- propagation phase: searches for other systems, connects to it, copies self to it and runs

May disguise itself as a system process

Concept seen in Brunner's "Shockwave Rider"

Implemented by Xerox Palo Alto labs in 1980's

# Software Engineering - Definitions

---

- (1969 – Fritz Bauer) Software engineering is the establishment and use of *sound engineering principles* in order to obtain *economically* software that is *reliable* and works *efficiently* on *real machines*
- (IEEE) The application of a *systematic, disciplined, quantifiable* approach to the *development, operation, and maintenance* of software; that is, the application of engineering to software

# **Knowledge Areas in v3(2014) of SWEBOk**

---

<b>Requirements</b>	<b>Configuration Management</b>
<b>Design</b>	<b>Quality</b>
<b>Construction</b>	<b>Processes</b>
<b>Testing</b>	<b>Models &amp; Methods</b>
<b>Maintenance</b>	<b>Engineering Management</b>
	<b>Project Management</b>
	<b>Economics</b>

# Emergent properties of software

Some requirements represent *emergent properties* of software—that is, requirements that cannot be addressed by a single component but that depend on how all the software components interoperate. The throughput requirement for a call center would, for example, depend on how the telephone system, information system, and the operators all interacted under actual operating conditions. Emergent properties are crucially dependent on the system architecture.

Experts consider **Security** as an *emergent property* of the software.

# Software quality

---

Software quality may refer:

- to desirable characteristics of software products,
- to the extent to which a particular software product possess those characteristics, and
- to processes, tools, and techniques used to achieve those characteristics

Experts defined variously

“conformance to requirements” - Phil Crosby

“achieving excellent levels of fitness for use” - Watt Humphrey

“market-driven quality” where the “customer is the final arbiter” - IBM

# Software quality

---

A healthy software engineering culture includes the understanding that tradeoffs among cost, schedule, and quality are a basic tenant of the engineering of any product.

The tradeoff is best decided by understanding four cost of quality categories: prevention, appraisal, internal failure, and external failure.

Prevention costs include investments in software process improvement efforts, quality infrastructure, quality tools, training, audits, and management reviews

Appraisal costs arise from project activities that find defects.

Costs of internal failures are those that are incurred to fix defects found during appraisal activities and discovered prior to delivery of the software product to the customer

External failure costs include activities to respond to software problems discovered after delivery to the customer.

# Requirement Models

---

**Scenario-based modeling** – represents the system from the user's point of view

**Flow-oriented modeling** – provides an indication of how data objects are transformed by a set of processing functions

**Class-based modeling** – defines objects, attributes, and relationships

**Behavioral modeling** – depicts the states of the classes and the impact of events on these states

# Use-Cases

---

A collection of user scenarios that describe the thread of usage of a system

Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way

Each scenario answers the following questions:

- Who is the primary actor, the secondary actor (s)?
- What are the actor’s goals?
- What preconditions should exist before the story begins?
- What main tasks or functions are performed by the actor?
- What extensions might be considered as the story is described?
- What variations in the actor’s interaction are possible?
- What system information will the actor acquire, produce, or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

# Misuse case (techtarget.com)

---

**Name:** Attack on "forgot password" functionality

**Summary:** A malicious user tries to attack the "forgot password" functionality in order to gain access to the Web application or guess a valid e-mail address

**Author:** Anurag Agarwal

**Date:** April 15, 2006

## Possible Attacks:

- SQL injection attack
- Brute force attack to guess a valid user
- Sniffing attack on e-mail sent with password on an insecure transmission channel

**Trigger Point:** Can happen anytime

**Preconditions:** None

## Assumptions:

- The attacker can perform this attack remotely over the Internet
- The attacker can be an anonymous user

# Misuse case (techtarget.com)

---

## Worst case threat (post condition) :

- Attacker gains entry into the company database and steals sensitive information
- Attacker is able to modify an existing e-mail address to its own e-mail address and mails the password to himself to gain unauthorized entry into the system

## Related business rule:

- The system should e-mail the password to a valid e-mail address entered

## Capture guarantee (post condition) :

- Attacker cannot gain access to the database to steal or modify information
- Attacker cannot identify the e-mail address of a valid user
- Attacker cannot view the password sent in an e-mail to an e-mail address of a valid user

# Misuse case (techtarget.com)

---

## Potential misuser profile:

- Script kiddie
- Skilled attacker

**Threat level:** High

## Mitigation steps:

- SQL injection attack
  - List all the mitigation steps to avoid a SQL injection attack.
- Brute force attack
  - Accept first name, last name along with e-mail address
  - Have proper error handling so as not to reveal information to the attacker
  - Delay 3 to 5 seconds before re-entering the e-mail address
  - Lock out page for that IP address after 10 attempts
- Sniffing attack
  - Send password e-mail on a secure transmission channel with strong encryption

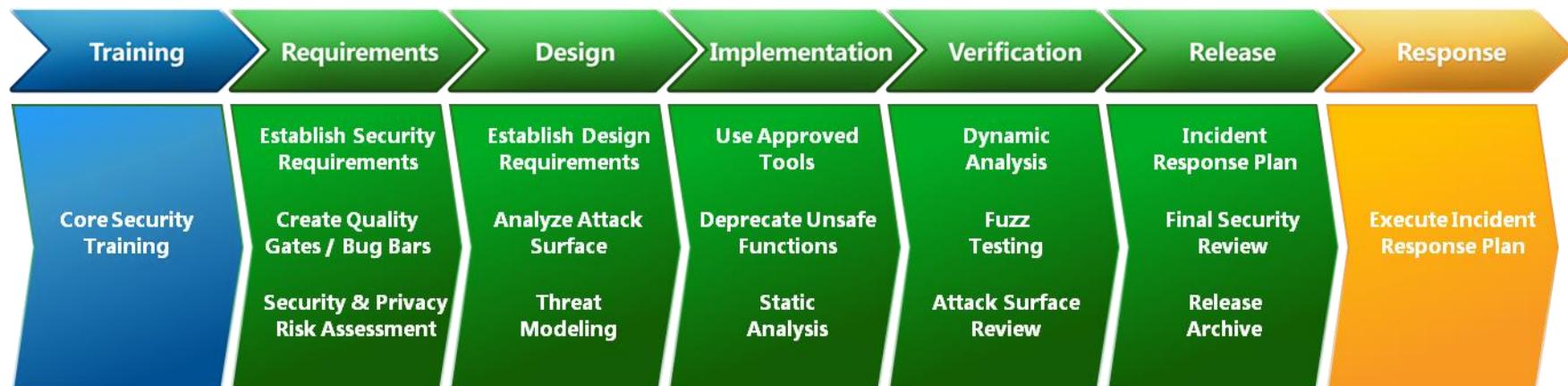
# Security Development Lifecycle

The Security Development Lifecycle (SDL) is a security assurance process developed by Microsoft as a company-wide initiative and a mandatory policy to reduce the number and severity of vulnerabilities in software products.

The Microsoft SDL is based on three core concepts —

- *education,*
- *continuous process improvement, and*
- *accountability*

# Security Development Lifecycle



# Securing a Computer Based System

A computer-based system has three separate but valuable components (Assets) :

- Hardware
- Software
- Data

## Vulnerabilities

- Weaknesses in a system that may be able to be *exploited* in order to cause loss or harm
  - e.g., a file server that doesn't authenticate its users

## Threats

- A loss or harm that might befall a system
  - e.g., users' personal files may be revealed to the public

# Characteristics of Computer Intrusion

By computing system, we include

- Hardware
- Software
- Storage media
- Data and
- People

A system is most vulnerable at its weakest point

- A robber will not attempt to penetrate a 2-inch-thick metal door if a window gives easy access

# Threat Model

---

When designing a system, we need to state the threat model

## Threat Model

- Set of threats we are undertaking to defend against
- Whom do we want to prevent from doing what?

## Attack

- An action which exploits a vulnerability to execute a threat
- e.g., telling the file server you are a different user in an attempt to read or modify their files

# Threats to Assets

---

According to Pfleeger, the threats are

- **Interruption** – an asset is destroyed, unavailable or unusable (*availability*)
- **Interception** – unauthorized party gains access to an asset (*confidentiality*)
- **Modification** – unauthorized party tampers with asset (*integrity*)
- **Fabrication** – unauthorized party inserts counterfeit object into the system (*authenticity*)

# Who are Attackers

---

One approach to prevention is to understand who carries out attacks and why

## Amateurs

- Ordinary computer professionals or users who, while doing their jobs, discover they have access to something valuable. Amateurs may be disgruntled employees who vow to get even with management

## Crackers or Malicious Hackers

- Attempt to access computing facilities for which they have not been authorized (often students who see it as victimless crime). Some carry out for curiosity, personal gain or self-satisfaction

# Who are Attackers

---

## Career Criminals

- Understands the targets of computer crime; often begin as computer professionals, then shift to crime finding payoff. “They don’t want to write a worm that destroys your hardware. They want to assimilate your computers and use them to make money”

## Terrorists

- They use computers in three ways
  - Targets of attack – denial of service, web site defacement attacks are popular to attract attention to the cause and bring undesired negative attention to the targets of attack
  - Propaganda vehicles – inexpensive way to get a message to many
  - Methods of attack – use computers to launch attacks

# Structured Approach to Threat Modeling

People often use an approach centered on

- Models of their assets (Valuable things they have),
- Models of attackers (People who might go after assets), or
- Models of their software (Common way to attack is via the deployed software)

Centering on one of these is preferable to using approaches that combine them because the combinations tend to be confusing

According to Adam Shostack, first two sets of models help in engaging with non-technical people and third type of models are important for software development

# What Threat Modelling is (not)

What threat modelling is	What threat modelling is not
A team activity	An activity performed by a single team member in isolation
An activity that helps identify security vulnerabilities in a variety of software applications	Just for large software projects
An activity that should be performed for every iteration or sprint during agile development	An activity done once during the lifecycle of the project

# Responsibilities

---

## Building a threat model (at Microsoft)

- Program Manager (PM) owns overall process
- Testers
  - Identify threats in analyze phase
  - Use threat models to drive test plans
- Developers create diagrams

# Customers / Work Products

---

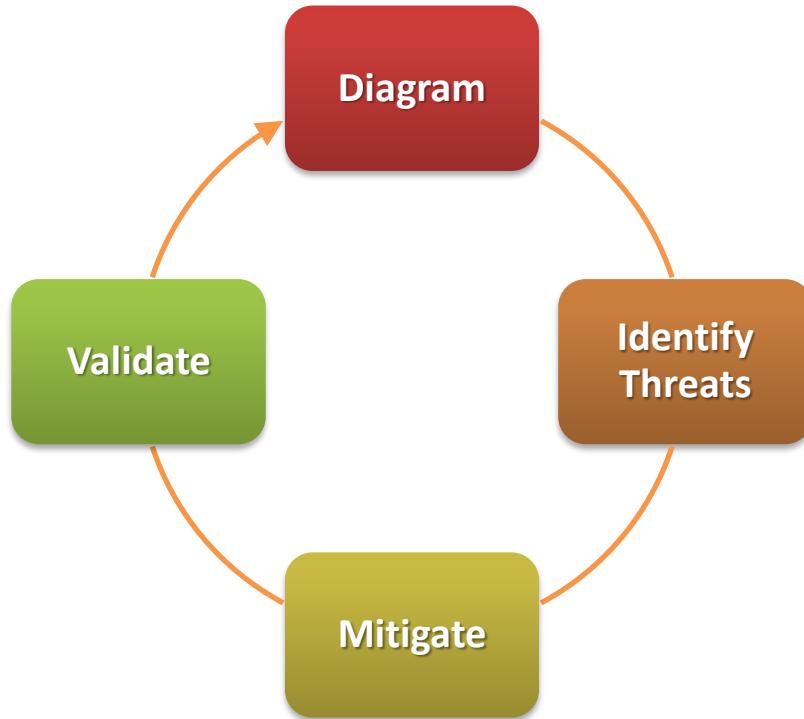
## Customers for threat models

- Your team
- Other features, product teams
- Customers, via user education
- “External” quality assurance resources,  
such as pen testers

## Threat model documentation

- The product as a whole
- The security-relevant features
- The attack surfaces

# The Process in a Nutshell



# Importance of Requirements Engineering

Some studies have shown that requirements engineering defects cost 10 to 200 times as much to correct once the system has become operational than if they were detected during requirements development.

According to Charette[2005], Requirements problems are among the top causes of the following undesirable phenomena

- Projects are significantly over budget, go past schedule, have significantly reduced scope, or are cancelled
- Development teams deliver poor-quality applications
- Products are not significantly used once delivered

# Requirements Engineering Challenges

Requirements Engineering on individual projects often suffers from the following problems:

- Requirements identification typically does not include all relevant stakeholders and does not use the most modern or efficient techniques.
- Requirements are often statements describing architectural constraints or implementation mechanisms rather than statements describing what the system must do.
- Requirements are often directly specified without any analysis or modeling. When analysis is done, it is usually restricted to functional end-user requirements, ignoring
  - 1) quality requirements such as security,
  - 2) other functional and nonfunctional requirements, and
  - 3) architecture, design, implementation, and testing constraints.

# Requirements Engineering Challenges

---

Requirements specification is typically haphazard, with specified requirements being

- ambiguous,
- incomplete (e.g., nonfunctional requirements are often missing),
- inconsistent,
- not cohesive,
- infeasible,
- obsolete,
- neither testable nor capable of being validated, and
- not usable by all of their intended audiences.

Requirements management is typically weak, with ineffective forms of data capture

- e.g., in one or more documents (rather than in a database or tool) and missing attributes.
- often limited to tracing, scheduling, and prioritization, without change tracking or other configuration management.

# Quality Requirements

---

Project teams often neglect *quality* requirements, such as performance, safety, security, reliability, and maintainability.

- Developers of certain kinds of mission-critical systems and systems in which human life is involved, such as the space shuttle, have long recognized the importance of quality requirements and have accounted for them in software development.
- In many other systems, however, quality requirements are treated in an inadequate way. Hence we see the failure of software associated with power systems, telephone systems, unmanned spacecraft, and so on.

This inattention to quality requirements is exacerbated by the desire to keep costs down and meet aggressive schedules.

# Security Requirements Engineering

According to BSI[09], if security requirements are not effectively defined, the resulting system cannot be **evaluated** for success or failure prior to its implementation

Operational environments and business goals often change **dynamically**, with the result that security requirements development is not a one-time activity.

Requirements engineering research and practice pay a lot of attention to the functionality of the system from the user's perspective, but little attention is devoted to what the system should **not** do [Bishop 2002]

# Security Requirements Engineering

Users have implicit assumptions for the software applications and systems to be secure and are surprised when they are not. These user assumptions need to be translated into security requirements for the software systems when they are under development.

It is important for requirements engineers to think about the attacker's perspective and not just the functionality of the system from the end-user's perspective.

- An attacker is not particularly interested in functional features of the system, unless they provide an avenue for attack. Instead, the attacker typically looks for defects and other conditions outside the norm that will allow a successful intrusion to take place.

# Security Is Not a Set of Features

Security features such as password protection, firewalls, virus detection tools etc. are, in fact, not security requirements.

They are rather implementation mechanisms that are intended to satisfy unstated requirements, such as authenticated access

A systematic approach to security requirements engineering will help avoid the problem of generic lists of features and take into account the attacker's perspective.

No convenient security pull-down menu that will let you select “security” and do the needful.

# Security Is Not a Set of Features

Security is an **emergent** property of a system, not a feature

Because security is not a feature, it cannot be bolted on after other software features are codified, nor can it be **patched** in after attacks have occurred in the field. Instead, security must be built into the product from the ground up

Most cost-effective approach to software security incorporates thinking beyond normative features and maintains that thinking throughout the development process

Every time a new requirement, feature, or use case is created, the developer or security specialist should spend some time thinking about how that feature might be unintentionally misused or intentionally abused

# Thinking Beyond Normal

---

When we design and analyze a system, we're in a great position to know our systems better than potential attackers do.

We can leverage this knowledge to the benefit of security and reliability, by asking and answering the critical questions:

- Which assumptions are implicit in our system?
- Which kinds of things make our assumptions false?
- Which kinds of attack patterns will an attacker bring to bear?

# Thinking like an attacker

---

System's creators are not the best security analysts of that system.

'Thinking like an attacker' is extremely difficult for those who have built up a set of implicit assumptions

System Creators make excellent subject matter experts (SMEs).

Together, SMEs and security analysts can ferret out base assumptions in a system under analysis and think through the ways an attacker will approach the software

# SQUARE Process Steps

---

1. Agree on definitions
2. Identify security goals
3. Develop artifacts to support security requirements definition
4. Perform (security) risk assessment
5. Select elicitation techniques
6. Elicit security requirements
7. Categorize requirements as to level (e.g., system, software) and whether they are requirements or other kinds of constraints
8. Prioritize requirements
9. Inspect requirements

# Understanding Software Architecture

The software architecture of a program or computing system is the structure or structures of the system which comprise

- The software components
- The externally visible properties of those components
- The relationships among the components

Software architectural design represents the structure of the data and program components that are required to build a computer-based system

An architectural design model is transferable

- It can be applied to the design of other systems
- It represents a set of abstractions that enable software engineers to describe architecture in predictable ways

# Importance of Software Architecture

Representations of software architecture are an enabler for communication between all stakeholders interested in the development of a computer-based system

The software architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity

The software architecture constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together

# General Objectives of Software Architecture and Design

---

## Completeness

- Supports the full scope of the defined requirements

## Stability

- Consistently performs as intended within its defined operational context

## Flexibility

- Can adapt to changing conditions
- Decompose such that selected components can be replaced going forward with minimal impact to the software

## Extensibility

- Leverages industry standards
- Long-lived and resistant to obsolescence

## Scalability

- Operates effectively at any size and load

# Security-Specific Objectives of Software Architecture and Design

---

## Comprehensive functional security architecture

- Security features and capabilities are fully enabled

## Attack resistance

- Contains minimal security weaknesses that could be exploited

## Attack tolerance

- While resisting attack, software function and capability are not unduly affected

## Attack resilience

- In the face of successful attack, the effects on the software are minimized
- Operates effectively at any size and load

# Architectural Risk Analysis

---

- The risk assessment methodology (Build Security In) encompasses six fundamental activity stages:
  - application characterization
  - architectural vulnerability assessment
  - threat analysis
  - risk likelihood determination
  - risk impact determination
  - risk mitigation

<https://buildsecurityin.us-cert.gov/articles/best-practices/architectural-risk-analysis/architectural-risk-analysis>

# Known Vulnerability Analysis

---

- Consider the architecture against a body of known bad practices or known good principles for confidentiality, integrity, and availability
  - e.g., the good principle of "*least privilege*" prescribes that all software operations should be performed with the least possible privilege.
  - Diagram the system's major modules, classes, or subsystems and circle areas of high privilege versus areas of low privilege. Consider the boundaries between these areas and the kinds of communications across those boundaries.

# Fundamental Design Concepts

---

Abstraction—data, procedure, control

Patterns—"conveys the essence" of a proven design solution

Separation of concerns—any complex problem can be more easily handled if it is subdivided into pieces

Modularity—compartmentalization of data and function

Information Hiding—controlled interfaces

Functional independence—single-minded function and low coupling

Refinement—elaboration of detail for all abstractions

Aspects—a mechanism for understanding how global requirements affect design

Refactoring—a reorganization technique that simplifies the design

*The beginning of wisdom (for a software engineer) is to recognize the difference between getting program to work, and getting it right*  
— M A Jackson

# Design Principles for Software Security

---

- Securing the Weakest Link
- Defense in Depth
- Failing Securely
- Least Privilege
- Separation of Privilege
- Economy of Mechanism
- Least Common Mechanism
- Reluctance to Trust
- Never Assuming that your Secrets are Safe
- Complete Mediation
- Psychological Acceptability
- Promoting Privacy

<https://buildsecurityin.us-cert.gov/articles/knowledge/principles/design-principles>

# Classes of Patterns

---

**Design-level patterns.** Design-level patterns describe how to design and implement pieces of a high-level system component, that is, they address problems in the internal design of a single high-level component, not the definition and interaction of high-level components themselves.

- Secure Factory
- Secure Strategy Factory
- Secure Builder Factory
- Secure Chain of Responsibility
- Secure State Machine
- Secure Visitor

# Thank You!