



SEZG566/SSZG566

Secure Software Engineering

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

T V Rao



- *The slides presented here are obtained from the authors of the books, product documentations, and from various other contributors. I hereby acknowledge all the contributors for their material and inputs.*
- *I have added and modified slides to suit the requirements of the course.*



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Definitions and concepts of security

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.

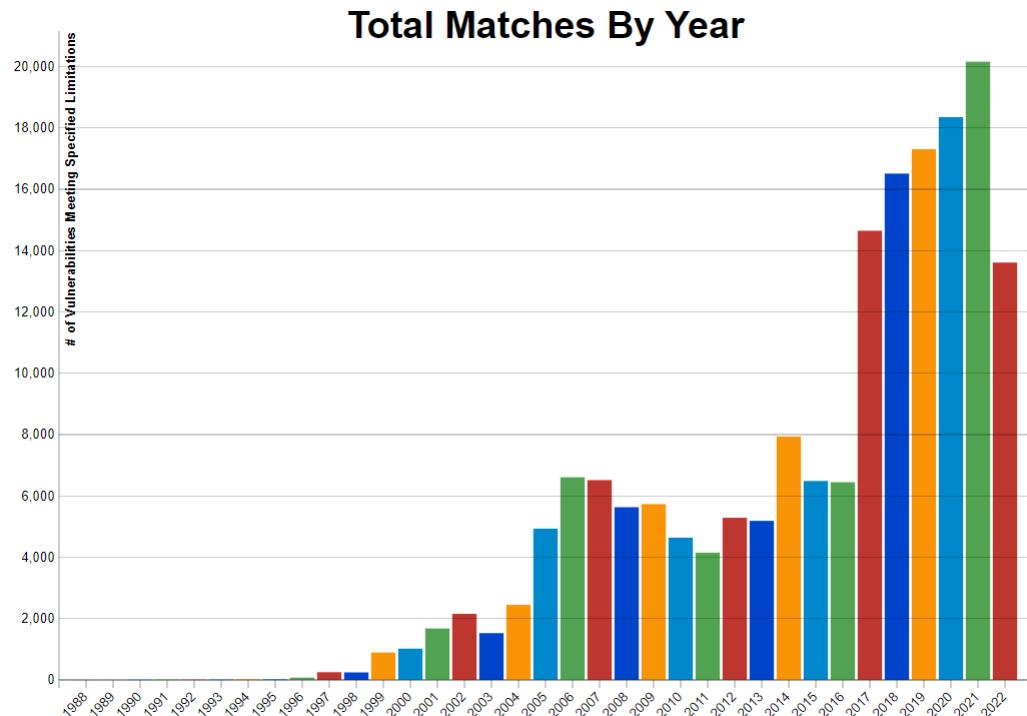
As per some experts, we are in era of

- 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

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

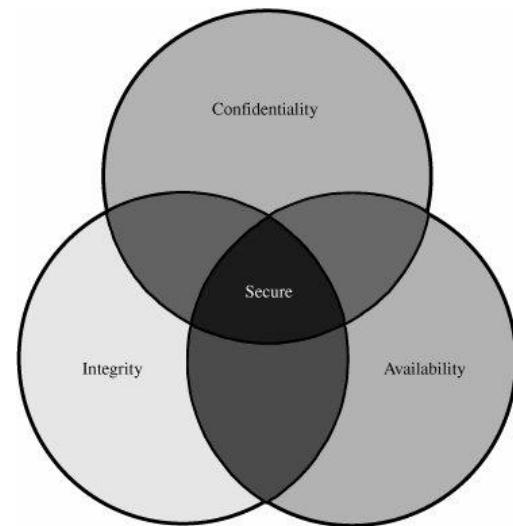
Source : National Vulnerability Database of NIST (National Institute of Standards and Technology)

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



Security implies Confidentiality, Integrity, and Availability

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

Key Security Concepts

Confidentiality

- Preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information

Integrity

- Guarding against improper information modification or destruction, including ensuring information nonrepudiation and authenticity

Availability

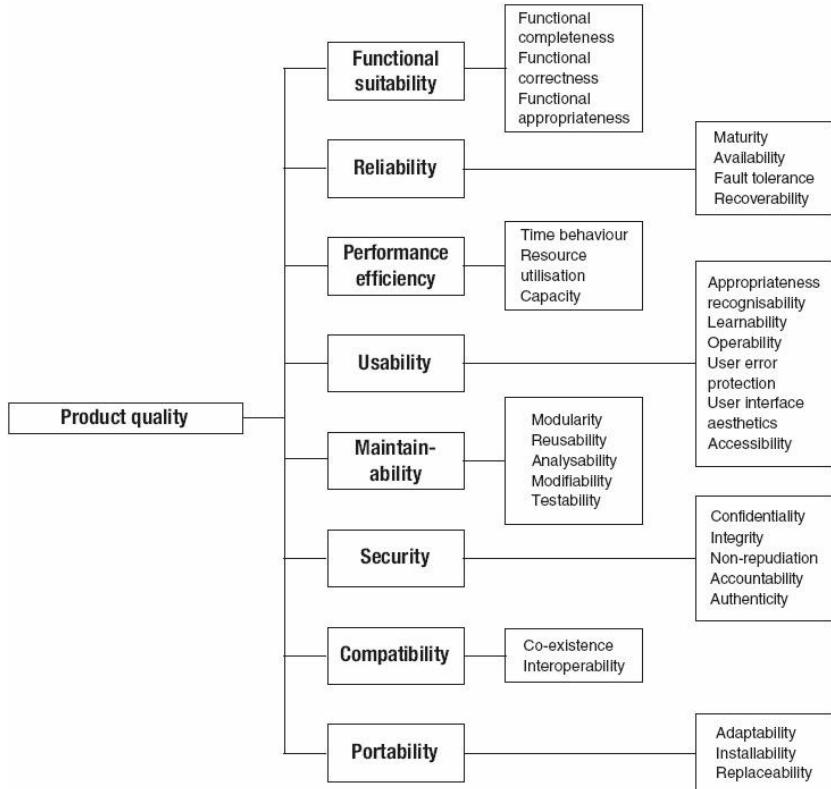
- Ensuring timely and reliable access to and use of information

Two additional properties

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 Department of Defense (DoD) defines software assurance as follows:

- *System assurance (SA) is the justified confidence that the system functions as intended and is free of exploitable vulnerabilities, either intentionally or unintentionally designed or inserted as part of the system at any time during the life cycle. This ideal of no exploitable vulnerabilities is usually unachievable in practice, so programs must perform risk management to reduce the probability and impact of vulnerabilities to acceptable levels*

A more practical definition (given by SEI CMM) emphasizes risk management by balancing cost and potential loss

- *the level of confidence we have that a system behaves as expected and the security risks associated with the business use of the software are acceptable*

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

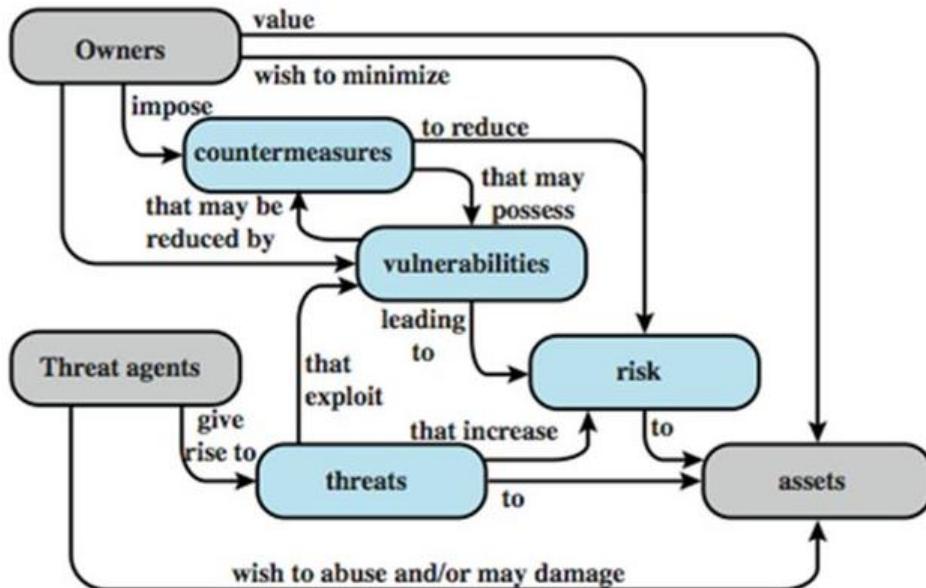
The most critical difference between secure and insecure software lies in the nature of the processes and practices used to specify, design, and develop the software

- 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 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

Privacy – GDPR

What does GDPR Compliance Look Like for Companies?

- “Data protection will be as significant as antitrust or anti-corruption in terms of compliance risk.” – *Hunton & Williams*
- Many small companies including news sites blocking EU users
- Product team compliance includes
 - Changes, often major, to back-end data logging
 - User interface changes
 - New tools for access, correction, portability, etc.
- Legal team compliance includes
 - Data Impact Assessments
 - Internal record-keeping
 - Renegotiating commercial contracts
 - Changing user Terms of Service
 - In some cases appointing Data Protection Officer resident in EU



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Threats to Software/Assets

The Asymmetric Problem of Security

Basics of Secure Design Development Test

– www.microsoft.com

Hacking a Politician email

Hacker chooses to hack politician's mail account and possibly impact US 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/>



How Apple and Amazon Security Flaws Led to My Epic Hacking _ WIRED

If it wasn't clear before, it certainly is now: Your username and password are almost impossible to keep safe.

Nearly 443,000 e-mail addresses and passwords for a Yahoo site [were exposed late Wednesday](#). The impact stretched beyond Yahoo because the site allowed users to log in with credentials from other sites -- which meant that user names and passwords for Yahoo ([YHOO, Fortune 500](#)), Google's ([GOOG, Fortune 500](#)) Gmail, Microsoft's ([MSFT, Fortune 500](#)) Hotmail, AOL (AOL) and many other e-mail hosts were among those posted publicly on a hacker forum.

What's shocking about the development isn't that usernames and passwords were stolen -- that [happens virtually every day](#). The surprise is how easily outsiders cracked a service run by one of the biggest Web companies in the world.

The group of seven hackers, who belong to a hacker collective called D33Ds Company, got into [Yahoo's Contributor Network](#) database by using a rudimentary attack called a [SQL injection](#).

Attack Surfaces

Consist of the reachable and exploitable vulnerabilities in a system

Examples:

Open ports on outward facing Web and other servers, and code listening on those ports

Services available on the inside of a firewall

Code that processes incoming data, email, XML, office documents, and industry-specific custom data exchange formats

Interfaces, SQL, and Web forms

An employee with access to sensitive information vulnerable to a social engineering attack

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

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.

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

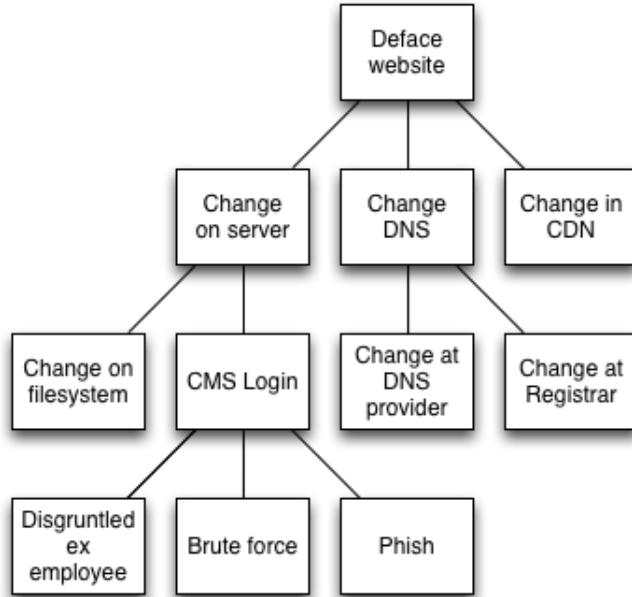
Attack trees

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

<http://ertw.com/blog/2015/01/06/thinking-about-cyber-security/>



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Malware Nomenclature

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-router 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)

<https://www.theguardian.com/world/2022/jan/14/ukraine-massive-cyber-attack-government-websites-suspected-russian-hackers>

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 structure (abstract)

```
program V :=  
  
{goto main;  
 1234567;  
  
  subroutine infect-executable :=  
    {loop:  
      file := get-random-executable-file;  
      if (first-line-of-file = 1234567)  
        then goto loop  
        else prepend V to file; }  
  
  subroutine do-damage :=  
    {whatever damage is to be done}  
  
  subroutine trigger-pulled :=  
    {return true if some condition holds}  
  
main:  main-program :=  
       {infect-executable;  
        if trigger-pulled then do-damage;  
        goto next;}  
  
next:  
}  
}
```

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

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

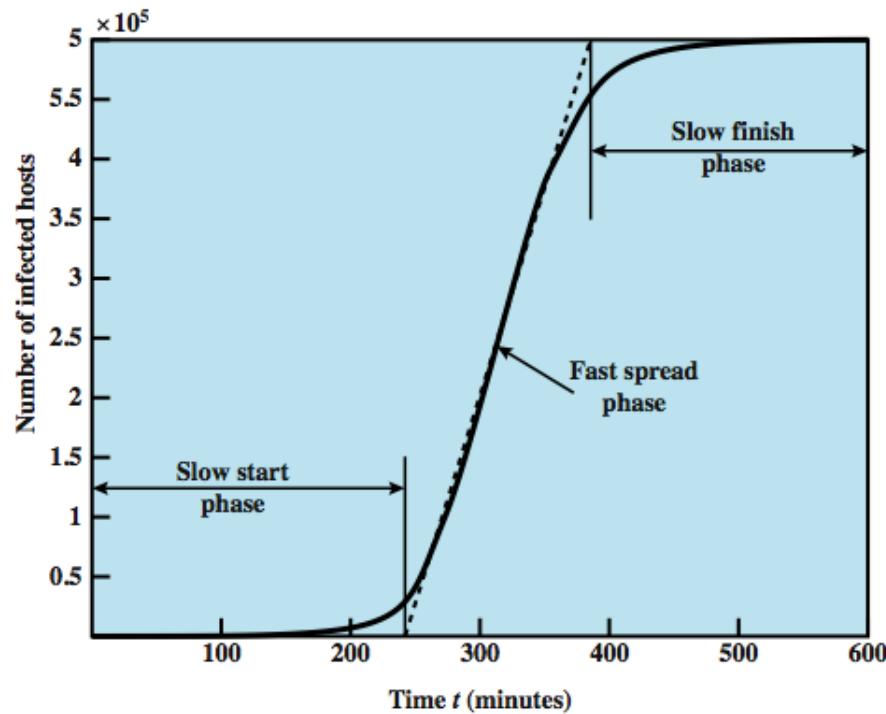
Various attacks on UNIX systems

- 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)



Recent Worm Attacks

Melissa	1998	E-mail worm First to include virus, worm and Trojan in one package
Code Red	July 2001	Exploited Microsoft IIS bug Probes random IP addresses Consumes significant Internet capacity when active
Code Red II	August 2001	Also targeted Microsoft IIS Installs a backdoor for access
Nimda	September 2001	Had worm, virus and mobile code characteristics Spread using e-mail, Windows shares, Web servers, Web clients, backdoors
SQL Slammer	Early 2003	Exploited a buffer overflow vulnerability in SQL server compact and spread rapidly
Sobig.F	Late 2003	Exploited open proxy servers to turn infected machines into spam engines
Mydoom	2004	Mass-mailing e-mail worm Installed a backdoor in infected machines
Warezov	2006	Creates executables in system directories Sends itself as an e-mail attachment Can disable security related products
Conficker (Downadup)	November 2008	Exploits a Windows buffer overflow vulnerability Most widespread infection since SQL Slammer
Stuxnet	2010	Restricted rate of spread to reduce chance of detection Targeted industrial control 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)



<https://www.gartner.com/en/articles/7-top-trends-in-cybersecurity-for-2022>

<https://www.gartner.com/en/newsroom/press-releases/2022-02-24-gartner-says-the-cybersecurity-leader-s-role-needs-to>

The DarkSide ransomware group was responsible for the Colonial Pipeline Company ransomware incident in May 2021, which led to the company's decision to proactively and temporarily shut down the 5,500-mile pipeline that carries 45 percent of the fuel used on the East Coast of the United States.



DarkSide Ransomware As a Service Group

SUBMIT TIPS VIA TELEPHONE OR THE FBI WEBSITE BELOW

Follow-on contacts to be established through WhatsApp, Telegram, Signal, or other platform of reporting party's choosing

1-800-CALL-FBI
(1-800-225-5324)

<https://tips.fbi.gov>

Supply Chain Attacks

A software supply chain attack occurs when a cyber threat actor infiltrates a software vendor's network and employs malicious code to compromise the software before the vendor sends it to their customers.

- Newly acquired software may be compromised from the outset, or a compromise may occur through other means like a patch or hotfix.
- The compromised software then compromises the customer's data or system
- These types of attacks affect all users of the compromised software and can have widespread consequences for government, critical infrastructure, and private sector software customers

https://www.cisa.gov/sites/default/files/publications/defending_against_software_supply_chain_attacks_508_1.pdf

Supply Chain Attacks

Lifecycle Phase	Example of Threat
Design	Hijacked Cellular Devices. 2016 – A foreign company designed software used by a U.S. cell phone manufacturer. The phones made encrypted records of text and call histories, phone details, and contact information and transmitted that data to a foreign server every 72 hours.
Development & Production	SolarWinds. 2020 – An IT management company was infiltrated by a foreign threat actor who maintained persistence in its network for months. The threat actor left the network only after it had compromised the company's build servers and used its update process to infiltrate customer networks.
Distribution	End-User Device Malware. 2012 – Researchers from a major U.S. software company investigating counterfeit software found malware preinstalled on 20 percent of devices they tested. The malware was installed in new desktop and laptop computers after they were shipped from a factory to a distributor, transporter, or reseller.
Acquisition & Deployment	Kaspersky Antivirus. 2017 – An overseas-based antivirus vendor was being used by a foreign intelligence service for spying. U.S. government customers were directed to remove the vendor's products from networks and disallowed from acquiring future products from that vendor.
Maintenance	Backdoors Embedded in Routine Maintenance Updates. 2020 – Thousands of public and private networks were infiltrated when a threat actor used a routine update to deliver a malicious backdoor.
Disposal	Sensitive Data Spillage. 2019 – A researcher bought old computers, flash drives, phones and hard drives, and found only two properly wiped devices out of 85 examined. Also found were hundreds of instances of personally identifiable information (PII) spillage, including Social Security numbers, passport numbers, and credit card numbers.

https://www.cisa.gov/sites/default/files/publications/defending_against_software_supply_chain_attacks_508_1.pdf

The Open Web Application Security Project (OWASP) is a worldwide not-for-profit charitable organization focused on improving the security of software. “Our mission is to make software security visible, so that individuals and organizations worldwide can make informed decisions about true software security risks.”

There are thousands of active wiki users around the globe who review the changes to the site to help ensure quality. Has a global group of volunteers with thousands of participants.

<https://us-cert.cisa.gov/>

(CISA is part of the Department of Homeland Security)



Department of Homeland Security (part of US government) offers information and campaigns for awareness of cybersecurity.

The cybersecurity part is available at

<https://www.dhs.gov/topic/cybersecurity>

CISA – Cybersecurity and Infrastructure Security Agency

“At the Computer emergency response teams (CERT) Division of the Software Engineering Institute (SEI) of Carnegie Mellon University(CMU), we study and solve problems with widespread cybersecurity implications, research security vulnerabilities in software products, contribute to long-term changes in networked systems, and develop cutting-edge information and training to help improve cybersecurity.”

“We are more than a research organization. Working with software vendors, we help resolve software vulnerabilities. We develop tools, products, and methods to help organizations conduct forensic examinations, analyze vulnerabilities, and monitor large-scale networks. We help organizations determine how effective their security-related practices are”.

<https://cert-in.org.in/>

The Indian Computer Emergency Response Team (CERT-IN) is an office within the Ministry of Electronics and Information Technology of the Government of India. It is the nodal agency to deal with cyber security threats like hacking and phishing. It strengthens security-related defence of the Indian Internet domain.



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

Computer Security: Principles and Practice by William Stallings, and Lawrie Brown
Pearson, 2018.

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

Threat Modelling by Adam Shostack, John Wiley 2014



Thank You!



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

SEZG566/SSZG566

Secure Software Engineering

T V Rao



- *The slides presented here are obtained from the authors of the books, product documentations, and from various other contributors. I hereby acknowledge all the contributors for their material and inputs.*
- *I have added and modified slides to suit the requirements of the course.*



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Phases in software development

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

Requirements	Configuration Management
Design	Quality
Construction	Processes
Testing	Models & Methods
Maintenance	Engineering Management
	Project Management
	Economics

Some Authors refer two columns as Primary & Supporting Activities
Pressman calls them Framework and Umbrella Activities

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.

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

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.

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.



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Software development – Supporting Activities

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



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Work products during SDLC

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

SafeHome (from Pressman)

SafeHome: The home security function would protect against and/or recognize a variety of undesirable “situations” such as illegal entry, fire, flooding, carbon monoxide levels, and others. It’ll use our wireless sensors to detect each situation. It can be programmed by the homeowner, and will automatically telephone a monitoring agency when a situation is detected.

Objects described for SafeHome might include the control panel, smoke detectors, window and door sensors, motion detectors, an alarm, an event (a sensor has been activated), a display, a PC, telephone numbers, a telephone call, and so on. The list of services might include configuring the system, setting the alarm, monitoring the sensors, dialing the phone, programming the control panel, and reading the display (note that services act on objects).

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 (searchsoftwarequality.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 (searchsoftwarequality.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 (searchsoftwarequality.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 Modeling

Data Flow Diagram

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

Process Specification

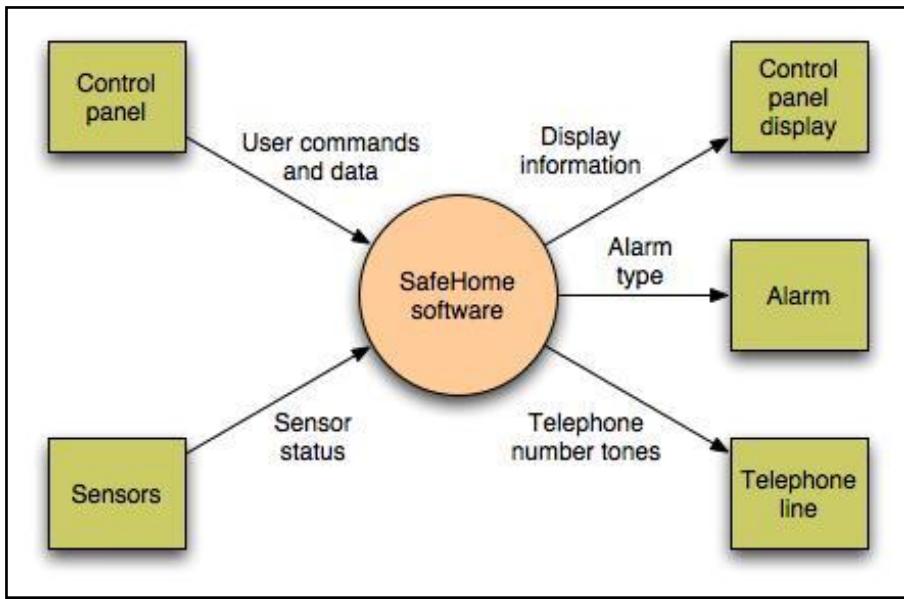
- Describes data flow processing at the lowest level of refinement in the data flow diagrams

Data Flow Modeling

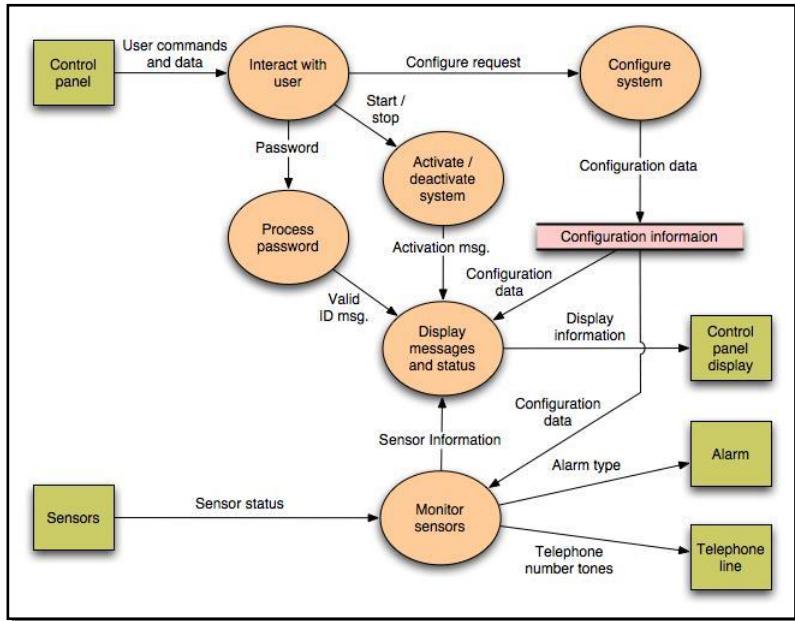
Guidelines

- 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)





BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Security implications to SDLC

What is OWASP SAMM?

SAMM stands for Software Assurance Maturity Model.

“Our mission is to provide an effective and measurable way for all types of organizations to analyze and improve their software security posture. We want to raise awareness and educate organizations on how to design, develop, and deploy secure software through our self-assessment model. SAMM supports the complete software lifecycle and is technology and process agnostic. We built SAMM to be evolutive and risk-driven in nature, as there is no single recipe that works for all organizations.”

OWASP 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 15 security practices, which are grouped into five business functions

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

OWASP SAMM Overview

Business functions	Governance	Design	Implementation	Verification	Operations
Practices	Strategy & Metrics Create & promote Measure & Improve	Threat Assessment Application risk profile Threat modeling	Secure Build Build process Software dependencies	Architecture Assessment Architecture validation Architecture compliance	Incident Management Incident detection Incident response
	Policy & Compliance Policy & standards Compliance management	Security Requirements Software requirements Supplier security	Secure Deployment Deployment process Secret management	Requirements-driven Testing Control verification Misuse/abuse testing	Environment Management Configuration hardening Patch & update
	Education & Guidance Training & awareness Organization & culture	Secure Architecture Architecture design Technology management	Defect Management Defect tracking Metrics & feedback	Security Testing Scalable baseline Deep understanding	Operational Management Data protection Legacy management
	Stream A Stream B	Stream A Stream B	Stream A Stream B	Stream A Stream B	Stream A Stream B

SAMM Checklist

OWASP SAMM Toolkit.xlsx

(<https://owaspSAMM.org/assessment/>)

OWASP SAMM Practices

SAMM prescribes methodology for practice that includes

- Assessment
- Results
- Success metrics
- Costs
- Personnel (Roles)
- Levels



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Security implications to SDLC

Security Development Lifecycle

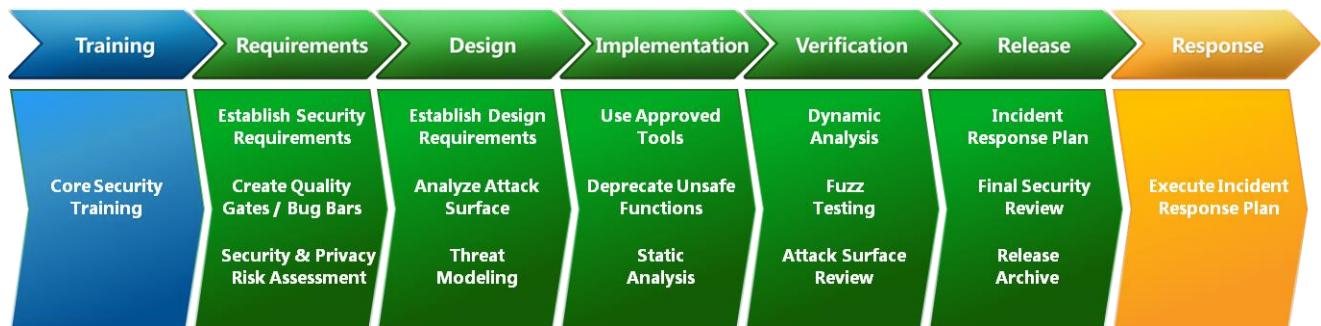
Security Development Lifecycle

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



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Security in Agile Development

The Manifesto for Agile Software Development

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

Kent Beck et al

What is “Agility”?

Effective (rapid and adaptive) response to change

Effective communication among all stakeholders

Drawing the customer onto the team

Organizing a team so that it is in control of the work performed

Yielding ...

Rapid, incremental delivery of software

Security in Agile Development

1. Utilize the hacker in that developer

A developer with the right tools, training and mindset can uncover the most common security pitfalls in code.

—Developers must be empowered through training and the provision of the right tools that will enable them to analyze code before submitting it to quality assurance or testing teams.

All development team members must be aware of the most common vulnerabilities in each system they develop.

—With developers' ratio to security specialists standing at about 100:1, it is only plausible to pass the responsibility of securing systems to the developers. They can then perform routine scans of their code and fix any issues that may arise during the core development process.

Team leaders can decide to incorporate the security team members into individual development teams to discover vulnerabilities early in development.

2. Always consider the “evil” user interacting with the system

Many developers envision a perfect software without taking into account the possibility of an “evil” user interaction

Team players should be made aware of the possible ways an attacker may interact with the system.

This is a continuous process starting with the statement of security-based user stories.

Only a few clients will talk about security in user stories. Security team has to identify critical points and valuable assets of the software that may be targeted by an attacker. They can enumerate possible attack goals and advise the development team in advance.

Security in Agile Development

3. Uphold continuous integration practices, tools and platforms

There are tools to integrate with development platforms and give insightful information about code.

These include code integration tools, code scanners and shared repositories.

Code analysis tools are available to help debug and refactor buggy code, thereby improving the system's overall safety.

Upholding continuous integration best practices work and help improve the quality of the software, thereby assuring the security of users, user data and vendors.

4. Review user stories with every iteration and adapt as necessary

Agile methodologies have short iterations, resulting in the release of new or improved software system features.

—Therefore, it is possible to review all user stories that necessitate iteration and how they relate to the system's general security before releasing it for public use.

The team leader must also organize peer review of code and how frequently this occurs. Two pairs of eyes are often better than one during code review.

The software must be adapted to better its reliability and security if vulnerabilities are discovered in the review process.

Security in Agile Development

5. Innovate with security

Agile teams neglect security due to absence of robust, agile security practices and testing tools.

Progressive teams have to innovate workable security policies and developing tools based on such policies.

Most CI tools are open to customization.

Therefore, an organization or team can choose a tool that works with their development practices and customize it to serve the most common security concerns affecting their code.

6. Cultivate the culture of security

Organizational culture influences how things are generally done.

If team leaders have high regard for secure systems, the juniors are likely to copy the same and practice it when writing code.

Educating the team and organizing seminars for security and agile development can also help members adopt the security needs of software quickly.

Every new team member should be taken through the necessary software security procedures and guidelines of organization.

Furthermore, combining all the other security aspects of agile development with a supportive and dynamic security culture will boost the general feel of your software.



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

DevOps and DevSecOps

DevOps

A set of practices that combines software development (Dev) and IT operations (Ops)



Why DevOps? Iterative and Rapid Software release cycles mean software moves between development team to IT operations team repeatedly. It means strong binding is needed between two teams.

DevOps is complementary with Agile software development; several DevOps aspects came from the Agile methodology

DevOps Practices

Continuous integration and continuous delivery (CI/CD)

- CI - merge code changes frequently into the main code. CD - frequent, automated deployment of new versions into a production environment

Version Control

- tracking revisions and change history to make code easy to review and recover

Agile software development

- emphasizes team collaboration, customer and user feedback, and rapid change through short release cycles

Infrastructure as code

- defines system resources and topologies in a descriptive manner that allows teams to manage those resources as they would code

Configuration management

- managing the state of resources in a system including servers, virtual machines, and databases

Continuous monitoring

- having real-time visibility into the performance and health of the entire stack, from the underlying infrastructure to higher-level software components

What is DevSecOps?

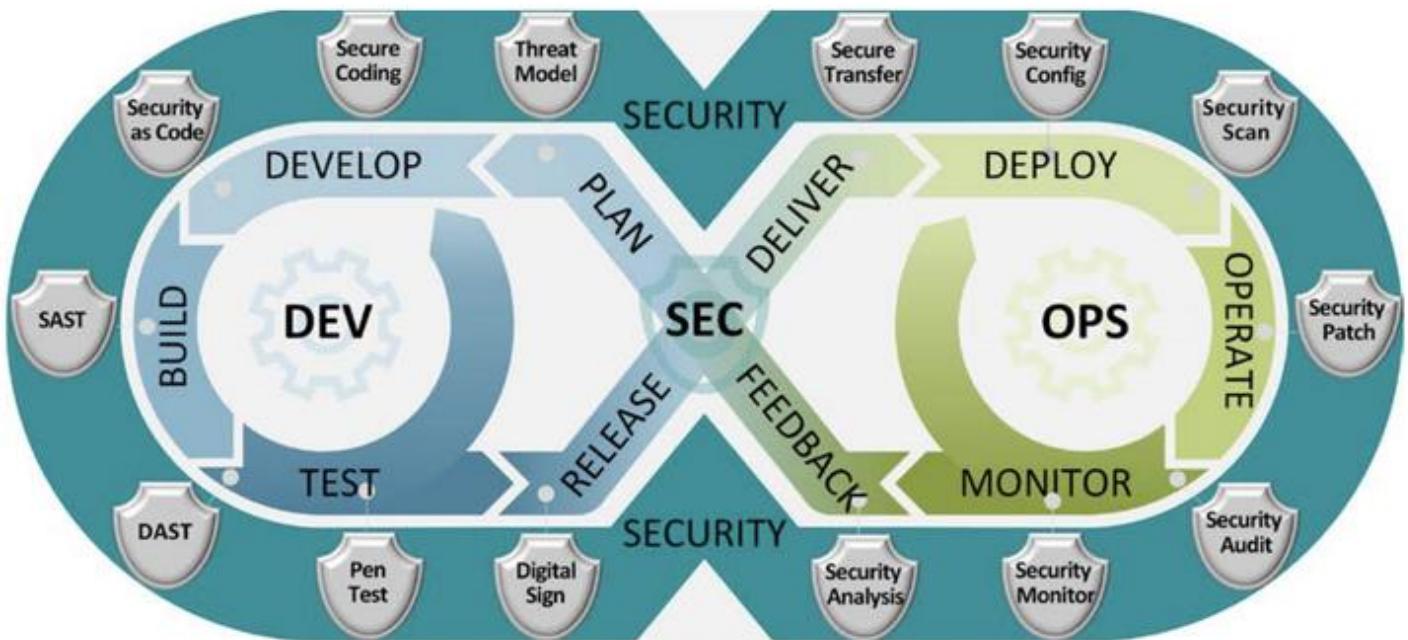
DevSecOps—short for *development, security, and operations*—automates the integration of security at every phase of the software development lifecycle, from initial design through integration, testing, deployment, and software delivery.

- In the past, security was incorporated to software at the end of the development cycle by a separate security team and was tested by a separate QA team

DevSecOps integrates application and infrastructure security seamlessly into Agile and DevOps processes and tools.

DevSecOps makes application and infrastructure security a shared responsibility of development, security, and IT operations teams, rather than the sole responsibility of a security silo

DevSecOps Software Lifecycle



Best Practices for DevSecOps

Shift left

- It encourages software engineers to move security from the right (end) to the left (beginning) of the DevOps (delivery) process.
- Shifting left allows the DevSecOps team to identify security risks and exposures early and ensures that these security threats are addressed immediately

Security education

- development engineers, operations teams, and compliance teams should be familiar with the basic principles of application security and other security engineering practices

Traceability, auditability, and visibility

- Traceability helps to track configuration items across the development cycle to where requirements are implemented in the code
- Auditability is for ensuring compliance with security controls
- Visibility means the organization has a measure the heartbeat of the operation, send alerts, has awareness of cyberattacks as they occur, and provide accountability

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

sei.cmu.edu

www.microsoft.com

www.devopedia.org/devops

www.ibm.com



Thank You!



SEZG566/SSZG566

Secure Software Engineering Threat Modelling

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

T V Rao



- *The slides presented here are obtained from the authors of the books, product documentations, and from various other contributors. I hereby acknowledge all the contributors for their material and inputs.*
- *I have added and modified slides to suit the requirements of the course.*



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Threat Modelling Concepts

Securing a Computer Based System

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

- Hardware
- Software
- Data

Vulnerabilities

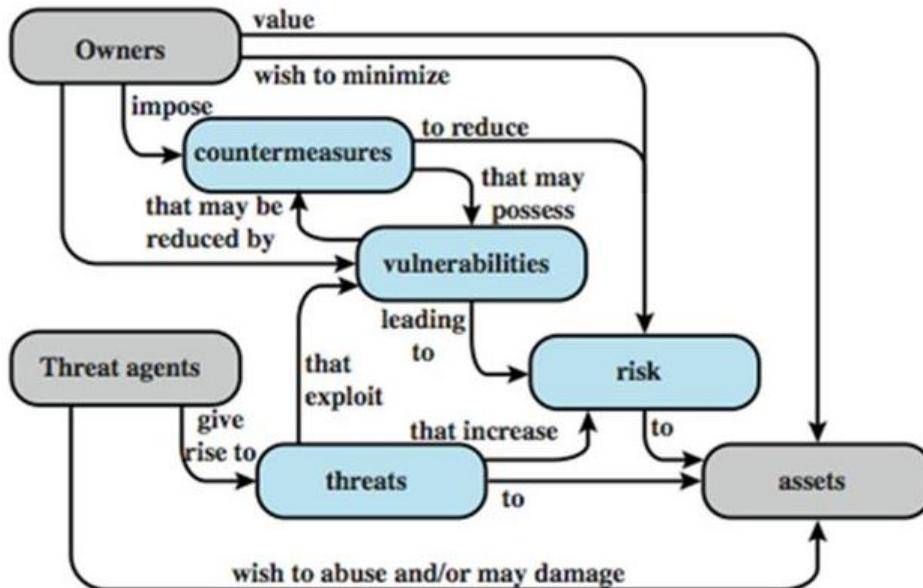
- Weaknesses in a system that may be able to be *exploited* in order to cause loss or harm
 - e.g., a file server that doesn't authenticate its users

Threats

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

Security Concepts and Relationships

Review



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

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 (M-O-M triad)

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



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

OWASP Threat Modelling Process

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.

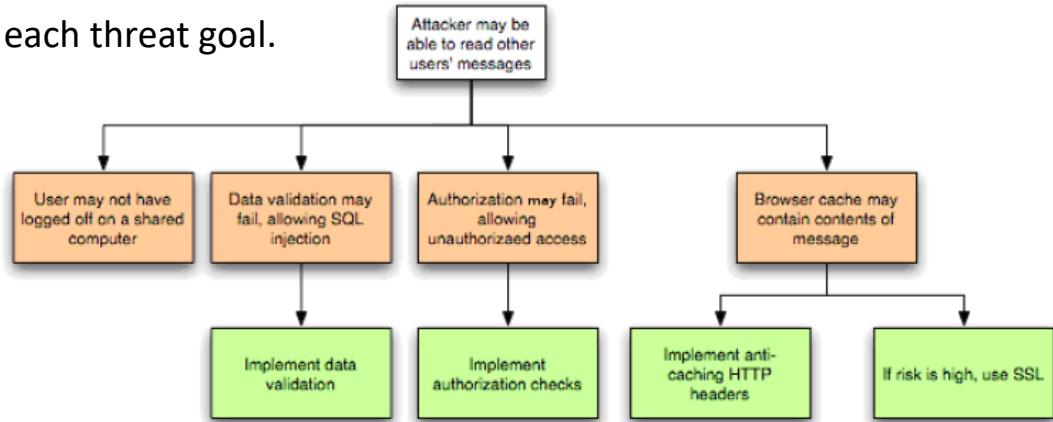
- data sources,
- processes,
- data flows, and
- interactions with users.

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

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

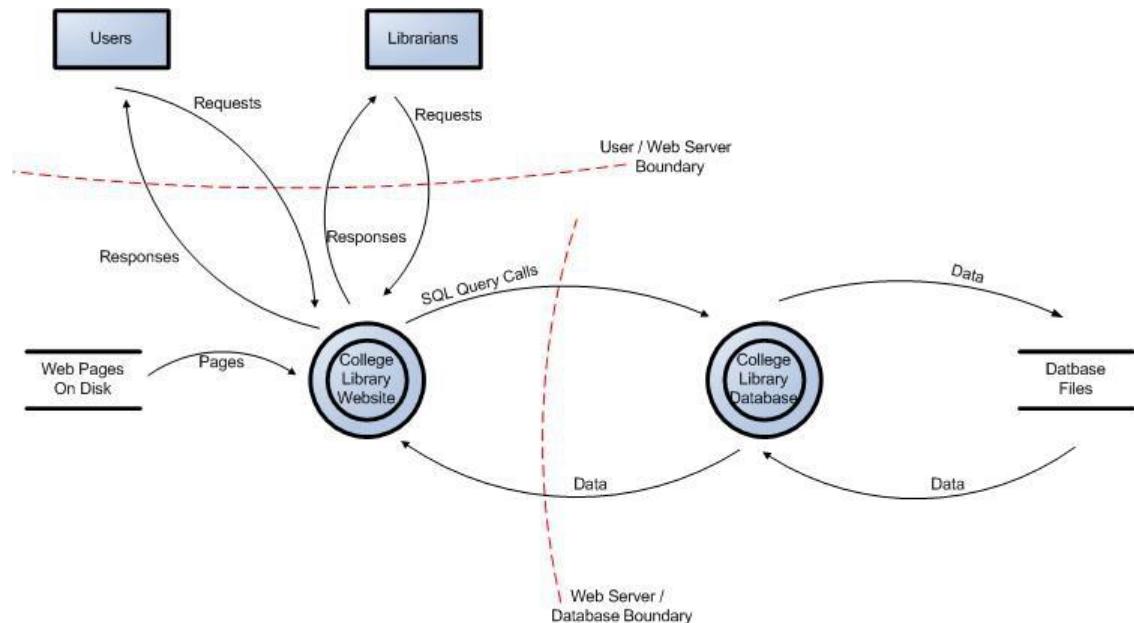
ID	Name	Description	Trust Levels
1	Library Users and Librarian	Assets relating to students, faculty members, and librarians.	
1.1	User Login Details	The login credentials that a student or a faculty member will use to log into the College Library website.	(2) User with Valid Login Credentials, (4) Librarian, (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

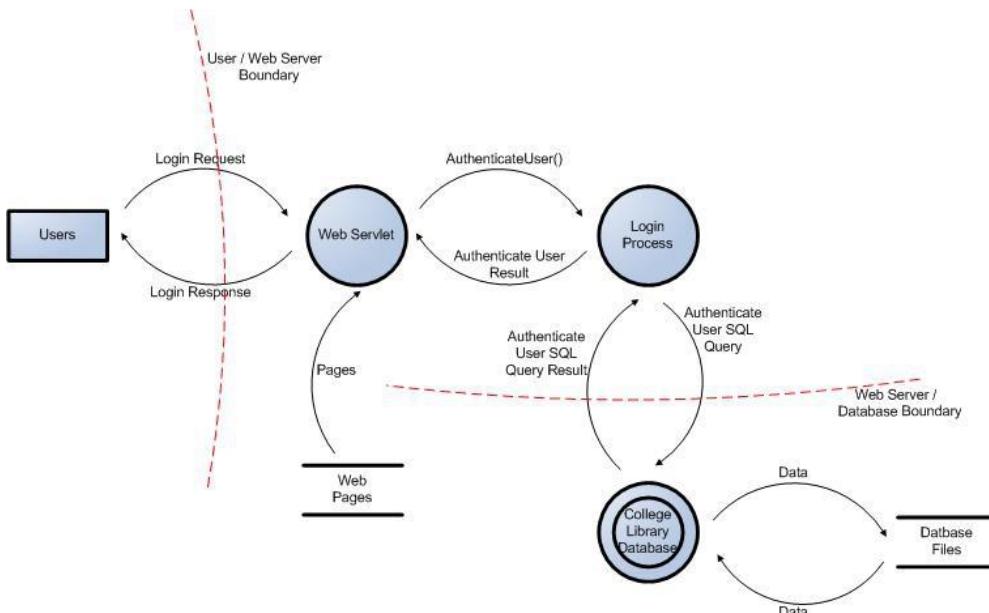
Knowledge of Assets, Entry points, etc. help in creating DFDs.

- 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

Threat Perspectives

Threat categorization helps to identify threats.

- Threat categorization to help identify threats from the attacker
 - (STRIDE)
- Threat categorization to help identify threats from the defensive perspective
 - Application Security Framework(ASF)

Example Countermeasures (ASF)

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



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

SDL Threat Modeling

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



- Defining security requirements.
- Creating an application diagram.
- Identifying threats.
- Mitigating threats.
- Validating that threats have been mitigated.

Define Security Requirements

Security requirements may come from

- industry standards,
- applicable laws, and
- history of past vulnerabilities

Security requirements provide a foundation of vetted security functionality for an application

Allow developers to reuse the definition of security controls and best practices

Prevent the repeat of past security failures

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

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.

Threat: Spoofing

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

Threat: Tampering

Threat	Tampering
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	Repudiation
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

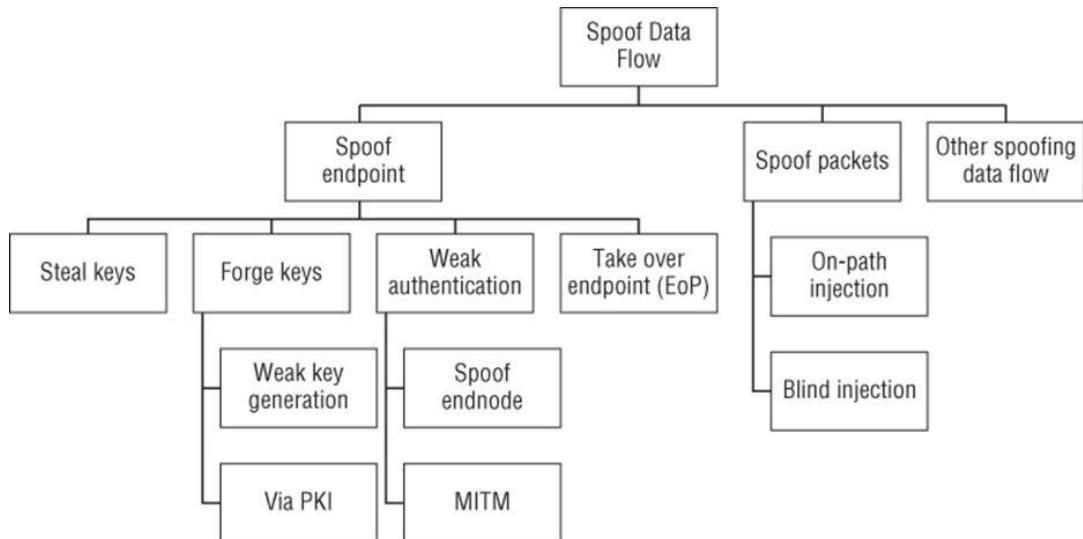
Threat: Denial of Service

Threat	Denial 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

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

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

Cyber security and resilience for Smart Hospitals by ENISA

<https://www.enisa.europa.eu/publications/cyber-security-and-resilience-for-smart-hospitals>



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Privacy Threat Modelling

LINDDUN

LINDDUN was created at KU Research University, Belgium

It is Systematic elicitation and mitigation of privacy. LINDDUN Stands for

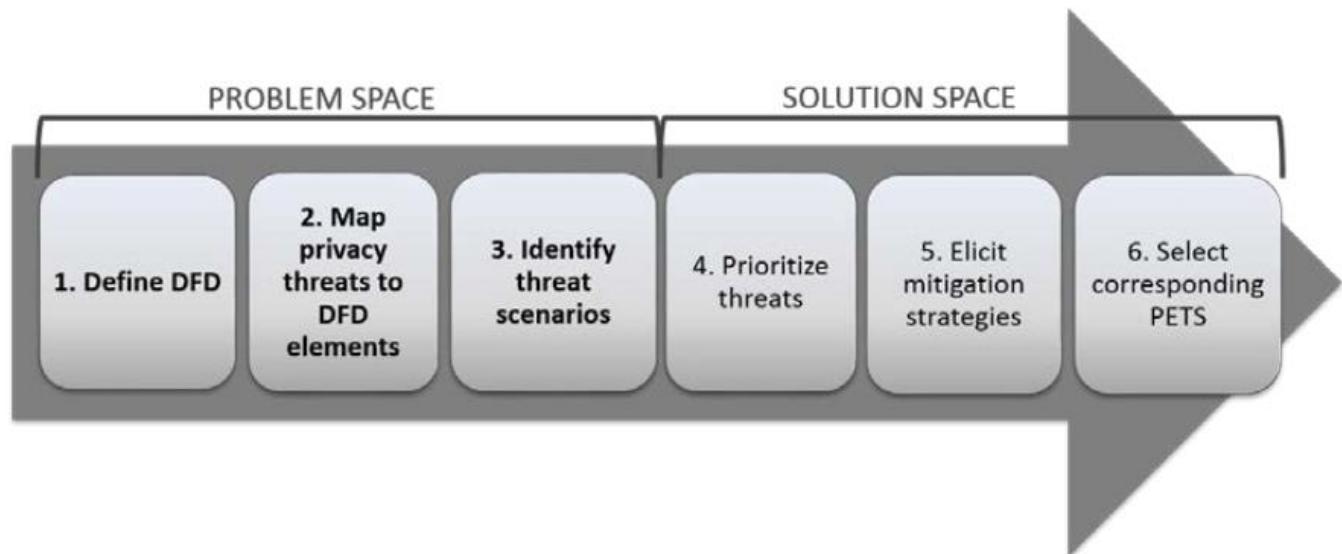
- Linkability,
- Identifiability,
- Non-Repudiation,
- Detectability,
- Disclosure of Information,
- Unawareness,
- Non-Compliance

- Linkability - Being able to sufficiently distinguish whether 2 IOI (items of interest) are linked or not, even without knowing the actual identity of the subject of the linkable IOI
- Identifiability - Being able to sufficiently identify the subject within a set of subjects (i.e. the anonymity set) Or Not being able to hide the link between the identity and the IOI (an action or piece of information).
- Non-Repudiation – works differently in the context of privacy
- Detectability - An attacker can sufficiently distinguish whether an item of interest (IOI) exists or not
- Disclosure of Information - Exposing information to someone not authorized to see it.
- Unawareness - Not understanding the consequences of sharing personal information in the past, present, or future.
- Non-Compliance - Not following the (data protection) legislation, the advertised policies or the existing user consents

Non-Repudiation in Privacy context

- In the context of privacy, non-repudiation works differently from a financial transaction perspective.
- Privacy context expects that attacker can not prove
 - Typical non-repudiation examples exist e.g.
 - Anonymous online voting systems, and
 - Whistleblowing systemswhere plausible deniability is required
- Non-repudiation is actually a security goal, for many systems (e.g. systems where payments are involved)

LINDDUN provides a systematic approach to privacy assessment, consisting of 6 steps



More Methods of Threat Modeling

<https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=524448>



<https://www.enisa.europa.eu/publications/cyber-security-and-resilience-for-smart-hospitals>

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, 2018.

www.owasp.com

www.microsoft.com



Thank You!



SEZG566/SSZG566

Secure Software Engineering Security Requirements Engineering

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

T V Rao



- *The slides presented here are obtained from the authors of the books, product documentations, and from various other contributors. I hereby acknowledge all the contributors for their material and inputs.*
- *I have added and modified slides to suit the requirements of the course.*



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Security Requirements Engineering

What is a requirement?

It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.

This is because requirements serve dual function

- May be the basis for a bid for a contract - therefore must be open to interpretation;
- May be the basis for the contract itself - therefore must be defined in detail;
- Both these statements may be called requirements.

What is a "Requirement"

When the end result of the software development activity is a COTS (Commercial Off The Shelf) product, we term requirements as product specifications.

When the end result of the software development activity is to deliver the product to a single client in a project scenario, we use the term requirements

What is a "Requirement"

A requirement is a need, expectation, constraint or interface of any stakeholders that must be fulfilled by the proposed software product during its development

- **Need**—It is something basic without which the existence becomes untenable. It is the absolute minimum necessity if the system is to be useful
- **Expectation**—Expectation is an unstated need. It is expected that the development team brings expertise of software to bridge the gap in the needs stated by the user
- **Constraint**—It is a hurdle that the user has to live with
- **Interface**—It is the basis for interaction with the customers, suppliers, and peers of the user
- **Stakeholders**—A stakeholder is someone who is affected by the outcome of a endeavor, viz. end user, project team, marketing team to sell product, management

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 Overview

Inception—ask a set of questions that establish ...

- basic understanding of the problem
- the people who want a solution
- the nature of the solution that is desired, and
- the effectiveness of preliminary communication and collaboration between the customer and the developer

Elicitation—elicit requirements from all stakeholders

Elaboration—create an analysis model that identifies data, function and behavioral requirements

Negotiation—agree on a deliverable system that is realistic for developers and customers

Specification—can be any one (or more) of the following:

- A written document
- A set of graphical models
- A formal mathematical model
- A collection of user scenarios (use-cases)
- A prototype

Validation—a review mechanism that looks for

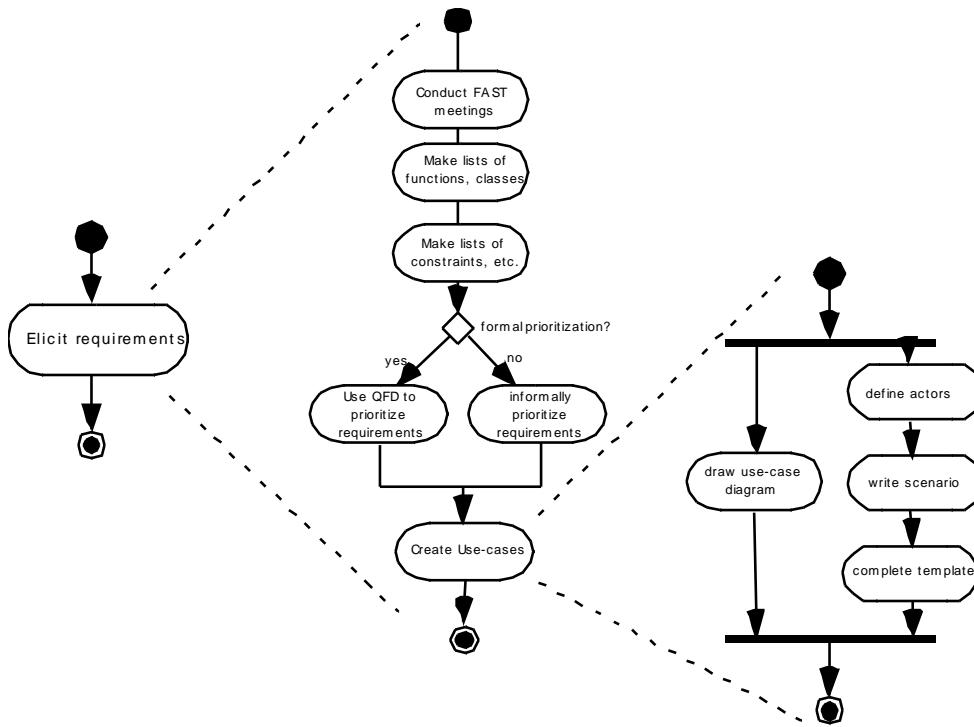
- errors in content or interpretation
- areas where clarification may be required
- missing information
- inconsistencies (a major problem when large products or systems are engineered)
- conflicting or unrealistic (unachievable) requirements.

Requirements management

Eliciting Requirements

- Meetings are conducted and attended by both software engineers and customers
- Rules for preparation and participation are established
- An agenda is suggested
- A "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
- A "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used
- The goal is
 - to identify the problem
 - propose elements of the solution
 - negotiate different approaches, and
 - specify a preliminary set of solution requirements

Eliciting Requirements



Quality Function Deployment

-Dr Yoji Akao et al (1966)



QFD is a focused methodology for carefully listening to the voice of the customer and then effectively responding to those needs and expectations – American Society for Quality

QFD translates needs of customer into technical requirements for software

Function deployment determines the “value” (as perceived by the customer) of each function required of the system

A requirement may be

- Normal - Stated
- Expected - Implicit
- Exciting - Beyond the above two

Information deployment identifies data objects and events

Task deployment examines the behavior of the system

Value analysis determines the relative priority of requirements

Requirements Engineering Challenges

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

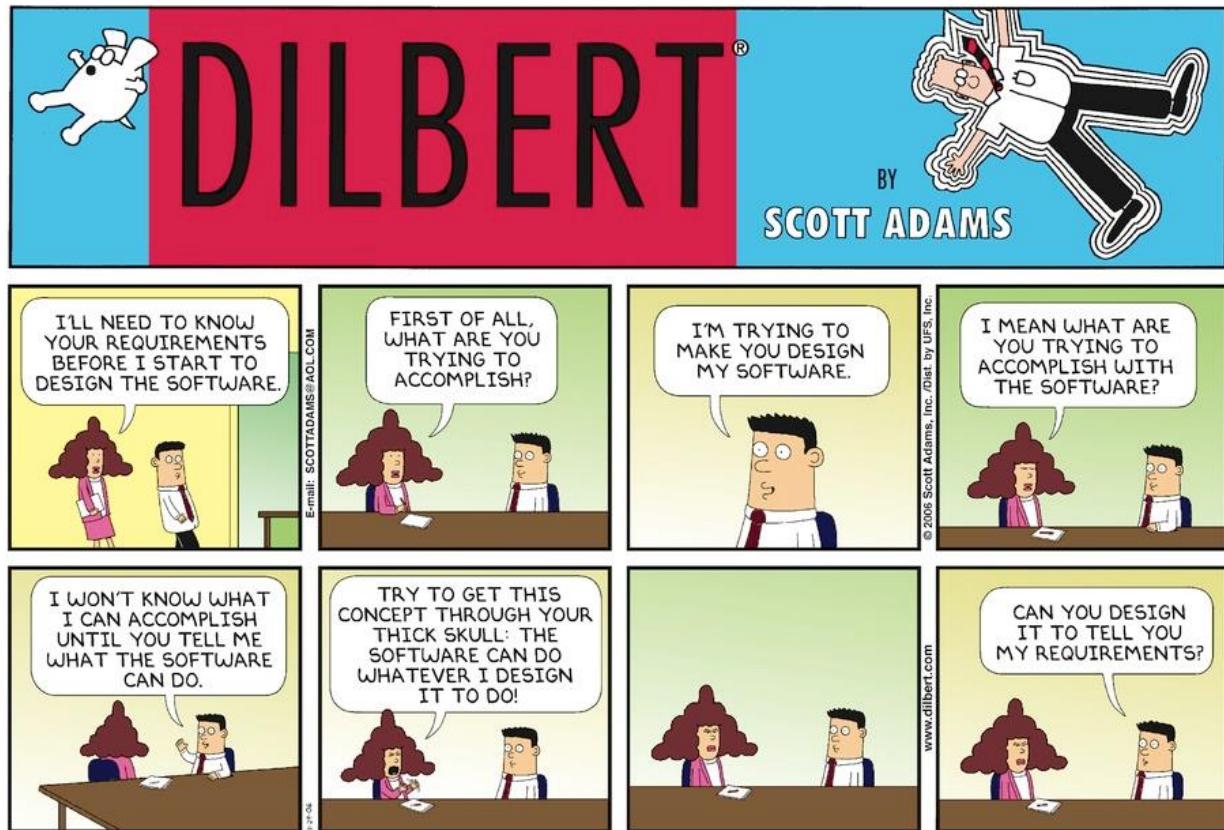
- Requirements identification typically does not include all relevant stakeholders and does not use the most modern or efficient techniques.
- Requirements are often statements describing architectural constraints or implementation mechanisms rather than statements describing what the system must do.
- 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.

Elicitation is not easy

innovate

achieve

lead



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

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.



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

CMU SQUARE Process Model

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 requirements 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



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

CMU SQUARE Work Products

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

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?

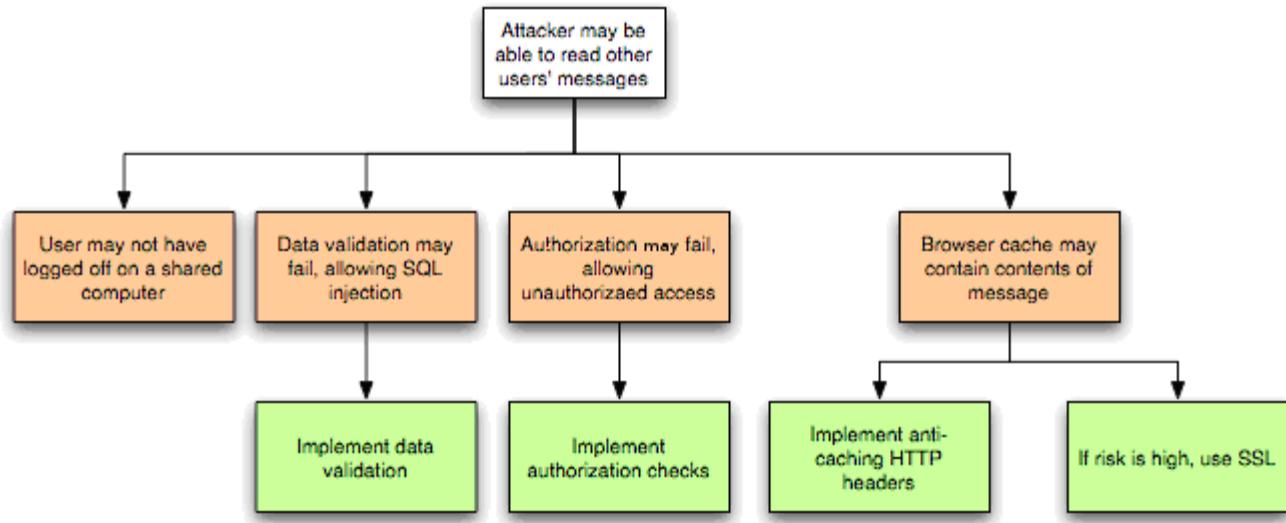
Attack Pattern Example

- **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.

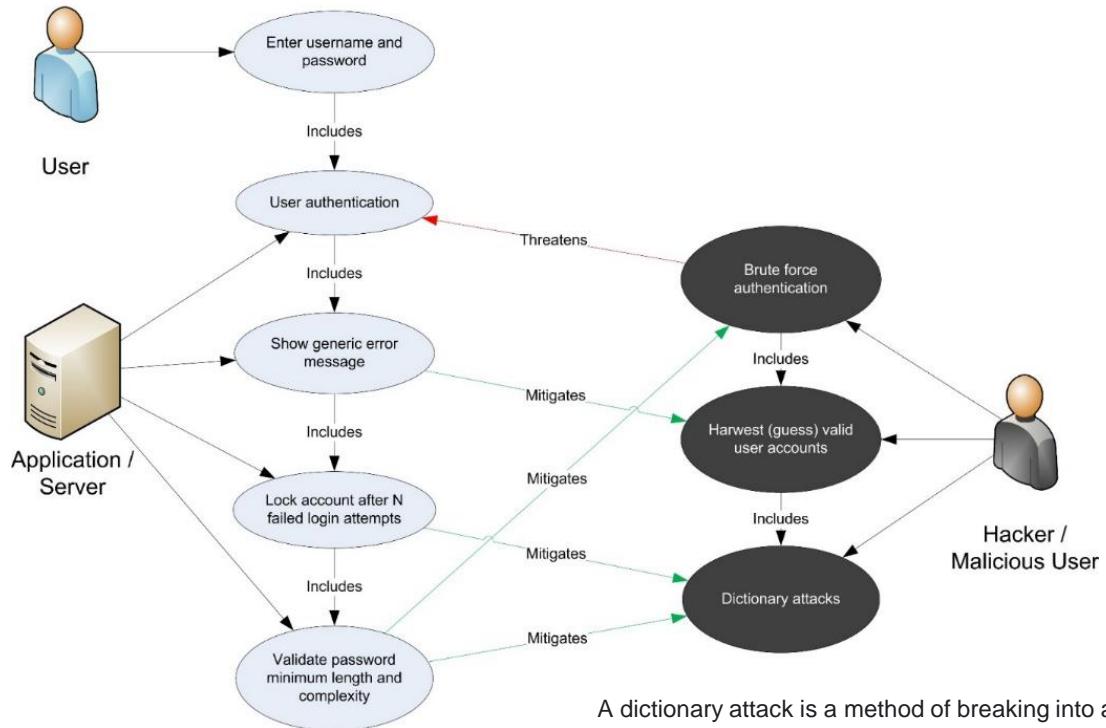
Attack Pattern Example

- **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

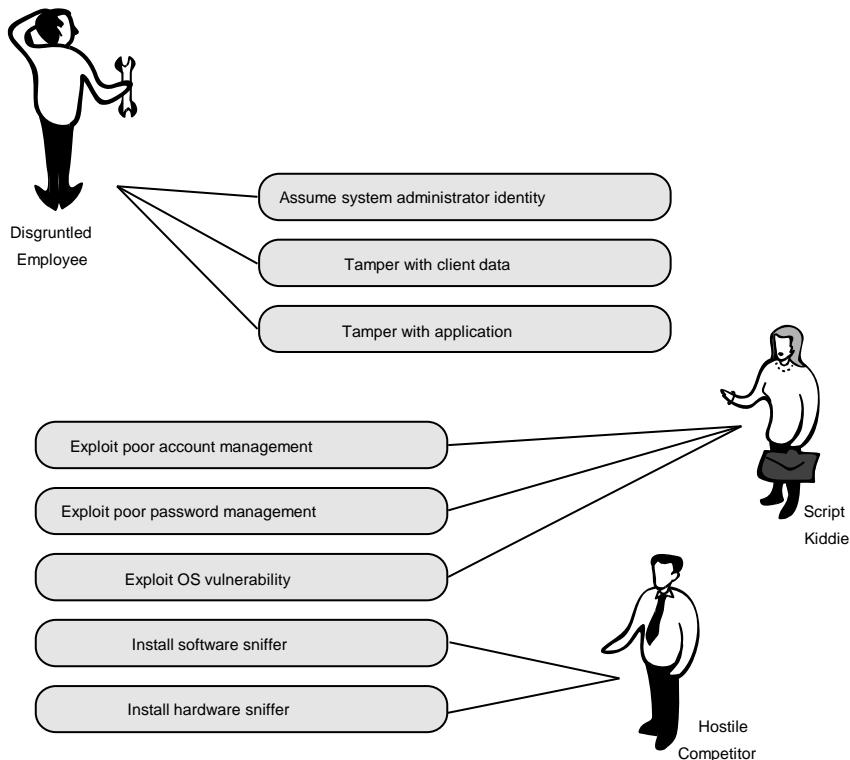


Use Case with Abuse Case



A dictionary attack is a method of breaking into a password-protected computer or server by systematically entering every word in a dictionary as a password

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.)



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

OWASP Recommendations

OWASP SAMM

SAMM (Security Assessment Maturity Model) divides activities associated with software development into 5 business functions for incorporating security

- Governance
 - Includes strategy, metrics, policy, compliance, education and guidance
- Design
 - Includes threat assessment, security requirements, and secure architecture
- Implementation
 - Includes secure build, secure deployment, and defect management
- 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 – SR1

(during the software requirements process)



Goal	Activities	Quality Criteria
Consider security explicitly during the software requirements process.	Identify security requirements	<ul style="list-style-type: none">Teams derive security requirements from functional requirements and customer or organization concernsSecurity requirements are specific, measurable, and reasonableSecurity requirements are in line with the organizational baseline
	Perform vendor assessments	<ul style="list-style-type: none">Consider including specific security requirements, activities, and processes when creating third-party agreementsA vendor questionnaire is available and used to assess the strengths and weaknesses of your suppliers

Increase granularity – SR2

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

Goal	Activities	Quality Criteria
Increase granularity of security requirements derived from business logic and known risks.	Standardize and integrate security requirements	<ul style="list-style-type: none"> Security requirements take into consideration domain specific knowledge when applying policies and guidance to product development Domain experts are involved in the requirements definition process You have an agreed upon structured notation for security requirements Development teams have a security champion dedicated to reviewing security requirements and outcomes
	Discuss security responsibilities with suppliers	<ul style="list-style-type: none"> You discuss security requirements with the vendor when creating vendor agreements Vendor agreements provide specific guidance on security defect remediation within an agreed upon timeframe The organization has a templated agreement of responsibilities and service levels for key vendor security processes You measure key performance indicators

Mandate security requirements process – SR3

(for all software projects and third-party dependencies)



Goal	Activities	Quality Criteria
Mandate security requirements process for all software projects and third-party dependencies.	Develop a security requirements framework	<ul style="list-style-type: none">A security requirements framework is available for project teamsThe framework is categorized by common requirements and standards-based requirementsThe framework gives clear guidance on the quality of requirements and how to describe themThe framework is adaptable to specific business requirements
	Align security methodology with suppliers	<ul style="list-style-type: none">The vendor has a secure SDLC that includes secure build, secure deployment, defect management, and incident management that align with those used in your organizationYou verify the solution meets quality and security objectives before every major releaseWhen standard verification processes are not available, you use compensating controls such as software composition analysis and independent penetration testing

Assessment Matrix for Security Requirements

Score ->	0.0	0.2	0.5	1.0
Do project teams specify security requirements during development?	No	Yes, for some applications	Yes, for at least half of the applications	Yes, for most or all of the applications
Do stakeholders review vendor collaborations for security requirements and methodology?	No	Yes, some of the time	Yes, at least half of the time	Yes, most or all of the time
Do you define, structure, and include prioritization in the artifacts of the security requirements gathering process?	No	Yes, some of the time	Yes, at least half of the time	Yes, most or all of the time
Do vendors meet the security responsibilities and quality measures of service level agreements defined by the organization?	No	Yes, some of the time	Yes, at least half of the time	Yes, most or all of the time
Do you use a standard requirements framework to streamline the elicitation of security requirements?	No	Yes, for some applications	Yes, for at least half of the applications	Yes, for most or all of the applications
Are vendors aligned with standard security controls and software development tools and processes that the organization utilizes?	No	Yes, some of the time	Yes, at least half of the time	Yes, most or all of the time

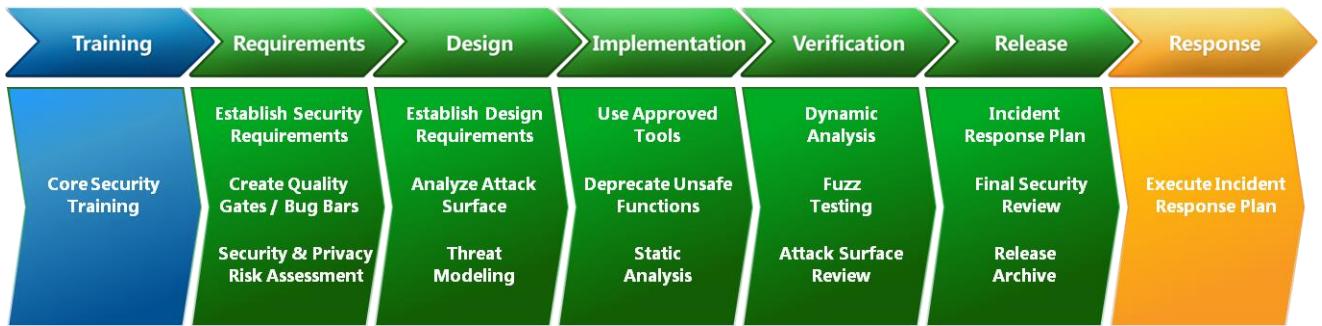


BITS Pilani

Pilani | Dubai | Goa | Hyderabad

SDL Recommendations

Security Development Lifecycle (SDL)



SDL Requirements Phase

The Requirements phase of the SDL includes

- Project Inception—when you consider security and privacy at a foundational level
 - Development teams identify key objectives and integrate security and privacy to minimizes disruption to plans and schedules
- Cost analysis—when you determine if development and support costs for improving security and privacy are consistent with business needs.

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

SDL - Agile

- SDL was developed by Microsoft for securing very large products, such as Windows and Microsoft Office
- The SDL team at Microsoft developed an approach that melds agile methods and security—the Security Development Lifecycle for Agile Development (SDL-Agile)
- There is a need to reconcile
 - Agile release cycles are short, there isn't enough time for teams to complete all of the SDL requirements for every release
 - There are serious security issues that the SDL is designed to address, and these issues simply can't be ignored for any release

SDL-Agile Requirements

- SDL requirements are represented as tasks and added to the product and sprint backlogs
- SDL tasks are added to the backlog as non-functional stories
- SDL-Agile places each SDL requirement into one of three categories defined by frequency of completion
 - Every-sprint requirements - that are so essential to security that no software should ever be released without these requirements being met
 - Bucket Requirements - tasks that must be performed on a regular basis over the lifetime of the project but that are not so critical as to be mandated for each sprint
 - One-Time Requirements - that need to be met when starting a new project with SDL-Agile

Every-Sprint Requirements

- SDL requirements that are so essential to security that no software should ever be released without these requirements being met
- Examples are:
 - Run analysis tools daily or per build
 - Threat model all new features
 - Ensure that each project member has completed at least one security training course

Appendix P of Microsoft-SDL V5.2

Bucket Requirements

- Consists of tasks that must be performed on a regular basis over the lifetime of the project but that are not so critical as to be mandated for each sprint
- The table below contains a sampling of the tasks

Verification Tasks	Design Review	Planning
ActiveX fuzzing	Conduct a privacy review	Create privacy support documents
Attack surface analysis	Review crypto design	Update security response contacts
Binary analysis (BinScope)	Assembly naming and APTCA	Update network down plan
File fuzz testing	User Account Control	Define/update security bug bar

- Appendix Q of Microsoft-SDL V5.2

One-Time Requirements

- SDL requirements that need to be met when you start a new project with SDL-Agile
- SDL-Agile allows a grace period to complete each one-time requirement. The period generally ranges from one month to one year after the start of the project
- For example,
 - Choosing a security advisor has a one-month completion deadline,
 - Updating your project to use the latest version of the compiler is considered a potentially long, difficult task and has a one-year completion deadline
- Appendix R of Microsoft-SDL V5.2

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

www.microsoft.com



Thank You!



SEZG566/SSZG566

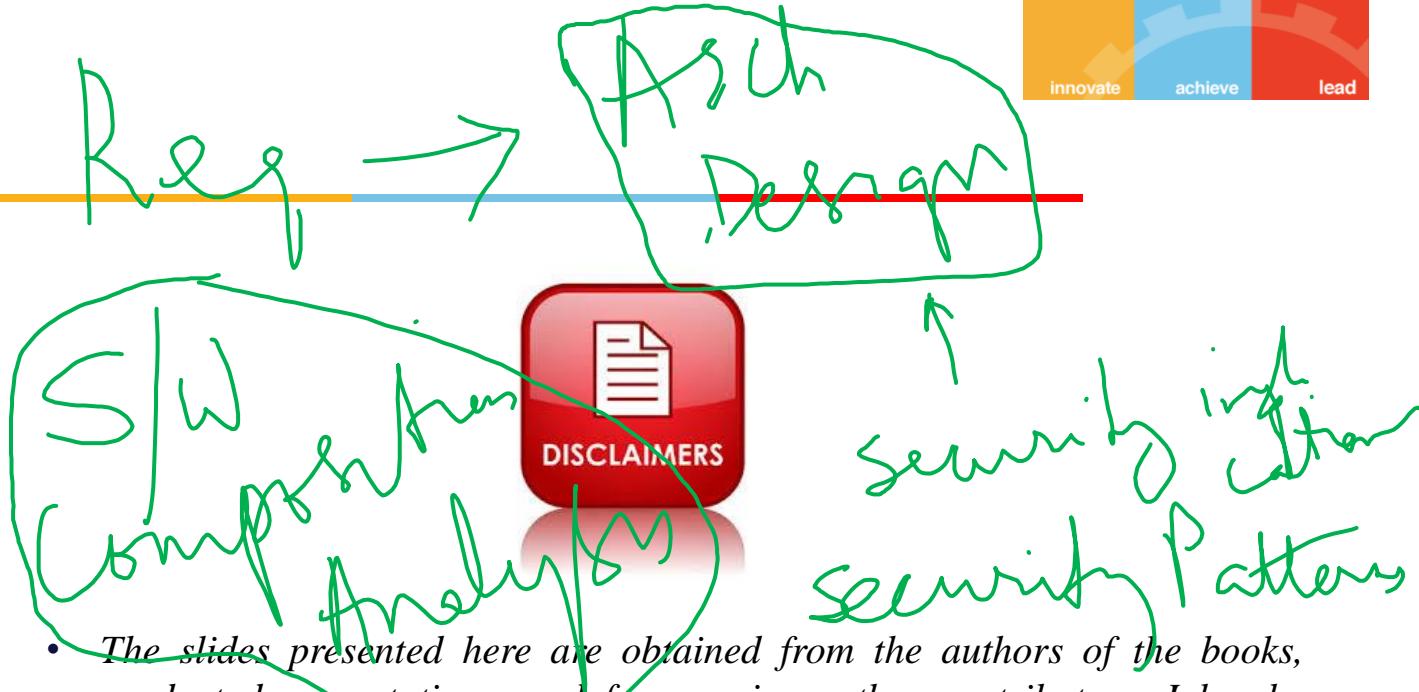
Secure Software Engineering

Architecting/Designing for Security

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

T V Rao



- *The slides presented here are obtained from the authors of the books, product documentations, and from various other contributors. I hereby acknowledge all the contributors for their material and inputs.*
- *I have added and modified slides to suit the requirements of the course.*



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Secure Architecture & Design - Introduction

Nomenclature (SWEBOk)

innovate

achieve

lead

Rev

Architectural Design

Design

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.

Business Problem

Technical Solution

Understanding Software Architecture



Client Server 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

catch fault flaw
early

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

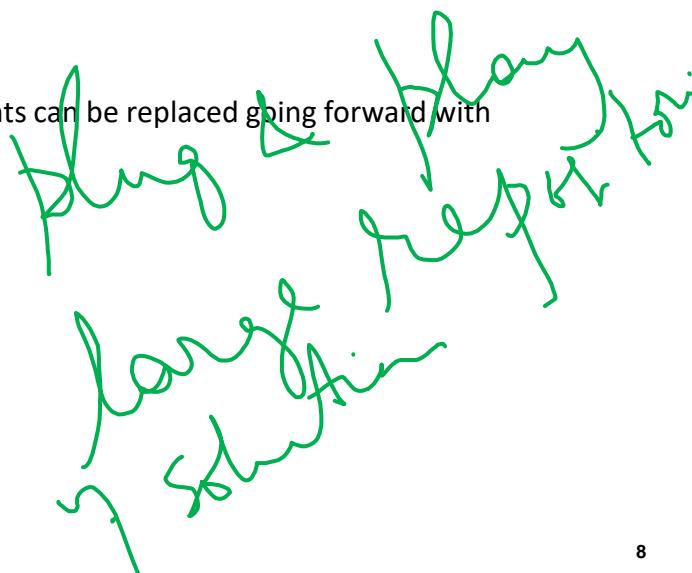
- Can adapt to changing conditions
- Decompose such that selected components can be replaced going forward with minimal impact to the software

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



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Secure Architecture Risk Analysis

Architectural Issues

innovate

achieve

lead



- 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).

CWE

CVE

Architectural Risk Analysis



- The risk assessment methodology encompasses six fundamental activity stages:
 - software characterization
 - architectural vulnerability assessment
 - threat analysis
 - risk likelihood determination
 - risk impact determination
 - risk mitigation

A large, hand-drawn green curly brace is positioned to the left of the final two items in the list. To the right of the brace, the words "Risk mgmt" are written in a cursive, handwritten style.

Software 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 following artifacts may be used 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

T1
SRS

Idea is to get comprehensive, yet concise picture of the nature of application

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

NIST
CVE

Slow testing
prove
NOT observe

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.

Mitre CWE

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.



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Principles for Secure Design

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

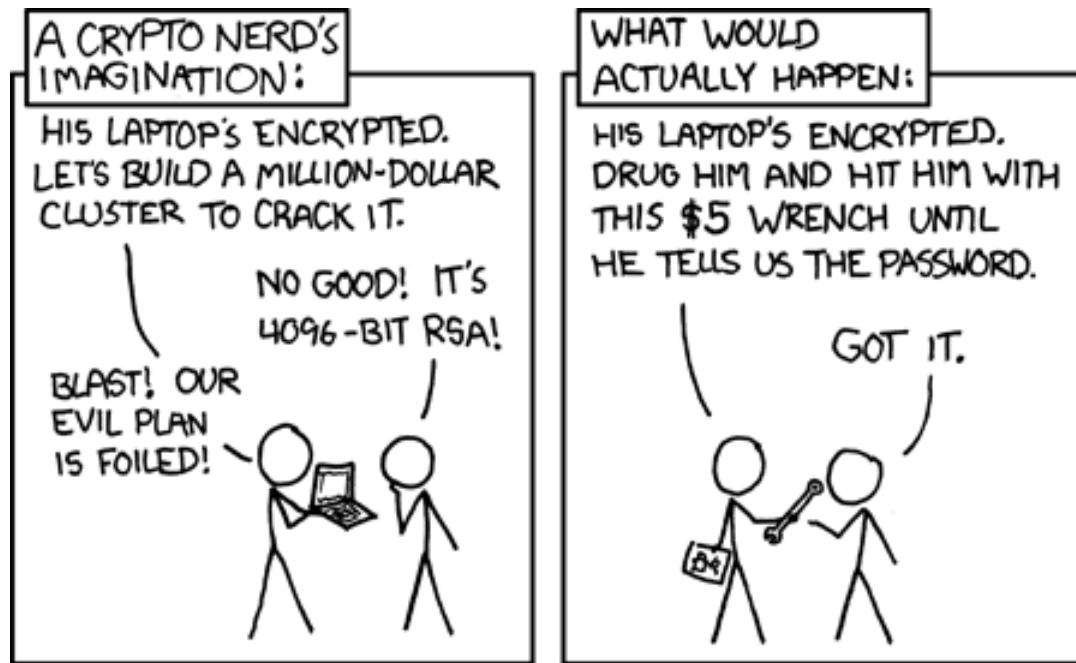
Design Principles for Software Security

- 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

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

Securing the Weakest Link



<https://xkcd.com/538/>

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
- A software system with authentication checks may prevent intrusion by subverting a firewall.
- Implementing a defense-in-depth strategy may 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.

Defense in depth

Success for an attack involves 5 key milestones

- First, the attacker must deliver their attack.
- Second, this attack must exploit a vulnerability (person, software).
- Next, that exploit enables the installation of the attacker's bad software.
- This installation allows the command and control centers of the attackers to connect to the compromised environment.
- At this point, the final state of the exploitation can occur where the ultimate goals of the attacker have been met.

Implementing a single mechanism of defense for each key milestone will provide depth of coverage.

Defense in depth is a structuring of IT security to slow or stop any given attack with multiple mechanisms across the **cyber kill chain**.

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 & Separation of 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."
- 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.

compliance
requirement



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Secure Architectural Patterns

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



decomposition

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

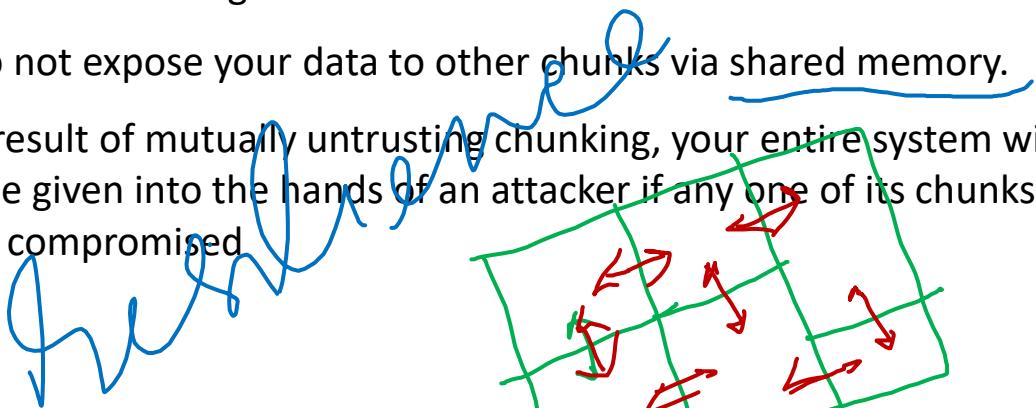
<https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=9115>

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

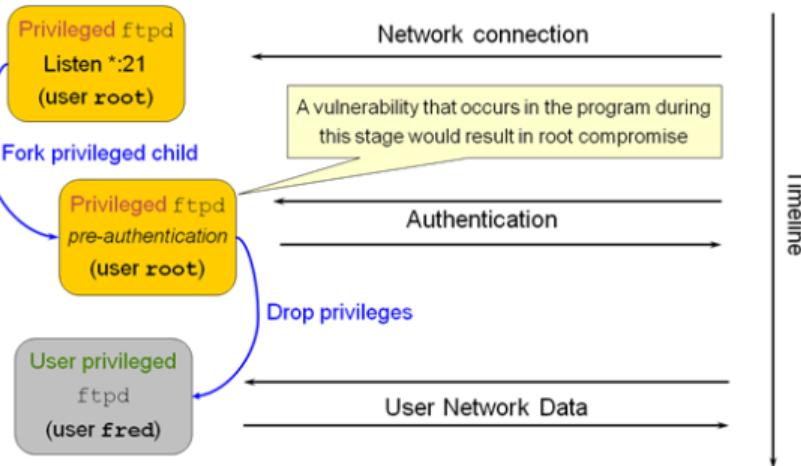
- Attacks in which versions of IE browser running in an account with administrator privileges is compromised
- Security flaws in Norton AntiVirus 2005 that allowed attackers to run arbitrary VBS scripts when running with administrator privileges
- A buffer overflow vulnerability in BSD-derived telnet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

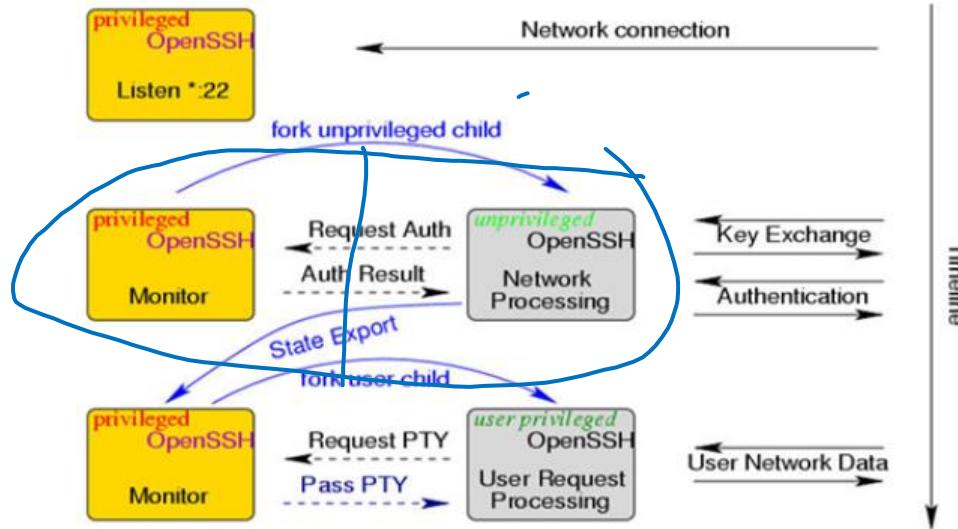
- The PrivSep pattern is a more specific instance of the Distrustful Decomposition pattern.
- Keep to a minimum the part of your code that executes with special privilege.
- If an attacker succeeds in breaking into software that's running at a high level of privilege, the attacker will be operating at 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.

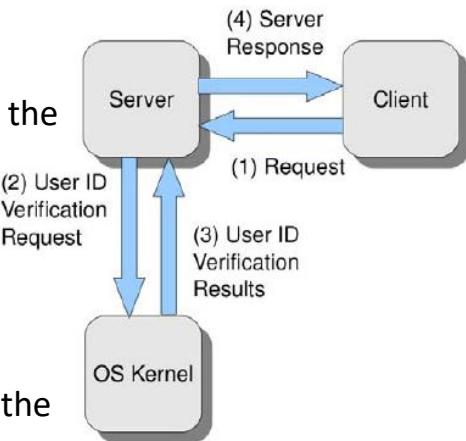
Defer to Kernel

The intent 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





BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Secure Design Patterns

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 requesting the functionality, 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



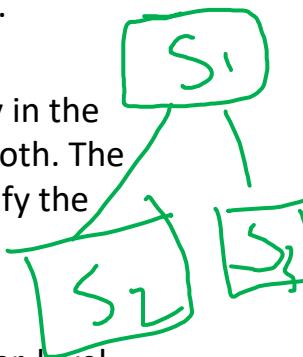
The intent of the Secure State Machine pattern is to allow a clear separation between security mechanisms and user-level functionality by implementing the security and user-level functionality as two separate state machines.

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

innovate achieve lead

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 ensures that the directories in which the files used by the program are stored can only be written (and possibly read) by the user 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

innovate

achieve

lead

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

VI A. Pingel

False positives

Software Composition Analysis

✓
False negatives

Software Composition Analysis



- Generally term is used for managing open source component use
- Involves scans of an application's code base to identify all open source components for
 - license compliance data, and
 - security vulnerabilities
- The scan covers identifying
 - direct dependencies &
 - transitive dependencies

Software Composition Analysis



- SCA tools are expected to perform the following with software vulnerabilities
 - Detection
 - Prioritization – identify risk/criticality of vulnerabilities
 - Remediation
- SCA tool requires
 - Comprehensive vulnerability database
 - Ability to patch

Log4j & Open Source Software



Can any SCA tool determine use of vulnerable component?

How often open source software is embedded in enterprise software?

What is the risk of using open source code?

<https://www.securitymagazine.com/articles/92368-synopsys-study-shows-91-of-commercial-applications-contain-outdated-or-abandoned-open-source-components>

<https://scantist.com/resources/blog/log4j-software-composition-analysis>

<https://www.forrester.com/blogs/log4j-open-source-maintenance-and-why-sboms-are-critical-now/>



Network Security

Modern Network Security Architecture



Large number of options available for organizational networks today means new requirements for network security

- Public Cloud
- Private Cloud
- Hybrid Cloud
- On-Premises

Each of the above come with wide variety of options and tools

Modern Network Security Architecture



- Zero-Trust Architecture (ZTA) is a network security paradigm that operates from the assumption that some actors on the network are hostile, and there are too many entry points to fully protect.
- Network firewall is aimed at preventing anyone from directly accessing the network servers that host an organization's applications and data
- Microsegmentation inhibits an attacker already on the network from moving laterally within it to access critical assets
- A secure web gateway (SWG) is responsible for connecting the user to the desired website and perform functions such as URL filtering, malicious content inspection, web access controls etc.
- DNSSEC strengthens authentication in DNS using digital signatures based on public key cryptography.

Modern Network Security Architecture



- Secure Access Service Edge (SASE) is an emerging framework that combines comprehensive network security functions, such as SWG, SD-WAN and ZTNA
- VDI and DaaS are both solutions for hosted desktops, the key difference between the two is that VDI is managed by the company itself, hosted and managed on-site or at a colocation facility, while DaaS is a managed infrastructure service, delivered by a provider and hosted in their data centres.
- Network Security Policy Management (NSPM) involves analytics and auditing to optimize the rules that guide network security, as well as change management workflow, rule-testing and compliance assessment and visualization. NSPM tools may use a visual network map that shows all the devices and firewall access rules overlaid onto multiple network paths.

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

www.microsoft.com

https://resources.sei.cmu.edu/asset_files/TechnicalReport/2009_005_001_15110.pdf



Thank You!



SEZG566/SSZG566

Secure Software Engineering

Security Testing

BITS Pilani

Pilani | Dubai | Goa | Hyderabad

T V Rao



- *The slides presented here are obtained from the authors of the books, product documentations, and from various other contributors. I hereby acknowledge all the contributors for their material and inputs.*
- *I have added and modified slides to suit the requirements of the course.*



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Testing for Security

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

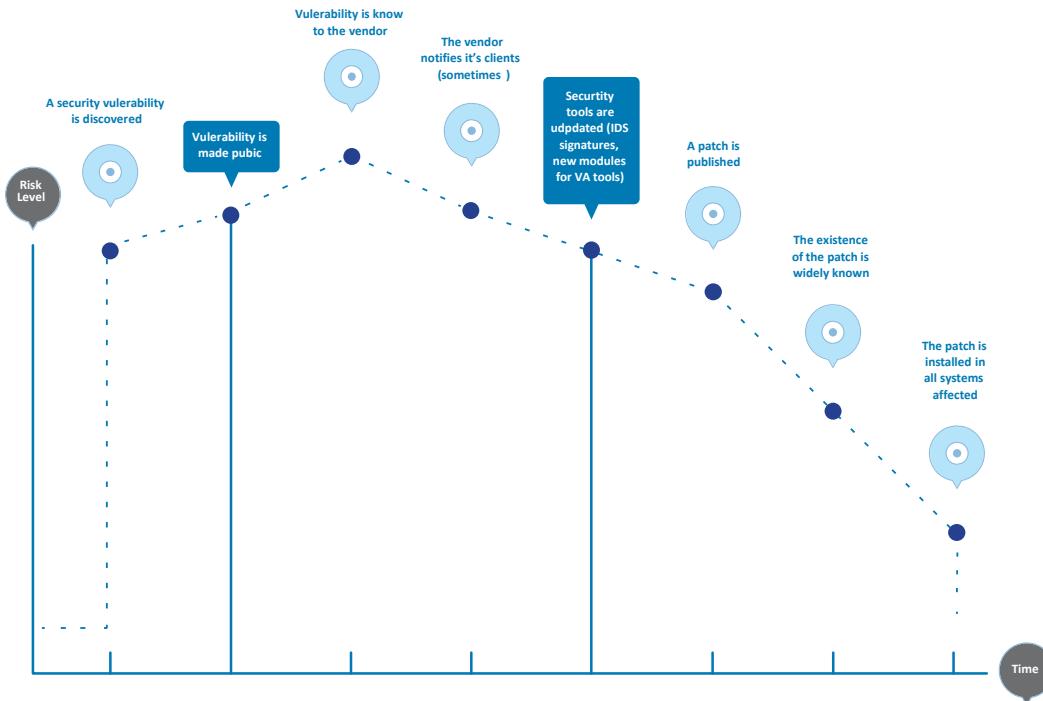
A number of software tools exist to support *post facto* security analysis of installed computer systems. Examples include

- Nmap (Network Mapper) a free and open-source network scanner.
- Burp Intruder is a powerful tool for automating customized attacks against web applications.
- Metasploit scans vulnerabilities backed by open-source database of known exploits.

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 (OWASP)





BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Source Code Analyzers

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)

Some of the tools are listed at

<https://resources.infosecinstitute.com/topic/secure-coding-top-15-code-analysis-tools/>

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.

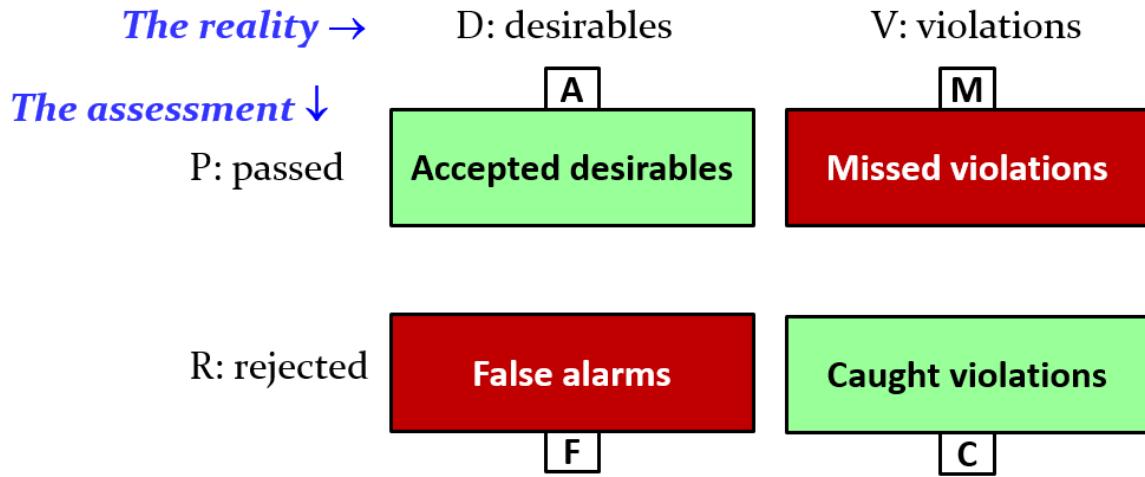
Pointer-aliasing analysis refers to any static technique that tries to solve this problem by tracking which pointers point to what locations

- 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

Soundness vs Completeness of Source Code Analyzers



In two categories, marked in green, assessment is right: accepted desirables (A), rightly passed, and caught violations (C), rightly rejected.

In the other two, marked in red, the assessment is off the mark: missed violations (M), wrongly passed; and false alarms (F), wrongly accepted.

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 (or false alarms).
- 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.



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

White Box Security Testing

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



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Black Box Security Testing

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



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Fuzz Testing

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/>)

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.

Fuzzing – Pros & Cons

Advantages	Issues
<ul style="list-style-type: none">• Simple to design and perform automated tests• Find bugs or crashes not easily visible via other testing techniques• Bugs found are sometimes severe and include defects that could be exploitable in the wild, including unhandled exceptions, crashes, memory faults/leaks and so on• Usually very inexpensive to implement	<ul style="list-style-type: none">• Generally finds very simple faults• Often takes an extremely long time to run• Crashes can often be difficult to analyze, especially when using black-box fuzzing• Mutation templates for applications with complex inputs can often be time-consuming to produce



BITS Pilani

Pilani | Dubai | Goa | Hyderabad

Penetration Testing

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)



<https://insights.sei.cmu.edu/blog/10-types-of-application-security-testing-tools-when-and-how-to-use-them/>

<https://insights.sei.cmu.edu/blog/decision-making-factors-for-selecting-application-security-testing-tools/>

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.

Hacker Hat Types

Black hat hackers

- are criminals who break into computer networks with malicious intent.
 - They may also release malware that destroys files, holds computers hostage, or steals passwords, credit card numbers, and other personal information
 - Hacking can operate like big business, boast partners, resellers, vendors, and associates, and they buy and sell licenses for malware to other criminal organizations for use in new regions or markets.
 - Some countries are lax with them unless they harm their own citizens

White hat hackers

- sometimes also called “ethical hackers” or “good hackers”, exploit computer systems or networks to identify their security flaws so they can make recommendations for improvement.
 - White hat hackers use the same hacking methods as black hats, but with the permission of the system owner first.
 - work with network operators to help fix the issue before others discover it.
 - Never cross the law.

Gray hat hackers

- a blend of both black hat and white hat.
 - Often look for vulnerabilities in a system without the owner's permission.
 - If issues are found, they report them to the owner, sometimes requesting a small fee to fix the problem.

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

Computer Security: Principles and Practice by William Stallings, and Lawrie Brown Pearson, 2018.

<http://resources.infosecinstitute.com>

www.owasp.com

www.microsoft.com



Thank You!