CIS 418/518 – Secure Software Engineering
Software Security Fundamentals

Jagadeesh Nandigam

School of Computing
Grand Valley State University
nandigaj@gvsu.edu

Outline

Introduction

- Computer security can be defined as providing **confidentiality, integrity, and availability** (C.I.A.) assurances to **users or clients** of **information systems**.
- Network security, hardware security, and perimeter security are important, but building better software that is secure is even more important.
- Software security deals with engineering software so that it continues to function correctly under malicious attack in order to provide confidentiality, integrity, and availability assurances to its users or clients.

CIA – Confidentiality, Integrity, and Availability

- **Confidentiality** – The assurance that the information system will keep the user's private data private. Attacks on confidentiality are known as disclosure attacks. This occurs when confidential information is disclosed to individuals against the owner's wishes.
- **Integrity** – The assurance is that the information system will preserve the user's data. Attacks on integrity are called alteration attacks, when information has been maliciously changed or destroyed so it is no longer in a form that is useful to the owner.
- **Availability** – The assurance is that the user can have access to his resources when they are needed. Attacks on availability are called denial attacks, when requests by the owner of the services or data are denied when requested.
- `https://whatis.techtarget.com/definition/Confidentiality-integrity-and-availability-CIA`

Class Activity

Q1. Classify the following as a confidentiality, integrity, or availability attack: The attacker changes my account settings on Facebook so my pictures are visible to the world.

Q2. Classify the following as a confidentiality, integrity, or availability attack: A virus deletes all the .PDF and .DOCX files on my computer.

Q3. Classify the following as a confidentiality, integrity, or availability attack: A terrorist hacks into the White House homepage and defaces it.

'!='s of Security

- Security Software != Secure Software
- Security Features != Secure Features

Terminology: Black vs. White vs. Gray Hats

- **Black Hats**: Those who attempt to break the security of a system without permission (bad guys - the attackers)
  - First Generation: Hackers ⇒ Black Hats motivated by curiosity and pride.
  - Second Generation: Criminals ⇒ Black hats motivated by promise of financial gain.
  - Third Generation: Information Warriors ⇒ Black hats motivated by ethical, moral, or political goals.
- **White Hats**: Those who attempt to break the security of a system while sharing two common characteristics (good guys - the defenders)
  - Ethics: Their activity is bounded by rules, laws, and a code of ethics.
  - Defense: They play on the defensive side. Any activity in an offensive role can be traced to strengthening the defense.

Terminology: Black vs. White vs. Gray Hats

- **Grey Hats**: First generation black hats motivated by the challenge of finding vulnerabilities and increasing system security.
  - They operate outside the law but to protect legitimate users.
  - They do not have the malicious intent typical of a black hat hacker.

Attacker's Advantage

- The black hats (the attackers) have innate advantages over the white hats (the defenders).
  - The defender must defend all points; the attacker can choose the weakest point.
  - The defender can defend only against known attacks; the attacker can probe for unknown vulnerabilities.
  - The defender must be constantly vigilant; the attacker can strike at will.
  - The defender must play by the rules; the attacker can play dirty.

Some Useful Links

- `https://community.norton.com/en/blogs/norton-protection-blog/what-difference-between-black-white-and-grey-hat-hackers`
- `http://www.rasmussen.edu/degrees/technology/blog/types-of-hackers/`
- `https://www.wired.com/2016/04/hacker-lexicon-white-hat-gray-hat-black-hat-hackers/`
- `http://www.pctools.com/security-news/zero-day-vulnerability/`

Terminology (cont.)

- **Asset** – An asset is something of value that a defender wishes to protect and the attacker wishes to possess. Obvious assets include credit-card numbers or passwords. Other assets include network bandwidth, processing power, or privileges. A user's reputation can even be considered an asset.
- **Threat** – A threat is a potential event causing the asset to devalue for the defender or come into the possession of the attacker. Common threats to an asset include transfer of ownership, destruction of the asset, disclosure, or corruption.
- **Vulnerability** – A threat to an asset cannot come to pass unless there exists a weakness in the system protecting it. This weakness is called a vulnerability. It is the role of the software engineer to minimize vulnerabilities by creating software that is free of defects and uses the most reliable asset protection mechanisms available.

Terminology (cont.)

- **Risk** – A risk is a vulnerability paired with a threat. If the means to compromise an asset exists (threat) and insufficient protection mechanisms exist to prevent this from occurring (vulnerability), then the possibility exists that an attack may happen (risk).
- **Attack** – An attack is a risk realized. This occurs when an attacker has the knowledge, will, and means to exploit a risk. Of course not all risks result in an attack, but all attacks are the result of a risk being exploited.
- **Mitigation** – Mitigation is the process of the defender reducing the risk of an attack. Attacks are not mitigated; instead risks are mitigated. There are two fundamental ways this can be accomplished: by reducing vulnerabilities or by devaluing assets.

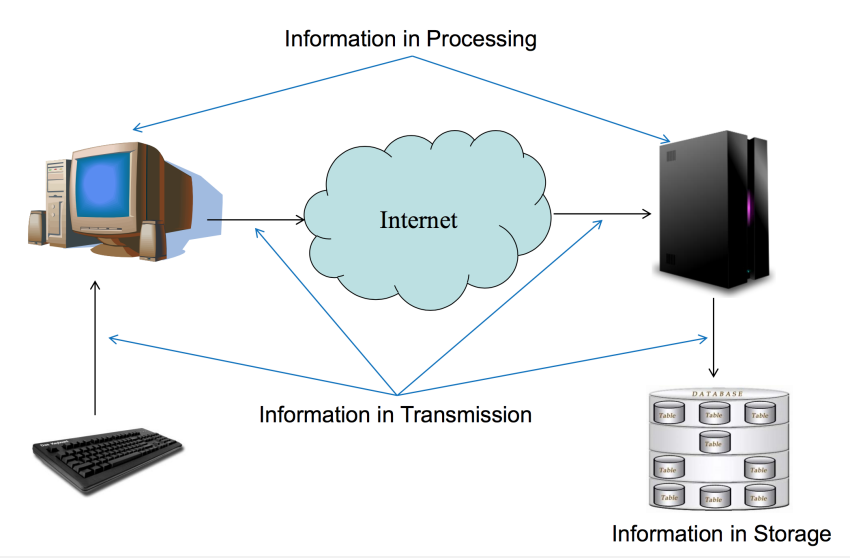
Terminology (cont.)

- **Exploit** – An exploit is a script, plan, or tool that executes against a vulnerability, leading to a security compromise.
- Exploitation Frameworks
  - Metasploit penetration testing framework (www.metasploit.com)
  - Zed Attack Proxy (ZAP) Tool (https://owasp.org/www-project-zap/)

Security as a Software Issue

- Organizations use software-intensive systems that are connected to the Internet to store, process, and transmit sensitive information.
- As a result of global connectivity, software systems and the sensitive information they handle are more vulnerable to unintentional and unauthorized use.
- The three states of information:
  - Information in storage (or at rest)
  - Information in use/processing
  - Information in transit
- Knowing and understanding the three states of information helps to know what security measures should be employed to ensure confidentiality, integrity, and availability of sensitive information.
  - Strong encryption, backups, use of resilient storage, authentication, authorization, use of secure channels of communication, etc.

The Three States of Information



What do these things have in common?

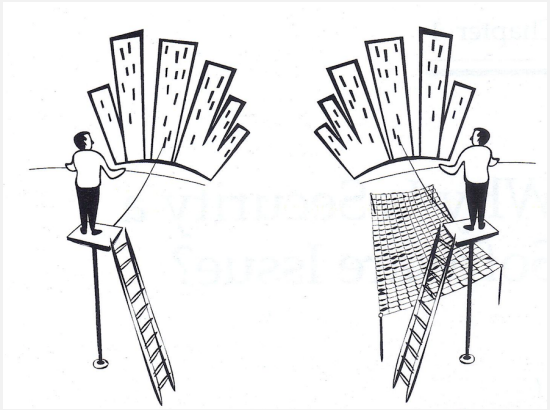
- Desktop and Web applications
- Web services (SOA)
- Middleware
- Browsers
- Servers
- Routers
- Firewalls
- Public key infrastructure (PKI) systems
- Wireless devices
- Cell phones and PDAs
- Smart appliances
- Digital homes

Software is Everywhere

- Software is everywhere – from end-user applications to consumer devices to infrastructure items to security apparatus.
- Real attackers go after "bad software", no matter where it lives.
- Software is the problem, per experts in the field:
  - "75% of hacks occur at the application level" - Gartner, Inc.
  - "Over 70% of security vulnerabilities exist at the application layer, not the network layer" - Gartner, Inc.
  - "If only 50% of software vulnerabilities were removed prior to production ... costs would be reduced by 75%" - Gartner, Inc.
  - "92% of reported vulnerabilities are in applications not in networks" - NIST
  - "64% of developers are not confident in their ability to write secure code" - Bill Gates

Building Software without Security in Mind

- Developing software without security in mind is like walking a high wire without a safety net.
- The degree of risk can be compared to the distance you can fall and the potential impact.



The Trinity of Trouble

- Why is software security a bigger problem now than in the past?
- Three trends, making up the *trinity of trouble*, have a large influence on the growing problem of software security.
  1. Connectivity
  2. Extensibility
  3. Complexity

The Trinity of Trouble: Connectivity

- Growing connectivity of computers through the ubiquitous Internet increased both the attack vector (the route by which an attack is carried out) and the ease with which an attack can be made.
- An attacker no longer needs physical access to a system to exploit vulnerable software.
- As a result, there are more software systems to attack, more attacks, and greater risks from poor software security practices than in the past.
- Wireless, mobile, web services, and service-oriented architecture (SOA) made matters worse.
- The attackers are now at your virtual doorstep.

The Trinity of Trouble: Extensibility

- Today's applications support extensions (sometimes referred to as *mobile code*) through add-ons, plugins, scripting, controls, components, and applets.
  - Consider the plug-in architecure of Web browsers and IDEs
- From an economic standpoint, extensible systems are attactive because they enable flexible interfaces and features.
- But extensible systems also make it hard to prevent software vulnerabilities from slipping in as unwanted extensions.

The Trinity of Trouble: Complexity

- Unbridled growth in the size and complexity of modern software systems.
- Use of unsafe programming languages (e.g., C and C++) that do not protect against simple kinds of attacks, such as buffer overflows.
- "more lines, more bugs" is the rule of thumb.
- More code results in more defects and, in turn, more security risk.
- Even a small "Hello World" program may result in a large code base (in executable space) on platforms such as .NET, J2EE, Rails, etc.
- Imagine a large base of code underneath your small application running on these platforms. Possible security risks in such a large code base?

Threats to Software Security

- Software is subject to two general categories of threats:
  - Threats during development (mostly insider threats)
  - Threats during operation (both insider and external)

Threats to Software Security

- Threats during development (mostly insider threats)
  - A software engineer can sabotage the software at any point in its development life cycle.
  - Carried out through intentional exclusions from, inclusions in, or modifications of the requirements specification, the threat models, the design documents, the source code, the assembly and integration framework, the test cases and test results, or the installation and configuration instructions and tools.

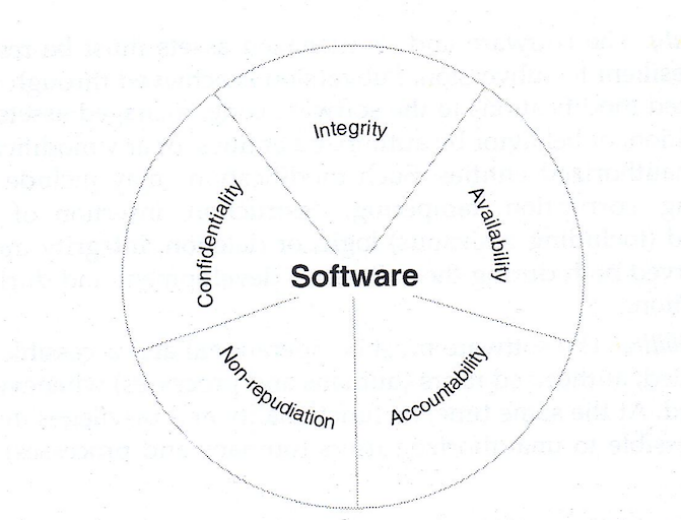
Threats to Software Security

- Threats during operation (both insider and external)
  - A software system that runs on a network-connected platform has its vulnerabilities exposed to attackers during its operation.
  - Attackers can, using the Internet, study about successful attacks and reports of security vulnerabilities in a wide range of software programs.
  - Attackers can access and often develop (and share) exploit scripts.
  - To be 100 percent effective, defenders must anticipate all possible vulnerabilities, while attackers need find only one to carry out their attack.

Properties of Secure Software

- Core properties of secure software
  - Properties who presence (or absence) are the ground truth that makes software secure (or not).
- Influential properties of secure software
  - Properties that do not directly make software secure but do make it possible to characterize how secure software is.

Core Properties of Secure Software



Core Properties of Secure Software

- Confidentiality – software and its managed assets are obscured or hidden from unauthorized entities.
- Integrity – software and its managed assets must be resistant and resilient to subversion or unauthorized modifications.
- Availability – software must be operational and accessible to its intended, authorized users whenever it is needed. It must also be inaccessible to unauthorized users at all times.

Core Properties of Secure Software

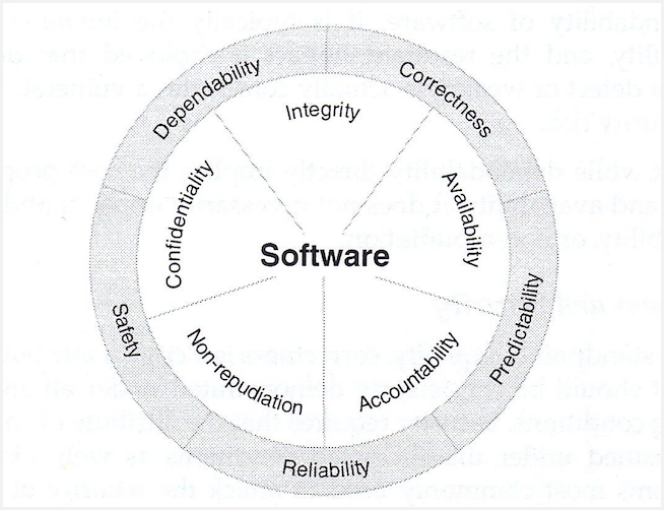
Two additional properties are required in software entities that act as users (e.g., proxy agents, Web services, peer processes):

- Accountability – All security-relevant actions of the software-as-user must be recorded and tracked, with attribution of responsibility.
- Non-repudiation – Software-as-user must be prevented from disproving or denying responsibility for actions it has performed. This ensures that the accountability property cannot be subverted or circumvented.

Core Properties of Secure Software - Examples

- The effects of a security breach in software can be described in terms of the effects on the core properties.
- Few sample attacks and their effects on core properties:
  - Successful SQL injection attack on an application to extract personally identifiable information from its database would be a violation of its confidentiality property.
  - Successful cross-site scripting (XSS) attack against a Web application could result in a violation of both its integrity and availability properties.
  - Successful buffer overflow attack that injects malicious code in an attempt to steal user account information and then alter logs to cover its tracks would be a violation of all five core security properties.

Influential Properties of Secure Software



Influential Properties of Secure Software

- Dependability – property of software that ensures that the software always operates as intended. It directly implies the core properties of integrity and availability.
- Correctness – software's expected behavior must be correct under anticipated and unanticipated conditions.
- Predictability – software never deviates from correct operation under anticipated and unanticipated conditions.
- Reliability – software must exhibit predictable, correct execution even in the face of malicious attacks on defects or weaknesses.
- Safety – A failure in the security of software, especially one that is intentional and malicious, can directly change the operational and environmental presumptions on which safety is based.

Influencing Security Properties of Software

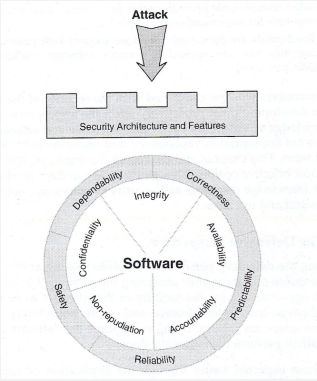
- It takes two perspectives to influence the security properties of software.
  - The Defensive Perspective
  - The Attacker's Perspective

The Defensive Perspective

- The Defensive perspective involves looking at the software from the **inside out**.
  - Addressing *expected* issues with *security architecture and features*
  - Addressing *unexpected* issues with *application defense/security*
  - Addressing *unexpected* issues with *software security*

The Defensive Perspective – Security Architecture and Features

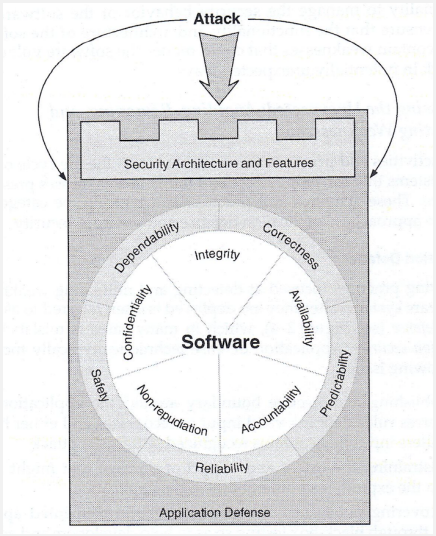
- Address expected security issues with security architecture and features (authentication, authorization, access control, permissions, privileges, and cryptography).



The Defensive Perspective – Application Security

- Address unexpected security issues with **application defense or security** using techniques and tools used for securing networks, operating systems, and middleware services.
- These include vulnerability scanners, intrusion detection tools, and firewalls or security gateways. These measures strengthen the boundaries **around the application**.
- Application security is about protecting software after it's already built.
- Infrastructure (IT) people are operators (not builders) who use reactive approaches to provide application defense.

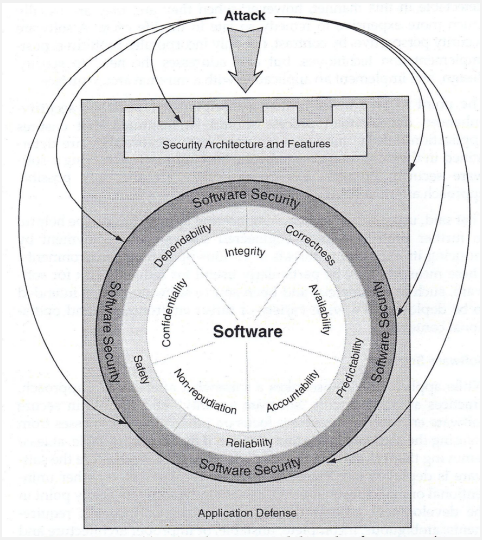
The Defensive Perspective – Application Security



The Defensive Perspective – Software Security

- Address unexpected security issues with **software security** with focus on preventing weaknesses from entering the software in the first place using a wide variety of security-focused practices during software development.
- Software security, which is a process of designing, building, and testing software for security, takes a proactive approach to identify and remove problems in the software itself.

The Defensive Perspective – Software Security



The Attacker's Perspective

- Involves looking at the software from the **outside in**.
- It is much easier to find vulnerabilities in software than it is to make software secure.
- An attacker simply need to find one exploitable vulnerability to achieve his or her goal.
- Attackers have been learning how to exploit software for several decades, but the general software development community has not kept up-to-date with the knowledge that attackers have gained.
- **Attack patterns** provide a powerful mechanism to represent the attacker's perspective.

Clasification of Attacks

- Attacks on CIA assurances are collectively called DAD for Disclosure, Alteration, and Denial. All attacks fall into one or more of these categories.
- STRIDE model is a more elaborative taxonomy of attacks developed by Microsoft (more on this later in the course!)
  - Spoofing – Pretending to be someone other than who you really are
  - Tampering – Changing data in some way
  - Repudiation – Process of denying or disavowing an action
  - Information Disclosure – Exposing confidentail data against the wishes of the owner of the information
  - Denial of Service – Making service unavaialable to legitimate users
  - Elevation of Privilege – Ability to achieve greater privilege than he/she normally would have under his/her current identity

Software Attacks/Weapons

- Software attacks/weapons exploit vulnerabilities in computing systems in an automated way.
- A software engineer needs to know about software weapons because these are the tools black hats use to compromise a legitimate user's confidentiality, integrity, and/or availability.
- The vast majority of attacks follow well-established patterns or tools.

Software Attacks/Weapons

- Rabbit – A rabbit is malware payload designed to consume resources.
- Bomb – A bomb is a program designed to deliver a malicious payload at a pre-specified time or event.
- Adware – Adware is a program that serves advertisements and redirects web traffic in an effort to influence user shopping behavior.
- Trojan – A trojan horse is a program that masquerades as another program. The purpose of the program is to trick the user into thinking that it is another program or that it is a program that is not malicious.
- Ransomware – Ransomware is a type of malware designed to hold a legitimate user's computational resources hostage until a price is paid to release the resources (called the ransom).

Software Attacks/Weapons

- Back Door – A back door is a mechanism allowing an individual to enter a system through an unintended and illicit avenue.
- Virus – A virus is a classification of malware that spreads by duplicating itself with the help of human intervention.
- Worm – A worm is similar to a virus with the exception of the relaxation of the constraint that human intervention is required.
- SPAM
- Rootkit – A term associated with software designed to help an unauthorized user obtain root privilege on a given system.
- Spyware – Spyware is a program hiding on a computer for the purpose of monitoring the activities of the user.
- Botware – Botware (also called Zombieware or Droneware) is a program that controls a system from over a network.

Attack Patterns

- A mechanism, similar to *design patterns*, for capturing and communicating attacker's perspective from knowledgeable experts and communicating it to development teams.
- Apply the problem-solution paradigm of design patterns in a destructive context.
- Describe the techniques that attackers might use to break software.
- Contain sufficient detail about how attacks are carried out to enable developers to help prevent them.
- Can be used to teach software development community both how software is exploited in reality and how to avoid such attacks.

Attack Pattern Components

- The following typical information is captured for each attack pattern:
  - Pattern name and classification
  - Attack prerequisites
  - Description
  - Targeted vulnerabilities or weaknesses
  - Method of attack
  - Attacker goal
  - Attacker skill level required
  - Resources required
  - Blocking solutions
  - Context description
  - References
- Useful information on attack patterns
  - `https://www.us-cert.gov/bsi/articles/knowledge/attack-patterns`
  - `https://www.us-cert.gov/bsi/articles/knowledge/attack-patterns/introduction-to-attack-patterns`

Publications of Attack Patterns

- Common Attack Pattern Enumeration and Classification (CAPEC) at `https://capec.mitre.org/`
- Common Weakness Enumeration (CWE) at `https://cwe.mitre.org/`
- Common Vulnerabilities and Exposures (CVE) at `https://cve.mitre.org/`
- Mitre Attack Matrix at `https://attack.mitre.org`

Defect, Bug, Flaw, and Risk

- Defect – Both implementation vulnerabilities and desgin vulnerabilities are defects.
- Bug – A bug is an implementation-level software problem.
  - Can be relatively easily discovered and remedied.
  - Source code analyzers are effective in detecting a wide range of implementation bugs, including buffer overflow vulnerabilities, format string bugs, resource leaks, and simple race conditions.
- Flaw – A flaw is a problem at a deeper level. A flaw is certainly instantiated in software code, but it is also present (or absent) at the design level.
- Risk – Flaws and bugs lead to risk. Risks are not failures. Risks capture the probability that a flaw or a bug will impact the purpose of the software.

Examples of Bugs and Flaws

- Software security problems are divided 50/50 between bugs and flaws.

Bugs	Flaws
Buffer overflow: stack smashing	Method over-riding problems (subclass issues)
Buffer overflow: one-stage attacks	Compartmentalization problems in design
Buffer overflow: string format attacks	Privileged block protection failure (`DoPrivilege()`)
Race conditions: TOCTOU	Error-handling problems (fails open)
Unsafe environment variables	Type safety confusion error
Unsafe system calls (`fork()`, `exec()`, `system()`)	Insecure audit log design
Incorrect input validation (black list vs. white list)	Broken or illogical access control (role-based access control [RBAC] over tiers)
	Signing too much code

Software security defects come in two basic flavors, each of which accounts for approximately 50% of software security problems.

Range of Defects

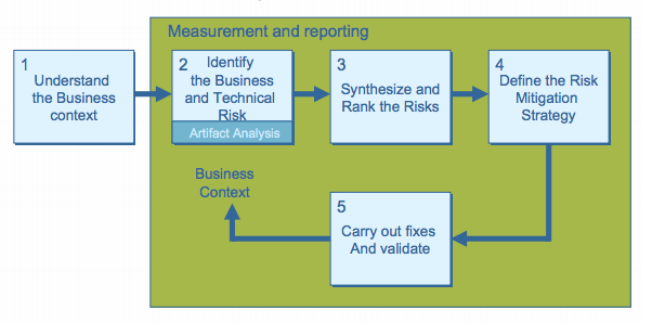
- Flaws and bugs exist along a continuum of defects. Determining whether a defect is a flaw or a bug is difficult.
- Security defects in software systems range from local implementation errors to midrange vulnerabilities to much higher-level design mistakes.
- Low-level coding errors (such as a call to gets() in C/C++) can be detected with good precision using a very simple lexical analysis without knowing anything about the rest of the code, its design, or the execution environment.
- Midrange vulnerabilities (e.g., detecting race conditions) involve interactions among more than one location in code.
- Design-level (e.g., error-handling and recovery systems that fail in an insecure way) vulnerabilities require more expertise to find and are hard to automate.

Solving the Software Security Problem – The Three Pillars

- Applied Risk Management
- Software Security Touchpoints
- Sofware Security Knowledge

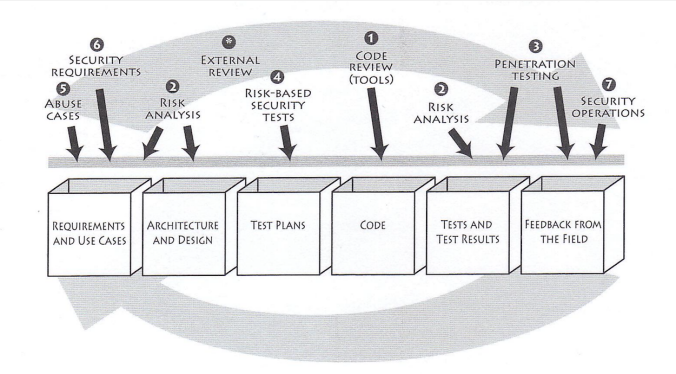
Applied Risk Management

- A risk management framework that supports iterative risk identification and mitigation that is integrated throughout the SDLC.



Software Security Touchpoints

- Application of software security best practices (or touchpoints) to the various artifacts produced during software development.
- Touchpoints – Abuse/misuse cases, security requirements, risk-based security tests, code review with tools, risk analysis, penetration testing, and security operations.



Software Security Knowledge – A Unified View

- Arm software practitioners with software security knowledge to provide a solid foundation for software security practices.
- Seven knowledge catalogs – principles, guidelines, rules, vulnerabilities, exploits, attack patterns, and historical risks
- Three knowledge categories
  - Prescriptive knowledge – principles, guidelines, and rules
  - Diagnostic knowledge – attack patterns, exploits, and vulnerabilities
  - Historical knowledge – historical risks

References

- Gary McGraw, "Software Security - Building Security In", Chapter 1, Addison Wesley.
- James N. Helrich, *Security for Software Engineers*, Chapters 0 – 3, CRC Press.
- Julia H. Allen, et. al., "Software Security Engineering - A Guide for Project Managers", Chapters 1 & 2, Addison Wesley.
- "Vulnerabilities, Threats, and Attacks" (pdf file available on Blackboard)