Module 1: Foundations and Concepts

CYBR 515: Software Security

Summer 2024

Mohammad Jamil Ahmad, Ph.D,

Table of contents

- Introduction to Software Security
- The Role of Trust in Software Security
- Principles of Information Security: CIA Triad
- The Gold Standard: Authentication, Authorization, Auditing
- Software source code, bug, fault, error, and weakness

What is a Software?

- Definition: Software is a set of instructions, data, or programs used to operate computers and execute specific tasks. It includes applications and the operating system that helps run a computer and manage its resources.
- Types of Software:
  - System software: Manages the hardware components and provides resources that the application software uses (Operating Systems)
  - Application software: Performs specific tasks for users, ranging from productivity applications to more complex applications like databases and financial software (all other apps you use)

Evolution of Software

- **Early Development:**
  - Initially, software was built for specific, often singular, tasks without much flexibility or user input.
  - Software was developed primarily for government, military, and large businesses.
- **Rise of Personal Computing:**
  - The 1980s and 1990s saw a boom in personal computing, leading to the development of more user-friendly software for general public use, including graphical user interfaces (GUIs) and commercial operating systems.

Evolution of Software

- **Internet and Software Development:**
  - The advent of the Internet revolutionized software development, promoting rapid growth in web-based applications and services.
  - The shift towards open-source projects in the late 1990s and early 2000s allowed for more collaborative development environments.
- **Modern Trends:**
  - Recent trends include the development of mobile apps, cloud computing, and AI-driven software, emphasizing the need for continual adaptation and innovation.

Shift in Security Paradigm

- **Early Days:**
  - Initially, software was used in controlled environments with limited connectivity, making security a secondary concern relative to functionality.
  - Security threats were relatively low, and the software was often built without robust security measures.
- **Change in Landscape:**
  - As the Internet grew, software began connecting vast networks and handling sensitive data, leading to an increased risk of cyber threats.
  - High-profile security breaches affecting government agencies, corporations, and individuals have highlighted vulnerabilities in software.

Modern Security Focus:

- Today, security is a top priority in software development. The industry adopts security-first approaches, such as DevSecOps, to integrate security measures from the earliest stages of software development.
- Compliance with global data protection regulations (e.g., GDPR, HIPAA) has become mandatory, reinforcing the need for secure software.

Future Outlook:

- Ongoing challenges in cybersecurity, such as the rise in ransomware and phishing attacks, keep pushing the boundaries of software security innovation.
- Emerging technologies like blockchain and quantum computing are reshaping how security is implemented in software systems.

Software Security

- ***Information Security*** refers specifically to the protection of data and how access is granted.
- **Software security** is a broader term that focuses on the design, implementation, and operation of software systems that are trustworthy, including the reliable enforcement of information security.
- **Software security** centers on the protection of digital assets against an array of threats, an effort largely driven by a basic set of security principles that the rest of this chapter will discuss.
- These foundational principles, along with other design techniques covered in subsequent chapters, apply not only to software but also to designing and operating bicycle locks, bank vaults, or prisons.

Trust

- Trust is fundamental but often overlooked.
- Software security is deeply dependent on trust due to the complexity and interconnectedness of modern systems.
  - No single entity can build an entire technology stack from scratch—reliance on third-party components is inevitable.
  - Major systems are built on layers of technology: from hardware up through operating systems and applications.
- Choices in hardware and software often driven by features and price, but each choice also represents a trust decision.
- Trust involves assessing the risk of malice (intentional harm) and incompetence (errors or negligence).

Balancing Trust and Security

- Over-trusting can lead to security breaches if the trusted party fails.
- Under-trusting can result in redundant, unnecessary protective measures.
- Trust operates on a more intuitive level, unlike technical analysis which is explicit and detailed.
- Developing a "gut feel" for software security through experience and intuition can be as effective as detailed analyses.

Competence, Imperfection, and Trust in Software Security

- Inherent Imperfections in Software
  - Most cybersecurity incidents exploit flaws or misconfigurations in software, often introduced despite the good faith efforts of developers and IT staff.
  - Software, inherently created by imperfect humans, is bound to contain **bugs**, some of which may be exploitable by attackers.
- Software licenses typically disclaim nearly all liability, reinforcing a 'buyer beware' approach.
- The inevitability of software bugs means that vulnerabilities exist and can be discovered and exploited.

Trust Decisions in Software Adoption

- Trusting established companies with strong track records in providing reliable software and hardware is common and generally safe.
- Decisions become more complex with less proven entities; the open-source model offers transparency but requires vigilant oversight to mitigate risks from malicious or buggy contributions.

Legal and Regulatory Safeguards

- Despite the absence of guarantees in software security, legal, regulatory, and business frameworks exist to mitigate risks associated with trust decisions.
- No company guarantees perfect security, highlighting the permanent risk of imperfection in software products.

Human Capacity for Trust

- Humans are naturally adept at assessing trust, though traditionally in face-to-face interactions rather than digital.
- Making informed, intentional trust decisions is crucial, even though they can never be perfect. Past performance is not a reliable indicator of future results.

Understanding the Spectrum of Trust

- Trust is not a binary state but is granted in varying degrees based on the context and risk involved.
- Examples range from high-stakes scenarios like major surgery, where patients entrust their lives and consciousness to medical professionals, to everyday situations with built-in safety measures (e.g., credit card limits, valet keys).

Implementing Trust But Verify in Software

- Bridging the gap between complete trust and total distrust, this policy is particularly effective in managing software security.
- Involves granting access or privileges with a system of checks and balances to ensure accountability.

Auditing as a Verification Tool

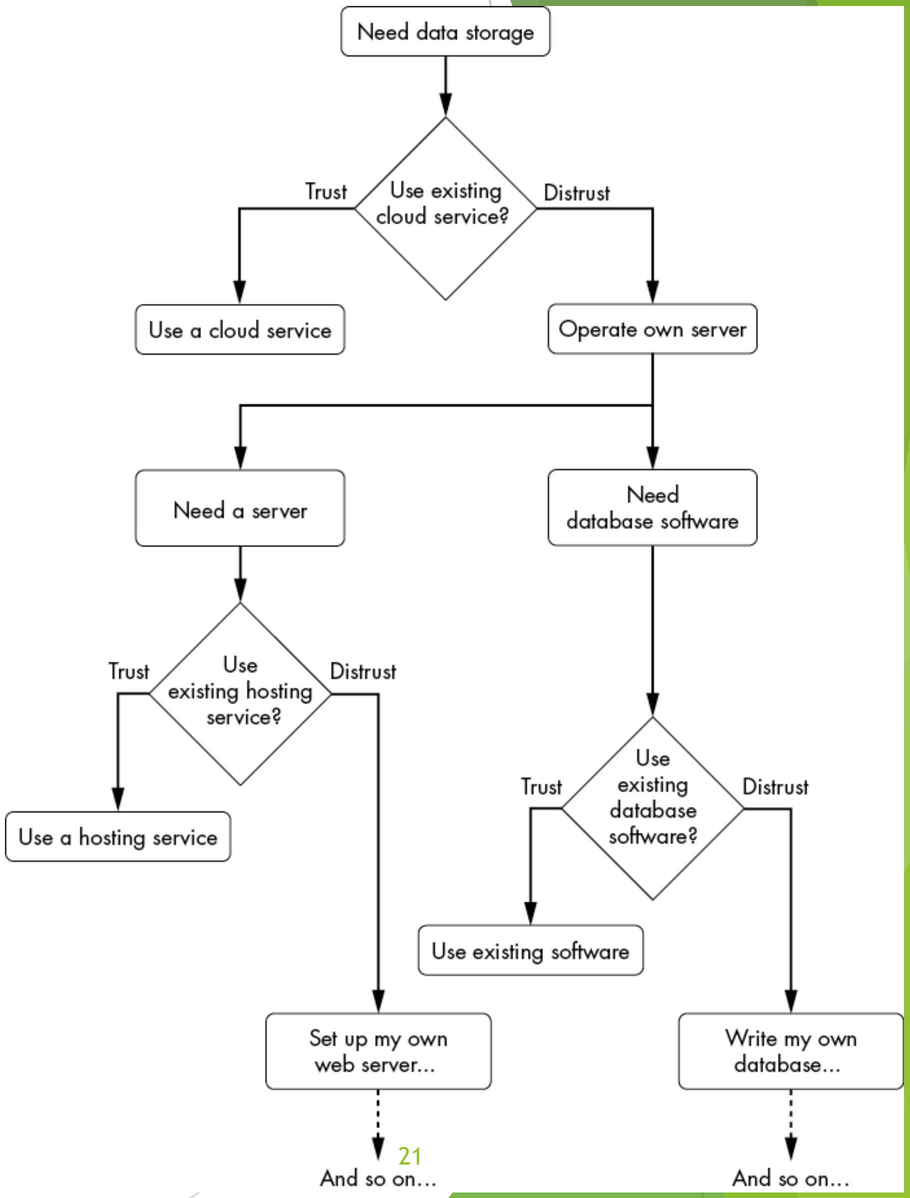
- Combines automated and manual auditing to efficiently and effectively monitor activities and enforce security policies.
- Automated tools handle large volumes of data for routine checks, while manual efforts focus on exceptions and critical decisions.

Navigating Trust Decisions in Software

- In software, the fundamental choice is whether to trust or not to trust specific components and permissions.
- Systems may enforce permissions, but the decision to allow or disallow remains a binary, crucial decision.
- Erring on the side of distrust can be safe if at least one alternative meets trust criteria.
- Excessive distrust may lead to the necessity of creating custom solutions, potentially increasing complexity and effort.

Decision Tree Analogy

- Trust decisions simplify the decision-making process, akin to cutting branches off an otherwise infinite decision tree.
- Trusting a component can eliminate the need for deeper security analysis of that component.
- Example: Lack of trust in available cloud storage services leads to operating your own service, which involves further trust decisions about hosting and database management.
- Each layer of distrust adds complexity and necessitates additional secure implementations.



Handling Inputs with Care

- Treat all inputs, especially from public internet sources, with high levels of scrutiny and security, considering them as potentially untrustworthy.
- Opportunistically add safety checks to inputs, even trusted ones, to reduce system fragility and prevent error propagation.

The Role of Implicit Trust in Software Projects

- Implicit trust refers to the reliance on a stack of technology components without thorough vetting, assumed to be secure based on the vendor's reputation.
  - Includes hardware, operating systems, development tools, libraries, and other dependencies.
- While implicit trust is a necessity in software development due to practical constraints, it is crucial to remain aware of what is being trusted.
  - Evaluate and reconsider these trust levels, especially before expanding the scope of the project or integrating more third-party components.

Strategies for Promoting Trustworthiness

- Being open about how your software works allows customers to assess its reliability and security for themselves.
- Transparency is not just about open source code; it also involves clear communication about product functionalities and operational procedures.

Here are some suggestions of basic ways to enhance trust in your work:

- Transparency engenders trust
- Involving a third party builds trust through their independence
- When problems do arise, be open to feedback
- Specific features or design elements can make trust visible

Information Security: Classic Principles

1. The data access requirements principles C-I-A, the goal of a secure system:
   - Confidentiality
   - Integrity
   - Availability
2. The data control and monitoring principles (Gold Standard), describes the means to secure a system:
   - Authentication
   - Authorization
   - Auditing

Information Security's C-I-A

- We traditionally build software security on three basic principles of information security: *confidentiality*, *integrity*, and *availability*.
- Formulated around the fundamentals of **data protection**, the individual meanings of the three pillars are intuitive:

1. **Confidentiality:**
   - Allow only authorized data access—don't leak information.
2. **Integrity:**
   - Maintain data accurately—don't allow unauthorized modification or deletion.
3. **Availability:**
   - Preserve the availability of data—don't allow significant delays or unauthorized shutdowns.

Confidentiality

- **Understanding What Constitutes Private Information:**
  - Clearly define what counts as sensitive or private information in design documents to avoid misunderstandings.
  - Treat all externally collected information as private by default, relaxing this only with a clear, justified policy.
- Users may expect privacy by default; accidental entry of sensitive information in the wrong fields.
- Legal and regulatory obligations, such as GDPR, may dictate stringent privacy standards based on user geography.

Confidentiality

- **Determining Access:**
  - Define clear rules for who can access private information and under what circumstances.
  - Disclosure decisions are trust-based and should be well-documented to explain the rationale behind access permissions.

The Underestimated Risks of Data Leaks

- **Potential for Misuse**
  - Minor data pieces can be more revealing when combined, leading to unintended insights or reidentification.
- Example: Combination of seemingly innocuous data like ZIP code, age, and profession could lead to deanonymization.

Integrity

- Integrity refers to the authenticity and accuracy of data, safeguarded from unauthorized changes and deletions.
- Ensures data remains true to its original form and authorized modifications.
- Keeping an accurate record of the data’s origin and all authorized changes enhances integrity by providing a verifiable history.

Integrity

- **Understanding What Constitutes Private Information:**
  - Clearly define what counts as sensitive or private information in design documents to avoid misunderstandings.
  - Treat all externally collected information as private by default, relaxing this only with a clear, justified policy.
- Users may expect privacy by default; accidental entry of sensitive information in the wrong fields.
- Legal and regulatory obligations, such as GDPR, may dictate stringent privacy standards based on user geography.

Strategies to Safeguard Integrity

- Data Backup and Version Control
- Advanced Integrity Verification

Potential Risks to Data Integrity

- **Multiple Tampering Vectors**
  - Data tampering can occur in storage, during transmission, or as a result of actions by authenticated users.
  - Common examples include client-side script modifications, data interception, and authorized changes made under false pretenses.

Availability

- Availability ensures that information and resources are accessible to authorized users when needed.
- Attacks on availability can make services temporarily or permanently inaccessible, impacting both users and businesses.

- **Simple Attacks**
  - Basic forms of attacks include overwhelming a server with high volumes of traffic, mimicking legitimate service requests.

Common Threats to Availability

1. **Denial-of-Service (DoS) Attacks**
    1. Anonymous DoS attacks, often demanding ransoms, are a significant threat to internet services.
    2. Typically involve sending excessive traffic to disrupt service availability.

Other Threats

1. Malformed requests causing crashes or loops.
2. Overloads on storage, computation, or communication capacities.
3. Attacks affecting software, configuration, or data integrity, causing delays even with backups.

The Gold Standard

- *Aurum* is Latin for gold, hence the chemical symbol "Au," and it just so happens that the three important principles of security enforcement start with those same two letters:
- **Authentication:**
  - High-assurance determination of the identity of a principal
- **Authorization:**
  - Reliably only allowing an action by an authenticated principal
- **Auditing:**
  - Maintaining a reliable record of actions by principals for inspection

  - Note: A *principal* is any reliably authenticated entity: a person, business or organization, government entity, application, service, device, or any other agent with the power to act.

The Gold Standard

- *Aurum* is Latin for gold, hence the chemical symbol "Au," and it just so happens that the three important principles of security enforcement start with those same two letters:
- **Authentication:**
  - High-assurance determination of the identity of a principal
- **Authorization:**
  - Reliably only allowing an action by an authenticated principal
- **Auditing:**
  - Maintaining a reliable record of actions by principals for inspection
  - Note: A *principal* is any reliably authenticated entity: a person, business or organization, government entity, application, service, device, or any other agent with the power to act.

The Gold Standard

- The Gold Standard acts as the enforcement mechanism that protects C-I-A. We defined confidentiality and integrity as protection against *unauthorized* disclosure or tampering, and availability is also subject to control by an authorized administrator.
- The only way to truly enforce authorization decisions is if the principals using the system are properly authenticated. Auditing completes the picture by providing a reliable log of who did what and when, subject to regular review for irregularities, and holding the acting parties responsible.

Authentication

- An authentication process tests a principal's claims of identity based on credentials that demonstrate they really are who they claim to be.
- Evidence suitable for authentication falls into the following categories:
  - *Something you know*, like a password
  - *Something you have*, like a secure token, or in the analog world some kind of certificate, passport, or signed document that is unforgeable
  - *Something you are*—that is, biometrics (fingerprint, iris pattern, and such)
  - *Somewhere you are*—your verified location, such as a connection to a private network in a secure facility

Authorization

- A decision to allow or deny critical actions should be based on the identity of the principal as established by authentication.
  - Systems implement authorization in business logic, an access control list, or some other formal access policy.

Auditing

- Auditing involves the comprehensive monitoring of critical system events such as authentication, authorization, system updates, and administrative accesses.
- Ensures oversight and accountability, helping mitigate risks including internal threats.
- Audit logs must be secure and protected from tampering, even by administrators, to maintain their integrity as reliable records.

Managing Audit Logs Effectively

- Securing Log Integrity:
  - Implement mechanisms to prevent unauthorized log alteration to ensure that logs accurately reflect all actions without potential for tampering.
  - Use independent systems with different administrative controls for sensitive logs to avoid conflicts of interest and cover-ups.

Best Practices for Effective Auditing

1. **Monitoring and Analysis**
   - Regularly monitor logs for unusual activity, analyze anomalies, and follow up on suspicious actions to maintain system security.
   - Non-repudiation ensures actions logged are incontrovertibly linked to identified users, preventing denial of wrongful actions.
2. **Balancing Log Detail (The Goldilocks Principle)**
   - Avoid logging too much or too little information. Excessive details can overwhelm analysts, while insufficient data might miss critical signals.
   - Strive for optimal detail to ensure actionable insights without data overload.

Bridging Information Security and Privacy

- While the C-I-A (Confidentiality, Integrity, Availability) principles and the Gold Standard anchor information security, privacy intersects subtly but significantly.
- Both fields address the safeguarding of data but differ in scope and focus, highlighting unique and shared challenges.
- Privacy extends the principle of confidentiality by considering additional human factors such as customer expectations, legal regulations, and cultural aspects of data use.

Managing Privacy in Software Design

- Establish and adhere to clear policies for data acquisition, use, sharing, and deletion, ensuring these practices are transparent and comply with legal standards.
- Translate these policies into enforceable software features to maintain data integrity and privacy.

Privacy Challenges in Modern Contexts

- As digital interactions increase, handling personal data responsibly becomes more complex.
- Avoid unnecessary data collection; collect only what is needed for a defined purpose and delete when no longer required or when risks outweigh benefits.

Enhancing Privacy Protection Through Design

1. **Maximizing Transparency and Minimality**
   - Implement maximal transparency in data use policies, ensuring they are simple enough for all users to understand.
   - Minimize the collection of personally identifiable information to reduce potential risks and liabilities.
2. **Communication and Compliance**
   - Clearly communicate privacy expectations to users and maintain a high standard of compliance within the organization.
   - Proactively manage privacy by designing systems that inherently protect user data, addressing both expected and potential misuse.

Definitions to know

Software Source Code

- Source code is the set of human-readable instructions written in a programming language that a computer program is made of.
  - It dictates what the program does and how it operates.
- The code must be precise and syntactically correct to function as intended. Once written, source code is usually compiled or interpreted into machine code, which can be executed by a computer's processor.

Software Bugs

- A software bug is a broad term for any sort of error in a program's source code that causes unexpected or incorrect behavior in the program's execution.
  - Bugs can manifest as faults, errors, or weaknesses, depending on their nature and impact.
- Example: Suppose a calculator app gives the result of 8 when you input 3 + 2.
  - This is a bug because the program is not performing its intended function due to an error in the code.

Software Errors

- An software error refers to the incorrect or undesired state of the program's operation observed as a result of a fault.
  - It's the actual manifestation of a fault during the runtime.
- **Example:** Continuing from the above fault, if the program attempts to execute the division by zero, it results in a runtime error that may cause the program to crash or display an error message.

Software Weakness

- A weakness is a type of fault in the source code that can make the software vulnerable to security threats.
  - It's broader than a fault, as it pertains specifically to vulnerabilities that could be exploited.
- **Example**: If a web application does not sanitize user input for a form field, this is a weakness. An attacker could exploit this weakness to inject malicious SQL queries (SQL injection), potentially gaining unauthorized access to or manipulating the database.

Understanding Through An Online Shopping Cart Calculation 1/

- Imagine a feature in an online shopping cart that calculates the total cost, including taxes, based on the user's location.
- **Bug:**
  1. **Definition**: A general term for any incorrect or undesired behavior in the software.
  2. **Example**: Users notice that the total price displayed in the shopping cart sometimes does not match the sum of individual item prices plus taxes.
  3. **Explanation**: This discrepancy is a bug because it's not what the program should be doing according to its requirements.

Understanding Through An Online Shopping Cart Calculation 2/

- **Fault:**
  1. **Definition**: A specific mistake in the code that can lead to an error.
  2. **Example**: The source code incorrectly multiplies the item price by the quantity after adding the tax, rather than before. This sequence error in code logic is a fault.
  3. **Explanation**: This coding mistake causes the final price calculation to be incorrect under certain conditions, such as when multiple items are in the cart.

Understanding Through An Online Shopping Cart Calculation 3/

- **Error:**
  1. **Definition**: The actual incorrect behavior observed when the program is run.
  2. **Example**: When a customer buys multiple items, the cart incorrectly calculates the total by applying the tax calculation before adjusting for item quantity, leading to a higher total cost.
  3. **Explanation**: The error is the observable incorrect total price, which results directly from the fault in the calculation logic.

Understanding Through An Online Shopping Cart Calculation 1/

- **Weakness:**
  1. **Definition**: A fault in the code that specifically makes the software vulnerable to security risks.
  2. **Example**: The same shopping cart application does not verify whether the quantity values passed to the server are integers. This oversight is a weakness because it could be exploited through an injection attack by passing in script code or SQL commands instead of numbers.
  3. **Explanation**: If an attacker injects malicious code through the quantity field, it could lead to unauthorized actions such as data breaches or database manipulation, exploiting the weakness in data validation.

Summary:

- In the previous example:
  - The **bug** is the noticeable mismatch in total price calculations under certain conditions.
  - The **fault** is the specific incorrect order of operations in the source code during the price calculation.
  - The **error** is the wrong total price displayed to the user.
  - The **weakness** is the vulnerability in the system due to inadequate input validation, potentially leading to security breaches.