SA  
课堂作业报告

得分：

|  |  |
| --- | --- |
| 内容 | 探讨软件系统的安全属性 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 角色 | 小组成员 | QQ号 | 联系电话 | 得分 |
| 组长 | 陈德霖 |  |  |  |
| 文档管理员 | 陈溢嘉 | 1561308699 | 17796321047 |  |
| 其它组员 | 江隽轩 | 2642888754 | 15918790372 |  |
| 陈柏霖 | 2513728919 | 13652561908 |  |
| 钱靖 |  |  |  |

# 一、教材讨论题

1. **Write a set of concrete scenarios for security for an automatic teller machine. How would you modify your design for the automatic teller machine to satisfy these scenarios?**

**Scenarios for ATM Security**

1. **Physical Theft**: Criminals may attempt to steal cash from the ATM or the machine itself through ram raids or burglary.
2. **Logical Attacks**: Cybercriminals could install malware to compromise the system and steal financial data or manipulate transactions.
3. **Skimming**: Fraudsters attach devices to the ATM to capture card information and PINs.
4. **Vandalism**: ATMs may be targeted for vandalism, causing damage to the physical components and disrupting service.
5. **Social Engineering**: Attackers may deceive users or staff to gain access to sensitive information or insert malicious devices.

**Design Modifications for ATM Security**

1. **Physical Security Enhancements**:

**Bolt ATMs to the floor** and use reinforced materials to resist ram raids and burglaries.

Install **high-security locks** on all accessible parts and **tamper-evident seals** on critical components to detect unauthorized access .

1. **Logical Security Measures**:

Implement **intrusion detection systems** and **firewalls** to prevent unauthorized network access.

Regularly update the **ATM software** to protect against known vulnerabilities and ensure the use of **encrypted communication channels** for all transactions .

1. **Anti-Skimming Devices**:

Equip ATMs with **anti-skimming technology** such as sensors that detect foreign objects attached to card slots or cameras monitoring the PIN pad area.

Use **integrated card readers** that are less susceptible to skimming devices and ensure that any additions to the ATM, such as privacy shields, do not facilitate skimming .

1. **Vandalism Protection**:

Use **durability and vandal-resistant materials** for the ATM exterior and screens.

Install **security cameras** with **motion detection** capabilities to deter vandalism and provide evidence in case of an incident .

1. **Social Engineering Defenses**:

Train staff to recognize and respond to social engineering attempts.

Educate users about security practices when using ATMs, such as covering their PIN entry and being vigilant for any unusual ATM behavior or appearance .

1. **Regular Audits and Inspections**:

Perform **regular physical inspections** of ATMs to identify any signs of tampering or unauthorized modifications.

Conduct **security audits** to ensure compliance with industry standards and to identify potential weaknesses in the system .

1. **User Interface Security**:

Ensure that the **ATM user interface** cannot be manipulated to display false prompts or to log keystrokes.

Use **PCI PTS-approved EPPs** (Encrypting PIN Pads) for PIN entry to protect against PIN theft .

1. **Network Security**:

Secure all **communication interfaces** within the ATM and ensure that they do not accept unauthorized connection requests.

Implement **network isolation** techniques and use **intrusion detection/mitigation tools** to protect against network-based attacks .

1. **Software and Firmware Security**:

Ensure that the **ATM software is hardened** and configured to run with the least necessary privileges.

Implement **secure software update mechanisms** that verify the integrity and authenticity of the updates .

1. **Life Cycle Management**:

Manage the entire life cycle of the ATM, from manufacturing to decommissioning, with a focus on security at every stage.

Ensure that **decommissioned ATMs** are properly sanitized to remove all sensitive data

1. **One of the most sophisticated attacks on record was carried out by a virus known as Stuxnet. Stuxnet first appeared in 2009 but became widely known in 2011 when it was revealed that it had apparently severely damaged or incapacitated the high-speed centrifuges involved in Iran's uranium enrichment program. Read about Stuxnet and see if you can devise a defense strategy against it based on the tactics in this chapter.**
   1. **Resist Attacks**: Stuxnet exploited zero-day vulnerabilities in Windows. Regularly update and patch all systems, especially those running critical infrastructure, to protect against unknown vulnerabilities.
   2. **Encrypt Data**: Since Stuxnet targeted industrial systems, ensure that all programmable logic controller (PLC) configurations and operational data are encrypted to prevent unauthorized changes.
   3. **Attack System Detection**: Implement robust monitoring tools that can detect anomalies in industrial control systems, as Stuxnet manipulated physical processes.
   4. **Maintain Audit Trail**: Maintain detailed logs of all system activities, especially changes to PLC configurations, to trace back any unauthorized modifications.
   5. **Limit Exposure**: Stuxnet spread via infected USB drives. Restrict the use of USB drives and other removable media in industrial environments.
   6. **Recover from Attacks**: Have a disaster recovery plan that includes regular backups of critical systems and data, ensuring that you can restore operations quickly after an attack.
   7. **React to Attacks**: Develop an incident response plan that includes steps to isolate affected systems to prevent the spread of the virus.
   8. **Revoke Access**: Stuxnet exploited legitimate credentials. Implement strong authentication mechanisms and promptly revoke access for any compromised credentials.
   9. **Lock Computer**: Implement automatic system lockdown after a period of inactivity to prevent unauthorized access.
   10. **Detect Intrusion**: Use intrusion detection systems (IDS) that are capable of monitoring for the specific types of behavior exhibited by Stuxnet, such as unauthorized PLC modifications.
   11. **Change Default Settings**: Stuxnet exploited default settings. Change default passwords and configurations on all systems, especially industrial control systems.
   12. **Separate Entities**: Stuxnet moved laterally within networks. Segment networks to limit the spread of malware and ensure that critical systems are isolated.
   13. **Identify Actors**: Use identity and access management systems to track and control access to critical systems.
   14. **Authorize Actors**: Ensure that only authorized personnel have access to change system configurations or operational parameters.
   15. **Inform Actors**: Educate personnel on the risks associated with malware like Stuxnet and the importance of following security protocols.
   16. **Binding Time**: Since Stuxnet was a late-bound attack, ensure that there are mechanisms in place to validate and control the behavior of all system components, especially those loaded at runtime.
   17. **Choice of Technology**: Choose security technologies that are designed to protect against advanced persistent threats (APTs) and are capable of detecting and responding to complex attacks like Stuxnet.
2. **Some say that inserting security awareness into the software development life cycle is at least as important as designing software with security countermeasures. What are some examples of software development processes that can lead to more-secure systems?**

Integrating security awareness into the software development life cycle (SDLC) is crucial for creating more secure systems. Here are some key processes that can lead to enhanced security:

1. **Security Planning**: This involves identifying and documenting security requirements at the outset, considering features like access controls and data protection .
2. **Security Training**: Ensuring that all team members, including developers, testers, and operations staff, are trained in secure coding practices and are aware of the latest security threats and mitigation strategies .
3. **Threat Modeling**: This process involves identifying potential threats to a software system and designing mitigations to address those threats .
4. **Code Reviews**: Regular code reviews can help identify and fix security vulnerabilities before they are exploited .
5. **Static and Dynamic Analysis**: Using tools to automatically detect security vulnerabilities in code during development .
6. **Penetration Testing**: Simulating attacks on the system to identify vulnerabilities that can be exploited .
7. **Secure Deployment**: Ensuring that servers and environments are securely configured and that access controls are in place during deployment .
8. **Maintenance and Monitoring**: Regularly updating and patching software, monitoring for security events, and having incident response procedures in place .
9. **Security and usability are often seen to be at odds with each other. Security often imposes procedures and processes that seem like needless overhead to the casual user. But some say that security and usability go (or should go) hand in hand and argue that making the system easy to use securely is the best way to promote security to the user. Discuss.**

Security measures often make systems harder to use, while enhancing usability may compromise security. A typical example is password complexity requirements that hinder usability but improve security​. These conflicts stem from the different concerns of the two, with security focusing on system protection and ease of use focusing on user experience. But the idea that security and usability can coexist harmoniously is supported by many experts. Rather than being at odds, security and usability can complement each other when systems are designed with user experience in mind. Here are some possible ways proposed in some papers supporting this viewpoint:

**Collaboration Between Teams**: To bridge the gap between security and usability, the document encourages collaboration between security experts and usability designers. By working together, these groups can identify potential conflicts early in the design process and resolve them through shared patterns.

**Use of Design Patterns**: The primary solution to managing the trade-off is through the application of **design patterns**. These patterns offer established solutions to common problems where usability and security may conflict. They help developers integrate security into systems without compromising usability. For example, patterns like "Toggle Password Visibility" maintain security while improving user experience.

**Participatory Design Workshops**: The paper we find advocates organizing workshops where security and usability teams collaborate to create and refine these patterns. This participatory approach ensures that both aspects are considered in tandem, rather than in isolation.

The workshop consists of the following phases:

Distribute case stories: Distribute case stories that describe usability security issues to participants.

Identify patterns: Conduct a comparative analysis of each group's solutions to identify good design examples.

Use scenarios for validation: Provide a set of design patterns and problem scenarios for participants to choose the applicable patterns and apply them.

Document validated patterns and lessons learned: Culminate in a catalog of availability security design patterns.

1. **List some examples of critical resources for security that might become exhausted.**

There are several situations, which are:

1. **Denial of Service (DoS) Attacks**

Impacted Critical Resources:

**CPU**: Overwhelming the system with requests can exhaust processing capabilities, preventing it from responding to legitimate requests.

**Memory**: Excessive requests can consume available memory, causing the system to fail to handle new requests.

**Network Bandwidth**: Attackers can flood the network with traffic, blocking legitimate user access.

1. **Distributed Denial of Service (DDoS) Attacks**

Impacted Critical Resources:

**Network Bandwidth**: Multiple sources can quickly consume available bandwidth, making the system inaccessible.

**Processing Power**: The need to handle simultaneous requests from many sources can exhaust CPU and memory resources.

**Connection Resources**: Systems may have a limited number of open file handles or network connections that can be exhausted by DDoS attacks.

1. **Buffer Overflow Attacks**

Impacted Critical Resources:

**Memory**: Inputting data that exceeds the buffer's limit can overwrite critical memory, causing crashes or security vulnerabilities.

**Processing Power**: If malicious code is executed, it can utilize CPU resources for further attacks.

1. **Privilege Escalation Attacks**

Impacted Critical Resources:

**System Permissions**: Attackers exploit vulnerabilities to gain higher access levels, compromising system security.

**File System and Data**: With elevated permissions, attackers can access, modify, or delete crucial system files and user data.

1. **Resource Exhaustion Attacks**

Impacted Critical Resources:

**Memory**: By making numerous requests, attackers can deplete available memory, leading to instability or crashes.

**File Handles or Network Connections**: By consuming many resources, attackers can prevent legitimate users from accessing the system.

1. **SQL Injection Attacks**

Impacted Critical Resources:

**Database**: Attackers can gain control over databases, allowing access to, modification of, or deletion of sensitive data.

**Storage Space**: Inserting excessive data can exhaust the database's storage capacity.

1. **Password Cracking Attack**s

Impacted Critical Resources:

**CPU and Processing Power**: Brute force or dictionary attacks consume significant processing resources as they attempt to crack user passwords.

**Security and Privacy Data**: Successful password cracking can lead to unauthorized access to sensitive information.

1. **Phishing Attacks**

Impacted Critical Resources:

**User Data and Privacy**: Phishing scams trick users into revealing sensitive information, such as usernames, passwords, or bank details, leading to data breaches.

1. **Malware**

Impacted Critical Resources:

**CPU and Memory**: Malware can consume system resources (e.g., for cryptocurrency mining or sending spam) and degrade performance.

**Storage Resources**: Some malware may occupy storage space or encrypt files for ransom (ransomware).

1. **List an example of a mapping of architectural elements that has strong security implications. Hint: think of where data is stored.**
2. **Which of the tactics in our list will protect against an insider threat? Can you think of any that should be added?**
3. **In the United States, Facebook can account for more than 5 percent of all Internet traffic in a given week. How would you recognize a denial-of-service attack on Facebook.com?**

Signs of a denial-of-service attack on Facebook.com might include:

1. **Unusually slow network performance:** If the site loads slowly or not at all, it could indicate an attack.
2. **Inability to access the website:** Widespread reports of users being unable to access Facebook might suggest a DoS attack.
3. **Sudden surge in network traffic:** A sharp increase in traffic could be a result of attackers sending a flood of requests.
4. **Flood of error messages:** Frequent error messages such as "service unavailable" or "server error" might indicate that the server cannot handle the requests.
5. **Drastic increase in spam emails:** A sudden influx of spam or malicious requests could be part of the attack.
6. **Monitoring for network anomalies:** Keeping an eye on network traffic for unusual patterns, such as a sudden increase in traffic or unusual packets, is crucial.
7. **Security measures:** Having security measures like firewalls and intrusion detection systems in place can help identify and mitigate potential attacks.

If there is suspicion of a DoS attack on Facebook.com, it would be advisable to contact the internet service provider and use security tools to analyze traffic and block malicious packets.

1. **The public disclosure of vulnerabilities in production systems is a matter of controversy. Discuss why this is so and the pros and cons of public disclosure of vulnerabilities.**

The public disclosure of vulnerabilities in production systems is indeed a contentious issue. There are several reasons why this is so, and it's important to consider both the pros and cons of such disclosure.

**Pros of Public Disclosure:**

1. **Raising Awareness:** Public disclosure can alert a wide audience to the existence of a vulnerability, which can lead to increased vigilance and a quicker response from the affected organizations
2. **Pressure for Quick Fixes:** When vulnerabilities are made public, there is often public pressure on companies to address the issue promptly, which can lead to faster remediation.
3. **Transparency and Trust:** Organizations that are transparent about their security issues can build trust with their users and the public, demonstrating a commitment to security.

**Cons of Public Disclosure:**

1. **Risk of Exploitation:** Public disclosure can provide would-be attackers with detailed information on how to exploit a vulnerability, potentially before a fix is widely available.
2. **Damage to Reputation:** Companies may be reluctant to disclose vulnerabilities publicly due to the potential damage to their reputation and the trust of their customers.
3. **Legal and Economic Challenges:** There can be legal barriers and a lack of cooperation among stakeholders, as well as limited market incentives for security researchers to participate in coordinated vulnerability disclosure.

**Responsible Disclosure as a Middle Ground:**

A middle ground that is often advocated is responsible or coordinated disclosure. In this model, vulnerabilities are reported privately to the organization first. The details are then disclosed publicly once a patch is available, sometimes with a delay to allow for widespread patching. This approach aims to balance the need for quick fixes with the desire to minimize risk to users.

In conclusion, the decision to publicly disclose vulnerabilities should be made carefully, weighing the potential benefits of increased awareness and pressure for fixes against the risks of exploitation and reputational damage. Responsible disclosure policies can help navigate this complex issue.

# 二、思考题

1. **功能性需求与质量属性的关系？**

功能性需求描述了系统必须执行的功能以及它必须遵守的行为或反应。质量属性需求则是对功能性需求或整个产品的其他方面的要求，例如执行某项功能的速度、对错误输入的弹性或在特定硬件上的性能等。简而言之，功能性需求关注系统“做了什么”，而质量属性需求关注系统“如何做”以及“做得怎么样”。

1. **质量属性与软件架构的关系？**

软件架构通过其设计的结构和元素行为来满足质量属性需求。架构的不同结构（如模块化结构、组件-连接器结构、分配结构）直接影响系统的质量属性，如性能、可靠性、可用性等。质量属性是在选择和设计架构时必须考虑的关键因素，因为它们定义了系统应该如何表现以及如何满足利益相关者的需求。

1. **什么是质量属性？**

质量属性（Quality Attribute, QA）是衡量系统满足其利益相关者需求的程度的属性。它们是系统的可测试或可测量的特性，用于指示系统在某个利益相关者关心的维度上的表现“好坏”。质量属性可以是运行时属性（如可用性、性能、安全性）或开发时属性（如可修改性、可测试性）。

1. **如何理解质量属性？**

质量属性是衡量软件系统满足其利益相关者需求的能力的一系列特征。它们通常描述了系统在特定方面的性能，如速度、可靠性、安全性等。质量属性通常涉及以下几个方面：

1. **需求分析**：识别和理解利益相关者对系统性能的期望和需求。
2. **场景构建**：构建具体的场景或用例，以描述系统在特定情况下的行为和性能。
3. **度量和测试**：为每个质量属性定义可度量的指标和测试方法，以便可以评估系统是否满足这些属性。
4. **权衡识别**：理解不同质量属性之间可能存在的权衡，如提高安全性可能会牺牲一定的性能。
5. **设计决策**：考虑如何在软件架构和设计中做出决策以支持所需的质量属性。
6. **如何理解架构决策的7个分类？**

架构决策是帮助架构师在设计软件系统时，系统地考虑和解决关键问题的一种方法，可以按照以下七个分类来理解：

1. **分配责任**：确定系统的重要职责，并决定如何将这些职责分配给运行时和非运行时元素。
2. **协调模型**：定义系统元素之间的协调机制，如同步或异步通信。
3. **数据模型**：选择数据抽象及其操作，以及如何组织数据。
4. **资源管理**：确定如何管理和共享资源，如CPU、内存和网络。
5. **架构元素之间的映射**：定义软件元素与环境元素之间的映射关系。
6. **绑定时间决策**：决定在软件生命周期的哪个阶段以及如何实现元素之间的绑定。
7. **技术选择**：选择实现架构决策的具体技术，如编程语言、数据库、中间件等。

复制分享

# 三、课堂作业小组讨论与结论

示例：

【以下仅供参考，不足之处是没有会议记录】

通过这次课程作业，我们学会了如何分工、合作一同完成任务，并且在检索、查阅、阅读相关文献的过程中，学会了如何去解决问题、思考问题，与组员一同讨论克服困难。

在检索文献的时候，我们碰到了一些问题，有些文献很难查找，但是我们的组员通过分享在查阅过程中的心得和经验，一同解决了问题。我们最后总结了，Google学术搜索是最适合检索论文的，但是可能有一些文献还是很难找到，可能在IEEE文献搜索网站上也能找到，这样我们就基本解决了问题。我们总结了一下，学术论文，尤其是国外的学术论文，我们必须通过特有的途径来搜索，目前我们能够运用的便是谷歌学术搜索、IEEE论文搜索引擎以及学校图书馆的数据库【优点：说明清楚了，团队成员如何发现具体问题、具体困难，如何提出具体的解决方案】。

在阅读文献的时候，我们其实还是有很多不太能理解的地方，毕竟我们大多数对软件体系结构知识的理解还不算是很透彻，也很少有组员有相关的开发经验或者是比较深刻的理解。软件体系结构（Software Architecture）本身也是一个很大的学术领域，有着很长的学术历史，是一个涉及了广大学术领域的学科，各个国家的著名科学家和学术研究人员在这其中花了很大的人力和物力。所以其实正真想掌握这门学科的话，不是说读几篇论文就能理解和明白的，应该是在一个正真庞大的项目中，亲自去体验这个过程，并且理解、总结、分析，和各大学术界的泰斗进行交流，这样才能算是开始理解软件体系架构。如果要深入发展，我觉得就算是花上数个世纪都不为过，因为学无止境！在学术领域里面我们要学会谦虚，虚心好学才能使人不断地进步。

当然，当下而言对我们来说，便是踏踏实实，认认真真的完成每一次作业，在一次又一次的作业中，我们相信终能够锻炼自己，逐渐提升自己的团队合作能力和学术研究能力！

【优点：深刻理解了软件架构的复杂性和发展性，指出了如何通过长期积累、持续更新学习的方式，掌握软件架构，控制好复杂性和发展性】