Secure Design Principles

Prof. Dr. Marc Rennhard, Dr. Stephan Neuhaus
Institut für angewandte Informationstechnologie InIT
ZHAW School of Engineering
rema | neut @zhaw.ch

This topic is covered, e.g., in the following books and online resources:

- *J. Viega and G. McGraw, Building Secure Software, ISBN-13: 978-0201721522* (Chapter 5: Guiding Principles for Software Security)
- *N. Dasawani, C. Kern and A. Kesavan, Foundations of Security, ISBN-10: 1590597842* (Chapter 2: Secure Systems Design, Chapter 3: Secure Design Principles)
- *https://www.owasp.org/index.php/Security_by_Design_Principles*

The slides in this chapter partly contain information from these books and online resources.

Content & Goals

Content

- Ten Secure Design Principles you should keep in mind when thinking about security during software development
- Exercise to apply the principles to an e-banking scenario

Goals

- You know and understand the secure design principles and can provide examples for each of the principles
- You can apply the principles to any given scenario and assess whether a principle is met or violated

Secure Design Principles

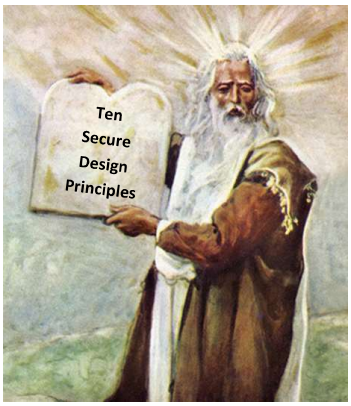
- Secure Design Principles are very fundamental security guidelines that provide a good general basis for securing IT systems

- «Fundamental» means the following:
  - They have established themselves over time based on a lot of experience in practice → they have demonstrated their effectiveness
  - They are technology-independent → they are relevant and applicable in virtually every IT system
  - They are time-agnostic → they were valid many years ago, they are valid today, and they will most probably also be valid in the future

- When developing secure systems, it's important to keep these principles in mind
  - Using them appropriately in your system should help you avoiding many security problems

Secure Design Principles Overview

- In this chapter, we are discussing the following ten principles
  - Secure the Weakest Link
  - Defense in Depth
  - Fail Securely
  - Principle of Least Privilege
  - Separation of Privileges
  - Secure by Default
  - Minimize Attack Surface
  - Keep Security Simple
  - Avoid Security by Obscurity
  - Don't Trust User Input and Services



- This is not an official list and the order does not reflect the relative importance of the principles
  - When looking at books / online resources that deal with secure design principles, then these are the ones that are typically discussed

Secure the Weakest Link (1)

- An IT system usually consists of several different components
  - Client-side software (e.g., browser or app), client OS, client hardware, network and communication protocols, server hardware, server OS, server-side application, database,…
  - But also users, administrators, support personnel, support processes,…
- The overall security is determined by the weakest component
  - Attackers try to identify the weakest component of a system as this is where they will most likely succeed
- This means that one should try to identify and mitigate the weak links during development of an IT system
  - Identifying the weak links is usually done by performing threat modeling and penetration tests in combination with risk analysis
  - Good risk management should address the highest risks first and not the ones that are easy to mitigate
    - This guarantees that security investments are made at the right places

Security is determined by the Weakest Link

This is very important to understand. If security is done well in 99 places in a system but not well in one place, then this one place determines the overall security, because an attacker will usually only need one «weak place» to successfully execute an attack. So security is not an «average value» over all the security measures, but it's determined by the weakest component overall.

Secure the Weakest Link (2)

- Example: Assume the communication between client and server is secured with TLS that is configured correctly and that uses strong ciphers
  - Attackers will most likely not try to break TLS, as the success probability is virtually zero → TLS (used correctly) is not a typical weak link
    - Therefore, it's probably pointless to invest more to secure the communication
  - The attackers will focus their attacks on other more promising areas, e.g., the users or vulnerable client or server components
    - Correspondingly, it's much more reasonable to make further security investments also in these areas where a true benefit can be expected
- Typical weak links:
  - Weak passwords, especially if users are not forced to fulfill minimum password strength requirements (length, character mix etc.)
  - Wrong usage of cryptography, e.g., using outdated algorithms, too short keys, insecure cipher modes, insecure random number generators
  - Missing/incorrect input validation, which is the main cause of many attacks
  - Insecure support processes, e.g., related to password reset
  - People, e.g., social engineering attacks on end users

Defense in Depth (1)

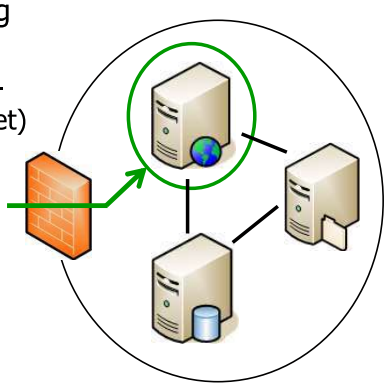
- Defense in Depth means that risks should be managed with multiple diverse defensive strategies
  - If one layer of defense fails, another layer may prevent a successful attack

Defense in Depth and Buffer Overflows

We have already talked about defense in depth when discussing buffer overflows: The first layer of defense should be secure programming, i.e., making sure no buffer overflow vulnerabilities are in the code. If a vulnerability still occurs, we then can still hope that the various technical defenses (stack canaries, ASLR etc.) help to prevent exploitation of the vulnerability.

Defense in Depth – Example (1)

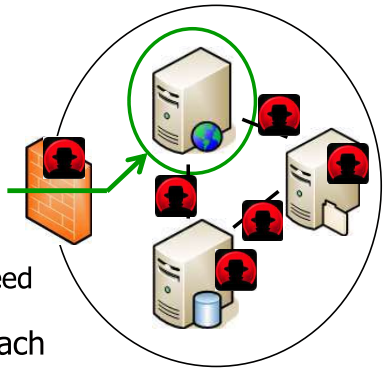
- A web application consisting of multiple server components that are located in the same server network
- The development team decides that the following security measures should be implemented:
  - Harden the web application server, as this component can be reached from the outside (the Internet)
    - Install only required software and latest patches, use a secure software development process for the web application
  - On the border of the server network, use a router-firewall which allows only access to the web application server and only with HTTPS



- In addition, they decide the following:
  - Communication between the server components can be done in the clear, because for the attacker it's virtually impossible to get into the server network
  - Servers that are not directly exposed don't require much security attention, because they cannot be reached by an attacker from the outside

Defense in Depth – Example (2)

- At a first glance, this sounds reasonable
- But what if an attacker compromises the router-firewall?
  - He gets access to all non-encrypted traffic in the network (ARP spoofing)
  - He sees the other server components and may likely find vulnerabilities that can be exploited to compromise them
  - → The current security approach is not a defense in depth approach as the attacker has to overcome only one layer of defense to succeed



- To improve security, a defense in depth approach should be implemented, which should include:
  - Encrypting the traffic also between the server components, which prevents the attacker from reading and manipulating it
  - Harden not only the exposed, but also the apparently «hidden» server components to avoid that they can easily be compromised

Web Applications Firewall (WAF)

With the measures above, compromising the firewall won't allow the attacker to read transmitted data or to easily attack the «hidden» server. But the web application itself or the underlying application server can still be attacked directly as it is exposed to the Internet. To implement a defense in depth approach there as well, one could use a WAF in front of the web application server. In this case, the attacker now would have to defeat / compromise two systems: the WAF and the web application server.

Using a lot of Security Measures is not the same as Defense in Depth

Sometimes, defense in depth is misunderstood. It does not mean «using a lot security measures in general is a good thing», instead it means «using more than one security measure to protect from specific attacks is a good thing». For instance, using TLS, a packet filtering firewall and system hardening are all good and important to protect access to a target server, but it's not a defense in depth approach, because these security measures protect from totally different attacks. But placing a server in a locked room and encrypting the hard disk is defense in depth, as the attacker now has to solve two problems – getting into the room to steal the server (or its hard disk) AND breaking the encryption.

Defense in Depth (2)

- Defense in Depth also means thinking beyond preventive measures
  - This means that you have additional security measures that «take over» in case the preventive measures have failed and the attack takes or took place
- A powerful strategy should include mechanisms to prevent, detect, contain and recover from attacks
- Example: How do banks defend against robbery?
  - Security guards outside the bank and cameras can prevent attacks
  - Detecting the attack is usually quite simple in the brick-and-mortar world
  - Having bank tellers stationed behind bulletproof glass may help containing the attack if violence breaks out (protecting the lives of the tellers)
  - Another containment measure is that two people – who are rarely in the bank at the same time – must each provide a key to open the vault with the really valuable stuff
  - Putting the money in a specially prepared briefcase that emits some sort of paint when the robbers open it may help recovering from the attack
  - Appropriate insurance is also a method to recover from the attack

Containing an Attack

This means to have measures that make sure that once an attack has happened, the damage that can be done by the attack is limited (contained).

Defense in Depth (3)

Prevent / Detect / Contain / Recover in the cyberworld using an online password guessing attack as example:

- Password-guessing attacks can be prevented by requiring users to pick passwords with certain quality requirements and by inducing a time delay after a few wrong passwords
- Password-guessing attack attempts can be detected by monitoring server logs for large numbers of failed login attempts from the same or a few IP addresses (an IDS can do this)
- Assuming an attacker manages to capture username / password pairs, one can contain the attack by denying all logins from suspicious IP addresses (assuming this can be detected) and by changing the passwords
- Finally, to recover from the attack, one could do more detailed monitoring of the accounts and source IP addresses involved in the attack

Diversity in Defense

Diversity in Defense is often mentioned along with Defense in Depth. The idea is to use multiple heterogeneous systems that do the same thing. For instance, using different operating systems to store multiple backup copies will likely reduce the risk of a full backup loss in case a particular operating system type is infected by malware. Of course, Diversity in Defense comes at a cost, as you need experts in your IT staff for multiple systems and the complexity increases (e.g., you have to monitor and apply patches to different types of systems).

Fail Securely (1)

- Fail securely means that a failure (which will happen in complex systems) must not compromise the security of the system
  - When this principle is violated, it's typically because of poor code, poor operational procedures, or poor configuration
- Example of poor code:

```
boolean isAdmin = true;
try {
  /* doAdminLogin() presents a view to the user where he can enter
     username and password. When receiving the credentials, they
     are checked. If the credentials are valid credentials of an
     administrator, doAdminLogin returns true, otherwise false. If
     an error happens during the login, an exception is thrown. */
  isAdmin = doAdminLogin();
} catch (Exception ex) {
  log.write(ex.toString());
}

// If isAdmin is true, grant access to admin functions...
```

Question: What can go wrong here?

Example of Poor Code

Assume the code is part of a program that allows to do administrative tasks and the purpose of the code section is to authenticate the user to make sure that only legitimate administrators get access.

Fail Securely (2)

- Poor operational procedures are often driven by the desire to «keep things running at all costs» in case of a failure

- Example: A company uses correctly configured firewalls to protect access to its various systems
  - The company also has a spare firewall in case one of the firewalls becomes inoperable
  - That firewall is configured to «let through everything» so it can easily replace a malfunctioning firewall with minimal service interruption
  - An attacker who knows this can try to disable a firewall (e.g., by remotely exploiting a vulnerability that crashes the firewall); as a result of this it may be that the firewall is replaced with the spare one
  - Until the new firewall is correctly configured, this gives the attacker a significant advantage as he may now be able to access (and attack) systems that normally wouldn't be accessible for him

Fail Securely (3)

- An example of poor configuration is a system that uses a secure communication protocol, but the system is configured to also support older versions of the protocol, which are not secure anymore
  - E.g., because older clients supporting only an older version are still used
  - But this means that if using the secure protocol version fails, the system gets less secure → fail securely principle is violated
- Even worse, this may be exploited in a version downgrading attack
  - In this attack, the attacker forces both endpoints to use the older, insecure version although both would support the new one
  - To do so, the attacker acts as a man-in-the-middle and modifies some protocol messages to convince both client and server that the other side only supports an older protocol version
- Examples:
  - Downgrading of the NTLM protocol to its predecessor LM, which allows cracking even complex user passwords
  - Downgrading of TLS 1.2 to SSL 3.0, which may allow an attacker to get access to the plaintext data

Version Downgrading Attack on NTLM

In case you visited the module IT-Sicherheit (IS) you may remember the lab “Password Cracking”, where you cracked LM password using rainbow tables.

Version Downgrading Attack on TLS (POODLE attack)

Details about how TLS can be downgraded to SSL 3.0, which can then be decrypted using a padding oracle attack, can be found here: *https://access.redhat.com/blogs/766093/posts/1976403*

Fail Securely (4)

Another example: Credit card payment authorization

- To prevent credit card fraud, vendors use terminals to check a card before payment is authorized
  - Requires the user to enter the correct PIN
  - The terminal contacts the credit card company to make sure the card is not reported stolen or that the credit limit has not yet been reached
  - In addition, the transaction is denied if any suspicious spending pattern is detected

- But what if the terminal or the communication link is down?
  - Vendors still sometimes use old credit card imprinters as backup, where none of the checks above take place (only needs the card and a handwritten signature)

- This means that if the system fails, it gets less secure
  - This can be exploited in a «physical version downgrading attack»
  - To execute the attack (to make purchases with a stolen card), cut or jam the communication link before shopping

Principle of Least Privilege (1)

- A user or program should be given the least amount of privileges necessary to accomplish a task
  - The purpose of this is to make sure a malicious user or a compromised / malicious program can only cause limited damage
- One reason why this principle is often violated is laziness
  - It's usually simpler to write or run a program when it gets full access rights as one does not have to think about permissions to access low level system resources (e.g., files)
  - Likewise, it may be simpler to give employees local admin rights on their workplace computers to install the software they need for working
- Example of good practice: Postfix
  - Postfix is a popular *nix mail server
  - It is heavily modularized and each component (e.g., delivery to local mail-boxes, send outgoing e-mails) gets minimal access rights to do its job
  - There's just one (small) component running as root (master daemon) as binding to ports 25/465/587 is only possible by a process that runs as root

Postfix

Communication between the Postfix components, which are all individual processes, takes place via inter-process communication techniques that are offered by every modern operating system. On *ix systems, this happens via Unix Domain Sockets.

An overview of the anatomy of Postfix can be found here: *http://www.linuxjournal.com/article/9454*

Principle of Least Privilege (2)

Examples of bad practice:

- Up to Windows XP, many programs required administrator privileges to function correctly and consequently most users worked with full administrator rights
- Many server programs still run with root privileges under *nix by default, which has repeatedly caused security problems in the past (e.g., sendmail)
- Web applications accessing a database often do this with a database user that has full access rights (sometimes including rights to modify the database schema)

Separation of Privileges (1)

- The goal of separation of privileges is preventing that a single user can carry out and conceal an action (or an attack) completely on his own
  - The purpose of this is to enforce that the action can only by carried out and concealed through the collusion of multiple persons

- Separation of privileges is very prevalent in the physical world, e.g.:
  - At ZHAW, ordering any goods requires always at least two signatures
  - In airplanes, there must be two pilots in the cockpit to help (among other things) preventing deliberate crashes
  - Often, this is referred as the four-eyes principle

- In the IT world, separation of privileges is usually enforced by separating the entity that approves an action, the entity that carries out an action, and the entity that monitors an action

Principle of least Privilege and Separation of Privilege

These two principles are often mixed up, but they are two completely different things:

- The first one – principle of least privilege – means a person or program gets only the minimally necessary rights to do a task. The motivation for this is to minimize damage in case a person or a (compromised) program performs illegitimate actions. An example is not giving normal users administrator rights on servers.
- The second one – separation of privilege – means that a single person cannot carry out and conceal an action completely on his own. The goal of this is to prevent that specific attack can be carried out at all by a single person (or that the attacks are not attempted as the risks of being detected are too high). An example is making sure that an employee, who is responsible for ordering office furniture in a company, cannot abuse this to order furniture for his own (personal) purpose. Another example is making sure that an administrator, who «by definition» has administrator rights, cannot easily abuse this to do malicious actions.

Separation of Privileges (2)

Some examples of separating privileges in IT systems:

- Application workflows
  - If an application should provide a workflow that enforces a four-eyes principle, this can usually easily be integrated into the application code
  - E.g., in an application where employees can order goods, an order is only executed after his superior logs in and approves the order
- Software development lifecycle
  - E.g., software developers should not be allowed to deploy a new release of the software on the production system to prevent that a malicious developer can easily include malicious functionality
  - Typically solved by having different people / teams that are responsible for software development → testing and approval → deployment
- Administrative functions
  - E.g., database administrators should not have root access to the underlying system
  - To prevent that a malicious database administrator who changes data directly in the database can remove his traces from the DB and system logs

Database Administrators without root Access

In this case, the actual system administrator of the system where the DBMS is running configures the DBMS so it does the desired logging. This configuration typically requires root rights and cannot be adapted by any other user of the system. In addition, the log files can only be written by the DBMS, which is usually enforced by using a special user to run the DBMS (in the case of the MySQL DBMS, this is for instance the user *mysql*) and giving only this user write access to the logs. As a result of this, the database administrator, who typically uses a tool to connect to the DBMS to do DB administration tasks, can neither alter the log settings of the DBMS nor the log files, as he does not have root access to the system.

Separation of Privileges (3)

In practice, separating privileges is often difficult:

- In small companies, there are often simply not enough people to enforce separation of privileges
- Especially with administrative functions, enforcing the principle is often hard or even impossible
  - A system administrator with root access can usually perform any function an hide all the traces
  - The administrator doing the backups and having access to the backup media usually can get access to a lot of sensitive company data
  - ...

Trust in Administrators

Administrative access to critical systems should only be given to administrators that are highly trusted, e.g. because they have been part of the company for a long time and have earned their reputation or because background checks of newly hired administrators did not reveal any skeletons in the closet.

System Administrator with root Access

One can still do something against a malicious system administrator. In high-security environments such as in the case of banks, access to critical systems is usually only possible from within the computing center of the bank, which is physically well protected. Remote administration of these critical systems is not possible, so the system administrator must be physically present in the computing center to to administrative tasks. To enforce a four-eyes principle in this case, one make sure that the administrator cannot go alone into the computing center, but only when he is accompanied by a second person who observes and writes down the tasks performed by the administrator.

Backup Administrator

Also in this case, separation of privileges can be done. To do this, the actual system administrator of the systems that should be backed up installs the backup software on these systems and configures it – using his root rights – so that the backup software uses a key, which can, e.g., be located in a file, to encrypt the backup. The backup software itself typically runs with root rights as it has to be able to access all other files on the system so one can make sure, using file access permissions, that the key also can only be accessed with root rights. The backup administrator uses the backup program to do additional settings such as how many times to backup, where the backups should be written to etc., but to do this, no root rights are needed and correspondingly, the backup administrator cannot get access to the key. As a result of this, the backup administrator won't be able to read plaintext data from the backup media.

Secure by Default

- Programs and operating systems should be implemented / delivered / installed with secure default configurations
- This includes, e.g., the following:
  - Security features provided by the application / the system should be turned on by default (e.g., two-factor authentication)
  - The default behavior of a newly installed desktop OS should be that the endpoint firewall and automated software updates are enabled
  - When using a TLS library (e.g., openssl or JSSE), the default behavior should be to support only secure protocol versions and cipher suites
  - If a user is added to an application or a system, he should only get the minimal rights per default; elevated rights must be configured explicitly
- Secure by Default has improved a lot during the past 10-15 years
  - Browsers supported insecure SSL versions / cipher suites → fixed
  - Automated update mechanisms were the exception → standard today

Minimize Attack Surface

- The attack surface of an IT system is the sum of the different points where an attacker can try to perform his attacks
  - This includes accessible services (open ports), web service APIs, web forms – basically any piece of code on a system that can somehow be reached by an attacker (directly or indirectly)
  - Obviously, the larger the attack surface, the more likely an attacker will be successful

- Some best practices to minimize the attack surface
  - Make sure that the software you develop only contains the necessary features (fewer features → less code → smaller attack surface)
  - Turn off features that are not or only rarely used (e.g., file sharing)
  - Only install the necessary software on productive systems, especially make sure that no unnecessary exposed services are available
  - Use packet-filtering firewalls to make sure that services that should only be available internally are not exposed to the Internet

Keep Security Simple (1)

- Make sure that in your software (or IT system), security is designed and implemented as simple as possible and only as complex as necessary
  - It's much more difficult to develop and maintain a complex approach in a secure way
  - Likewise, it's much more difficult to analyze and test a complex approach with respect to security – the likelihood you are going to miss something important is very high
- Make sure that it is simple to use your software (or IT system) in a secure way
  - Users shouldn't have to read manuals so they learn how to use a system in a secure way (you can try, but they probably won't read them anyway)
  - Users shouldn't have to make important security decisions when using the program and shouldn't be expected to do security configurations correctly (that's why «Secure by Default» is important)
  - Don't give the users the option to circumvent security as they likely will
    - Especially when security is perceived as a usability hindrance

Keep Security Simple (2)

Some good practices to follow:

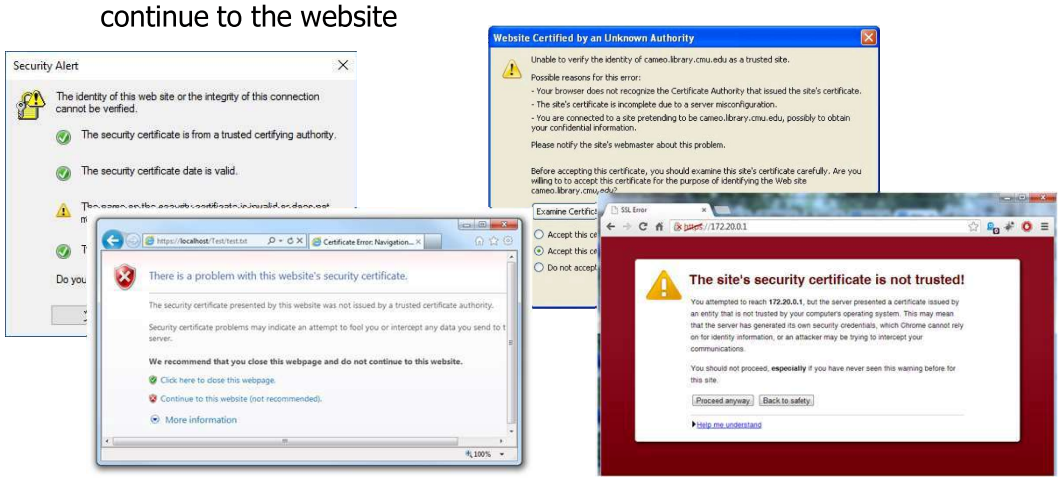
- Select a decent, as simple as possible software design
  - It's the basis also for sound security design
- Re-use proven software components or technologies when appropriate
  - This is especially important with security functions such as secure communication protocols and cryptographic algorithms
- Implement security-critical functions only once and place them in easily identifiable program components (e.g., in a separate security package)
  - This will make it easier to use security-critical functions in your software
  - For instance, use a single *checkPassword()* or *checkAccess()* method in your software and take care to review these methods for correctness
- Do not allow the users to turn off important security features
  - E.g., if your security analysis has shown you that a two-factor authentication is necessary to achieve the desired level of security, then don't give the users the choice to turn off the 2nd factor for convenience

Spreading Security Functions in the Code

One often sees that security code is spread throughout a program. This can mean that access control is implemented at different places in different ways, which naturally significantly increases the probability for security vulnerabilities.

Keep Security Simple (3)

- Don't give the users the option to temporarily circumvent security
  - As they most likely will – because they want to «get things done»
- The classic poor example is the certificate warning of browsers
  - Many studies confirm that approx. 50% of all users ignore them and continue to the website



Studies about Ignoring Certificate Warning

Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. Presented at 18th Usenix Symposium, August, 2009. Available at *http://lorrie.cranor.org/pubs/sslwarnings.pdf*

Adrienne Porter Felt, Alex Ainslie, Robert W. Reeder, Sunny Consolvo, Somas Thyagaraja, Alan Bettes, Helen Harris, and Jeff Grimes. Improving SSL Warnings: Comprehension and Adherence. April 2015. Available at *https://adrifelt.github.io/sslinterstitial-chi.pdf*

Keep Security Simple (4)

- For the average user, a certificate warning more looks like this:

The screenshot above was taken from *http://www.usenix.org/events/sec09/tech/slides/sunshine.pdf*

Avoid Security by Obscurity (1)

- Security by obscurity means that the security of the system mainly relies on the secrecy of the design or implementation of the system
  - I.e., the system is only «secure» as long as the attacker does not know or understand how it works internally
- Security by obscurity nearly always fails in practice if there's a determined attacker with access to the binary code or to the hardware implementation of the code (e.g., a chip)
- The reason is because of powerful reverse engineering methods (e.g., disassemblers, decompilers) that can be applied to binaries / hardware
  - It's very hard to hide the actual functionality of, e.g., a proprietary cryptographic algorithm
  - Hiding a secret such as a cryptographic key in a code binary is virtually impossible (e.g., search for randomness)
  - You can use code obfuscation to make the task for the adversary more difficult, but a determined adversary will most likely find out in detail how your software works

Code Obfuscation

The purpose of code obfuscation is making reverse engineering more difficult by making decompiled code more difficult to read. Code obfuscation can be achieved through one or more of the following methods:

- Source or binary structure obfuscation - A source code obfuscator accepts a program source file, and generates another functionally equivalent source file, which is much harder to understand or reverse-engineer. This is useful for technical protection of intellectual property when source code must be delivered for public execution purposes.
- Data Obfuscation - This is aimed at obscuring data and data structures. Techniques used in this method range from splitting variables, promoting scalars to objects, converting static data to procedure, change the encoding, changing the variable lifetime etc.
- Control Flow Obfuscation - This aims at changing the control hierarchy with logic preservation. Here false conditional statements and other misleading constructs are introduced to confuse decompilers, but the logic of the code remains intact.
- Preventive Obfuscation - Here the focus is on protection against decompilers and reverse engineering methods. Renaming metadata to gibberish or less obvious identifiers is one such technique, like defining function InterestCalculation() as x().

Source: http://palpapers.plynt.com/issues/2004Dec/code-obfuscation/

Code obfuscation may certainly be a good idea if your software provides, e.g., an ingenious algorithm (that you don't want to publish) to perform a non security-critical operation such as processing an image in an photo editor program, but it shouldn't be used to hide the functionality of security-critical operations from an adversary.

Avoid Security by Obscurity (2)

- Several examples undermine this
  - CSS (Content Scramble System) for the cryptographic «protection» of DVD video content (e.g., movies)
  - The failure of many software copy protection mechanisms
- Test for security by obscurity:
  - Ask yourself: If I were publishing the entire design and source code of my software, would it still be secure?
  - If the answer is «no», then you're most likely relying on security by obscurity (at least partly) – which means you should fix the software
- Final note: security by obscurity is not a totally bad thing, it's only bad if you use it as the only security measure
  - If your software is basically secure, then using security by obscurity «on top of it» is reasonable and may increase overall security
  - Example: Proprietary ciphers used by the military can be expected to be designed in a secure way, but their design and implementation is not publicly available (in contrast to, e.g., AES)

Software Copy Protection

The difficulty with software protection mechanism is that somewhere in the code, a “check” takes place whether, e.g., a valid license key is available or whether the software was activated online. With enough effort, it is usually possible to find that code section and adapt it (directly in the binary) so that the actual check is circumvented and the software can be used.

Don't Trust User Input and Services

- IT systems often get input from their users and use 3rd party services to perform operations
- In general, it's a good idea to not implicitly trust the received data and always consider it potentially harmful
  - Because the user may be an attacker and the service may be malicious or compromised
- Therefore, always validate the received data before processing it
  - Check the length and format of the data, e.g., if a numeric value between 0 and 1'000'000 is expected, make sure the data corresponds to this
- Example: A shopping center uses an external service for the loyalty program it offers its customers
  - To check their current bonus points, customers can log in at the web application of the shopping center; for this the external service is queried
  - If the web application does not validate the received bonus points value, this may be abused by the external service to inject JavaScript code into the web page presented to the customer (e.g., to get the session ID)

Secure Design Principles – Exercise (1)

Assume a web-based e-banking application, which should provide «good security». For the following statements, decide whether it's a good or bad choice and with which secure design principle(s) it is associated.

- To support Internet Explorer 6 users, the server not only supports TLS 1.0 – 1.3, but also SSL 3.0
- Users are allowed to disable the 2nd authentication factor (login code received by SMS) in their account settings
- A web application firewall (WAF) that can detect typical web application attacks (SQL injection, Cross-site scripting etc.) is used in front of the web application server

Secure Design Principles – Exercise (2)

- The web application server runs with root privileges, as this is required to write the logs

- To increase security, users get a hardened browser on a read-only USB stick, which must be used to access the e-banking application

- To further increase security, this browser uses a proprietary and secret encryption algorithm (developed by the bank) for TLS encryption

Secure Design Principles – Exercise (3)

- If a system administrator wants to get access to the e-banking computing center, he needs a written approval signed by his superior; this approval must then be presented to the guard at the entrance

- When an e-banking account is created, the default setting is that no transaction confirmation (code received by SMS) is used when the user does a payment; but the user can enable this in the account settings

- The e-banking web application runs on Ubuntu Linux where all software that is not absolutely needed has been removed

Summary

- Secure Design Principles are very fundamental security guidelines that provide a good general basis for securing IT systems
  - Secure the Weakest Link
  - Defense in Depth
  - Fail Securely
  - Principle of Least Privilege
  - Separation of Privileges
  - Secure by Default
  - Minimize Attack Surface
  - Keep Security Simple
  - Avoid Security by Obscurity
  - Don't Trust User Input and Services
- They have demonstrated their effectiveness in practice, are relevant and applicable in virtually every IT system, and will most probably also be valid in the future
- Keeping these principles in mind during software development should help you to avoid many security problems