Lecture Notes: Security of IT-Systems

Jonas Otto

February 21, 2021

Todo list

Contents

Chapter 1

Fundamentals

1.1 Motivation and Introduction

The number of catalogued vulnerabilities is rising rapidly over time, ever since IT existed. Security is all over the headlines both in dedicated news agencies and in mainstream media. Ransomware attacks are popular recently and have highly visible and critical targets such as hospitals and large companies. Running a networked computer system with 100% security is neither feasible nor possible. But strong security is still achievable, and defending most attackers is possible.

1.2 Terminology

When talking about "secure" systems, what is usually desired is a "dependability", meaning a system that shows no unexpected or unacceptable behavior. A dependable system should fulfill multiple goals, including

- Availability
- Reliability
- Safety
- Integrity
- Maintainability
- (Confidentiality)

The three major security goals are Confidentiality, Integrity and Availability (**CIA**). The main difference between dependability and security is that the security is usually assessed from the viewpoint of a potential attack, while dependability is considered in a more general context. Security can be seen as a

precondition of having a dependable system. A secure system is a system that can achieve the three mentioned goals even facing an attacker.

Confidentiality Protection of information against unauthorized access

Integrity Protection against unauthorized change and destruction

Availability Protection against rendering IT resources inaccessible

A **threat** is defined by a potential error in the system, which could enable an attacker to violate those objectives. A **vulnerability** is a concrete fault in the system that threaten one or more security objective. A threat would be for example the possibility of a DDOS attack towards a web service, a lack of resources to cope with the attack is a vulnerability. If an attacker exploits the vulnerability, this is called an **attack**.

The concepts of safety and security shall be differentiated.

When analyzing a system with regards to it security, two factors are considered: The first is **threat potential**, which estimates the likelihood of each potential attack against the system. The second is the **damage potential**, which asks what the impact to the system would be if an attack succeeds. Likely high-impact attack scenarios can be mitigated by either reducing the impact of a successful impact or by taking security measures reducing the likelihood of a successful attack.

A few more useful definitions:

Identification Assignment of an identifier

Authentication Verification of an identity

Authorization Assignment of permissions

Access Control Protection of resources against unauthorized access

Privacy Protecting personal information

1.3 Attacks and Defenses

1.3.1 Attacks

Attack can be categorized by many measures, like intention, approach or point of attack.

Categories by intention can be:

Denial of Service Making an IT system unabailable to users

Information Theft Access to confidential information by unauthorized persons

Intrusion Bypassing access control to gain access to a system

Tampering Modification of stored or transmitted data

By approach:

- Masquerading
- Eavesdropping
- Authorization Violation
- Loss/Modification
- Denial/Repudiation
- Forgery
- Sabotage

By point of attack:

- Network
- Network services
- Operating system/Applications
- User

Another possible method of categorization is "STRIDE" categorization, which stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service and Elevation of privilege.

An attack often follows similar patterns. The first step is collecting information of the system. This may expose a possible attack vector to the attacker, which can then be used/tested. This is often repeated, as the first attack is not necessarily successful. After a successful intrusion the next step is often privilege escalation. This may allow the attacker to cover their tracks, install back doors, etc. If the main goal of the attack is not reached yet, this point may be used as the starting point for the next attack, towards the initial goal.

Understanding and talking about attacks is vital when dealing with security, as this is the very point we are trying to defend against.

1.3.2 Security Mechanisms and Policies

A Policy is a statement of what is, and what is not allowed. This allows the distinction into "authorized" and "unauthorized" that we made already.

A Security mechanism is a method, tool or procedure that attempts to enforce such policies. We can categorize those measures into prevention, detection and recovery. It is possible to employ multiple measures, for example a gate as a way of prevention, and a camera pointed at the gate which provides detection of a successful attack.

Security through obscurity or prohibiting reverse engineering and attacking is not an alternative to real security.

Security mechanisms can never be judged by themselves. During risk assessment, it is always necessary to evaluate the measures in the context in which they are employed. The security of a system is determined by the weakest link in the chain.

Chapter 2

Cryptography

2.1 Cryptographic Hash Functions and Random Numbers

2.1.1 Hash Functions

A hash function is defined as a function $h : D \mapsto S$ with $|D| > |S|$. A hash function is expected to fulfill more desired properties:

- Compression: $|D| \gg |S|$
- Chaotic: Maximal change of the hash with minimal change in input
- Surjective: |S| is fully used
- Efficient calculation

Even with those properties, a hash function is not considered a cryptographic hash function. Even a CRC fulfills those properties. The additional properties of a **cryptographic** hash functions are:

One way function: Also called **first pre-image resistance**, this implies that given a hash, an input producing that hash can not be computed efficiently.

Weak collision resistance: Also called **second pre-image resistance**, implies that given a hash $h = \text{hash}(m)$, an input $m' \neq m$ with $\text{hash}(m') = h$ can not be efficiently found.

(Strong) collision resistance: No m and $m' \neq m$ can be found efficiently with $\text{hash}(m) \neq \text{hash}(m')$. In contrast to weak collision resistance, a specific hash value is not required here.

Hash functions can be used in combination with a key in the form of **Message Integrity Code** and **Message Authentication Code** to provide an equivalent to digital signatures using symmetric cryptography. The hash based MAC **HMAC** is calculated as a hash over a combination of the message with a key.

2.1.2 Random Number Generators

Pseudo-Random Number Generators **PRNG**s which produce a deterministic sequence of numbers from an initial seed are not suitable for cryptographic applications such as key generation due to their predictability. Non-Deterministic RNGs exist and often rely on external physical processes like noise in electronic circuits. They do however often have a low data rate which is not sufficient for all applications. A combination of a PRNG which is (periodically) seeded by a true, non-deterministic RNG is an approach used in practice.

Criteria for good RNGs are:

- Statistical distribution as desired
- Independence: No repeating sequences, invariant to environmental conditions
- Speed of generation
- Long periodicity (PRNGs) / Non-reproducibility

2.2 Encryption

One important distinction between encryption schemes is the concept of symmetric and asymmetric ciphers. Symmetric encryption is also called secret key cryptography and relies on the presence of a secret key at both endpoints of the communication. Asymmetric or public key encryption separates the key into a public and private key for both parties. Information about the secret key is never shared, and it can be used to decrypt messages which are encrypted using the corresponding public key.

The algorithm itself is never considered secret, the security shall only depend on the secret keys (Kerckhoffs principle).

Additional actors in a encryption scheme that are to be considered are the passive (eavesdropping) attacker (*Eve*), the active (man-in-the-middle) attacker (*Mallory*) and a trusted third party (*Trent*).

2.2.1 Symmetric Encryption

Symmetrical encryption approaches usually follow the pattern of using basic bitwise operations to form building blocks such as Feistel networks which are then combined or repeated in multiple rounds. A challenge arises when the input

of the cipher is of variable length. Stream ciphers are suited to such a problem, but the more common approach is to divide the input into blocks and use a block cipher, operating with a fixed input size, multiple time. Simply applying the same cipher with the same key to each block (*electronic code book*) however is not a good approach since repeated blocks in the input will be directly reflected in the ciphertext. Alternatives include integrating the previous ciphertext into the next block or including a counter in each block, to avoid repeating inputs to the cipher altogether.

A standard block cipher used today is the *Advanced Encryption Standard* **AES**, a block cipher with variants for different key lengths such as 128, 192 or 256 bit.

2.2.2 Asymmetric Encryption

Asymmetric encryption makes encryption possible even without a previously agreed on private key.

2.2.3 Diffie-Hellman Key Exchange

The goal of the DIffie-Hellman key exchange algorithm is to establish a private key between two users Alice and Bob, without compromising the secret key to an attacker that has full visibility of the entire communication between Alice and Bob.

Algorithm 1: Diffie-Hellman Key Exchange

Result: private key k
choose public modulus p (prime);
choose public base g (primitive root modulo p);
begin Alice
  choose secret a;
  $A = g^a \mod p$;
  publish A;
end
begin Bob
  choose secret b;
  $A = g^b \mod p$;
  publish B;
  $k = A^b \mod p$;
end
begin Alice
  $k = B^a \mod p$;
end

This results in the same private key k for both Alice and Bob. Generating this

private key requires knowledge of one of the secrets a or b, which are only known to the corresponding party.

Diffie-Hellman itself is not secure against **Man-in-the-Middle Attacks**. If an attacker *Mallory* intercepts the communication, they could perform a separate key exchange with both Alice and Bob. They would both have a shared key with Mallory, unaware that the real communication partner is not Alice or Bob. Mallory can then decrypt incoming messages, read or modify them, and re-encrypt them using the second key.

2.2.4 RSA

RSA is an encryption scheme which allows encrypted communication without first establishing a symmetric secret key (using Diffie-Hellman for example). Each participant calculates both a public key and a secret key. The public key can be used by any participant to encrypt messages, which can only be decoded using the corresponding secret key, which is only known to the owner of the key.

2.2.5 Digital Signatures

If the integrity of a document and identity of the author are of concern, but the contents are not necessarily encrypted, digital signatures are used. In digital signatures, the signature is generated for a specific message (usually a hash of the document) with the private key of the author. The verification of the signature is possible using the corresponding public key and again the message. This is similar to the reverse of the public key encryption scheme.

2.2.6 Strength of Cryptographic Approaches

The security of cryptographic algorithms can be categorized in the following categories:

- **Empirically secure:** The approach is secure because no attacks against it have been successful, and analysis has not found a specific weakness
- **(Formally) proven secure:** The encryption is proven to be a mathematically hard problem
- **Unconditionally secure:** "An attacker cannot extract any information from the encrypted text"

The only unconditionally secure approach is the **one-time pad**, where the key and message have the same length and a unique, random key is used for every transmission. Since the key has no inherent correlation, the probability of recovering any arbitrary plaintext is identical, which makes even brute force attacks impossible.

Chapter 3

Identification and Authentication

3.1 Identification

The identity of an entity shall have the following properties:

- Uniqueness
- Unchangable Linking
- Lifelong validity
- No Transferability

In order to identify an entity, an **identifier** has to be defined. The identifier should meet the above criteria and should be able to determine an identity *within a given context.*

Identifiers have the purpose of both accountability and access control. They can be applied to both subjects (users, processes, ...) and objects (files, URLs, ...), humans and machines, and can be temporary or persistent.

For authentication, a separate *proof of identity* is usually required:

3.2 Authentication

Authentication is the process of confirming whether a second party is indeed who they claim to be, to a specified level of confidence. There are three basic forms of authentication:

- **Something you know** (passwords)

- **Something you have** (smart cards)
- **Something you are** (biometrics)

Combinations of those increase the security (**Multi-Factor-Authentication**).

Password Authentication is based on the *something you know* factor. Examples include unix passwords, PINs or secret code words. They can also easily be used to authenticate groups, by distributing the password to every entity in the group. A weakness of passwords is that an attacker can learn and reuse it. A possible solution are *one-time passwords*.

One-time Passwords are only used once, an example would be a TAN list for online banking. They can also be part of a challenge-response-protocol, where the two parties agree on a secret function beforehand, and authentication happens by verifying the function response to a challenge.

Hardware Tokens take a similar approach in generating some kind of one-time use token, but those are generated by dedicated hardware, shifting the factor to *something you have*. They might have an additional input such as a pin, or, as in the case of popular 2FA sulutions, the current time. The **HOTP** (HMAC-based One-Time Password algorithm) generates short time passwords using a counter (time) and a pre-shared secret key.

Biometric Authentication has to be differentiated into *verification* and *recognition*. In verification, the used specifies its identity, and the system authenticates the used if biometric verification succeeds. In recognition, the system recognizes the user amongst multiple known users without further input.

Biometric authentication systems can fail in two ways: **False negative** means that a user is incorrectly rejected, a **false positive** means that a user is wrongly accepted. The threshold on accepting a authentication attempt has to be chosen in a application specific way, depending on which fault is more acceptable. Figure 3.1 shows the relationship between the *False Acceptance Rate* and *False Rejection Rate* with a varying threshold. A measure of the security of the authentication system could be the *Equal Error Rate*.

3.3 Password Security

Passwords which are short or badly chosen can easily be cracked. Brute-force or dictionary attacks guess the password eighter randomly of from a list of known (pass-)words. Brute-force attacks are easily feasible for passwords up to ~8 characters in length, useful rules on possible guesses and dictionary attacks can lead to success for even longer passwords. An advantage for the attacker is when the attack can be executed *offline*, such as by stealing the file containing the

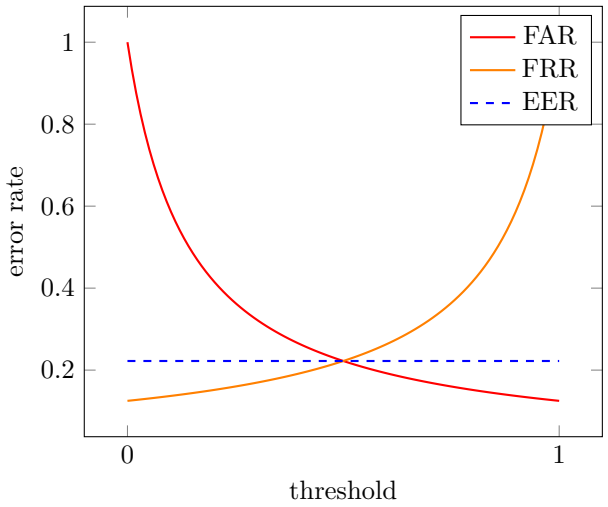

Figure 3.1: *False Acceptance Rate* and *False Rejection Rate* for biometric authentication

password hashes. This removes the bottleneck of the authentication mechanism of the target and allows for distributed attacks.

A common protection measure is to use a **SALT**. A salt is a random value that gets appended to the password before hashing, and then gets stored alongside the password hash. While this does not protect a single password against the mentioned attacks, it prevents reuse of a hash that has already been calculated. Otherwise, it would be possible to just compare the hashes to known hashes of popular passwords.

Another consideration is access to the password hashes. While the actual cryptographic security is only influenced by the hash function, preventing offline attacks by properly protecting the hashes forces the attacker to execute much slower online attacks. Those online attacks can be slowed even further by limiting the number of invalid authentication attempts or introducing an increasing delay after failed authentication attempts (*back off*), and by using "slow" hash functions. The previously popular measure of password aging (requiring passwords to be changed after a certain amount of time) is discouraged, since it promotes the use of weak but easy to remember passwords.

Other attacks focus on the specific implementation of the authentication mechanism and exploit vulnerabilities that allow login even without actually obtaining the correct password, or allow changing or resetting passwords even with insufficient privileges.

3.3.1 Time Memory Trade-off

In attacks on passwords a trade-off between time and memory has to be made, the two extremes being the brute-force attack and the fully pre-calculated dictionary/codebook attack.

One possible solution is a **Variable Length Lookup Table**. Those rely on hash chains: For many initial values, a chain of hashes (length $n_{\max}$) each is calculated. Only the initial and end-value are for each chain is then stored. When a certain hash shall then be cracked, it will be hashed $n_{\max}$ times until a chain is found which end-state matches the calculated hash. Once the chain is found, it can be restored using the stored initial state which results in a chain containing the to-be-cracked hash and the password as the state immediately preceding that value.

An improvement to lookup time can be made by making the chains variable length, and introducing an end criterion, for example a certain number of zero-bits at the end of the hash (**Distinguished Codepoints**). This reduces the number of end-lookups significantly.

Duplication arising from hash collisions are addressed by **Rainbow Tables**: A round-specific reduction function (hash space →password space) is introduced. So even if a state is already present in a different chain (but at another round in the chain), the original chain continues separately.

3.4 Network Authentication

Authentication: Network Authentication

Chapter 4

Access Control

Access control combines authentication and authorization. Chapter 3 showed how the identity of a subject can be confirmed, the question is now whether the subject is **authorized** to access a specific resource. This decision is generally performed by a **reference monitor**, on the basis of some **access policy**, which may be set by an administrator or the owner of the object.To differentiate the terms *subject* and *object*:

A **subject** is the *acting entity*, which intends to carry out an operation requiring access. Examples include persons, processes or network nodes. A subject can also be the object of an access operation at another time.

An **object** is the unit that is *being accessed*. Examples include files and directories, database records, computers, processes, mailboxes or applications.

The **access operation** is the type of operation of a subject in relation to an object. Different sets of access operations can be defined. A file system might for example define *read* and *write* operations (or *observe* and *alter*). There might be additional operation like *execute* or *append*, which can be handled independently.

4.1 Access Control Matrix

The access policy can be described using an *access control matrix*. The matrix contains the set of every allowed operation for every possible pair of subject and object. This however immediately presents a problem, as this approach scales badly. For N subjects (users) and M objects (files), a $N \times M$ matrix would have to be stored.

Two implementations of the access control matrix may be more appropriate depending on the application:

4.1.1 Access Control List

In an Access Control List (*ACL*), a list of subject-operation pairs is stored with each object. A file might for example contain a list of all users that have access to that file, and which exact operations they are allowed to execute on the file. If a user is not allowed to interact with the file in any way, the entry may be omitted.

4.1.2 Capabilities

In direct contrast to the ACL, in a capabilities based system the subjects contain a list of objects and the corresponding operations which the subject is authorized to execute on the object. In the file system example, each user would contain a list of files which the user is allowed to access.

4.2 Models for Access Control

More detailed access control models are in use:

DAC **Discretionary Access Control** is the approach as explained above. Access control decisions are based on a subject and object. Subjects define the access rights of an object (file owner for example).

MAC **Mandatory Access Control** is oriented towards clearance levels, as one might expect in a government or military setting. The access rights are usually defined system-wide, and are not changed by a subject. More information on this in section 4.4.

RBAC In **Role Based Access Control** access to objects is not defined per subject, but per *role*. Roles are given to subjects, and may change.

ABAC **Attribute Based Access Control** is even more abstract, where both subjects and objects have attributes, and access control decisions are made by flexible comparison of attributes.

Real world systems often implement aspects of multiple of those models, and no model is the best or definitive answer to access control.

4.3 Example: Linux

Linux access control, ACLs

4.4 Multilevel Security, Bell-Lapadula-Model

In multilevel security, both objects and subjects are classified. Classes may for example be *Unclassified*, *Confidential*, *Secret* and *Top Secret*. Access is only

granted if the subject has at least the same classification as the object. Access is denied if the subject has a lower classification than the object. The **Bell-Lapadula-Model** is an implementation of this access control model:

To define this model, consider a set of subjects S, a set of objects O and the set of operations $A = \{\text{execute}, \text{read}, \text{append}, \text{write}\}$. For two security levels $a, b \in SL$, there always exists a greatest lower bound $l \in SL : (l \leq a, l \leq b$ and $l$minimal) and a least upper bound h.

The system is in a state at all times. A state is a triple (b, M, f) with

- $b \subseteq S \times O \times A$ the set of current accesses
- $M = (M_{so})_{s \in S, o \in O}$ the current access matrix
- $f = (f_s, f_c, f_o)$ with
  - $f_s : S \mapsto SL$ the maximum clearance of each subject
  - $f_c : S \mapsto SL$ the current clearance of each subject (this requires $f_c(s) \leq f_s(s)$ for each subject s)
  - $f_o : O \mapsto SL$ the security classification of each object

The current clearance f_c is chosen by each subject, within the limit set by the maximum clearance f_s.

A state is secure, if the following properties are met:

- **Simple Security Property**: For all $(s, o, a) \in b$ with $a =$ read or $a =$ write: $f_o(o) \leq f_s(s)$ (Each subject which is currently reading or writing has the necessary maximum clearance)
- ***-Property**:
  - For all $(s, o, a) \in b$ with $a =$ append: $f_c(s) \leq f_o(o)$
    (No append operation is executed with higher clearance than the object)
  - For all $(s, o, a) \in b$ with $a =$ write: $f_c(s) = f_o(o)$
    (Each write operation is executed with minimal clearance)
  - For all $(s, o, a) \in b$ with $a =$ read: $f_c(s) \geq f_o(o)$
    (Each read operation is executed with a sufficient current clearance)

  This property enforces the correct selection of the current clearance f_c. Appending happens "upwards", writing on the same level, and reading "downwards".
- **Discretionary Security Property**: For all $(s, o, a) \in b$: $A \in M_{s,o}$

4.5 POSIX Capabilities

POSIX capabilities are an implementation of a capability based access control scheme. An example would be the right to open raw sockets on linux, which is usually reserved to the root user. This is however necessary for the `ping` utility, which should be accessible to every user, even without setting the `setuid` bit on the binary. The POSIX capability system now allows giving the specific right to open raw sockets to the specific binary. This also adheres to the principle of least privilege a lot better than always executing `ping` with full root rights. Capabilities are also not reserved to files, but can be given to individual processes as well.

Chapter 5

Malware

5.1 Buffer Overflow Attacks

Buffer overflows are one of the main security vulnerabilities found today. Modern mitigations make them not as trivial in practice as in theory. The most common form is a **stack overflow**. The goal is to overwrite the return address of the current stack frame, usually by exploiting unchecked array-access which allows writing to the address where the return address is located. The attacker could instruct the program to return to a different part of the program, for example skipping authentication checks. If write access to an executable page is also present, the attacker could also first inject code and then jump to that code, executing arbitrary instructions. If the place of the injected shellcode is not known exactly, one technique is to insert `NOP` instructions before the actual exploit (*NOP sled*).

Mitigations include canaries, which are values placed between the variables and return address on the stack to indicate buffer overflows. Injection of shellcode is usually protected against by not allowing memory pages with both write and execute permissions. Type-safe languages also offer features making access to memory outside of variables impossible. Address space layout randomization makes guessing memory locations more difficult.

5.2 Introduction to Malware

Malware is a generic term for software, which is designed to perform a function undesirable or harmful to the user. Categorization of malware is possible by the approach of spreading:

- A **virus** is a program which spreads by abusing other (harmless) programs
- A **worm** spreads autonomously over a network

- A **trojan** disguises itself as a harmless program

Especially malware that spreads over the network can be further categorized, the main distinction being the amount of interaction a user has to perform in order to be infected, which can lead from downloading and opening an attachment to only visiting a website or even zero-interaction vulnerabilities.

Initial infection may also happen completely offline, for example via the "lost USB stick" tactic in a more targeted attack.

The malicious payload can assume many forms and achieve a multitude of goals. Some examples of payload functionality include:

- Deleting data
- Spying, exfiltrating data
- Enabling remote access
- Denial Of Service (*DOS*)
- Physical damage
- Encryption of data (and demanding ransom)
- Abusing resources (crypto mining)

More often than not, those are motivated by financial gains, which is apparent with the recent waves of large scale ransomware attacks.

Malware may also be categorized into mass infection and targeted attacks (Advanced Persistent Threat *APT*). While the former is focused entirely on maximizing the number of infected hosts, the latter is focused on a specific target.

5.3 Botnets and Targeted Attacks

Botnets are established with no particular payload. The initial infection happens with a *dropper*, which connects to a command and control server. The actual payload is then downloaded from this server, which allows the botnet to be rented out to perform a number of different attacks that all benefit from a large amount of infected machines.

Chapter 6

OS Security

Operating system- and **host-security** has the goals of protecting stored data, running processes and the operating system itself. This necessitates some form of authentication, to distinguish between users on a multi-user system, as well as an access control system which assigns permissions to users. A useful concept applied here is isolation, which is applied to users and processes. Raw hardware access is usually restricted, and only allowed to privileged code within the operating system. Protection against external access may also be part of host security.

6.1 Concepts and Reference Monitors

A **reference monitor** is an abstract machine which mediates all accesses to objects by subjects (see chapter 4). Reference monitors can be implement on any level of the system. Reference monitors in the systems hardware include the MMU and privileged execution modes. Kernel level reference monitors are for example implemented in the file system or capability system. Applications can also contain reference monitors, which may be the case in web server applications, or run completely inside another program which implements a reference monitor such as the Java virtual machine or a database engine.

The **security kernel** is the hardware, firmware and software of a trusted computing base which implements the reference monitor. It must mediate all accesses, be protected from modification and be verifiable as correct.

The **trusted computing base (TCB)** is the totality of protection mechanisms within a computer system. This includes hardware, firmware and software which is responsible for enforcing a security policy. The enforcement of the security policy must only depend on the TCB, the rest of the operating system need not to be trusted.

6.2 Virtualization and Mandatory Access Control

6.2.1 Virtualization

There exist many reasons to use virtualization, but the goal of virtualization from a security perspective is full isolation of systems and applications. The possibility to roll back an entire system to a known-good state in case of compromise also presents an advantage.

Levels of virtualization are distinguished based on the role and position of the **virtual machine monitor VMM**. In **native virtualization**, the VMM directly interfaces with the hardware. In **user mode virtualization**, the VMM only interfaces with the host OS, and not directly with the hardware. A hybrid approach is **dual-mode virtualization**, where a host OS exists, but some form of direct access to the hardware by the VMM is possible.

The interaction of the VMM with the **guest OS** provides another way of categorization: In **full virtualization**, the guest OS runs unmodified, as on real hardware. This is usually assisted by various hardware extensions such as *AMD-V* and *Intel VT-x*, special support by the MMU and passthrough of system busses such as PCI.

Paravirtualization refers to a mode of virtualization in which the guest OS is aware of the virtualization and has some adaptations to the host OS. Hardware drivers for example can be replaced with components that directly interface with the VMM.

6.2.2 Isolation

Isolation is the main benefit of virtualization from a security perspective. Errors in applications can be contained effectively, programs with different security requirements can be separated, and malware analysis is possible without effecting the host system. The host system can employ detailed monitoring, and perform effective intrusion detection from the outside. The isolation of common dependencies between applications prevents the compromise of multiple applications by compromise of the common dependency.

As an example, a payment processor may be located in a dedicated VM with a secure operating system. The non-critical application such as a frontend which exposes a large attack surface runs in a separate VM and interfaces with the payment application only through a closely monitored interface.

6.2.3 Security Enhanced Linux

Linux provides the **Linux Security Module (LSM)** interface. Once a user triggers a security sensitive activity in the kernel through a syscall (for example: `read`), the LSM hook is executed, which the LSM module registers. The LSM

module can then for example deny the filesystem access. **SELinux** is such a module. SELinux supports a variety of mandatory access control schemes, the main one discussed here is **type enforcement**. The idea is that both subjects and objects are assigned security labels. A central security policy determines which subjects can execute which operations on the objects based on the labels. The acting entity is always a process. Processes are assigned to **domains**, for which access rules to objects are specified. SELinux supports two modes of handling processes without a specified domain: assigning all of those to a default `unconfined` domain or creating an individual domain for each process.

Files are tagged with security **labels**. The label my include a user identity, role, type or domain. The last field implements the type enforcement: for regular files, this expresses the type of the object, but for executables this determines the domain of the process once it is executed.

Beyond the simple type enforcement, **roles** can be defined. Users can have roles, and users can change the current role if they have sufficient privilege. Specific rights can then be given to specific roles.

Multi-category and multi-level security is also implemented in SELinux, implementing the Bell-Lapadula model (see section 4.4).

6.2.4 AppArmor

AppArmor: describe example from video 6b

6.3 Use-Case: iOS Security

iOS Security

Chapter 7

Embedded and Hardware Security

Security in embedded systems is of special concern. Embedded systems are often harder to patch, and often have direct real-world impact, sometimes in a safety critical way. Non-mainstream operating systems are in use, and the CPU might not offer all desired security features (see below). This results in a large attack surface on a system that is not as well understood security-wise as desktop- or server applications.

In the following, security will be considered from a lower level than before, down to the hardware. Many of the security measures considered before can be circumvented if lower-level access is present. Access control on a file system for example is worthless if the attacker can connect the disk to another system and dump all the contents. Software security is difficult if it relies on libraries that can be exchanged by an attacker.

Placing security mechanisms at the lowest possible layer circumvents those attacks.

7.1 Introduction and x86 Privilege Levels

At any point while an x86 processor is executing instructions, it is in some **privilege level** or **protection ring**, usually indicated by a number where higher numbers represent less privileged execution:

(-1) Virtual Machine Monitor, Hypervisor

0 Operating system kernel

1 Rest of the operating system

3 I/O Drivers etc.

4 Application software

Switching to a lower privilege level must be protected, while switching to a higher level (less privileges) is usually easy. One important feature made possible by this is protection of memory segments. Each memory segment contains a Descriptor Privilege Level *DPL*, and each process is assigned a privilege level. If the Current Privilege Level $CPL > DPL$, the CPU generates a protection fault and prevents access to memory accessible only to lower privilege levels. This for example prohibits applications from modifying operating system data structures.

Upgrading of the privilege level (setting it to a lower value) may happen for example when a syscall is initiated and execution is passed to the operating system. This provides a well defined "gate" to those privilege levels.

In order to prevent an application to misuse the privilege level of the syscall, by for example instructing it to copy data into the process that the application would otherwise not have access to, an additional *Requested Privilege Level* may be introduced (**confused deputy problem**).

7.2 Isolation and HW-based Attacks

The privilege levels introduced above provide a means of isolation between the operating and user programs, often called *userspace* and *kernelspace*. Isolation is often advantageous to security as it provides a barrier with defined interfaces between components. A ubiquitous form of isolation is that between multiple processes on a single computer. Address space, access rights and file descriptors are completely separate for multiple processes. Special security-critical functions such as handling of private keys may even be delegated to a dedicated hardware security module.

Attacks which circumvent those measures and leak data through channels other than the predefined interfaces are called **side-channel attacks**: Such side-channels may be for example power consumption or electromagnetic emission of a device. Countless such side-channels have been found, with varying degree of real-world usage. Most mitigations against such attacks have been broken by even cleverer side-channel attacks. Recent examples of side-channel attacks exploit interference between adjacent DRAM cells in main memory or cachelines that don't get evicted after branch prediction has failed.

7.3 HW-based Security Mechanisms

7.3.1 Trusted Platform Module

IDK, seems like something the copyright-industry would come up with...

7.3.2 Physical Unclonable Functions

See Rührmair et. al.

Chapter 8

Software Security

Chapter 9

Network Security

Chapter 10

Web Security

All components of a web application, the browser, the web server and the underlying infrastructure are all subject to various attacks:

The **connection** is vulnerable to misuse of TCP/IP mechanisms, such as *SYN flooding*. Eavesdropping and man-in-the-middle attacks are of concern to any network connection, and as such also for web applications.

Web servers or **web applications** are often the target of an attack. Various techniques such as XSS, injections and authentication bypasses are explained later in this chapter.

Attacks against the **browser** exploit flaws in the renderer, engine, through scripting or plugins like java and flash.

10.1 Transport Layer Security

TLS presents a solution to secure TCP/IP connections. Its goals are to provide authentication, protection, and confidentiality: Servers are authenticated using X.509 certificates, clients can optionally authenticate the same way. The transferred data is encrypted. TLS is implemented on top of TCP/IP and provides an API very similar to normal TCP sockets.

10.1.1 Handshake Protocol

The TLS handshake protocol establishes a TLS session, including authentication, negotiation of cryptographic primitives and negotiation of a symmetric session key. One TLS session allows for multiple parallel TLS connections.

Figure 10.1 shows an overview of the TLS handshake: In **phase 1**, the client offers a list of supported cipher suites to the server. The server selects a cipher

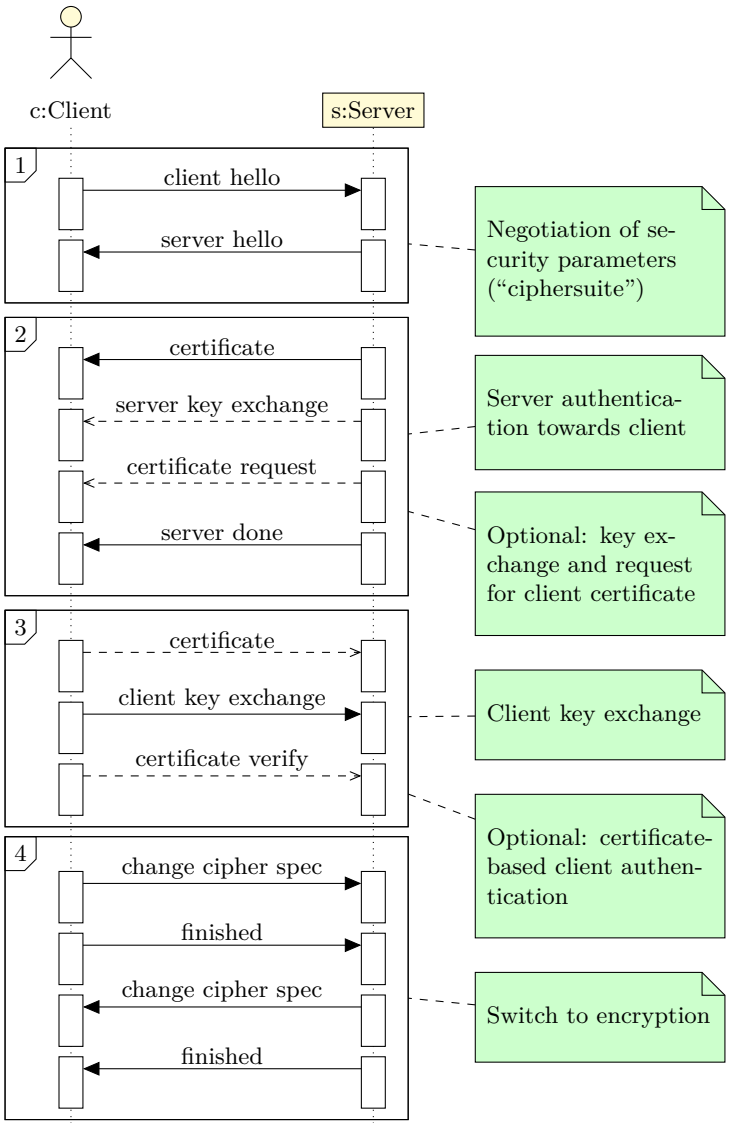

Figure 10.1: TLS Handshake Protocol

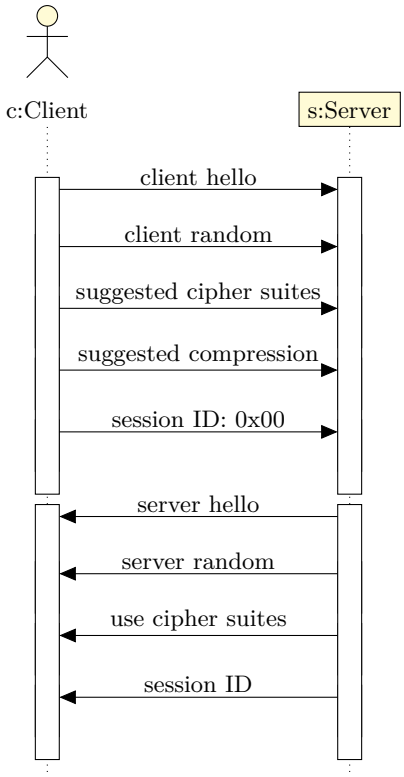

Figure 10.2: TLS Handshake Protocol: Hello messages (phase 1)

suite, and establishes a session ID. Random values for later key creation are also exchanged. Figure 10.2 shows phase 1 in detail.

In **phase 2**, the server sends its certificate, and optionally requests a client certificate for authentication of the client.

In **phase 3**, the client sends its certificate, if requested. The client has verified the server using the certificate, and can now use the severs public key for establishing a symmetric session key. The client generates a `PreMasterSecret`, from which client and server can derive the symmetric key using the random values exchanged in phase 1. The `PreMasterSecret` is RSA-encrypted using the servers public key and then sent to the server.

In **phase 4**, the symmetric session key is established. The integrity of the handshake is verified by exchanging hashes over the derived key and the previous messages. All following communication happens encrypted.

10.1.2 Record Protocol

Once the handshake is complete, the handshake protocol takes over and handles transfer of data. The record protocol is completely separate to the handshake protocol, making it theoretically possible to execute the handshake completely offline. The record protocol handles, in order: Fragmentation, Compression, Message Authentication Code, Encryption. Receiving works the same way in the opposite order. MAC and encryption use separate keys for both directions.

10.2 Injection Attacks

In injection attacks, some specially crafted input to the web application leads to unwanted effects.

10.2.1 Cross Site Scripting

Many web applications accept some kind of textual input, which then gets shown to a different user on the website, such as a forum- or social media post. If no special care is taken, this means that anyone can place arbitrary HTML, including scripts, on the website in the targets browser. This is referred to as *Cross Site Scripting* or **XSS**. A typical goal of such an attack is to steal cookies, which might allow the attacker to take over the login session of the target. Exfiltration of the cookie might happen via the query parameters of an image that gets loaded from the attackers server via the injected script. Protection against those kinds of attack include validating the input, and removing possibly dangerous characters, or making sure that user generated content is only interpreted as text by the browser, and not executed. Other measures include instructing the targets browser to not make any request to external servers, which hinders exfiltration.

10.2.2 SQL Injection

User inputs such as login credentials are often used as parts of SQL queries. If the input contains valid SQL syntax, and is just appended to a query, it might change the query in a way such that it always returns a certain value to bypass authentication, or execute arbitrary statements with the privilege of the web server application. Even escaping the input data might not solve the issue, as second order injections might circumvent that. Again, it is important to ensure input data is only handled as text and no situation exists where it is potentially executed as code.

Chapter 11

Data Protection and Privacy

11.1 Privacy Motivation

privacy motivation

11.2 Privacy by Design and PETs

11.2.1 Anonymous Communication: TOR

TOR implements a form of onion routing and a mix-network: A message for a server is encrypted with its public key. It is however not directly sent to this server, but through another server, which requires an additional layer of encryption for the intermediate server. (In practice, this asymmetric encryption may be replaced by a key exchange and symmetric encryption for the actual data, for performance reasons). This is repeated such that there are three intermediate *onion routers* between the source and destination. This way, none of the routers sees both the source and destination: The first router knows about the source, the last router knows about the destination, the middle router does not know the source or destination.

Attacks are possible by correlation of packets inside the network, incorrect usage of DNS or identification by the message payload.

11.2.2 Blind Signatures

A blind signature is needed when a document shall be signed without revealing it to the signing party. This works by first "blinding" the message, signing the

blinded message, then unblinding the signed, blinded message. This results in a signed form of the original message.

These kind of blind signatures are useful for electronic cash, which should provide anonymity, verifiable authenticity and protection against double spending. An analogy works as following: The user who wants to spend money puts an empty piece of paper with carbon paper in a sealed envelope. The bank signs that envelope on the outside, which makes the envelope with the paper inside worth 1$. While signing the envelope, the signature was printed on the paper inside the envelope. This signed paper is now used as payment. The bank has however not seen the actual paper inside the envelope. Double spending is prevented by containing a serial number on the paper, which the merchant verifies with the bank before accepting payment.

11.2.3 Group Signatures

A group of users has a shared public key, but each user has an individual private key. A message should now be signed with a private key, and verification should be possible via the group public key. Group anonymity is provided since it is only confirmed during validation that a member of the group produced the signature, but not which exact member.

A problem is that the group manager, who issues keys to the members, can spoof any participants identity and acts as a kind of trusted third party. The group manager might also be able to identify the individual member from a signature.

11.2.4 Attribute-Based Credentials

Using **Attribute Authentication**, the user can provide proof of an attribute without revealing their own identity. Multiple authentications should not be linkable. Examples could include public transport tickets, where the owner should only be required to prove authorization to travel on the current route, without revealing identity or entire travel plans. ABCs can also implement pseudonymous or not-at-all-anonymous authentication.

In an ABC scenario, the ABC authority first authorizes an issuer, and issues a device (smartcard) to the user. The issuer can then issue credentials containing attributes stored on the card. The card can then generate a selective proof revealing some attribute to a relying party.