Rovesti Gabriel

[nome della società]

Advanced topics in computer network and security simple (for real)

Summary

[Course introduction 2](#_Toc147906824)

[How papers are made and carry out research 4](#_Toc147906825)

[Containers and Kubernetes Security 5](#_Toc147906826)

# Course introduction

(Host: Mauro Conti)

The course is structured this way:

1. Advanced Topics

We will cover, through lectures and talks from invited speakers, recent and relevant security issues in traditional and novel technologies, such as:

* IoT and Cyber-Physical System
* (Adversarial) Machine Learning for Security
* Blockchain
* Advanced Cryptography Applications
* User Authentication

1. Students Presentations

* Students present to the class a given topic
  + Group of about 3 students
  + The topics are assigned (from a list available on course website) through a bidding phase, at the end of the Part I
  + Topics are like the one presented in Part I
* Students are also required to:
  + Send provoking questions regarding the topics presented by other groups
  + Interact with the presenting group during the lecture

Each group (as identified in Part II) is evaluated through a final project:

* The goal is identify improvement directions of a state-of-the-art problem
* The topic should be “close” to the one presented in Part II (topic shall be identified together with the lecturer)
* The work should be supported by experiments
* Essay (about 10 pages) + presentation of the project

The grading criteria works like this:

Immagine che contiene testo, schermata, Carattere, linea

Descrizione generata automaticamente

Going deeper, this is the evaluation:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Projects can be:

* Proposed by the group and discussed with the teaching team to evaluate the feasibility
* Selected through a list of proposals presented by SPRITZ group members. There will be a project presentation in early December

# How papers are made and carry out research

(Host: Mauro Conti)

There is a methodology in doing research, just like the scientific method (make observations, form hypotheses, experiment, analyze, report and reproduce results). In our case, it’s reading papers, attending talks, thinking and discussing (not falling into the *survivorship bias*, so focusing on some parts of the process thinking we’re collecting the right data, when actually we’re not, to analyze a particular phenomenon). Also, we make patents to create original ideas, and something not seen before (these are considered factual for the most part).

When we’re doing research, we’re at the edge of the knowledge, when we’re consolidating facts and things nobody knows to build new knowledge and facts. We’re not alone in this, it’s made by the community and other researchers as well, consolidating a clear understanding of themes. Keep in mind we’re designing sound experiments, constantly looking at data, criticizing the overall work and questioning constantly.

Papers usually range from 6/10 pages up to 30, made on the outside by:

* a title
* list of authors (affiliations)
* abstract to give the general idea and context
  + give the good idea to the right people, to try to invest further into your text (keep it short)
* introduction on the problem and its history, the motivation behind and the scientific motivations about the work and the contributions
* a section about related work, comparing what’s already there with our works, highlighting what our work does more than others

Inside instead, we structure like:

* description of the proposal, giving background knowledge, a formal definition of the problem and its method and the overall components
* experimental evaluation, implanting the experimenting and describing the tools used, presenting results and discussing limitations (supporting claims validly)
* conclusions, summarizing contributions and future research conclusions

The *review process* is made by picking a venue of evaluation to other scientists (journal/conference) keeping an eye to the deadline submission.

The venues can either be:

* scientifical journals, places established for several years, where there is a board of experts responsible of evaluating papers (chief/associate editors/reviewers)
* conferences, mostly one shot in a specifical place (many chairs like conference, submission, publicity chairs, etc./program committee members/reviewers)
  + it’s important to understand the quality of the conferences, looking into rankings
  + useful link: <https://people.engr.tamu.edu/guofei/sec_conf_stat.htm>

The chairs are responsible for the reviews and very few are accepted. We disclose information ethically, presenting good for a career’s sake but also having a good paper. To read papers, good places are IEEE Xplore, ACM Digital Library, Google Scholar, dblp, Springer Link, etc.

To assess a paper, it’s important to read it, analyzing the person impact, the author reputation, citations (more advanced: assessing researchers, looking for citing, h-index [used to quote the impact of a paper and uses as the index of number of citations by other authors at least that same number of times. For instance, an h-index of means that the scientist has published at least papers that have each been cited at least ] and more).

Also, one can look for the citation graph that describes the citations within a collection of documents, linking all the citation in between and see how problems were linked and solved.

# Containers and Kubernetes Security

(Host: Alessandro Brighente)

We’re talking about security inside the supply chain, given the different connections between people and software. In fact, usually there’s the need for software updates, which also regards these kinds of topics, being fast in developing and delivering software (abstract of the context).

In this case, we’re talking about containers, which are standards units of software packaging up code and dependencies to run software quicker and bundling a specific configuration of code libraries, configuration files between different environments. They virtualize the OS, so they are lighter but constrained to the OS itself. Also, they can be used for Cloud-Native apps, relying on containers for a common operational model, increasing software flexibly building new architectures (when there’s a big number of containers, we need orchestrators).

They are different from virtual machines, virtualizing the underlying hardware component so that multiple OS machines can ran and access effectively hardware resources, containing OS image, libraries, apps. The applications need an hypervisor, creating an infrastructure to abstract all the VM images. They are less portable but can be scaled slowly because they are more resource-intensive.

One very well-known platform providing the isolation feature is Docker; consider the *containers threat model*, because containers can give access to the machine, so giving overprivileged pieces of code can exploit vulnerabilities on the system, given badly configured hosts.

A good overview of vulnerabilities:

1. **Insecure Networking:**

Insecure networking in the context of containers refers to the use of inadequate or unprotected network configurations, which can expose containers and their data to various security risks. This includes:

* + **No Network Segmentation:** Containers by default can communicate with each other and the outside world. Without proper network segmentation, a compromised container can potentially access or attack other containers on the same host.
  + **Unencrypted Traffic:** Communication between containers and external services may occur over unencrypted channels, making it vulnerable to eavesdropping. This could result in sensitive data exposure or unauthorized access.
  + **Lack of Network Policies:** Without network policies, containers may have overly permissive network access, allowing unintended inbound or outbound traffic, which can be exploited by attackers.

1. **Secret Exposure:**

Secret exposure involves the inadvertent disclosure of sensitive information, such as API keys, passwords, or encryption keys, within containerized applications. This can happen due to:

* + **Improper Storage:** Secrets stored within container images or configuration files can be easily accessed by anyone with access to the container, potentially leading to unauthorized access to services or data.
  + **Logging Secrets:** Insecure logging practices may lead to secrets being written to log files, which could be accessible to unauthorized users.
  + **Environment Variables:** Secrets passed as environment variables can be visible within the container, especially if other processes or users can inspect the environment variables.

To mitigate this risk, it's essential to use secure secret management tools and practices, such as Kubernetes Secrets or Vault, to store and securely manage sensitive information.

1. **Container Escape:**

Container escape refers to a security breach in which an attacker manages to break out of the container's isolated environment and gain access to the underlying host system. This can be extremely dangerous, as it can potentially compromise the entire host. It may happen due to:

* + **Kernel Vulnerabilities:** If a container exploits a kernel vulnerability, it can escape its confined environment and interact with the host system.
  + **Misconfigurations:** Inadequate container configurations, especially related to namespaces and cgroups, can create vulnerabilities that attackers may exploit to escape.
  + **Legacy Linux Capabilities:** Certain Linux capabilities can be used by containers to gain excessive privileges and break out of the container.

To prevent container escapes, it's crucial to keep host systems and container runtimes up-to-date with security patches, enforce strict access controls, and follow best practices in container configuration and isolation.

1. **Vulnerable Code Exploits:**

Vulnerable code exploits occur when an application or its dependencies contain known security vulnerabilities that attackers can exploit. In the context of containers, this can happen in several ways:

* + **Insecure Images:** Using container images with outdated or unpatched software components can expose your application to known vulnerabilities.
  + **Image Scanning:** Failing to regularly scan container images for known vulnerabilities can result in the deployment of images with exploitable weaknesses.
  + **Zero-Day Vulnerabilities:** While not known, attackers may discover new vulnerabilities in containerized applications. Regular security updates and monitoring are essential to respond to zero-day threats.

To mitigate the risk of vulnerable code exploits, maintain an up-to-date inventory of container images, apply security patches promptly, and use container image scanning tools to identify and address known vulnerabilities in your images.

Usually, the applications can run in the user space, so if the user wants to access a file, it should ask the kernel to do so; the interface allowing the user space to make there are requests are system calls/syscalls (inside of Linux, there are more than 300 syscalls, both inside and outside the system).

When you execute a file, the process that gets started inherits the User ID (slide code example):

We can also change the ownership, but still cannot run it unless root (slide code example):

If we set the UID bit (*setuid*, creating executable permissions and *setgid*, to affect both files and directories) (slide code example):

To prove more granularity over privileges, there are *capabilities*, which need to be set since version 2.2 of Linux OS and are assigned to assign specific privileges (in case of containers, what they do in a machine). A good overview here: <https://man7.org/linux/man-pages/man7/capabilities.7.html>

We can set capabilities for command we have *setcap* and *getcap* to know what they are.

We have *control groups*, which are a kernel feature to allow an admin to allocate resources such as CPU, memory, and I/O bandwidth to groups of processes. Managing them involves reading and writing to files/directories within those hierarchies, seeing them as listing content to directories.

Cgroups are like processes in that:

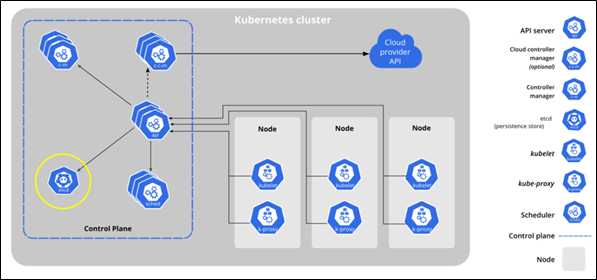
* they are hierarchical, and
* child cgroups inherit certain attributes from their parent cgroup.

The fundamental difference is that many different hierarchies of cgroups can exist simultaneously on a system. If the Linux process model is a single tree of processes, then the cgroup model is one or more separate, unconnected trees of tasks (i.e. processes).

In those, we have namespace, which wrap global system resource in an abstraction that makes it appear to the processes within the namespace that they have their own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace but are invisible to other processes. One use of namespaces is to implement containers.

We can use the *unshare* command, creating a new process from kernel to the program and then some namespace from parent to allow doing this (just like containers do). In the case of Docker, we use the *docker build* command, following the instructions from Dockerfile, carefully managing resources and APIs actions. Remember any user can trigger *docker build* and performing *docker run*.

We need an orchestrator to have a lot of containers, so we need an orchestrator, to automate software production; for this we have software like *Kubernetes*, designed by Google initially, providing a framework to distribute the system resiliently, operating with clusters.



The smallest units deployable are *Pods*, which are environments where multiple containers run and define a trust boundary. Each one has its own IP access, data linked to them, etc. What matters is that they have no identity, security context, encryption or something like that (we need to set all these things ourselves).

The whole point is that, by default, there are a lot of unsecure configurations, poisoned or reuse of pieces of code which are not secure, so defending against supply chain attacks is a top priority.

# spritzmater – Security partner of innovation

(Host: 2nd talk – Federico Turrin)

We talk about cyber-physical system as system that interconnect physical processes (operation technology – OT) and information technology (IT). Contrary to traditional IT systems, which only access the internet, these kinds of system connect to the internet, but also physical connections to defined.

We need confidentiality, integrity and availability for those systems (with the last one being very important). Today’s vehicle networks are mostly based on those, for example with a technology called the CAN bus to have access to the antennas or vehicle music.

A paradigm used to control and monitor vehicles are Vehicle-to-Grid (V2G), with wide spreading electric vehicles, enabling secure communication. Air Ground, instead, we have Automatic Dependent Surveillance (ADS-B), to monitor everything specifically.

We also see Industrial Control Systems, categorized also as Critical Infrastructures are nuclear power plants and water treatment systems (damaging them means several damages). In this model, we have a concept of demilitarized zone to prevent threats and firewalls for system, supervising and monitoring system accordingly to how much problem can cause the coming of other entry points (between operation, control and operational devices).

Cyber-attacks on this system enabled of many dangerous impacts, even health and lives at times (famous example: Stuxnet). In this case, we lose customer trust or even take fines. Protocols are designed with no auth, no integrity or TCP/IP protection, adapting to communication and connection to legacy devices. These systems are called SCADA - Supervisory Control And Data Acquisition, working on such protocols like Modbus, based on ladder logic programming to receive I/O, controlling everything downwards and remotely collecting data, while maintaining control on everything.

These systems need to be monitored real-time, working with legacy systems most of the time, so they’re not well suited for control systems, given their simpler network dynamics. The communication links lack on security features, also no physical protection and they have a long lifespan. These devices are continuously scanned, looking for leveraging tools to craft attacks.

An interesting piece of software is *Shodan*, an engine which continuously maps internet connected devices and shows ports, modules, names and details.

Some interesting search queries: <https://github.com/jakejarvis/awesome-shodan-queries>

To protect from these threats we need:

* Security by Design (according to standards, security best practices, challenging the current architectures, etc.)
* Continuous Security Assessment, testing always to many levels the security in networks via for example of VAPT tests, snapshotting the network and then trying to patch it