Rovesti Gabriel

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

Summary

[Course introduction 2](#_Toc148170872)

[How papers are made and carry out research 3](#_Toc148170873)

[Containers and Kubernetes Security 4](#_Toc148170874)

[spritzmater – Security partner of innovation 9](#_Toc148170875)

[Natural Language Processing in Cyber Threat Intelligence 11](#_Toc148170876)

# 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 and have the OS and its underlying resources belonging to it. This allows containers to be deployed everywhere and allows a usage for velocity and reliability purposes: cloud-native applications, making them ideal for building microservices architecture.

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, allowing multiple side-by-side containers and isolating dependencies from the underlying machine and package versions. There might be threats however; 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 host (attackers from all sides, external/internal/inadvertent).

Immagine che contiene testo, schermata, diagramma, linea

Descrizione generata automaticamente

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.

Immagine che contiene testo, Carattere, bianco, schermata

Descrizione generata automaticamente

We can also change the ownership, but still cannot run it unless root.

Immagine che contiene testo, Carattere, bianco, algebra

Descrizione generata automaticamente

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

Immagine che contiene testo, Carattere, schermata, bianco

Descrizione generata automaticamente

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 (cgroups)*, 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, allowing use a set of pseudo-filesystems that involves reading and writing to files and directories within these hierarchies
* child cgroups inherit certain attributes from their parent cgroup.

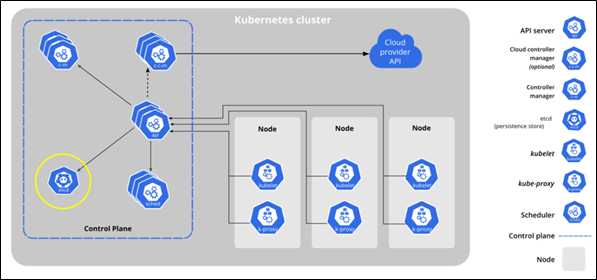
Creating a subdirectory in the memory directory creates a cgroup, where the kernel automatically populates the directory with the various files that represent parameters and statistics about the cgroup. When you start a container, the runtime creates cgroups for it. By default resources (e.g., memory) are not limited, so if a process is allowed to consume unlimited memory, it can starve other processes on the same host. This can open to a possible *resource exhaustion attack*: use as much memory as possible. They are created automatically within platforms like Docker.

If cgroups control the resources a process can use, namespaces control what it can see, which wrap global system resources 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, this way isolating hostname of a machine from other things.

We can use the *unshare* command, creating a new process from kernel to the program and then some namespace unshared from parent (just like containers in managing resources work, but you need root to do so). In the case of Docker, we can list the whole processes in the host which are independent from the host machine and the *docker build* command to create images, setting instructions in a so-called Dockerfile, carefully managing resources and APIs actions. Remember any user can trigger *docker build* and performing *docker run*.

Anyone who has access to a container image can access any file included in that image. That’s why inside images we have *image layers*, where each one is stored separately, meaning that you must be careful not to store sensitive data, even if a subsequent layer removes it.

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



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 the and a trust boundary for its data. What matters is that they have no identity, security context, encryption or something like that (we need to set all these things ourselves). The lifecycle of a pod is controlled by the *kubelet*, the Kubernetes API server’s deputy, deployed on each node in the cluster to manage and run containers (traffic unencrypted here may be easily sniffed by compromised pods/nodes).

By default, every pod can see and talk to every other pod in the cluster and workloads can escalate to host network interface controller, having unrestricted accesses to resources, given the absence of identity, encryption and environmental restrictions in querying data. That’s why they are powerful tools, but they may come with unsecure configurations, poisoned, or reuse pieces of code that are not secure: they may come with unsecure configurations, poisoned, or reuse pieces of code that are not secure. Defending against supply chain attacks is a top-priority for companies.

# spritzmater – Security partner of innovation

(Host: 2nd talk – Federico Turrin)

We talk about *cyber-physical systems* as systems that interconnect physical processes (operation technology – OT) and information technology (IT) (some examples might be smart grid, smart cars, industrial control system, e-Health devices). 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 (Controller Area Network (CAN) bus is a widely used standard in the automotive industry for communication between various electronic components within a vehicle) to have access to the antennas or vehicle music.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Today's vehicle networks are truly distributed electronic systems, via the simple protocol of CAN buses, like drive-by-wire," where electronic sensors and control units manage functions like throttle control, steering, and braking. Also, they provide steering aids, safety features in engine/braking/airbag/navigation control.

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. Here there is *no security at all*, so eavesdropping and injection/modification of messages can happen at any time (e.g. Opensky network).

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 business/environment/human lives). 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). An example of its architecture:

Immagine che contiene testo, schermata, Carattere, software

Descrizione generata automaticamente

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 (e.g. Modbus, which consist in a simple master/slave communication with no security by design).

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.

Immagine che contiene testo, schermata, software, diagramma

Descrizione generata automaticamente

Other systems may be:

* sensors, measuring physical and generating continuously domain values
* actuators, which perform actions on the system and generate categorical values
* PLC – Programmable Logic Controllers, which receive I/O and are based on ladder logic programming, structured and sequential
* HMI – Human Machine Interface, made with a GUI to manage system state, alarms, errors and also allowing inputs

The SCADA systems are at the higher level of the ICS hierarchy, so they monitor and control acquired from field sites, this way managing the communication between the various devices. They 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. This might not be enough and can lead to outdated results, while exploring vulnerabilities and devices. 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

# Natural Language Processing in Cyber Threat Intelligence

(Host: Puthuvath Vinod)

We analyze an attack as:

* Reconnaissance of an attack, gathering information and vulnerabilities
* Attack to the corporate network
* Expansion, leveraging vulnerabilities to other machines in the network
* Keeping control inside of the network

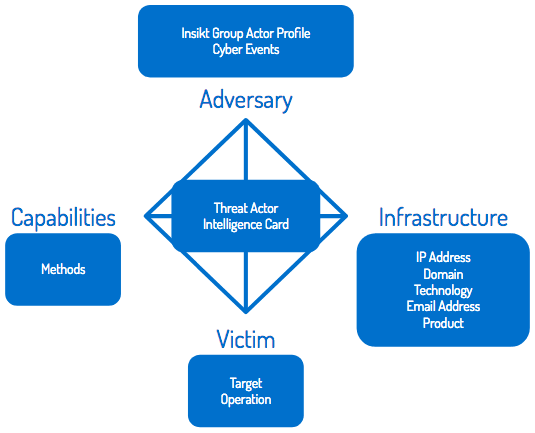
Inside industries, we try to analyze a lot of data, filtering between a lot of useless data, just to understand what’s going on. The main goal is to infiltrate inside of a network with a purpose, given the vulnerability of cyberspace.

*Cyber threat intelligence* (CTI) is data that is collected, processed and analyzed to understand a threat actor’s motives, targets and attack behaviours. We do preprocessing to extract knowledge from data, understanding the attacker’s behaviour, finding out what will happen. This is evidence-based knowledge about existing or emerging menaces/hazards to inform decisions about specific attacks. This way, we will understand tactics, techniques and procedures.

We identify the Indicators of Compromise (IOCs), actionable pieces of CTI to identify and block attacks. They are forensic artifacts signals that a host or a network might have been compromised. IOC can be collected by Structured Threat Feeds or unstructured documents (like blogs, articles, social network, etc.).

The indicators use as parameters:

* Network
* Social
* Blockchain
* Cryptography
* Attack
* Vulnerability
* Location



A famous model available is the *Diamond Model*, which focuses on four core elements: adversary, infrastructure, capability and victim. It helps analysts to understand relationships between these elements and interactions during a cyber-attack.

IOC-centric CTI systems are questionable though:

1. They can’t depict the technical details
2. Attackers frequently change infrastructure, so having static indicators may be limited

We have CTI extraction methods:

* Dictionary based
  + Using a list of predefined terms so they cannot capture previously unseen CTI data (they yield high false positive and negative rates)
* Rule based
  + Here CTI data is represented with letters and numbers, trying to find patterns or rules
* Deep learning based
  + Try to gather a lot of data, giving satisfactory performances when enough will be gathered

The goal is trying to gather as much CTI data as possible, defining *tactics* as high-level strategies used by threat actors (classical example: phishing). We define *techniques* as the specific methods or tools employed with tactics, include social engineering (playing with emotions), forcing victims to click malicious links or techniques. To prevent this, we try to outline the step-by-step process followed by adversaries to execute their tactics and techniques.

The goal of analyzing data is finding links to similar reports of a specific problem and correlating that to specific companies and specific entities. In natural language processing, we want to find processes to extract entities and associate parts of speech and terms to specific things (specifically, it’s defined as *named entity recognition*) or grammatically/semantically understand the term (if it’s malware, if it’s a verb, noun, etc.) via a 1 to 1 mapping.

Each data will be tagged via some terms and via specific libraries (e.g. Python), we can name data and recognize the specific entities (basing itself to English text meanings and not cybersec meanings, e.g. virus which could be a malware and not a disease). Speech tagging, by itself, it’s not a big deal: we just give a tag to each word, segmenting things with no problems. Naming entities, also, assigns labels to spans of text, more difficult because of ambiguity in meanings. There will be something called *cross-type confusion*, where same entities can refer to people, locations, organizations, etc.

A model bases itself tries to categorize entity types and boundaries, having huge amounts of data to annotate things and label them according to abbreviations and acronyms without many variations or ambiguity. We then adopt a specific annotation strategy, taking every word in a sentence and tagging labeling sequence for span-recognition problem, where each word has a single label (each word will get categorized inside spans of interest according to the specific context); examples of tagging may be IO/BIOES tagging.

Now we have annotated data and for each sample, we want to label data according to human annotators. Using *Cohen’s kappa statistics*, we can calculate measure inter-rater reliability (how independent are the terms) for qualitative (categorical) words (so, how much words are linked together according to labels), putting them in an inference matrix and rate variables and words alike (also *Fleiss Kappa*, analyzing the relationship between multiple terms and correlations).

The goal is creating *dependency parsing*, determining relationships between words via libraries or toolkits. The approach is first crawling data inside the cyberspace, performing breadth-first crawling, starting from homepage and discover each link until no new one is found. This one is based on *blog scrappers*, comparing DOM’s pages to template of pages to drop outliers (in this domain, we talk about creating *focused crawlers*). Something specific might be a *topic collector*, where we filter IOC articles (usually longer ones, having lower dictionary) from non-IOC ones, selecting from articles up to *x* words together with their frequencies.

On this, a classification model can be classified over subsets of datasets, label the unknown sets and validating manually selected instances from the classified results. To identify sentences, we look for IOCs using regex and context terms, referring to existing standards. We then check relations between context terms and then parsing sentences into a dependency graph, finding main parts of sentences then parsing all the other parts and finding the shortest paths in between single words (transforming them into vectors and then checking all relationships between every single term).

The relation checking refers to the verb in between two entities in the graph, mapping the verb with action/relation in STIX (Structed Threat Information Expression) and find the relation. The concept of self-attention (which refers to understanding the context of a sentence and what do parts implicitly refer into sentences) is complicated for computers or machines (for example, “the AV did not detect the trojan because it was too complex; a PC does not understand what “it” refers to).

Basically, words are embedded, then transformed according to embeddings and representing values between key/query/value and finding out the attention scores. The *Transformer model* (which labels data weighting words and then stacking them into layers according to the specs) is what’s in mainly used in this context; for example, we have the BERT (Bidirectional Encoder Representations from Transformers), learning how contextualized words are bidirectionally linked between semantics and contexts. In this, you mask words and make the model identifying the missing words; also, we can make the model foresee what the next sentence will be.

Federated learning (FL) then aims to collaboratively train a ML model, keeping data decentralized and sharing model between servers, averaging the parameters then sharing all data, repeating the things until the model it’s closed. We can also have the P2P FL, selecting the specific server for distributing globally the model, training it with local data, exchanging models and aggregating data as well.