Attack & Defense
A network security case study

Enrico RUSSO
Università di Genova

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

https://cybersecnatlab.it

License & Disclaimer

License Information

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Goal

- Learn how an Attack/Defence competition works
- Learn how to apply the knowledge learned in the network security modules in practice
- Learn possible threat mitigation strategies
- Learn how to configure a simple firewall

Prerequisites

- Lecture:
  - *NS_0.1 – Network Fundamentals*
  - *NS_1.1 - Network Analysis and Monitoring*
  - *NS_1.2 - Securing internet communications*

Outline

- Attack and Defense (A/D) competitions overview
- Principles of authentication policies with SSH
- Network traffic control and analysis
- Threat mitigation strategies
- Flag submission

Outline

- Attack and Defense (A/D) competitions overview
- Principles of authentication policies with SSH
- Network traffic control and analysis
- Threat mitigation strategies
- Flag submission

A/D CTF competitions at a glance

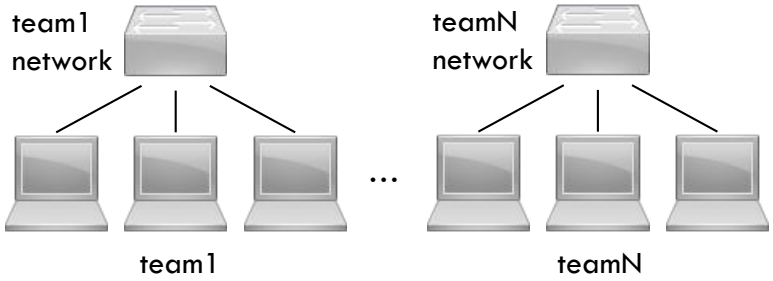
- Participants are in teams
- Each team has its own instance of an identical server hosting different *vulnerable* services (aka, **Vulnbox**)
- Vulnboxes are connected to a shared network
- Teams must attack other teams' services while protecting their own from being hacked

Motivation

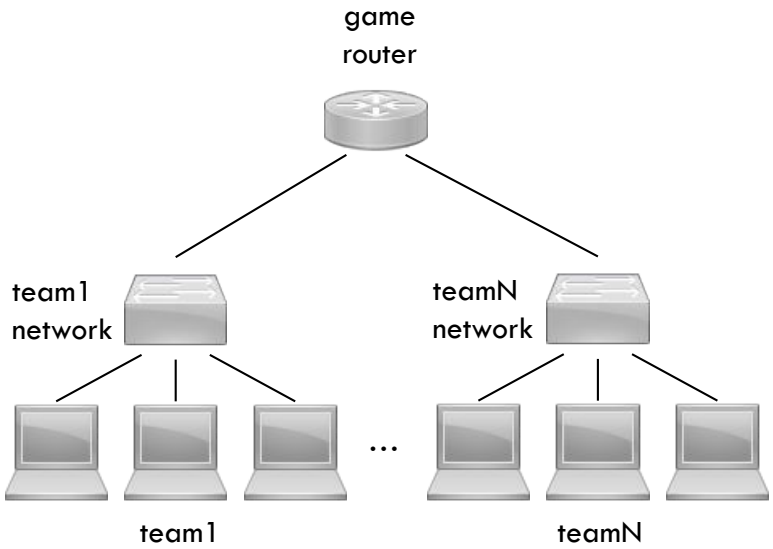
- Why attack and defense competitions?
  - A perfect scenario where learning and training both attacking and defensive skills
  - Teamwork oriented environment
  - The national competition will be an attack and defense

A/D CTFs: setup

- Each team has assigned a private network
- The above network hosts teams participants' laptops

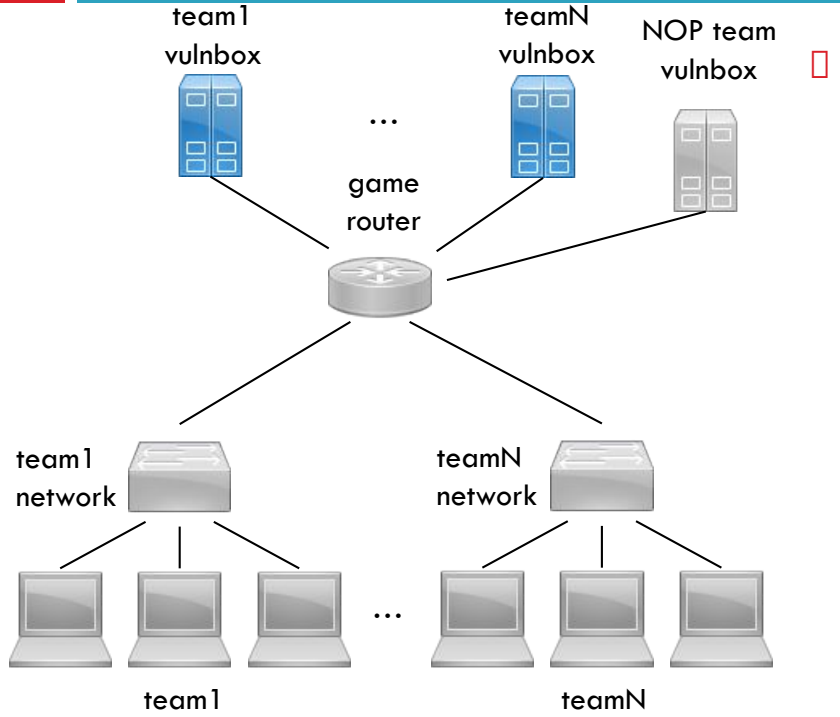


A/D CTFs: setup



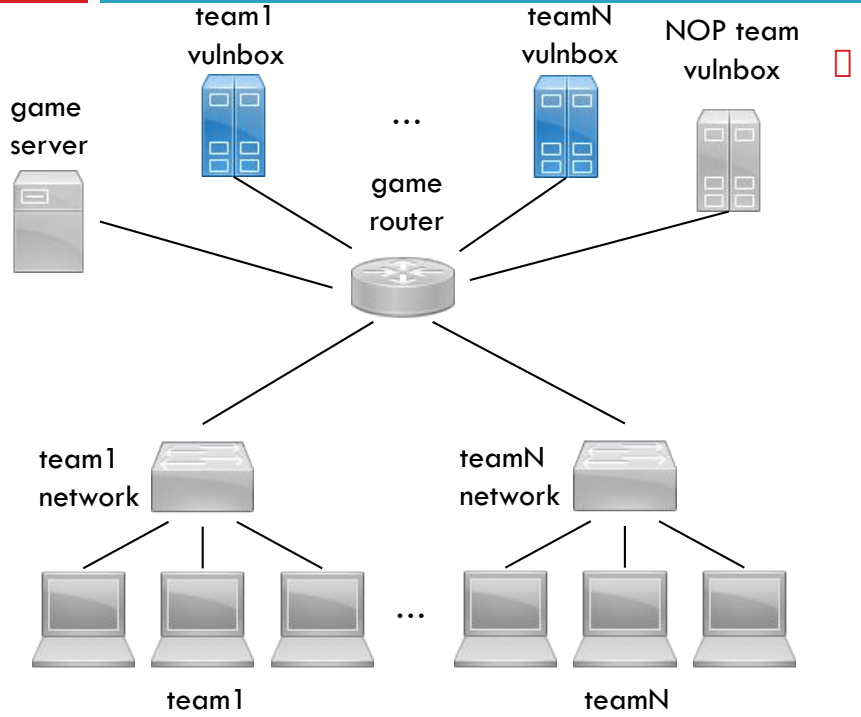
- Each team network is connected (physically or over VPN) to the **game router**

A/D CTFs: setup



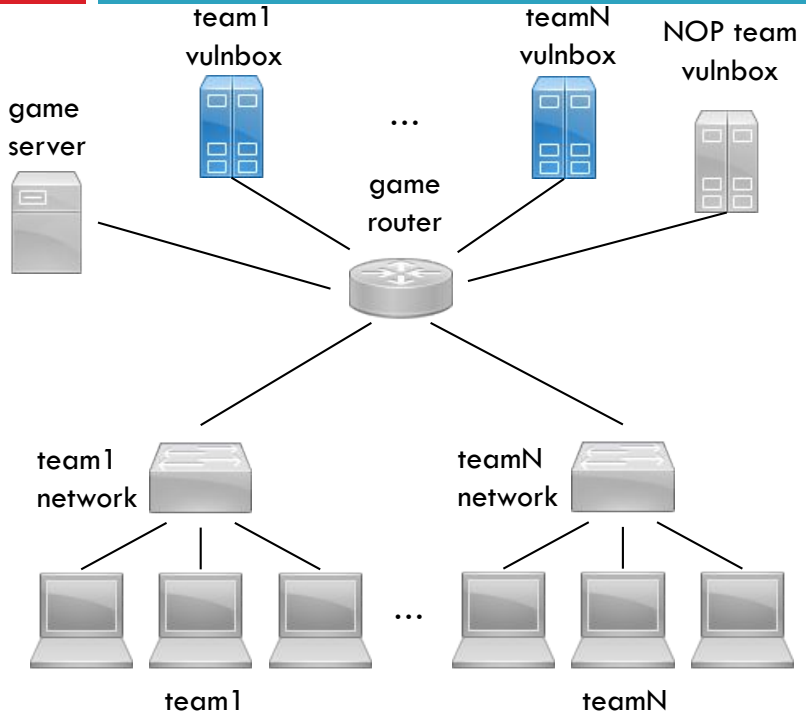
- Each team network is connected (physically or over VPN) to the **game router**
  - allows teams to manage their own **vulnbox** and attack others
  - gives access to a NOP *(Non-Playable)* team vulnbox that is available to teams for testing exploits (NOP team is not tied to the score)

A/D CTFs: setup



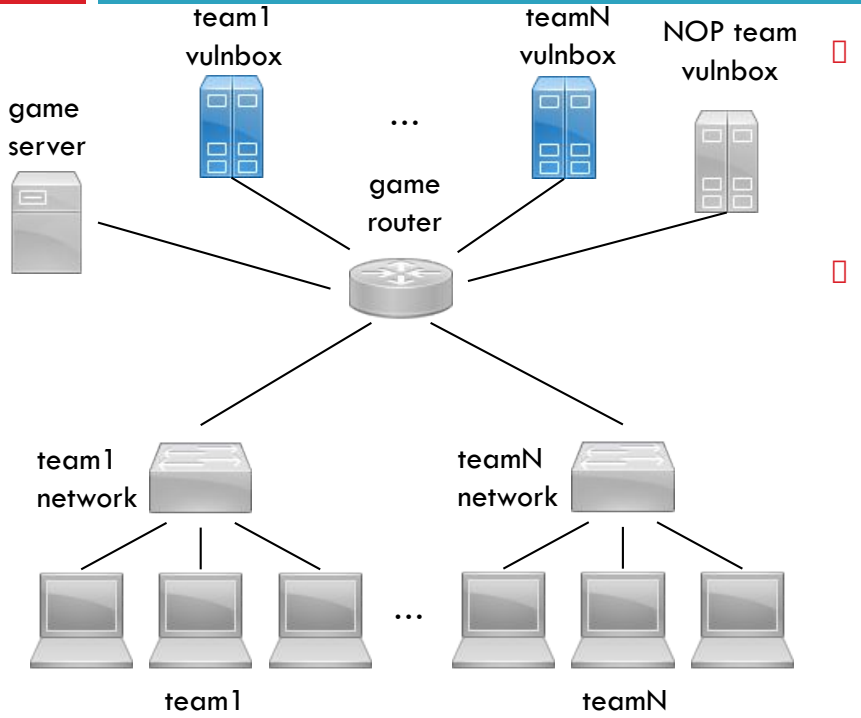
- Each team network is connected (physically or over VPN) to the **game router**
  - allows teams to manage their own **vulnbox** and attack others
  - gives access to a NOP *(Non-Playable)* team vulnbox that is available to teams for testing exploits (NOP team is not tied to the score)
  - connects the organizers' **game server**

A/D CTFs: Vulnbox



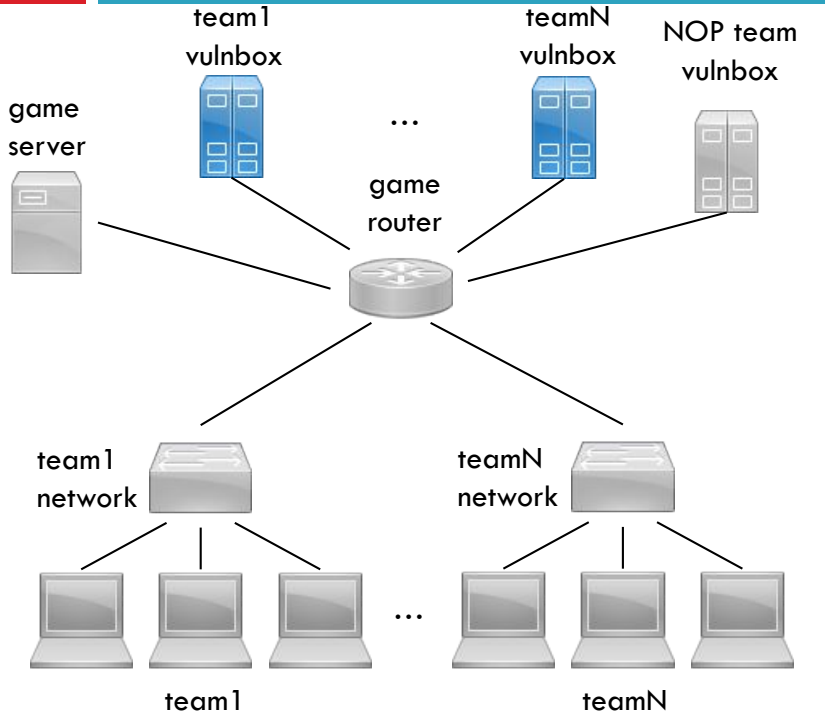
- Each team receives an *identical* **vulnbox**
- The vulnbox is a Linux virtual machine
- It contains different *dockerized* services
- Each service listens on a specific TCP/UDP port

A/D CTFs: Game Router and network



- Vulnboxes have known IP addresses and are accessible to everyone from the teams' networks (all other machines are off-limits!)
- Teams will run exploits from their own laptops
  - traffic to the vulnboxes is modified by the game router using Source Network Address Translation (SNAT)
  - SNAT is used to prevent attacks from being easily distinguished from legitimate traffic

A/D CTFs: Game Server



- Game Server is responsible for:
  - dispatching flags to the vulnboxes
  - checking services integrity
  - hosting the scoreboard and updating scores

A/D Tasks

- Find vulnerabilities in services running in the vulnbox
- Patch vulnerabilities (**defense points**)
- Attack vulnerable machines of other teams. Retrieved flags represent proofs of succesful exploitation (**attack points**)
- Keep the services up and running. **Service Level Agreement** (**SLA**) **points** indicate the availability and correct behavior of services

A/D Ticks

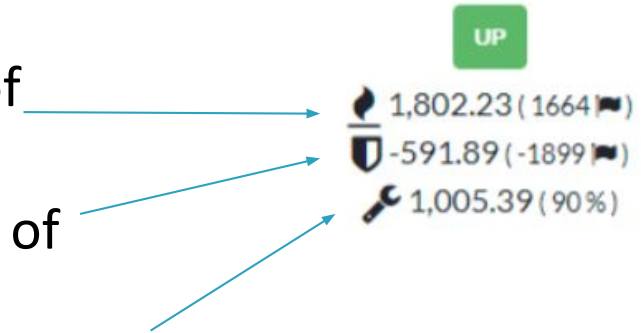
- The game is divided into rounds, each round last T (e.g., 120) seconds and it is called **tick**
- Every **tick** the game server add new flags to services of different vulnboxes
- Randomly, during each **tick**, the game server checks the integrity of services by interacting with them and by retrieving the flags through legitimate accesses

A/D Flags

- A flag is a string and always matched by a predefined regular expression (e.g., a string made up of 31 uppercase alphanumeric chars: [A-Z0-9]{31}=)
- Acquiring attack point requires the teams to submit stolen flags to the game server
  - manually: using an input form in the CTF portal
  - automatically: performing an HTTP POST request to the game server
- Unsubmitted flags are considered expired after a predefined number of ticks (e.g., 5)

A/D Scoreboard

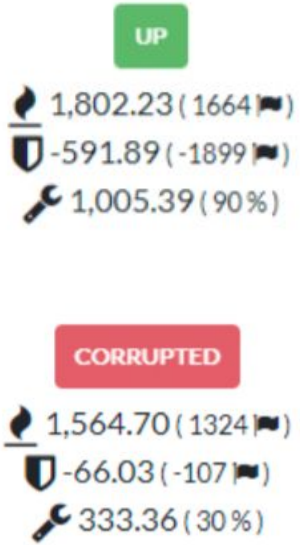
- For each service, the scoreboard determines the score based on the following elements
  - *Attack points*: based on the number of opponents flags retrieved
  - *Defense points*: based on the number of flags lost
  - *SLA points*: based on the time during which the service is working properly



A/D Scoreboard – Service status

- Every service can have at least three different states
  - *UP*: the service is working properly
  - *DOWN*: the service is not reachable
  - *CORRUPTED*: the service is up but some major features aren't available (e.g., the checker cannot retrieve flags)

Both DOWN and CORRUPTED lower the scoring in the same way.



Outline

- Attack and Defense (A/D) competitions overview
- Principles of authentication policies with SSH
- Network traffic control and analysis
- Threat mitigation strategies
- Flag submission

Access to the Vulnbox

- Vulnboxes provide an SSH server (listening on the well-known port TCP/22) that can be accessed by the teams' laptops
- They are configured with a user (root or a root enabled user via sudo* command) and a default password
- Login command** : *ssh remote_username@vulnbox_ip*

* *https://linux.die.net/man/8/sudo*

** *https://linux.die.net/man/1/ssh*

Access to the Vulnbox

- Default credentials are known to all the teams: change the password!
  - Linux command: *passwd* *
- A suggested solution for granting access to team members without sharing a common password
  - enable **Public Key Authentication** (PKA)
  - disable password authentication

* *https://linux.die.net/man/1/passwd*

SSH: configure PKA

- Each team member has to generate their own key pair
  - Linux command: *ssh-keygen* *
  - Notice that only the file with **.pub** extension (public key) can be shared and uploaded to the vulnbox
- Add authorized public keys inside the vulnbox
  - copy them in the *~/.ssh/authorized_keys* file
- Disable password authentication in the vulnbox changing the */etc/ssh/sshd_config* and adding
  - PasswordAuthentication no
- Restart the SSH daemon (e.g., *systemctl reload sshd* in *Debian based* distros)

* https://linux.die.net/man/1/ssh-keygen

Outline

- Attack and Defense (A/D) competitions overview
- Principles of authentication policies with SSH
- Network traffic control and analysis
- Threat mitigation strategies
- Flag submission

Firewall

- It is a good practice to configure the vulnbox in a way that
  - only the public services are exposed to the game network
  - any backend service (e.g., a local database) is filtered from being accessed from outside the server
- A firewall is an easy solution to block traffic from the outside network and allow only specific ports
  - a resource for configuring the firewall in *Debian based* distros is the **Uncomplicated Firewall** (**UFW**) *
  - UFW is installed by default but disabled

* https://wiki.ubuntu.com/UncomplicatedFirewall

UFW: example

A basic configuration to protect the vulnbox

- Set the default policies for **incoming traffic** (traffic directed to the vulnbox) to **DENY**
  - *ufw default deny incoming*
- Set the default policies for **outgoing traffic** (traffic generated from the vulnbox) to **ALLOW**
  - *ufw default allow outgoing*
- Allow access to SSH server
  - *ufw allow ssh*
- For each public service listening on [port] and protocol [tcp/udp]
  - *ufw allow [port]/[tcp/udp]*

UFW: example

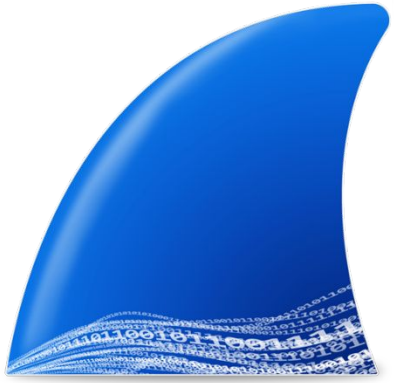
- Enable firewall
  - *ufw enable*
- View firewall status
  - *ufw status verbose*
- Disable firewall
  - *ufw disable*

UFW and Docker

- Attention should paid that Docker **override** UFW rules
  - Docker services with exposed ports (cf. **ports** * in the docker-compose file) imply an **implicit ALLOW rule** for the exposed port
  - All the exposed ports that do not comply with the policies defined in UFW need to be also disabled in the docker-compose file

* https://docs.docker.com/compose/compose-file/compose-file-v3/#ports

Traffic analysis



- Analyzing the network traffic is an essential task during the A/D competition
  - legitimate interactions help teams understand how services work
  - identifying attacks can facilitate patch and exploit development
- Traffic analysis requires
  - Dumping the interesting traffic to files
  - Analyzing saved dumps

Dump network traffic

- TCPDUMP * is the easiest tool to dump the network traffic
- TCPDUMP creates packet capture (pcap) files
- Create a pcap file every X seconds for each public service listening on [port] and protocol [tcp/udp]
  - *tcpdump -w /var/log/[tcp/udp]_port[port].pcap -G X '[tcp/udp] dst port [port]'*

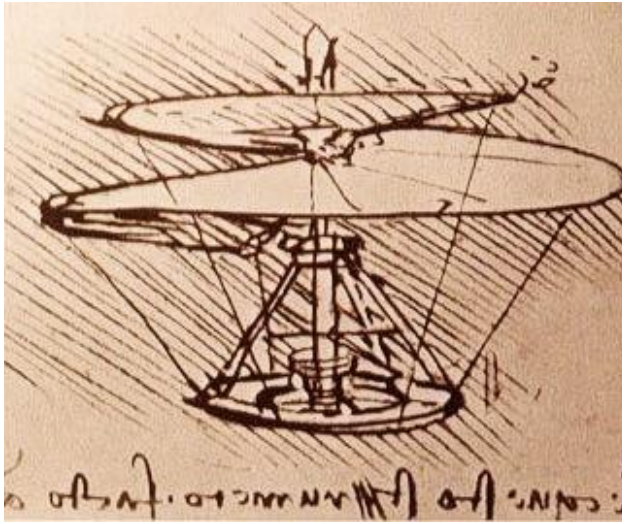
Example for creating a pcap file every 60 seconds for a service listening on TCP port 80: *tcpdump -w /var/log/tcp_port80.pcap -G 60 'tcp dst port 80'*

*https://www.tcpdump.org/manpages/tcpdump.1.html

Analyze dumped network traffic

- PCAP files can be analyzed using Wireshark
- Retrieve the PCAPs from the vulnbox to local laptops using the Secure Copy (SCP) command provided by the SSH daemon:
  - *scp remote_username@vulnbox_ip :/var/log/[pcap filename] [/local/directory]*
- Be aware of the limited space available inside the vulnbox: remember to delete old dumps periodically

Advanced traffic analysis



- Several A/D-specific solutions for analyzing traffic exist and can be used during the competition:
  - Flower: https://github.com/secgroup/flower
  - Caronte: https://github.com/eciavatta/caronte
  - Packmate: https://gitlab.com/packmate/Packmate

Outline

- Attack and Defense (A/D) competitions overview
- Principles of authentication policies with SSH
- Network traffic control and analysis
- Threat mitigation strategies
- Flag submission

Threat mitigation strategies

- During A/D competitions a threat is represented by an exploit stealing flags from a service. Mitigations occur with different (and alternative) remediations:
  1. Patch the service
  2. Implement application-level filtering
  3. Drop the service (but losing SLA points)

Threat mitigation: patches

- Developing a patch to fix the vulnerability represents the ideal (but often the most difficult) solution
- Applying a patch to binary services (i.e., without the source code) could be even harder. Some useful tools are:
  - Hex editors (e.g., WerWolv/ImHex: https://github.com/WerWolv/ImHex)
  - Reverse engineering tools (e.g., Cutter: https://cutter.re/)

Threat mitigation: app-level filtering

- Application-level filtering can (intercept/)block network traffic based on protocol properties or a specific pattern in payloads
- Different solutions can be adopted
  - Using the string extension of Linux iptables (https://ipset.netfilter.org/iptables-extensions.man.html)
  - Web Application Firewall (WAF) for HTTP protocol (e.g., ModSecurity: https://github.com/SpiderLabs/ModSecurity)
  - Network Intrusion Detection and Prevention Engines (e.g., Snort: https://www.snort.org/ or Suricata: https://suricata-ids.org/)

Outline

- Attack and Defense (A/D) competitions overview
- Principles of authentication policies with SSH
- Network traffic control and analysis
- Threat mitigation strategies
- Flag submission

Flag submission

- A/D CTFs require to retrieve a flag from multiple servers (one for each team) and multiple times (one for every tick)
- It is essential to automize the above process using scripts for exploiting vulnboxes and submitting retrieved flags to the game server

Flag submission tools

- Many flag submission tools are available
- Two prominent ones:
  - CTFsubmitter: https://github.com/TowerofHanoi/CTFsubmitter
  - DestructiveFarm: https://github.com/DestructiveVoice/DestructiveFarm

Attack & Defense
A network security case study

Enrico RUSSO
Università di Genova

CYBER CHALLENGE.IT

CYBERSECURITY NATIONAL LABORATORY

https://cybersecnatlab.it