

Basic Principles of Hardening – Tutorial

PA211 Advanced Topics of Cyber Security

November 8, 2022

Daniela Belajová, Pavel Čeleda, and Jan Vykopal

Sandbox preparation

1. Run `pa211_setup` command – only on school computers (nymfe).
2. Clone a new repository with the target (hardening sandbox):
<https://gitlab.fi.muni.cz/cybersec/pa211/hardening.git>
3. Change directory to `hardening/intro`. This directory contains **Vagrantfile**.
4. Run `vagrant up`.

Goals of this seminar

- Test your system against CIS Benchmark
- Create *security templates* for UFW
- Hands-on activities with other supportive tools
- This tutorial focuses primarily on **check of your system**. The actual configuration comes after.

Standards

CIS Benchmarks

Open Source Initiative

- Tools performing audit/hardening based on **CIS Benchmark**
 - As source e.g., **GitHub**
 - **Be careful, it is security risk to run any random script!**
 - Help at the beginnings, can't rely on them for 100%
- **Awesome Open Source**
 - Open-Source application catalog, 56 categories, more than 370,000 projects
 - Currently Top 56 CIS Benchmark Open-Source Projects (search for "CIS Benchmark")
 - Examples:
 - [CIS Ubuntu 20.04 Ansible role](#)
 - [Prowler for AWS security assessment](#)
 - [CIS Kubernetes Benchmark](#)
 - [CIS Benchmark for CentOS](#)



Task 1 – Check System Against CIS

- Introduce yourself with [debian-cis](https://github.com/ovh/debian-cis) tool
- Work on **server VM** (vagrant ssh server)
- QuickStart:
 - `$ git clone https://github.com/ovh/debian-cis.git`
 - `$ cd debian-cis`
 - `$ sudo cp debian/default /etc/default/cis-hardening`
 - `$ sudo sed -i "s#CIS_ROOT_DIR=. *#CIS_ROOT_DIR='$(pwd) '#"
/etc/default/cis-hardening`
- Run **ONLY** audit, do not apply changes to the system
(*--apply* vs. *--audit*) `$ bin/hardening.sh --audit-all --sudo`

Task 1 – Check System Against CIS

1. How many checks passed/failed?
2. Use **grep** command and filter key words ***time_sync, firewall, ...***
 - a. Which time synchronization packages/services are recommended by CIS Benchmarks for Debian 10 to be install/enabled? Are they? Which is/isn't?
 - b. Which of time sync packages are so-called "full featured" for CIS? What does it mean ?
 - c. Is some firewall enabled? Which one(s)?
 - d. (optional) Does CIS Benchmark for Debian 10 specify whether/which firewall you can/should choose?

Solution Q1

```
vagrant@server: ~
99.5.4.5.1_acc_logindefs_ [INFO] [DESCRIPTION] Check that any password that will be created will be SHA512 hashed and salted
99.5.4.5.1_acc_logindefs_ [INFO] Checking Configuration
99.5.4.5.1_acc_logindefs_ [INFO] Performing audit
99.5.4.5.1_acc_logindefs_ [ OK ] ENCRYPT_METHOD SHA512 is present in /etc/login.defs
99.5.4.5.1_acc_logindefs_ [ OK ] Check Passed
hardening [INFO] Treating /home/vagrant/debian-cis/bin/hardening/99.5.4.5.2_acc_shadow_sha512.sh
99.5.4.5.2_acc_shadow_sha [INFO] Working on 99.5.4.5.2_acc_shadow_sha512
99.5.4.5.2_acc_shadow_sha [INFO] [DESCRIPTION] Check that any password that may exist in /etc/shadow is SHA512 hashed and salted
99.5.4.5.2_acc_shadow_sha [INFO] Checking Configuration
99.5.4.5.2_acc_shadow_sha [INFO] Performing audit
99.5.4.5.2_acc_shadow_sha [ OK ] User root has a disabled password.
99.5.4.5.2_acc_shadow_sha [ OK ] User systemd-coredump has a disabled password.
99.5.4.5.2_acc_shadow_sha [ KO ] User vagrant has a password that is not SHA512 hashed.
99.5.4.5.2_acc_shadow_sha [ OK ] User intern2 has suitable SHA512 hashed password.
99.5.4.5.2_acc_shadow_sha [ KO ] Check Failed
hardening [INFO] Treating /home/vagrant/debian-cis/bin/hardening/99.99_check_distribution.sh
99.99_check_distribution [INFO] Working on 99.99_check_distribution
99.99_check_distribution [INFO] [DESCRIPTION] Check the distribution and the distribution version
99.99_check_distribution [INFO] Checking Configuration
99.99_check_distribution [INFO] Performing audit
99.99_check_distribution [ OK ] Your distribution is debian and the version is supported
99.99_check_distribution [ OK ] Check Passed
##### SUMMARY #####
Total Available Checks : 233
Total Runned Checks : 233
Total Passed Checks : [ 96/233 ]
Total Failed Checks : [ 137/233 ]
Enabled Checks Percentage : 100.00 %
Conformity Percentage : 41.20 %
root@server:/home/vagrant/debian-cis#
```


Solution Q2

- a. Packages *ntp*, *chrony* are not installed, *systemd-timesyncd* is enabled
 - CIS Debian Linux 10 Benchmark, v1.0.0 - 02-13-2020, p. 135
- b. Packages *chrony* and *NTP* are NTP implementations
 - *systemd-timesyncd* implements only SNTP (Simple NTP) client-side, therefore can't be time server and it is less accurate than NTP
- c. Ufw, iptables
- d. No, one of those provided is OK

Best Practices

Other Approaches and Tools

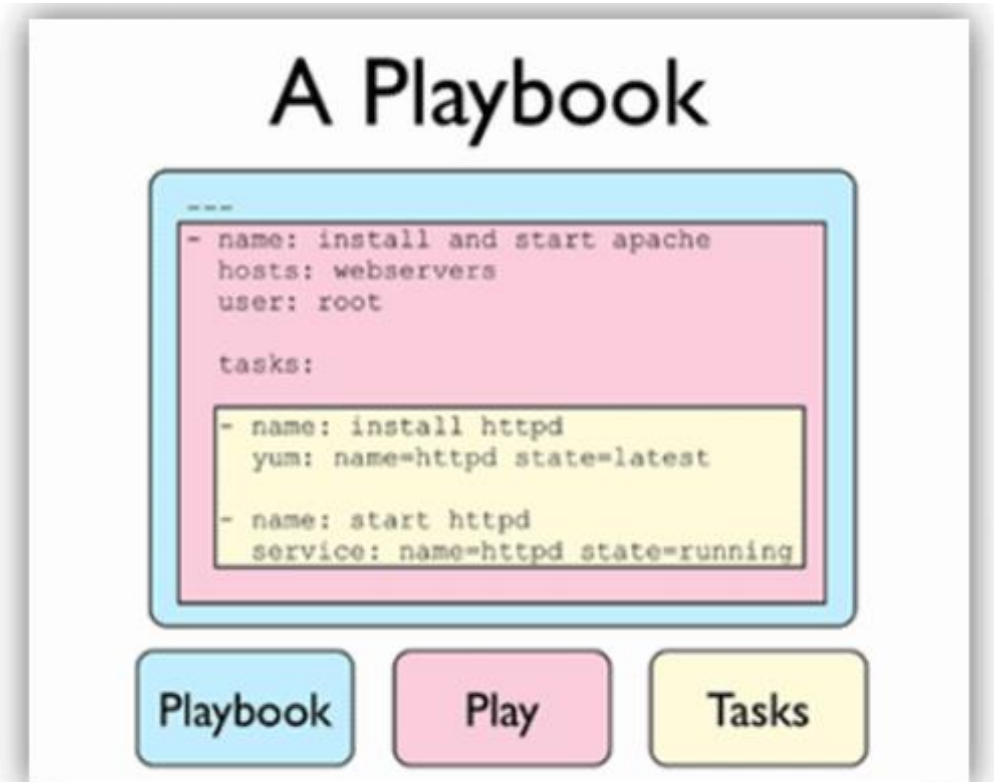
Security Templates

- Goal: To create **a configuration file** that sets all required settings for your server, database, tool, firewall, etc., **automatically** and can be **reused** on other (virtual) machines to provide the same functionality (and level of security).
- Automated configuration with *BASH scripts*?
 - You want to **describe the desired state, not how to get there**
 - What happen if you run commands more times than you want/should?
- Configuration management tools: **Ansible**, Puppet, CFEngine, etc.

Ansible crash course

- Ansible – automation of systems configuration
- Keywords:
 - **Playbooks** – yaml file, consists of play(s)
 - **Play** – executes part of the overall goal of the playbook, running one or more **tasks**
 - **Hosts** – the play's target (all, server, localhost)
 - **Modules** – Ansible main building blocks
 - More in [documentation](#)

Note: Ansible doesn't like **tabs**

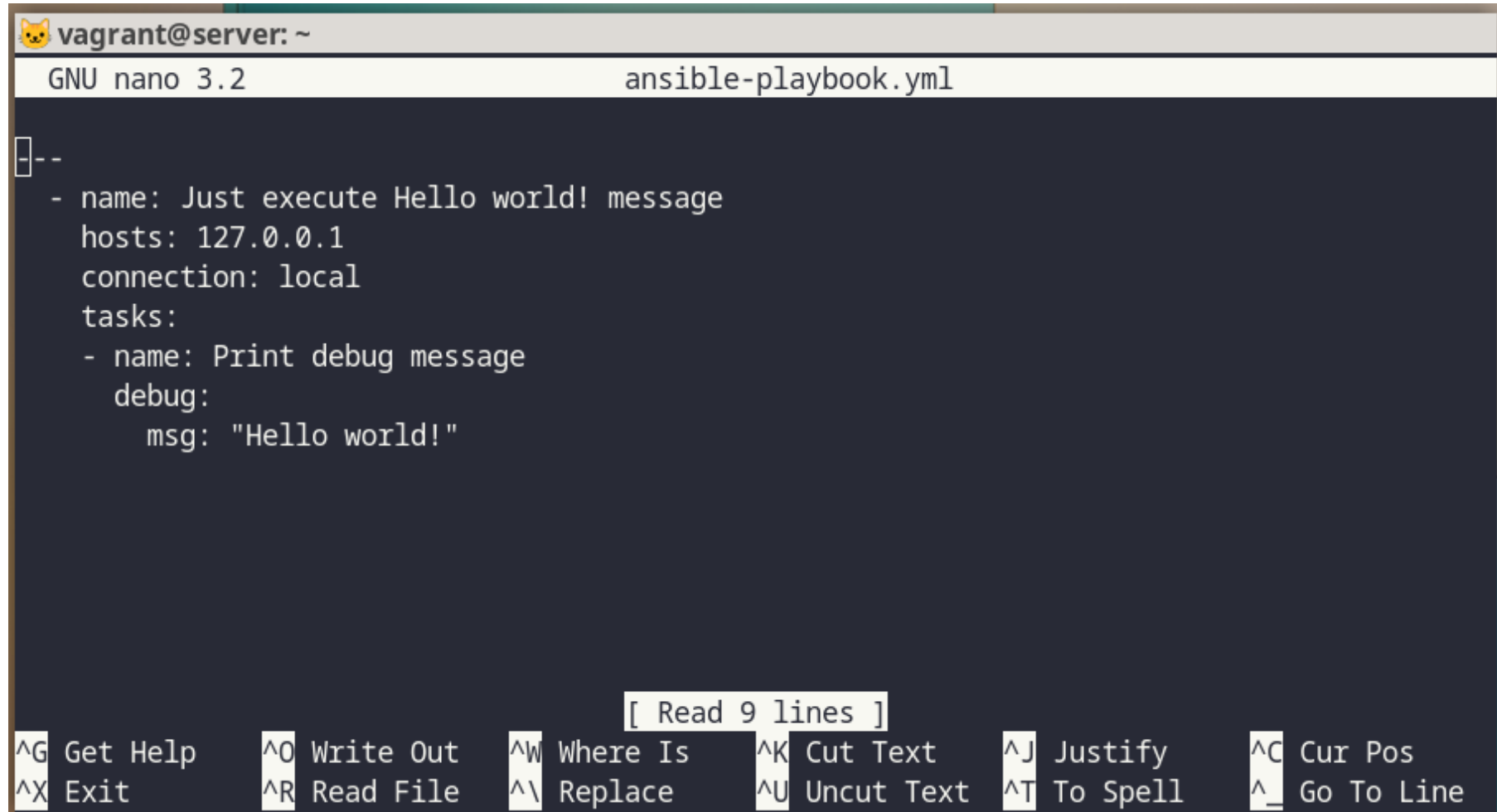


Task 1 - Run "Hello world!" playbook

- Create `helloworld.yml` playbook
 - Playbook will print debug message "Hello world!"
 - Use [Ansible Debug module](#)
 - As hosts use localhost (127.0.0.1)
 - Run the command `ansible-playbook helloworld.yml`

```
vagrant@server: ~  
vagrant@server:~$ ansible-playbook helloworld.yml  
[WARNING]: No inventory was parsed, only implicit localhost is available  
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'  
  
PLAY [Just execute Hello world! message] *****  
  
TASK [Gathering Facts] *****  
ok: [127.0.0.1]  
  
TASK [Print debug message] *****  
ok: [127.0.0.1] => {  
  "msg": "Hello world!"  
}  
  
PLAY RECAP *****  
127.0.0.1 : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Solution 1



The screenshot shows a terminal window with a dark background. The title bar at the top reads "vagrant@server: ~". Below it, the editor header shows "GNU nano 3.2" and the filename "ansible-playbook.yml". The main content area displays an Ansible playbook with the following structure:

```
--  
- name: Just execute Hello world! message  
  hosts: 127.0.0.1  
  connection: local  
  tasks:  
    - name: Print debug message  
      debug:  
        msg: "Hello world!"
```

At the bottom of the terminal, there is a status bar. On the left, it says "[Read 9 lines]". On the right, it displays a series of keyboard shortcuts for nano: ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, ^C Cur Pos, ^X Exit, ^R Read File, ^\ Replace, ^U Uncut Text, ^T To Spell, and ^_ Go To Line.

Task 2 – Configure UFW to follow CIS

1. Check current settings of UFW
 - Run `sudo ufw status numbered`
 - Ansible playbook for this setup is in */vagrant/hardening/provisioning-server/ufw.yml*
2. Configure UFW to follow CIS Benchmark for Debian 10 server
 - **Modify the provided playbook ufw.yml**
 - Use Ansible [UFW module](#), which requires `community.general` collection – it is already present, you can check it by running `ansible-galaxy collection list`

Task 2 – Configure UFW to follow CIS

- Create playbook that follows CIS Benchmark for Debian 10
 - Sections 3.5.2.1 - 3.5.2.5 of Debian 10 Guide
- 1. Ensure **default deny firewall policy** (Be careful, do not close your door!)
 - Deny incoming, outgoing, routed
- 2. Ensure **firewall rules exist for all open ports**
 - Non-loopback TCP open ports, NOT LOOPBACK
- 3. Ensure **outbound connections** are configured
 - No special requirements
- 4. Ensure **loopback traffic** is configured
 - The loopback interface is the only place that loopback network traffic should be seen. All other interfaces should ignore traffic on this network as an anti-spoofing measure.
- 5. Ensure **ufw service is enabled**
- 6. (not CIS) **Reject port** for Active Directory Microsoft service (SMB)
 - Do you know what is difference between reject/drop?

Solution 2

```
vagrant@server: /vagrant/hardening/provisioning-server/solution

vagrant@server:/vagrant/hardening/provisioning-server/solution$ sudo ufw status numbered
Status: active

      To Action From
      --
[ 1] 22 ALLOW IN Anywhere
[ 2] 80 ALLOW IN Anywhere
[ 3] 443 ALLOW IN Anywhere
[ 4] 2222 ALLOW IN Anywhere
[ 5] 445 REJECT IN Anywhere
[ 6] Anywhere ALLOW OUT Anywhere on all (out)
[ 7] Anywhere on loopback ALLOW IN Anywhere
[ 8] Anywhere ALLOW OUT Anywhere on loopback (out)
[ 9] Anywhere DENY IN 127.0.0.0/8
[10] 22 (v6) ALLOW IN Anywhere (v6)
[11] 80 (v6) ALLOW IN Anywhere (v6)
[12] 443 (v6) ALLOW IN Anywhere (v6)
[13] 2222 (v6) ALLOW IN Anywhere (v6)
[14] 445 (v6) REJECT IN Anywhere (v6)
[15] Anywhere (v6) ALLOW OUT Anywhere (v6) on all (out)
[16] Anywhere (v6) on loopback ALLOW IN Anywhere (v6)
[17] Anywhere (v6) ALLOW OUT Anywhere (v6) on loopback (out)
[18] Anywhere (v6) DENY IN ::1

vagrant@server:/vagrant/hardening/provisioning-server/solution$
```

Solution in */vagrant/hardening/provisioning-server/solution/ufw-cis.yml*

Ansible Role as Security Template

- What we have already done, but more complex
- Roles automatically load related *vars*, *files*, *tasks*, *handlers*, and other Ansible artifacts based on a [known file structure](#)
- Can be easily **reused** and **shared**
 - Minimize the risk of introducing mistake, once prepared "to be secure"
- Examples:
 - Role to set [iptables](#) rules and make them persistent
 - Open Source Initiative - [CIS Ubuntu 20.04 Ansible Role](#)

Bonus: Other Tools

Monitoring

Build your own arsenal

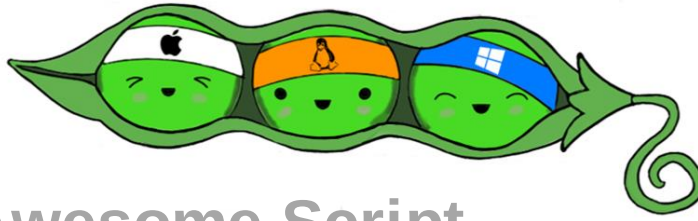
- No magic tools exist automatically solving hardening
- Do **NOT** use random tools changing your system settings!!!
 - They usually need sudo escalation to read/write specific files
- Build your own set of helping (trusted) tools + knowledge

Sandbox Preparation

- Return sandbox into useable form after playing with ufw
- Run **sudo ufw disable** and continue with following tasks

LinPEAS

Linux Privilege Escalation Awesome Script



- Check whether your system is vulnerable to **privileges escalation**
- Linux/Unix*/MacOS hosts
- LinPEAS is often used for penetration testing
- Overview of individual [checks and explanation](#)

LinPEAS – Generating report

- Access server VM `vagrant ssh server`
- Get LinPEAS
 - `wget https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh`
 - `chmod +x linpeas.sh`
- Run LinPEAS and view report
 - `./linpeas.sh > report.txt`
 - `less -r report.txt`

Task – Get to know the LinPEAS

1. Are there some recognized CVEs?
2. Can you find information about current *sudo* version?
3. Are there some *SUID/SGID binaries*?
4. Did LinPEAS find some forgotten passwords?

Solution

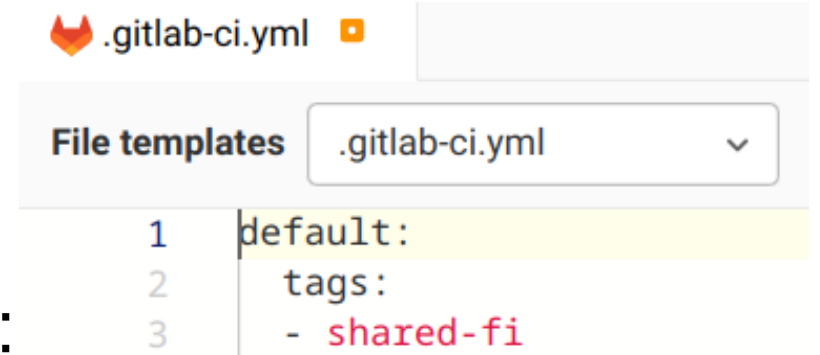
1. Sections **CVEs Check** (CVE-2022-2588) and **Executing Linux Exploit Suggester** (CVE-2019-13272, CVE-2021-3156, etc.)
2. In section **Sudo version** == 1.8.27 is potentially critical (red color), in Interesting Files -> SUID LinPEAS suggests to check if */usr/bin/sudo* is vulnerable
3. Sections **SUID, SGID**
4. Not really, e.g., section **Searching passwords in history files** reference legit code without forgotten passwords. In following sections, most of the files are legit ones which usually does not contain passwords --> but you would need to do closer look to be sure.

Gitlab Security

- Offers many **analyzers** (including open source)
 - You can integrate them into CI pipeline (e.g. merge to main)
- **SAST** – Static Application Security Testing
 - Checks your **source code** for known vulnerabilities
 - Other features - **dependency** scanning, **container** scanning, **secret** detection, **fuzz** testing
- **D(ynamic)AST** – automated web app security testing using OWASP ZAP (Zed Attack Proxy), i.e., attacking your application

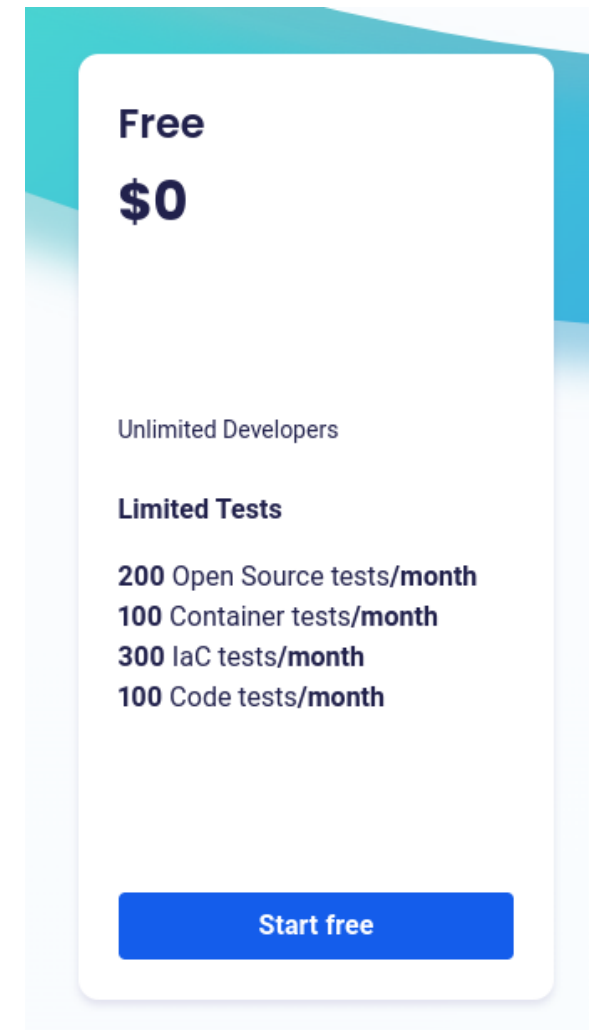
SAST – Test your repository

1. Create `.gitlab-ci.yml` in repository root with following tag if using gitlab.fi.muni.cz:
2. In the repo, go to *Security & Compliance* -> *Configuration*
-> *Enable SAST* and merge it with default settings into main
3. After completing pipeline, check results in *Vulnerability report*



Snyk

- [Security tool](#) which **scans your code**, dependencies, containers, etc. **for vulnerabilities**
- Supports integration with various tools:
GitHub/GitLab, Kubernetes, Docker Hub, IntelliJ, PyCharm, RubyMine, ...
- Free version is limited in number of tests per months
- Requires registration (Google/GitHub)
 - Then set integration with your favorite tools, import repos and run scanning...voila!



Snyk – Scan your repository

The screenshot displays the Snyk web interface for a repository scan. On the left, a sidebar contains filters for Issues (18), SEVERITY (High: 0, Medium: 17, Low: 1), PRIORITY SCORE (0-1000), STATUS (Open: 18, Ignored: 0), LANGUAGES (PHP: 18), and VULNERABILITY TYPES (Cross-site Scripting (X...): 13). The main area shows a search bar and a list of 18 issues. The selected issue is a Cross-site Scripting (XSS) vulnerability with a score of 614. The vulnerability is located in the file `src/modules/cli/users.php` at line 142. The code snippet shows a `print` statement that concatenates user input into an HTML page. The description states: "Unsanitized input from *data from a remote resource flows* into `_`, where it is used to render an HTML page returned to the user. This may result in a Cross-Site Scripting attack (XSS)." A link to learn more about this vulnerability is provided. At the bottom right, there are buttons for "Ignore" and "Full details".

Issues 18

Search...

18 of 18 issues

Group by none Sort by highest severity

M Cross-site Scripting (XSS) SCORE 614

SNYK CODE | [CWE-79](#)

```
138 | $user->setName($name);
139 | $user->setEmail($email);
140 | $user->setAdmin($admin);
141 | if ($manager->storeUser($user)) {
142 |     print "user ".$user->getUid()." was created\n";
```

Unsanitized input from *data from a remote resource flows* into `_`, where it is used to render an HTML page returned to the user. This may result in a Cross-Site Scripting attack (XSS).

[src/modules/cli/users.php](#) 22 steps in 1 file

[NEW](#) [Learn about this type of vulnerability and how to fix it](#)

[Ignore](#) [Full details](#)

Bonus: Pakiti

- Another example of [Ansible role](#)
- Pakiti: monitoring the patching status of Linux systems
 - Client/server model
 - Client on monitored machines regularly sends reports to Pakiti server
 - Pakiti server checks version against its database and collected information
 - Detection of vulnerabilities based on CVE identifiers
- Developed by CESNET
- Current version is almost 2 years old

Bonus: Grafana and Prometheus

- [Prometheus](#) (open-source)
 - Monitoring and alerting tool
 - Recording any numeric time series (metrics recorded over times), e.g. CPU usage
- Prometheus supports [Grafana](#) integration
 - Open-source analytics & visualization web application for databases
 - Integration also with other monitoring solutions: Elasticsearch, MySWL, Graphite, ...
- Grafana provides freely available [demo](#)
 - Use data source, e.g., 'Prometheus – Demo Dashboard'

How was it today?

And how was this part of the course?

Please fill in an **anonymous** exit ticket:

<https://muni.cz/go/pa211-22-09>



M U N I
F I