HAL
open science

MI-LXC: A Small-Scale Internet-Like Environment for Network Security Teaching

François Lesueur, Camille Noûs

▶ **To cite this version:**

François Lesueur, Camille Noûs. MI-LXC: A Small-Scale Internet-Like Environment for Network Security Teaching. ARES 2021: The 16th International Conference on Availability, Reliability and Security, Aug 2021, Vienna Austria, France. pp.1-6, 10.1145/3465481.3469181 . hal-03340749

HAL Id: hal-03340749

https://hal.science/hal-03340749v1

Submitted on 10 Sep 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

MI-LXC: A Small-Scale Internet-Like Environment for Network Security Teaching

François Lesueur[1,2] Camille Noûs[2*]

[1] INSA Lyon, CITI, EA3720, France
[2] Laboratoire Cogitamus, https://www.cogitamus.fr, France

https://github.com/flesueur/mi-lxc

August 2021

Abstract

MI-LXC is a framework to simulate an internet-like infrastructure on top of LXC to practice cybersecurity on a realistic environment. MI-LXC follows the *infrastructure-as-code* paradigm to program the topology of the system and the provisioning of the different hosts. This construction is highly customizable, allowing to create hosts ranging from webservers to graphical desktops. Provisioning of similar subsets of features on different hosts is attained through a template mechanism. MI-LXC currently provides 28 hosts in 11 AS, allowing to simulate BGP routing, DNS, SMTP, HTTP, Certification authorities as well as attacks against these protocols. In this article, we present the MI-LXC framework, the generated infrastructure and some labs on top of it. MI-LXC is a free software (AGPL).
Keywords : Cybersecurity, Cyberrange, Internet simulator, Training platform

1 Introduction

In the cybersecurity area, we need platforms to teach and train people on the complex systems they will have to deal with. CyberRanges, for instance, are integrated platforms simulating an information system, attackers, some scenario, etc. During a CyberRange training, participants should collaborate to fight the attacker.

However, because academic teaching requires pedagogic autonomy, we need free (as in freedom) tools to support it and we should be particularly skeptical when large companies gain some momentum in the education space. Thus, besides needing specific tools for cybersecurity teaching, we should not pave the way for large corporations to supply these tools. Yet, today, most CyberRanges are closed-source with strong ties to the defense industry. As such platforms become mandatory for teaching, we need them to be *compatible* with the central principles of academic education and thus must aliment some perspectives for free alternatives.

In this field, we present MI-LXC (*Mini-Internet using LXC*). MI-LXC allows to programmatically generate a small-scale internet-like environment running as LXC containers, which can in turn be used for cybersecurity training. The contribution of MI-LXC is two-fold. First, it is a Python framework to rapidly prototype interconnected information systems and to interact with these systems using both their commandlines and graphical desktops (Section 3). Second, it is an instantiation of this framework using JSON and Bash representing a minimalist internet (self-contained BGP, alternative DNS root, SMTP, HTTP, CA) on which we can practice cybersecurity (28 hosts on 11 AS, see Section 4). It is lightweight enough to run on students' laptops, who can thus use it autonomously, and does not come with complex dependencies (mostly LXC and its python bindings, which are packaged in every major

*Camille Noûs embodies the collegial nature of our work, as a reminder that science proceeds from disputation and that the building and dissemination of knowledge are intrinsically selfless, collaborative and open.

Linux distributions). Learners can play different roles during a session to understand how these different organizations are interacting (for instance intrusion scenario, network segmentation, IDS or certification authorities, see Section 5). We strongly believe that learning cybersecurity requires to understand indirect dependencies and complex interactions, which in turn requires such self-contained infrastructure that can be customized by both students and teachers (Section 6). We have been using it with students for 3 years and the current version is stable.

It is worth to be noted that, by itself, MI-LXC is *not* a cyberrange. Where cyberranges are generally heavy platforms structured around different scenarios, each scenario being both a specific topology and an associated challenge, MI-LXC runs on a standard PC to simulate the core internet infrastructure, allowing to study the security of internet and autonomous systems. It is not tailored towards competition among different teams but rather towards the analysis of cybersecurity issues in the real internet and the deployment of countermeasures, in a lightweight setup. Yet, MI-LXC could be used as a substrate for cyberranges to generate some core components of the infrastructure and addresses some similar teaching needs.

MI-LXC is a free software (AGPL) available at `https://github.com/flesueur/mi-lxc`.

2 Related work

Network classes have long used network simulators for practical work. Among others, we can cite GNS3[1], Kathara[1], NSG[4], Mininet[6] or Marionnet[2]. These tools are well reknowned and robust. They allow to specify, export and import complex network topologies, as well as to study network-level properties (routing, QoS, etc.). However, being network-oriented, they do not provide facilities to provision and differentiate the hosts in the network. Mininet, for instance, starts isolated bash processes to simulate the hosts but does not provide facilities to provision different filesystems, packages and configurations on these different hosts. They are well-suited to prototype network topologies, but not to prototype information systems with higher level services.

A set of recent tools uses Docker to isolate the different parts of the system to simulate (Labtainers[5], Seed[3], Kathara[1], Dockernet[3]). Docker is quite popular but is centered on the paradigm of one process per container. Rather than launching an init process, Docker containers run a single application and are then composed using docker-compose. It should be possible to run an init process in a docker (although there seems to be some incompatibilities with systemd at least) but it goes against the very basic philosophy of Docker. Thus, complying with the Docker philosophy does not allow to simulate realistic systems running on classical OS (multiple concurrent users, different processes, allowing an attacker to pivot). Moreover, these projects concentrate on the framework and some scenarios, but do not propose a generic internet-like sandbox.

SecGen[8] creates full VMs but is dedicated to CTF-style challenges. It creates voluntarily vulnerable images and plants flags inside. It is a tool to practice offense with artificial vulnerabilities, where we rather aim at practicing defense of up-to-date systems.

A few frameworks target the cyberrange usage, such as ADLES[2], Cyris[7], KYPO[9] or EduRange[10]. They typically focus on heavy virtualization (VMWare Vsphere, KVM, OpenStack, etc.) and thus require a dedicated hosting infrastructure and are quite complex tools. They are tailored towards the defense of a system with active attackers (which may be simulated) rather than towards the comprehension of internet interactions.

There are also, of course, several commercial cyberranges. We cannot ignore them but they are not well publicly documented. They have two major shortcomings. First, as stated in the introduction, we need free platforms for teaching: we need to be autonomous on our objectives, to be able to adapt and study a given platform. Second, they usually rely on some costly hardware, typically several high-grade PC to virtualize the whole infrastructure, and are thus not suited for students to run on their own laptops.

VM infrastructure building tools, such as Vagrant or Terraform, and provisioning tools, such as Puppet or Ansible, are of course also in the scope. But in fact, they tackle slightly different problems. Vagrant automates the VM creation but the biggest work in our case is, in fact, the recipes of VM creation.

[1] `https://gns3.com/`
[2] `https://marionnet.org`
[3] `https://github.com/gmiotto/dockernet`

Vagrant moreover does not support well LXC (only an unofficial and unmaintained plugin) and mostly targets heavy virtualization such as VirtualBox. Puppet or Ansible target the provisioning but not the creation process and network topology aspects. In fact, given LXC provides official Python3 bindings and that generated infrastructures do not have to be maintained in the long time, Vagrant, Puppet or Ansible do not provide clear benefits. Anyway, most of the work lies in the recipes to configure the different hosts in the simulated systems.

3 MI-LXC Framework

MI-LXC is an *infrastructure-as-code* framework generating a lightweight internet-like infrastructure which can run on standard hardware such as students' laptops. It is written in Python3, generates LXC containers and uses Python-LXC bindings to interact with these containers. The topology is defined in a few JSON files and the provisionning of an internet-like environment relies on simple bash scripts. This infrastructure can thus be easily modified, shared, versionned or updated. In this section, we present how to use the framework to specify an architecture.

3.1 Global topology

First, the global topology is defined in the `global.json` configuration file. This configuration file lists the different AS and specifies how they are interconnected. In the following excerpt, there is the definition of the "opendns" AS in the `global.json` file. This AS uses the as-bgp template and has 2 network interfaces: one is connected to a transit operator (transit-a) and one is connected to its LAN (opendns-lan).

```
"opendns": {
  "templates":[
    {"template":"as-bgp", "asn":"7",
      "asdev":"eth1",
      "neighbors4":"100.64.0.1 as 30",
      "neighbors6":"2001:db8:b000::1 as 30",
      "interfaces":[
        {"bridge":"transit-a",
          "ipv4":"100.64.0.30/24",
          "ipv6":"2001:db8:b000::30/48"},
        {"bridge":"opendns-lan",
          "ipv4":"100.100.100.1/24",
          "ipv6":"2001:db8:a100::1/48"}
      ]
    }
  ]
}
```

3.2 Local topology

Second, for each AS, a `groups/<asname>/local.json` configuration file describes its local topology. The `local.json` lists the internal hosts and how they are interconnected. The interconnections as well as the internal configurations can be modified during a lab, for example to study how to segment an information system. Here, we can see the definition of the "sales" host in a `local.json` file. This host is plugged on the ¡asname¿-lan bridge, receives some IP addresses and configures four templates: it is a SSHFS client, an LDAP client, a graphical mail client and does not use DHCP.

```
"commercial": {
  "container":"sales",
  "interfaces":[{"bridge":"lan",
    "ipv4":"100.80.0.2/16",
    "ipv6":"2001:db8:80::0:2/48"}
  ],
  "gatewayv4":"100.80.0.1",
  "gatewayv6":"2001:db8:80::0:1",
  "templates":[
    {"template":"sshfs", "server":"filer"},
    {"template":"ldapclient",
      "domain":"target.milxc",
      "server":"ldap"},
    {"template":"mailclient",
      "domain":"target.milxc",
      "mailname":"sales",
      "password":"sales",
      "login":"sales"},
    {"template":"nodhcp",
      "domain":"target.milxc",
      "ns":"100.80.0.1"}
  ]
}
```

3.3 Provisioning

Third, for each host, a simple bash script is used to provision it. For instance, this part of a provisioning script configures a DNS zone and adds a custom script to the host.

```
# disable systemd-resolved which conflicts with nsd
echo "DNSStubListener=no">>/etc/systemd/resolved.conf
systemctl stop systemd-resolved

# manage gozilla.milxc zone
apt-get update
DEBIAN_FRONTEND=noninteractive apt-get install -y unbound
cp dns.conf /etc/unbound/unbound.conf.d/

# Script to add a cert to the CA/Browser consortium
cp addcatofox.sh /usr/local/bin
chmod a+x /usr/local/bin/addcatofox.sh
```

3.4 Masters

Fourth, each host derives from a master image. Master images are listed in the `global.json` file and are provisioned exactly like hosts. We currently propose two masters: a Debian Buster for general-purpose hosts and an Alpine Linux for BGP routers.

3.5 Templates

Finally, hosts can use some templates, which allow to factorize redundant aspects. Templates are mostly bash scripts which look like provisioning scripts, except that they propose variables which are substituted by the supplied parameters. We currently propose templates for BGP routers, LDAP clients, mail clients, mail servers, nameservers or SSHFS clients. The following example allows to mount SSHFS network fileshares hosted on `$server`.

```
apt-get update
DEBIAN_FRONTEND=noninteractive apt-get install -y sshfs libpam-mount hxtools

echo -e "#!/bin/bash\nmknod -m 666 /dev/fuse c 10 229\nexit 0" > /etc/rc.local
chmod +x /etc/rc.local

echo "user_allow_other" >> /etc/fuse.conf

cp pam_mount.conf.xml /etc/security/
sed -i -e "s/\$server/$server/" /etc/security/pam_mount.conf.xml
```

Figure 1 illustrates running containers providing both command-line and X11 access.

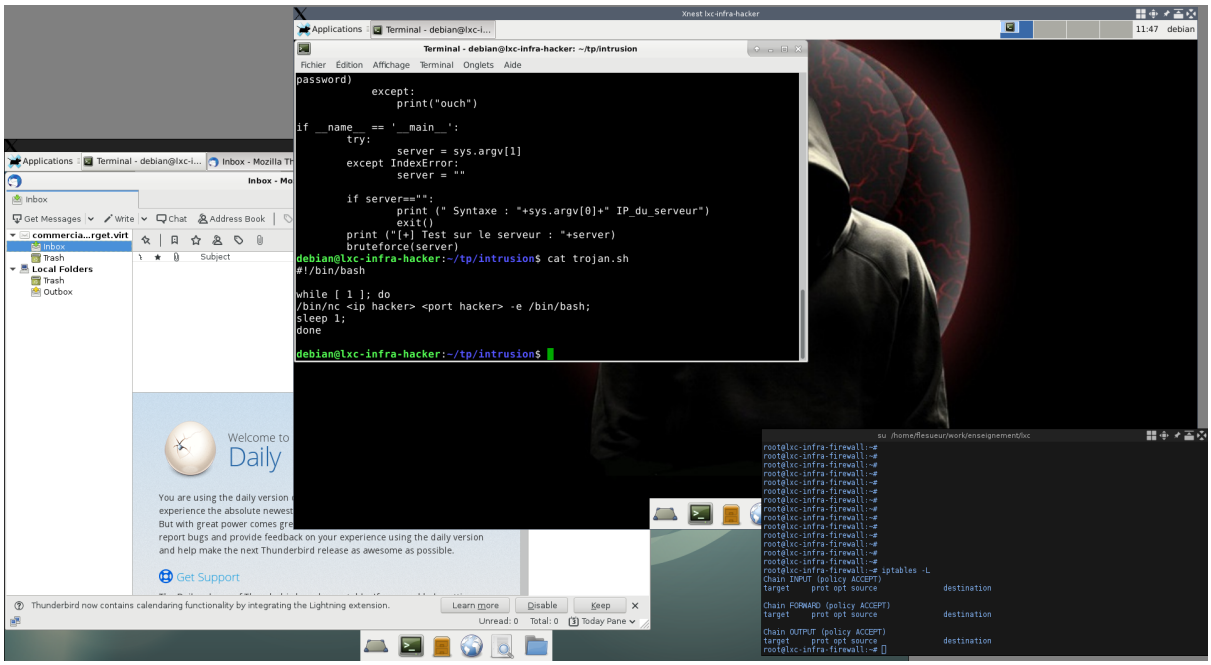

Figure 1: Screenshot of the desktop of the hosting machine, showing both command-line and X11 access to simulated hosts: on the left, there is the graphical desktop of some simulated host running a mail client; on the top, there is another graphical desktop of some other simulated host; on the bottom-right, there is a command-line access on a simulated firewall.

4 MI-LXC Topology

The currently running topology is illustrated on Figure 2. It consists in 28 hosts in 11 AS:

- The core network is represented by the two transit operators *transit-a* and *transit-b*. *Transit-a* also routes the whole infrastructure to the *real* internet through the LXC bridge on the host machine
- The `.milxc` TLD, used for internal domains (for instance `target.milxc`), is managed by the *milxc* AS
- Two alternative DNS roots, which allow to resolve the `.milxc` TLD as well as the *real* TLDs, are hosted on *root-o* and *root-p*
- An open DNS resolver is provided by *opendns*
- The commercial ISP *isp-a* provides internet access to several clients, who may be honest or malicious

- The certification authority *mica* uses the ACME protocol (like *Let's Encrypt*) to deliver HTTPS certificates
- The *gozilla* browser editor provides a web-browser which may accept new certificates roots
- The *target* enterprise hosts an organization system with SMTP, HTTP, DNS, LDAP, filer, intranet and desktop clients. It is voluntarily in a flat, unsecured network architecture
- The *ecorp* enterprise allows to run BGP attacks

The aim of this topology is to reproduce core internet concepts, such as BGP routing or open protocols (DNS, SMTP, HTTP). It is an insecure internet (as a large part of the real internet is), since the aim is to analyze these threats and to learn how to deploy security measures.

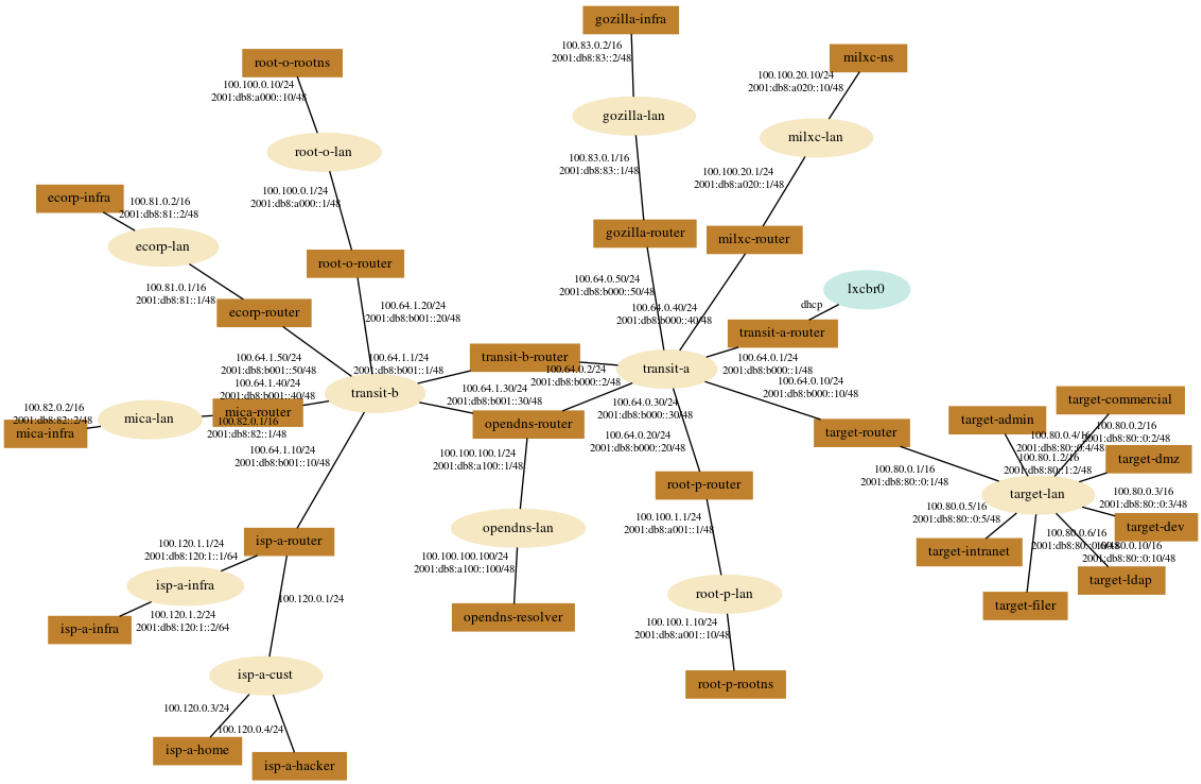

Figure 2: Currently implemented topology. Ovals are network bridges and rectangles are hosts.

5 Training Examples

In this section, we describe the 4 lessons of practical work (4 hours each) we propose with MI-LXC: an intrusion scenario, a network segmentation (firewall), an IDS deployment and the HTTPS security model with a global certification authority. The pedagogic material of these 4 lessons is available on the Github page (in French). We also introduce how MI-LXC can be used for students' projects.

5.1 Intrusion Scenario

Intrusion scenario, network segmentation and IDS deployment belong to the same course. This intrusion scenario allows to apprehend the type of tools an attacker may use, how a multi-step attack may be performed, what is pivoting and how we could defend from that.

The attacker is external and runs everything from *isp-a-hacker*. Her aim is to steal and delete confidential information hosted on an internal server of the *target* corporation. The proposed attack can follow this plan:

- A wiki is hosted on *target-dmz* and is accessible from the outside. The attacker can brute-force an administrative account, upload a trojan with a reverse shell and add a page to explain this is a security update.
- The attacker can then send a spoofed email to an internal employee, masquerading as an admin, asking him to follow the security update process. This process being described on a local server (on the forged wiki page), the employee has many reasons to trust it. These two first steps illustrate social engineering and students have many ideas on this topic !
- When the employee follows the process, he in fact launches a reverse-shell with the attacker on the other side.
- The attacker can then transfer nmap and scan the internal network with it. At this point, students have to map the whole network: hosts, listening softwares, banners, versions, ...
- An internal web server clearly hosts some sensitive content
- The employee can connect to this server with SSH, but the attacker needs to get this employee's password. Students can either choose to install a keylogger or use *LaZagne*, which is a tool digging into software configuration to find configured accounts. In this case, *LaZagne* can find the password of the mail account in the ClawsMail configuration.
- The attacker can then connect to the internal web server using SSH and the newly discovered password. She can find sensitive files there.
- There is another attack path, through the machine of a developer whose code is continuously deployed to this web server and can then lead to a continuous delivery attack.

5.2 Network Segmentation

This lesson is ran from the *target-admin* host to configure the firewall on *target-router*. First part is to gain some knowledge on iptables/nftables using simple examples. Then, having understood what can be filtered, the possible attack paths and the organization of the information system, students propose a network flow matrix. This intermediate step allows to discuss their choices until they provide a sound proposition. Finally, they can implement this matrix by adding some network interfaces and bridges, updating the network configuration in MI-LXC and configuring iptables on *target-router*.

After this session, students obtain a network where functional zones are clearly isolated and lateral movement are constrained. The employee who was compromised during the intrusion, for instance, cannot access the whole network anymore. Only expected flows are remaining and these flows will be monitored using an IDS during the following lesson.

5.3 IDS

During this lesson, students deploy an HIDS, a NIDS as well as some concentration and correlation tools. We mainly use OSSEC, Suricata, Prelude and Prewikka but some other choices are possible. Students first propose the artefacts they should observe and then elaborate signatures or process to detect them. They can for instance detect the brute-force (OSSEC or Suricata), appearance of new files (the uploaded trojan, with OSSEC), opening of the reverse-shell (Suricata), alteration of sensitive files (OSSEC), ...

This work allows to evaluate the possible control points, their soundness, but also how easy it is to evade such controls. Starting from that, we plan to propose something further, around NSM or threat intelligence and collaboration among different organizations.

5.4 Certification Authorities

This lesson is isolated from the precedent ones and is done in another course. During this lesson, students practice and understand the HTTPS security model with a global certification authority such as *Let's Encrypt*. The objective is to allow someone on *isp-a-home* to securely connect to `https://www.target.milxc` which is hosted on *target-dmz*.

First, students use a simple HTTP connection and attack it, using a DNS attack (zone modification) or BGP hijack (using the dedicated rogue AS *ecorp*): they thus can see the MitM problem we want to solve. Then, they create the CA on *mica-ca* using the SmallStep toolsuite: these tools provide an easy-to-use CA (no more openssl configuration) compatible with the ACME protocol (the automated challenges from Let's Encrypt). They can then ask for a certificate from *target-dmz* and configure their webserver. Then, to make it a global CA, MICA needs to be accepted by the browsers' editors, as in real life with the CA/Browser forum (we do not simulate a local CA but a global CA, to analyze the security model of HTTPS through internet). This part is simulated with Gozilla, an internal browser editor, which can accept this new CA and will then provide an updated browser to *isp-a-home*, with a new CA root in its trust store. The user can finally connect securely to `https://www.target.milxc`, without any warning. Finally, students examine what can happen if an attacker attacks this connection (there is a security error) or attacks the verification process by the CA (there is a wrongly emitted certificate). We conclude with some Certificate Transparency remarks.

5.5 Students' Projects

Besides these practical work, MI-LXC can act as a substrate for students' projects. Since MI-LXC runs directly on their own laptop rather than in the cloud. they can rapidly prototype some interconnected systems without any external registration or synchronization. They benefit from the framework and from the already deployed infrastructure. For instance, some students used it to centralize logs in an ELK stack or to explore how to assess a continuous improvement methodology of the security monitoring.

6 Usage

In this section, we describe how MI-LXC is used by students for practical work, by students for projects and by teachers for preparing specific infrastructures.

6.1 By students for practical work

Students run MI-LXC autonomously on their laptops. It can be installed in two different ways. First, there is an installation procedure on Linux, with a few dependencies (mostly LXC and its python bindings). Second, there is also a VirtualBox VM which can run on any OS ; this VM is created using Vagrant (the script is available in the code repository) and is directly downloadable. This allows a smooth bootstrap of the practical work.

Whether using the installation on Linux or the VM, next steps are similar. Once the infrastructure is created with `'./mi-lxc.py create'` (already done in the VM, requires 10-20 minutes depending on CPU, storage speed and network bandwidth), students start the system with `'./mi-lxc.py start'`. Then, they mostly use three commands:

- `'./mi-lxc.py print'` prints the current topology (as in Figure 2);
- `'./mi-lxc.py attach target-dmz'` opens a root shell on the target-dmz host;
- `'./mi-lxc.py display isp-a-home'` opens a graphical desktop on the isp-a-home host.

6.2 By students for projects

When students develop cybersecurity projects with MI-LXC, they need to modify the infrastructure. Typically, they would first add some hosts or change the interconnections. For that, they need to edit JSON files (`local.json` of the relevant AS) to describe the aimed topology. Second, they would modify some other hosts. For that, they need to edit provisioning scripts (relevant `provision.sh`) to suit their needs. Finally, to update the running infrastructure, they need to recreate the containers. They can either destroy all containers or destroy only specific containers (useful during the development), and then create them again. The two commands are:

- `'./mi-lxc.py destroy target-dmz'` to destroy only the target-dmz host;

- `'./mi-lxc.py create'` to create all needed hosts.

They can also directly configure the hosts when running MI-LXC since changes are permanent across reboots.

6.3 By teachers

Teachers can either reuse the provided MI-LXC infrastructure as is or provide a customized VM to their students. To provide a customized VM, they would add and edit hosts similarly as in the previous subsection. Finally, when everything is ready, they can create a VM using this precise deployment with `'vagrant up'` (in the `vagrant/` subdirectory). This will create and bootstrap a clean VirtualBox VM which can be tested and then exported as an OVA (which the student can import into their own VirtualBox).

7 Conclusion

MI-LXC allows to specify, program and deploy virtual infrastructures for cybersecurity training. It allows to run a simulated internet-like environment currently consisting of 28 hosts in 11 AS. This MI-LXC topology runs smoothly with 2GB of RAM and 6GB of disk (when installed directly on a Linux host, we also provide a ready-to-run VM needing 13GB of disk). We currently use it for several practical works: intrusion, network segmentation, IDS and certification authorities for HTTPS. The whole project consists of around 1000 lines of Python, 1000 lines of Bash and 300 lines of JSON, which makes it quite maintainable and easy to understand.

In the future, we expect some improvements as well as some new usages. First, we experimented some remote access control to the infrastructure on the students' laptops, using a VPN and X2GO. This yielded interesting results in the case of remote teaching we should develop. Then, we should investigate adding some background activity in the infrastructure, to simulate honest as well as malicious users and provide some background noise. Finally, we are also interested in adding other practical works, to teach NSM, hunting, collaborative threat intelligence or incident response.

References

[1] G. Bonofiglio, V. Iovinella, G. Lospoto, and G. Di Battista. Kathará: A container-based framework for implementing network function virtualization and software defined networks. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2018.

[2] Daniel Conte de Leon, Christopher E. Goes, Michael A. Haney, and Axel W. Krings. Adles: Specifying, deploying, and sharing hands-on cyber-exercises. *Computers & Security*, 74:12–40, 2018.

[3] Wenliang Du. Seed: hands-on lab exercises for computer security education. *IEEE Security & Privacy*, 9(5):70–73, 2011.

[4] Thomas Holterbach, Tobias Bühler, Tino Rellstab, and Laurent Vanbever. An open platform to teach how the internet practically works. *SIGCOMM Comput. Commun. Rev.*, 2020.

[5] Cynthia E Irvine, Michael F Thompson, Michael McCarrin, and Jean Khosalim. Labtainers: a docker-based framework for cybersecurity labs. In *Proc. 2017 USENIX Workshop on Advances in Security Education*, 2017.

[6] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[7] Cuong Pham, Dat Tang, Ken-ichi Chinen, and Razvan Beuran. Cyris: a cyber range instantiation system for facilitating security training. In *Proceedings of the Seventh Symposium on Information and Communication Technology*, pages 251–258, 2016.

[8] Z Cliffe Schreuders, Thomas Shaw, Mohammad Shan-A-Khuda, Gajendra Ravichandran, Jason Keighley, and Mihai Ordean. Security scenario generator (secgen): A framework for generating randomly vulnerable rich-scenario vms for learning computer security and hosting {CTF} events. In *2017 {USENIX} Workshop on Advances in Security Education ({ASE} 17)*, 2017.

[9] Jan Vykopal, Radek Ošlejšek, Pavel Čeleda, Martin Vizvary, and Daniel Tovarňák. Kypo cyber range: Design and use cases. 2017.

[10] Richard Weiss, Franklyn Turbak, Jens Mache, and Michael E Locasto. Cybersecurity education and assessment in edurange. *IEEE Annals of the History of Computing*, 15(03):90–95, 2017.