Building an IT Security Laboratory for Complex Teaching Scenarios Using ‘Infrastructure as Code’

Marcus Soll; Hendrik Helmken; Michel Belde; Sebastian Schimpfhauser; Felix Nguyen; Daniel Versick

2023 IEEE Global Engineering Education Conference (EDUCON), 2023

IEEE Xplore: https://ieeexplore.ieee.org/document/10125250

DOI: 10.1109/EDUCON54358.2023.10125250

Building an IT Security Laboratory for Complex Teaching Scenarios Using 'Infrastructure as Code'

Marcus Soll, Hendrik Helmken, Michel Belde, Sebastian Schimpfhauser, Felix Nguyen, Daniel Versick
NORDAKADEMIE gAG Hochschule der Wirtschaft
Elmshorn, Germany
{marcus.soll,daniel.versick}@nordakademie.de felix.nguyen@lhind.dlh.de

***Abstract*—There are increasing demands for IT security education which could be partly met by easier access to IT security laboratories. This paper proposes the use of 'Infrastructure as Code' (IaC) as a central building block for introducing dynamically adaptable teaching scenarios to laboratories in the context of IT security. The decision was made based on our didactical concept (which is built on Bloom's Taxonomy). The concept we propose is intended for use in a virtual laboratory, where the whole laboratory set-up is distributed over and contained within virtual machines. This way, we are able to build realistic, complex teaching scenarios. After comparing multiple IaC solutions, we decided to build our implementation on Terraform. The most important building blocks written in Terraform are presented. In addition, a user interface was created to meet demands of students and teachers. We will describe example teaching scenarios including one where students are tasked with gaining access to vulnerable data via a 2-step attack.**

***Index Terms*—Computer science education, educational technology, infrastructure as code, penetration testing, Terraform**

I. Introduction

If we look at the latest curriculum guidelines, the focus of IT security in university education is mainly on the theoretical side. In the guideline of the German *Gesellschaft für Informatik* [1], the usage of practical tools[1] is only considered for low contextualisation. Complex scenarios should only be analysed, which is often considered a more theoretical task. The guidelines of the ACM and IEEE for undergraduate degree programmes for computer science [2] only suggest analysing typical security problems, but do not suggest the usage of security related software tools, such as penetration tools or intrusion detection systems. The specialised guideline for cybersecurity curricula [3] mentions that the knowledge and usage of practical tools is beneficial, but the defined learning outcomes are written in a way that the preferred method of teaching (theoretical or practical) is not defined and therefore up to the teacher.

This situation is not optimal. Practical training in laboratories and hands-on experience can improve the overall domain-knowledge of students [4]. In addition, important learning outcomes can be trained such as instrumentation, experiment, learn from failure, creativity [5] or knowledge over industry context, working mindset or getting an overview over larger contexts [6]. Thus, giving the students laboratories for IT security to experiment in seems beneficial.

How can such a laboratory for IT security look like? In general, exercises in IT security can be table-top based (e.g. a discussion about a scenario) or full simulation (e.g. artificial systems like virtual computers and networks) [7]. For a full-fledged IT security laboratory, we imagine having a full simulation spanning at least one (if possible multiple) virtual systems where students can practise different exercises like analysing a computer for security vulnerabilities or analysing network traffic to identify attackers.

If we look a bit further, such a laboratory should also prepare for current challenges. One such challenge is operational security [8]: With the rise of *Industry 4.0* and *Internet of Things*, more and more operational technology is connected to networks thus open up new cyber security threads. One famous example of such a security thread is the Stuxnet virus, which was used around 2010 to sabotage physical production systems in the form of Iranian uranium centrifuges [9]. Future laboratories should thus be able to include exercises for operational security.

To strengthen the position of practical knowledge in IT security, we suggest to increase the availability of IT security laboratories at universities. These laboratories should be low cost and easily deployable, which would make it easier for universities to adopt them. Since cloud-based virtual laboratories can be easy to access globally and allow maximisation of resources [10], we want to explore this route by levering the advantages of modern *'Infrastructure as Code'* technology [11].

II. IT Security Labs in Teaching

Laboratories for IT security are not a new development. A good overview over the different types of IT security laboratories can be found in Topham et al. [10]. However, we can also look at more recent research on laboratories for IT security, where we can see multiple research paths: The creation of new laboratory types (e.g. personalised learning [12] or new domains like Android security [13]), embedding cybersecurity into different other programmes (e.g. social science [14], liberal arts [15] or even K-12 education [16]),

This research was part of the project *Flexibel kombinierbare Cross-Reality Labore in der Hochschullehre: zukunftsfähige Kompetenzentwicklung für ein Lernen und Arbeiten 4.0* (CrossLab), which is funded by the *Stiftung Innovation in der Hochschullehre*, Germany.

[1]German: *'typischen Angriffsmethoden und Werkzeuge benutzen'* [1, p. 27], the English translation would be close to *'usage of usual tools for penetration testing'*

and laboratory technology [17]–[19]. In our work, we want to follow the last-mentioned path.

There are already multiple (commercial) systems which allow to practise IT security in a virtualised environment, such as HackTheBox[2], TryHackMe[3] or Root-Me[4]. All of these services allow accesss to learning / practising scenarios containing one or multiple virtual machines while also containing gamification elements [20]. While we do not include gamification in our current laboratory systems, we want to provide similar learning scenarios to students. We believe that presenting the technology of our system enables other institutions to employ similar laboratories and thus increases the quality of IT security education. In future development, we want to include topics like operational security in our scenarios.

III. Didactical Concept

While designing the laboratory, one important aspect was the didactical concept behind it. Thereby, we can build a system which enables students to learn all learning outcomes designed in different courses for different degrees. We believe that this is important even before developing an actual course, since courses can only have a good didactical concept if the laboratory enables it. Therefore, we want to present the didactical concept we have in mind even when no full course is developed yet.

We follow the general trend in the Bologna Process in the European Union and describe the goals of our courses through learning outcomes / competencies [21]. According to Kennedy [21], a learning outcome is a description of what a student can achieve, which must be demonstrable by the students. Learning outcomes are based on Bloom's Taxonomy (see the revised version at [22]). Basically, you can sort learning activities from easy to hard in the following order [22]:

- Remember
- Understand
- Apply
- Analyse
- Evaluate
- Create

In our case, the first two levels (*remember* and *understand*) can also be addressed in a classical lecture. Thus, it is important that our laboratory system is able to address levels higher than *apply* of Bloom's Taxonomy. This has a few consequences for the design of our system:

1) **We need a system that allows us to build scenarios of different sizes, ranging from single machine set-ups to scenarios spanning multiple networks and machines.** This allows us to both build small scenarios for exercising with IT security tools (*analyse*) as well as building realistic scenarios for practising *evaluation* of scenarios as well as securing those scenarios (i.e. *create* in Bloom's Taxonomy).
2) **We want a system that students can experiment in without the fear of breaking something.** Some studies suggest that anxiety in laboratories might be a problem for students [23], [24]. By reducing anxiety, students can have the freedom to *analyse*, *evaluate* and *create* and thus have the opportunity to achieve those learning outcomes more easily.
3) **Both teaching and learning of all levels of Bloom's Taxonomy should be encouraged.** This enables students to gain deep knowledge of IT security.
4) **Support students with different learning speeds:** While students have different learning speeds, it is important that they master the required skills instead of going through assignments as quickly as possible [25]. It is therefore important that our laboratory system supports students of different learning speeds to achieve the desired learning outcomes.
5) **Support exercises for operational security.** As written in Sec. I, the security of operational technology is an important upcoming topic [8], thus new IT security laboratories should support them. While this is not a direct consequence of Bloom's Taxonomy, we still think it is important enough to be included in our requirements.

Thus, using IaC for an IT security laboratory seems to be a way to build laboratories with a sound didactical concept.

IV. Choosing an 'Infrastructure as Code' Solution

Since *'Infrastructure as Code'* (IaC) is an emerging trend in cloud-computing [11], we want to analyse whether IaC is feasible to be used for the construction of IT security laboratories. Infrastructure as Code is defined as a way of computer infrastructure automation which follows the practises of software development (i.e. the infrastructure definition is written as code, every change of code reflects a change in infrastructure) [26].

For our laboratory system, we want a system that

1) is not dependent on a single cloud provider
2) has a state management (calling the same configuration multiple times is idempotent)
3) uses a declarative configuration language (that can be added to a source code repository)
4) is available as open source

There are multiple solutions we can use as an IaC provider. Based on the popularity taken from [27], if we want at least a 10% usage we have 9 solutions. The results of our comparison can be seen in Tab. I. The comparison shows that three systems fulfil our requirements: Terraform, Chef and Puppet. Out of those three, we chose Terraform [28] as the foundation of our laboratory. Since we use Microsoft Azure[5] for different parts of the infrastructure of our university, we decided to use Microsoft Azure as the cloud provider for our laboratory.

[2]https://www.hackthebox.com/
[3]https://tryhackme.com/
[4]https://www.root-me.org/
[5]https://azure.microsoft.com/

TABLE I
COMPARISON OF POPULAR IAC SOLUTIONS. THE SOLUTIONS ARE COMPARED WHETHER THEY 1) ARE NOT DEPENDENT ON A SINGLE CLOUD PROVIDER; 2) HAVE A STATE MANAGEMENT; 3) USE A DECLARATIVE CONFIGURATION LANGUAGE; 4) ARE AVAILABLE AS OPEN SOURCE. PROVIDERS ARE BASED ON [27] WITH AT LEAST 10% USAGE.

IoC-Solution	1)	2)	3)	4)
AWS CloudFormation templates [29]			✓	
Azure resource manager templates [30]		✓	✓	
Terraform [28]	✓	✓	✓	✓
Google cloud deployment manager templates[6]			✓	(✓)
Ansible [31]	✓			✓
AWS systems manager [7]				
AWS OpsWorks [29]	uses Chef / Puppet			
Chef [32]	✓	(✓)	(✓)	✓
Puppet [33]	✓	✓	✓	✓

Now an IaC solution is chosen, our didactical requirements (see Sec. III) can be addressed as following:

1) **We need a system that allows us to build scenarios of different sizes, ranging from single machine set-ups to scenarios spanning multiple networks and machines.** Using IaC together with different cloud providers, we are able to scale the scenarios to any size without having to take hardware limits into consideration.
2) **We want a system that students can experiment in without the fear of breaking something.** Deploying learning scenarios in the cloud not only decouples them from any local infrastructure (thus negating the risk to the university), but most cloud providers allow restricting network access to deployed machines (thus isolating the scenario from the internet except a well-guarded gateway). In the event a student actually breaks a scenario, IaC can then deploy a completely fresh instance of that scenario with almost no work.
3) **Both teaching and learning of all levels of Bloom's Taxonomy should be encouraged.** While this is independent of whether we use IaC, it has the consequence that we actually need some sort of learning / teaching environment which motivates both students to learn as well as teachers to use the system.
4) **Support students with different learning speeds:** Using IaC for students does not only allow students to work through scenarios multiple times, it is easy to set-up laboratories for different target skills so slower learners can go through multiple scenarios if they wish. In theory, it is also possibly to dynamically generate learning scenarios according to students learning speeds and needs.
5) **Support exercises for operational security.** IaC can support us here by allowing for automated set-ups where such a connection is possible (e.g. automatically install needed drivers and software to a virtual computer). However, IaC alone is not enough here and future work is needed to actually connect operational technology (physical or virtual) to the laboratory.

[6]https://cloud.google.com/deployment-manager/v2beta1/templates
[7]https://aws.amazon.com/de/systems-manager/

V. USING 'INFRASTRUCTURE AS CODE'

In our laboratory, IaC allows us to model complex teaching scenarios. Experts (not necessarily teachers) have to manually design the teaching scenarios and write them down using the domain-specific language *HashiCorp Configuration Language* (HCL). The written scenarios are then added to the library of our software tool.

Once our software tool knows a scenario, we can automatically deploy that scenario to a cloud provider (in our case Azure). This means that our software tool can create and destroy scenarios based on demand without the need of any manual set-up of the teacher. The laboratories usually take a few minutes to start.

In those scenarios, the students can experiment freely without having the fear of breaking equipment or otherwise getting in trouble. Worst case, the current scenario gets deleted and the student gets access to a new instance of the same scenario. In addition, we can provide students with all necessary tools preinstalled on a virtual computer in the scenario, so no additional set-up by the students are needed.

This model also allows us to save costs: Instead of needing to buy and maintain equipment for enough students, which is most likely only needed a couple of times per year, we can deploy and pay on an 'as needed' basis.

A. Building Scenarios in Terraform

All scenarios are built as Terraform modules. This way, a single scenario definition can be used to deploy multiple instances. At the same time, new scenarios can be added by simply adding a new module to the main Terraform file (see Fig. 8 at the end of the paper). For this to work, a unified interface has to be present across all modules.

All modules must have three input variables. The name and the meaning of these variables is the same across all modules. These variables are:

- *"id"*: The unique id of one instance of the scenario. This *id* must be used in every resource name (e.g. use `"scenario1-$var.id-public-ip"` instead of `"scenario1-public-ip"`. Failing to comply with the naming convention might lead to name clashes, which has the consequence of different scenarios interfering with each other.
- *"azure_region"* and *"azure_rg"*: These variables are needed for modules to know which data centre they should put their resources to. While these can theoretically be set different for different modules, it still makes sense to choose the best ones for you and keep them consistent across all modules.

At the same time, we need some output variables in order to provide students with all required information to access the laboratory. These are the public IP, the user name as well as the password to connect to the machine. With this interface in place, we can now build a complete module.

An example scenario can look as follows: The input variables can be defined in a file **variables.tf** (see Fig. 6 at the

end of the paper) and the output variables in a file *output.tf* (see Fig. 7 at the end of the paper). With that in place, we can now write our actual scenario (see Fig. 9 at the end of the paper):

1) We first define a local variable `name` to save a more elaborate name including the scenario name (here: *example-lab*). While this is not strictly necessary, it helps with later identification in Terraform.
2) We can then set up the network. This needs a *virtual network* where we put all our machines in, at least one *subnetwork* (can be more for larger scenarios), a *public IP* which can be used by students to access the scenario and a *network interface* for every distinct machine network configuration (e.g. when a machine is on a different subnet configuration, a different *network interface* is needed).
3) We can use a *random password* Terraform resource to generate the passwords students use to access the scenario.
4) Finally, we set up the virtual machines. In this example scenario, we only use one standard Ubuntu virtual machine. This machine's network access is determined by the *network interface* we assign to it.

Unfortunately, Terraform needs all modules to be joined in a single root file (see Fig. 8 at the end of the paper). Besides some set-up, this file contains a *module* resource as well as an *output variable* for every scenario we define. We now want to instantiate laboratories by just providing IDs for laboratories to Terraform. To do this, we use different Terraform functions. `for_each` and `toset` are used to map the provided IDs to the modules to create multiple instances of those laboratories. `tomap` is used to collect the output variables of those instances. By doing it this way, we just need to provide the state of laboratories we want to have at the end and do not need to track which laboratories need to be created or destroyed (which will be managed by Terraform itself).

B. Creating Images for Virtual Machines

One of the challenges when creating scenarios is the set-up of images for virtual machines, especially because many virtual machines need manual set-up. We identified four possible ways of using images in scenarios:

1) Use a pre-existing image. This is only possible when no special set-up is needed, such as using Kali Linux[8] as the gateway machine.
2) Use a preexisting image and add a custom set-up script. The set-up script can be used by adding an `azurerm_virtual_machine_extension` ressource containing the script (see Fig. 1 for an example). This method is useful if only a few changes have to be performed on a base image and the new image is only used once.
3) Use the Azure Image Builder[9]. This method is best used when a single custom image is used multiple times (possibly even over multiple scenarios). It can be combined with 2) for increased flexibility.
4) Manually create the image and upload it to the *Azure Compute Gallery*. This is the most manual work, but it also allows the best way to customise the image. This is needed if a very special set-up is needed (e.g. when a specific software version with known vulnerabilities is needed or when software needs to be configured in a vulnerable way). Azure needs the image to be in the *Virtual Hard Disk (.vhd)* format.

```
resource "azurerm_virtual_machine_extension" "vmext" {
  name = "example-vmext"

  virtual_machine_id   = azurerm_linux_virtual_machine.example.id
  publisher            = "Microsoft.Azure.Extensions"
  type                 = "CustomScript"
  type_handler_version = "2.0"

  protected_settings = <<PROT
  {
    "script": "${base64encode(file("set-up_script.sh"))}"
  }
PROT
}
```

Fig. 1. Terraform code snippet to use a custom set-up script for a virtual machine.

VI. Students' & Teachers' Interface

To access the deployed learning scenarios, standard tools for remote connections like SSH or RDP can be used. While this provides rudimentary access, we aimed to have a system that encourages learning and teaching instead of simply allowing it (as discussed in Sec. III). We identified four major requirements for our system:

A. Students should be guided through the learning scenario when possible. Help text and self assessment questions should be easily available at any point.
B. Feedback to the students should be given quickly. This is important to us since no matter which learning theory is used, timely feedback seems to be important [34].
C. The students should be able to access the learning scenarios independently of their location or end device. This allows for greater flexibility by students.
D. Teachers should be able to track the progress of students. This should help teachers to identify struggling students and help them accordingly.

To fulfil all requirements, we decided to build a web application. Using a web application allows students to use the system independently of their location, device, or installed software (requirement C.). To access the laboratory, students just need a URL provided by the teachers[10].

The students' interface is split up into two parts (see Fig. 2 for the concept and 3 for an implementation): On the left

[8]you can find this image at the Azure Marketplace at https://azuremarketplace.microsoft.com/en/marketplace/apps/kali-linux.kali

[9]https://learn.microsoft.com/en-us/azure/virtual-machines/image-builder-overview

[10]The URL might be unique to each student or students might use the same URL to access the same instance of the scenario

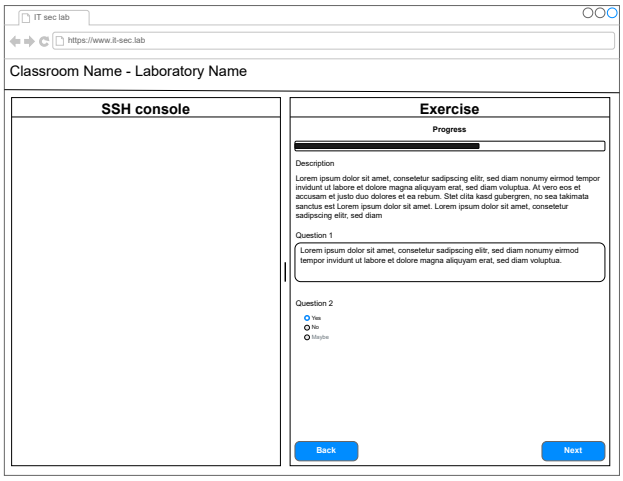

Fig. 2. Mock-up of the students interface. The interface is divided into virtual machine view (left) and task view (right).

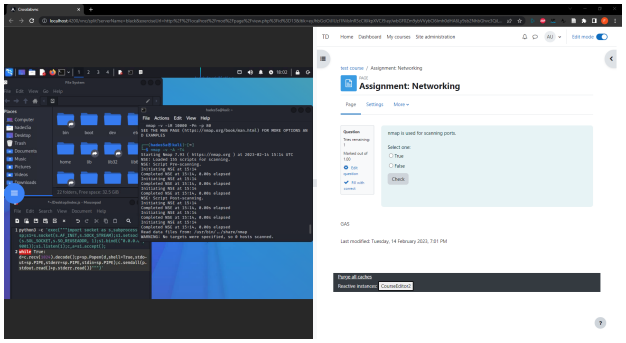

Fig. 3. Students interface. The interface is divided into virtual machine view using a VNC connection (left) and task view using a Moodle quiz (right).

side, students can access the learning scenario through the gateway virtual machine. For now, access is granted through SSH[11] or VNC[12]. On the right side, teachers can provide both instructions as well as self assessment questions to students (requirement A.). These questions can also be used to provide quick feedback whether tasks have been completed correctly (requirement B.).

A teacher has two ways of tracking user progress: He can either use standard tests of the learning platform (in our case Moodle) or use a web portal (see Fig. 4) we developed. Both solutions track the students progress based on answered self assessment questions (requirement D.). In addition, both solutions allow teachers to spot students who might have difficulties and are thus behind. This way, teachers can help those students and give appropriate feedback (requirement B.).

VII. Example Scenario

In this example scenario, students have to perform a penetration test [35] on a company network. This network has a single public server. Students are tasked with accessing

[11]using webssh: https://github.com/huashengdun/webssh
[12]using noVNC: https://github.com/novnc/noVNC

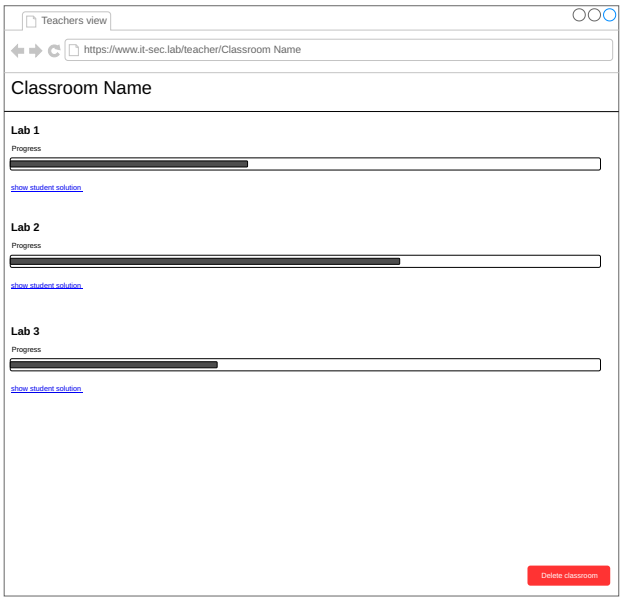

Fig. 4. Mock-up of the teachers' view. The progress of all students/laboratories can be monitored.

secret company data on a server in the private network of the company. A brief overview over the scenario can be seen in Fig. 5.

In a first step, students must analyse the public server. By doing a port scan [36], they find the two open ports 22 (SSH server) and 8080 (HTTP server). They can access port 8080 and get a normal website. Using an SQL injection [37], they get access to the usernames of all employees. The usernames of the employees can then be used to brute force [36] attack to get access to the SSH server. Now students have access to the internal network.

In the second step, students must now scan the private network. There, they find a second server. Using a port scan on the newly discovered machine, they find an active Windows file sharing / smb server (port 445). Using an old vulnerability like *EternalBlue* [38], they get access to the server and with it access to the secret data. This way, they can finish the scenario successfully.

We can now evaluate the learning outcomes students can achieve in this scenario by using Bloom's Taxonomy. (see Sec. III):

- *Understand*: Knowledge of the danger of exposing websites to the internet.
- *Understand*: Knowledge of the danger of SQL injections and how to test for them.
- *Understand*: Knowledge of the danger of weak authentification methods (i.e. weak password) for SSH.
- *Understand*: Knowledge of how to access private networks.
- *Understand*: Knowledge of the danger of not installing patches in a timely manner.
- *Apply*: Usage of the different tools involved (e.g. `nmap` [39] for port scanning, metasploit [40] for EternalBlue

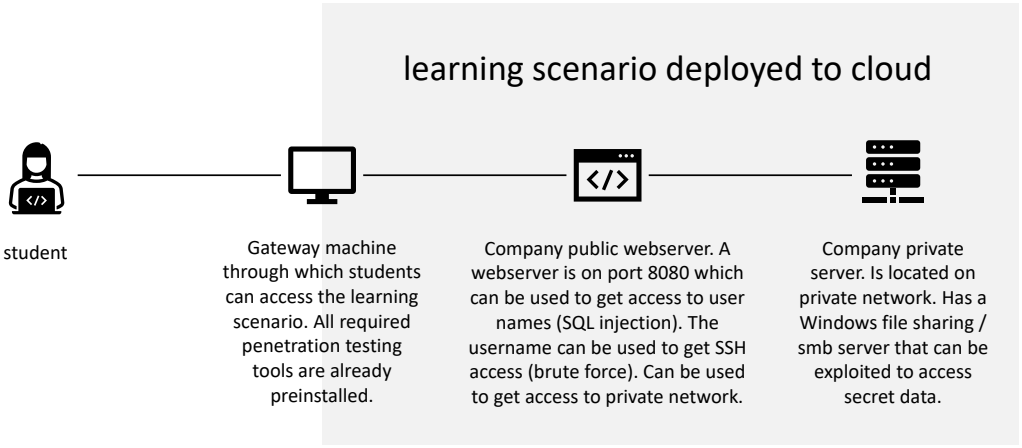

Fig. 5. Example scenario. Students have to gain access to a public server of a company. Using this server, they can access secret data on a server located in a private network.

exploitation).

Depending on how the teacher uses the scenario in his teaching, further learning outcomes higher on Bloom's Taxonomy can be achieved, e.g.:

- *Analyse*: The students can analyse where the problems of the scenarios are and how to avoid them.
- *Evaluate*: The students can give an evaluation of the overall security of the scenario.
- *Create*: Improvements of the presented systems in the scenario can be discussed. It is also possible that students should actually make the scenario more secure by fixing the identified problems, although this might be harder and more time consuming.

VIII. Conclusion and Outlook

This work mainly focuses on the technical and didactical part of our new IT security laboratory. Based on Bloom's Taxonomy as our didactical concept, we were able to show that *'Infrastructure as Code'* (IaC) seems like a good candidate to build IT security laboratory. After comparing multiple IaC solutions, we settled for Terraform to build our laboratory. We show the technology behind our laboratory, our user interface as well as an example scenario involving multiple virtual machines.

The laboratory is fully operational at the moment. It is planned to use the laboratory starting at the beginning of 2023. Accompanying the students at learning, we want to make studies on how effective the new laboratory is for learning and how complex scenarios can be used to improve IT security teaching. Long term, we hope that we can make the laboratory available to a wider audience.

Since operational security is an upcoming topic, we would like to include scenarios containing different physical devices. However, we have not yet explored how to add them into our Terraform modules. This should be possible by providing teaching scenarios containing virtual machines which are set-up to connect to physical systems. We plan to include them by connecting our laboratory into the CrossLab [41] infrastructure in future research, which would allow us to connect to different physical devices using a standardised protocol.

An other open topic is the creation of individual, automatically generated learning scenarios. Such scenarios would allow us to better adress the needs of individual students and thus help them improve their knowledge about IT security. While IaC has the capability to realise this, it is not yet implemented into our system and might be an interesting topic for future research.

References

[1] Gesellschaft für Informatik e.V., "Empfehlungen für Bachelor- und Masterprogramme im Studienfach Informatik an Hochschulen (Juli 2016)," 2016.

[2] The Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM), and IEEE Computer Society, *Computer Science Curricula 2013 - Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*, New York, NY, USA, 2013.

[3] Joint Task Force on Cybersecurity Education, *Cybersecurity Curricula 2017: Curriculum Guidelines for Post-Secondary Degree Programs in Cybersecurity*. New York, NY, USA: Association for Computing Machinery, 2017.

[4] F. L. Forcino, "The Importance of a Laboratory Section on Student Learning Outcomes in a University Introductory Earth Science Course," *Journal of Geoscience Education*, vol. 61, no. 2, p. 213–221, 2013. [Online]. Available: https://doi.org/10.5408/12-412.1

[5] L. D. Feisel and A. J. Rosa, "The Role of the Laboratory in Undergraduate Engineering Education," *Journal of Engineering Education*, vol. 94, no. 1, p. 121–130, 2005. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/j.2168-9830.2005.tb00833.x

[6] M. Soll and K. Boettcher, "Expected learning outcomes by industry for laboratories at universities," in *2022 IEEE German Education Conference (GeCon)*, 2022, pp. 1–6.

[7] R. S. Dewar, "Cybersecurity and cyberdefense exercises," ETH Zurich, Tech. Rep., 2018. [Online]. Available: http://hdl.handle.net/20.500.11850/314593

[8] B. C. Ervural and B. Ervural, *Overview of Cyber Security in the Industry 4.0 Era*. Cham: Springer International Publishing, 2018, p. 267–284. [Online]. Available: https://doi.org/10.1007/978-3-319-57870-5_16

[9] M. Baezner and P. Robin, "Stuxnet," ETH Zurich, Tech. Rep., 2017. [Online]. Available: http://hdl.handle.net/20.500.11850/200661

[10] L. Topham, K. Kifayat, Y. Younis, Q. Shi, and B. Askwith, "Cyber Security Teaching and Learning Laboratories: A Survey," *Information & Security: An International Journal*, vol. 35, no. 1, pp. 51–80, 2016.

[11] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba, "Adoption, support, and challenges of infrastructure-as-code: Insights from industry," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2019, pp. 580–589.

[12] Y. Deng, D. Huang, and C.-J. Chung, "Thoth lab: A personalized learning framework for cs hands-on projects (abstract only)," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, ser. SIGCSE '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 706. [Online]. Available: https://doi.org/10.1145/3017680.3022442

[13] J.-F. Lalande, V. Viet Triem Tong, P. Graux, G. Hiet, W. Mazurczyk, H. Chaoui, and P. Berthomé, "Teaching android mobile security," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 232–238. [Online]. Available: https://doi.org/10.1145/3287324.3287406

[14] C. M. B. Turner and C. F. Turner, "Analyzing the impact of experiential pedagogy in teaching socio-cybersecurity: Cybersecurity across the curriculum," *Journal of Computing Sciences in Colleges*, vol. 34, no. 5, p. 12–22, apr 2019.

[15] X. Mountrouidou, "Cyberpaths," *Journal of Computing Sciences in Colleges*, vol. 34, no. 3, p. 16, jan 2019.

[16] A. Konak, "Experiential learning builds cybersecurity self-efficacy in k-12 students," *Journal of Cybersecurity Education, Research and Practice*, vol. 2018, no. 1, 2018. [Online]. Available: https://digitalcommons.kennesaw.edu/jcerp/vol2018/iss1/6

[17] M. Knöchel, S. Karius, and S. Wefel, "Developing a web-based training platform for it security education," in *DELFI 2021*, A. Kienle, A. Harrer, J. M. Haake, and A. Lingnau, Eds. Bonn: Gesellschaft für Informatik e.V., 2021, pp. 223–228.

[18] M. F. Thompson and C. E. Irvine, "Individualizing cybersecurity lab exercises with labtainers," *IEEE Security & Privacy*, vol. 16, no. 2, pp. 91–95, 2018.

[19] N. Eliot, D. Kendall, and M. Brockway, "A flexible laboratory environment supporting honeypot deployment for teaching real-world cybersecurity skills," *IEEE Access*, vol. 6, pp. 34 884–34 895, 2018.

[20] K. M. Kapp, *The Gamification of Learning and Instruction: Game-Based Methods and Strategies for Training and Education*, 1st ed. Pfeiffer & Company, 2012.

[21] D. Kennedy, *Writing and Using Learning Outcomes: A Practical Guide*. Quality Promotion Unit, University College Cork, 2007.

[22] D. R. Krathwohl, "A revision of bloom's taxonomy: An overview," *Theory Into Practice*, vol. 41, no. 4, pp. 212–218, 2002. [Online]. Available: https://doi.org/10.1207/s15430421tip4104_2

[23] E. Kaya and A. Yıldırım, "Science anxiety among failing students," *Elementary Education Online*, p. 518–525, 2014.

[24] E. Ural, "The effect of guided-inquiry laboratory experiments on science education students' chemistry laboratory attitudes, anxiety and achievement," *Journal of Education and Training Studies*, vol. 4, no. 4, p. 217–227, 2016. [Online]. Available: https://eric.ed.gov/?id=EJ1095156

[25] P. Rouhani, "The role of time in self-directed personalized learning environments: An exploratory analysis," Ph.D. dissertation, 2019. [Online]. Available: https://dash.harvard.edu/handle/1/42081520

[26] K. Morris, *Infrastructure as Code*, 2nd ed. Sebastopol, CA: O'Reilly Media, Inc., 2020.

[27] Scribd and F. Software, "Usage of cloud configuration tools worldwide in 2022, current and planned," *Statista*, 2022. [Online]. Available: https://www.statista.com/statistics/511293/worldwide-survey-cloud-devops-tools

[28] Y. Brikman, *Terraform: Up & Running*, 2nd ed. O'Reilly Media, Inc., 2019.

[29] Amazon Web Services, Inc., "Overview of deployment options on aws," Tech. Rep., 2022, last Accessed:. 24.11.2022 15:07. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/overview-deployment-options/welcome.html

[30] D. Rendón, *Building Applications with Azure Resource Manager (ARM): Leverage IaC to Vastly Improve the Life Cycle of Your Applications*. Berkeley, CA: Apress, 2022. [Online]. Available: https://link.springer.com/10.1007/978-1-4842-7747-8

[31] M. Heap, *Ansible*. Berkeley, CA: Apress, 2016. [Online]. Available: http://link.springer.com/10.1007/978-1-4842-1659-0

[32] M. Marschall, *Chef Infrastructure Automation Cookbook*. Packt Publishing, 2013.

[33] S. Pandya and R. Guha Thakurta, *Hands-on Infrastructure as Code with Puppet*. Berkeley, CA: Apress, 2022, pp. 135–163. [Online]. Available: https://doi.org/10.1007/978-1-4842-8689-0_7

[34] M. Thurlings, M. Vermeulen, T. Bastiaens, and S. Stijnen, "Understanding feedback: A learning theory perspective," *Educational Research Review*, vol. 9, pp. 1–15, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1747938X12000656

[35] M. Bishop, "About Penetration Testing," *IEEE Security Privacy*, vol. 5, no. 6, p. 84–87, 2007.

[36] S. Shirey, "Internet security glossary," RFC 2828, 2000. [Online]. Available: https://www.ietf.org/rfc/rfc2828.txt

[37] A. Sadeghian, M. Zamani, and S. M. Abdullah, "A taxonomy of sql injection attacks," in *2013 International Conference on Informatics and Creative Multimedia*, 2013, pp. 269–273.

[38] S. B. Wicker, "The ethics of zero-day exploits—: The nsa meets the trolley car," *Commun. ACM*, vol. 64, no. 1, p. 97–103, 2020. [Online]. Available: https://doi.org/10.1145/3393670

[39] G. Lyon, *Nmap Network Scanning - The Official Nmap Project Guide to Network Discovery and Security Scanning*. Nmap Software LLC, 2022. [Online]. Available: https://nmap.org/book

[40] S. Raj and N. K. Walia, "A study on metasploit framework: A pen-testing tool," in *2020 International Conference on Computational Performance Evaluation (ComPE)*, 2020, pp. 296–302.

[41] I. Aubel, S. Zug, A. Dietrich, J. Nau, K. Henke, P. Helbing, D. Streitferdt, C. Terkowsky, K. Boettcher, T. R. Ortelt, M. Schade, N. Kockmann, T. Haertel, U. Wilkesmann, M. Finck, J. Haase, F. Herrmann, L. Kobras, B. Meussen, M. Soll, and D. Versick, "Adaptable digital labs - motivation and vision of the crosslab project," in *2022 IEEE German Education Conference (GeCon)*. Berlin, Germany: IEEE, Aug 2022.

APPENDIX
EXAMPLE TERRAFORM FILES FOR A BASIC SCENARIO CONTAINING A SINGLE UBUNTU VM

```
variable "id" {
  type = string
}

variable "azure_region" {
  type = string
}

variable "azure_rg" {
  type = string
}
```

Fig. 6. Input variables of each scenario. Saved as file *modules/example-scenario/variables.tf*.

```
output "vm_public_ip" {
  value = azurerm_linux_virtual_machine.example.public_ip_address
}

output "vm_user" {
  value = azurerm_linux_virtual_machine.example.admin_username
}

output "vm_password" {
  value = azurerm_linux_virtual_machine.example.admin_password
}
```

Fig. 7. Output variables of each scenario. Here, values are taken from the Azure-VM with the identifier 'example'. Saved as file *modules/example-scenario/output.tf*.

```
terraform {
  required_version = ">= 1.2.5"

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "3.24.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "3.4.3"
    }
  }

  backend "http" {}
}

provider "azurerm" {
  features {}
}

resource "azurerm_resource_group" "scenario_rg" {
  name     = var.ressource_group
  location = var.azure_region
}

module "example_scenario" {
  source = "./modules/example-scenario"

  for_each = toset(var.example_scenario_ids)

  id           = each.key
  azure_rg     = azurerm_resource_group.scenario_rg.name
  azure_region = var.azure_region
}

output "example_scenario_public_ip" {
  value = tomap({
    for key, service in module.example_scenario : key => service.vm_public_ip
  })
}

output "example_scenario_username" {
  value = tomap({
    for key, service in module.example_scenario : key => service.vm_user
  })
}

output "example_scenario_password" {
  value = tomap({
    for key, service in module.example_scenario : key => service.vm_password
  })
  sensitive = true
}
```

Fig. 8. Main Terraform file containing all modules. Saved as *main.tf*.

```
locals {
  name = "example-lab-${var.id}"
}

resource "azurerm_virtual_network" "network" {
  name                = "${local.name}-network"
  location            = var.azure_region
  resource_group_name = var.azure_rg
  address_space       = ["10.0.0.0/16"]

  tags = {
    exampleLabId = local.name
  }
}

resource "azurerm_subnet" "subnetwork" {
  name                 = "${local.name}-subnet"
  resource_group_name  = var.azure_rg
  virtual_network_name = azurerm_virtual_network.network.name
  address_prefixes     = ["10.0.0.0/24"]
}

resource "azurerm_public_ip" "public_ip" {
  name                    = "${local.name}-public-ip"
  location                = var.azure_region
  resource_group_name     = var.azure_rg
  allocation_method       = "Dynamic"
  idle_timeout_in_minutes = 4

  tags = {
    exampleLabId = local.name
  }
}

resource "azurerm_network_interface" "nic" {
  name                = "${local.name}-nic"
  location            = var.azure_region
  resource_group_name = var.azure_rg

  ip_configuration {
    name                          = "${local.name}-ipconfig"
    subnet_id                     = azurerm_subnet.subnetwork.id
    public_ip_address_id          = azurerm_public_ip.public_ip.id
    private_ip_address_allocation = "Dynamic"
  }

  tags = {
    exampleLabId = local.name
  }
}

resource "random_password" "password" {
  length      = 16
  special     = false
  min_lower   = 1
  min_numeric = 1
  min_upper   = 1
}

resource "azurerm_linux_virtual_machine" "example" {
  name                            = "${local.name}-vm"
  resource_group_name             = var.azure_rg
  location                        = var.azure_region
  size                            = "Standard_B1ls"
  admin_username                  = "adminuser"
  admin_password                  = random_password.password.result
  disable_password_authentication = false
  network_interface_ids           = [
    azurerm_network_interface.nic.id,
  ]

  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
    name                 = "${local.name}-vm-disk"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-jammy"
    sku       = "22_04-lts-gen2"
    version   = "latest"
  }

  tags = {
    exampleLabId = local.name
  }
}
```

Fig. 9. Example scenario containing a single Ubuntu VM. Saved as file *modules/example-scenario/main.tf*.