Labtainers: A Framework for Parameterized Cybersecurity Labs Using Containers

Cynthia E. Irvine
Naval Postgraduate School
irvine@nps.edu

Michael F. Thompson
Naval Postgraduate School
mfthomps@nps.edu

Jean Khosalim
Naval Postgraduate School
jkhosali@nps.edu

ABSTRACT

We have created a framework to simplify creation, deployment, and assessment of stand-alone cyber security lab exercises, intended for use on individual student computers. We are implementing this framework using Linux Docker containers. Each lab has one or more associated containers that ensure an execution environment consistent with the requirements of the software elements and activities within the lab. Lab-specific containers are automatically installed and configured on the student's Linux computing platform, (e.g., a VM) when the student starts the lab. Results of student lab activity are automatically collected and packaged when the student completes the lab, and these results are automatically evaluated on an instructor's computer, using similar Docker containers. Automated assessment of student labs makes it practical, (from an instructor's perspective), to individualize every instance of each lab such that students cannot easily submit results either created by another student or mined from the Internet.

KEYWORDS

Computer security education, laboratory exercises, containers, parameterization

1 INTRODUCTION

Laboratory exercises provide students with experiential learning opportunities that reënforce concepts presented in readings and lectures. In addition, they allow students to learn how many tools and techniques can be used to solve problems. In the context of cybersecurity, laboratory exercises must cover a wide range of topics and need to be organized so that their learning outcomes progress from fundamental concepts to in-depth understanding.

Despite the acknowledged value of experiential learning [9], cybersecurity educators often face a number of challenges when attempting to provide worthwhile laboratory exercises for classes. First, they may lack the institutional infrastructure required to host complex lab exercises that may involve networked systems. Second, they may not have the time or expertise required to develop high quality exercises. Just as educators often rely upon good textbooks that contain well-vetted homework problems, and that may be accompanied by slides and other materials, so can over-worked instructors benefit from materials that will help them provide useful laboratory activities. Third, operating system and application configuration is a significant source of frustration and distraction for students and instructors when working with computer science labs. When students are required to use their personal systems, the consequent platform diversity can force instructors to manage problems associated with exercise setup and potentially diverse results caused by platform differences. Platform diversity can also negatively impact students, who must devote a disproportionate amount of time to lab set up, relative to doing the lab exercise and achieving the intended learning objectives. Fourth, often the best learning takes place when students are engaged in exploratory exercises. Unfortunately, exploratory exercises, by their very nature, can involve many blind alleys and detours. Understanding what each student has done, where that student may have had difficulty and whether the lab was successfully completed can become very laborious for instructors. Last, students may share or reuse exercise results. The undesirable consequences of this behavior include: implicit punishment of diligent students who complete the exercises themselves; student-cheaters who do not learn the material yet receive undeserved high marks; and instructors are unable to identify and assist students having difficulty with the material. Individualized exercises could solve the problem; however, if each student is given an different exercise, the instructor is faced with a huge grading task. Furthermore, it is possible that the individualized exercises will vary in difficulty, making the exercise unfair.

To address these problems, we have developed *Labtainers*.

The Labtainers framework involves three roles:

Lab Designer This person is an expert on the material to be presented in the lab exercise and serves as the pedagogical authority with respect to the intended learning objectives. The lab designer creates the laboratory exercise. The lab designer might have no interaction with the students, although it is likely that the lab designer will work with instructors, or may be the instructor. This interaction will help ensure that the intended learning objectives of the exercise are addressed. The lab designer can easily modify existing labs to fine tune the learning experience or to update the labs to incorporate new tools and technologies The designer might also specify requirements for, or even create, tools to assist instructor assessment of students' deliverables.

Instructor The instructor interacts with the students and is responsible for ensuring that learning objectives are met. An instructor might also work with a lab designer to specify the objectives for new labs. The instructor must ensure that students have the prerequisite knowledge required to complete each exercise successfully, and must assess student performance on the exercise.

Student Students perform laboratory exercises and deliver their results to instructors. The intent is for students to have learned something as a result of doing an exercise.

This work was supported by NSF grant DUE-1140938. The views expressed in this material are those of the authors and do not reflect the official policy or position of the National Science Foundation, Department of Defense or the U.S. Government.

The overarching goal of the Labtainers project is to develop a framework for cybersecurity laboratory exercises that is:

Consistent and Fair Laboratory exercises will be delivered to students in a preconfigured environment. Different labs can offer different environments. Both students and instructors will be able to to generate consistent, reproducible results. Each lab environment will be homogeneous across the entire instructor and student population, despite likely heterogeneity across the underlying platforms. One exception is that per platform performance may vary. Students will not have to struggle to create lab environments, and, instead, can concentrate on the lab. Instructors can assess student work without having to contend with differences resulting from the way the labs were set up.

Parameterizable Each student will have a laboratory exercise that cannot be accomplished by simply copying the results of another student. Parameterization choices can ensure that all students have labs of the same difficulty. Lab designers will be able to parameterize the generation of expected results on a per student basis, thus streamlining the assessment process. Labtainers are not an absolute guarantee that students will not engage a surrogate to do the lab or that a student might outwit the parameterization mechanism for a particular exercise, but the work factor required to cheat is intended to disincentivize cheating.

Support Automatic Assessment The Labtainer framework is designed so that evidence can be collected to demonstrate that students have achieved intended intermediate and final goals of exercises. Information can be collected so that the instructor can determine where students become confused or go off track. The framework supports the use of assessment tools by storing student evidence in a format that can be easily digested by other applications. These tools may be generic, or may be created by the lab designer to address specific aspects of an exercise. Through the development of machine learning and other emerging analytical methods, it may be possible to create good metrics of exercise and experiential learning efficacy.

The remainder of this paper consists of three sections. In the next section of this paper, related prior work and an overview of containers, our enabling technology, are presented. The student workflow supported by the Labtainer framework, lab exercise definition, parameterization, and support for student assessment are discussed in Section 3. This is followed by a summary and discussion of future work.

2 BACKGROUND AND RELATED WORK

This section reviews relevant preëxisting work and then provides an overview of the container technology that enables Labtainers.

2.1 Existing Cybersecurity Lab Frameworks

Security labs often require extensive infrastructure and technical support to undertake educational activities such as capture-the-flag challenges and class-focused lab exercises. Because many programs are homed at institutions lacking sufficient resources for such facilities, several environments have been created to address this need.

Several alternatives are available to instructors who wish to offer cybersecurity labs: hands on experience involving physical machines, virtual machines hosted on an infrastructure-as-a-service (Iaas) platform, virtual machines hosted on each student's laptop, and containers executing either on the the student's Linux host or in a Linux virtual machine hosted on the student's system.

The use of virtual machines to teach computer science classes has been explored for over 30 years, e.g. [5]; however, low cost virtualization technology capable of easily supporting laboratory exercises has only been available in the past ten years, e.g. Hay et al. suggested the use of virtual machines to support security labs [11]. Relative to virtual machines, containers have a number of advantages.

On demand cloud computing resources, such as Amazon Web Services (AWS) [2] require special permission to run many simple network security exercises, such as port scans and penetration testing. [1]

Construction of an institutionally-owned and operated virtual machine farm is likely to require considerable initial hardware investment and technical expertise, as well as an ongoing operational tail for maintenance, user management, continuity of operations, and backups. Even institutional use of a proprietary system for managing VMs, such as vSphere [18], requires expertise, whereas less costly open source options, such as KVM [17], require even greater levels of expertise. If students are required to host a number of VM images on their personal computers or laptops, the resource requirements could quickly exceed the resources available on the host. In contrast, Linux containers [24] offer a less costly and less complex alternative that affords lab designers and instructors greater control, while not tethering students to a server farm.

Where the physical component cannot be virtualized, the solution may involve some combination of elements, both physical and virtual networked together.

DeterLab [13] makes available 24 homework exercises intended to be run on the DETER testbed. Students access exercises via the web, as do instructors. Providing a similar facility, RAVE (Remote Access Virtual Environment) [14, 22] provides multiple infrastructure centers and sharable lab exercises. A challenge for instructors is the level of sophistication required to construct, execute, and maintain exercises. EDURange [20] originated to offer a facility for competitive, interactive exercises. To provide a combination of flexibility and ease of use for instructors, it uses Amazon's AWS to support virtual machines and inter-VM networking. The Tele-Lab project [23] supports web-based access to virtual machines that can support individualized laboratory exercises and assessments. All of these enabling infrastructures require students to be connected to the infrastructure platform hosting the virtual machines. In contrast, Labtainers do not tether students to a server and allow self-pace and intermittent activity. In addition, containers allow a more constrained and tailored lab environment.

[1] https://aws.amazon.com/premiumsupport/knowledge-center/penetration-testing/

Although there are many descriptions of laboratory exercises used to support teaching cybersecurity, e.g. [21], fewer complete sets of materials are available.

The SEED project [6–8] is an initiative to develop an instructional environment and a collection of hands-on materials for computer security education. The SEED project currently offers 33 labs divided into three categories: vulnerability and attack, design and implementation, and exploration. All lab materials are freely available under the GNU Free Documentation License, allowing instructors and textbook authors to integrate and adapt lab materials into course curricula. The SEED project openly shares usage statistics and continues to receive thousands of lab downloads per month, suggesting that the materials continue to be used at a number of universities.

Wang has developed a set of lab exercises for IT security [19]. These lab exercises are broken down into the following taxonomy: Computer Security Labs, Network Security Labs, Cryptography Labs, Application Security Labs.

Parameterization of security labs was incorporated into Tele-Lab [23]. Parameters are predefined and stored in a parameter database. In Labtainers, parameterization is achieved by using metadata associated with the student and does not require a database. To create individualized labs, the PolyLab project developed techniques to provide randomization of exercises using hashes. [10] Exercises in steganography and networking were modified so that students were required to find answers that reflected their unique hash values. However, PolyLab does not support the virtualization provided by Labtainers.

Automated assessment has been explored for programming courses, e.g. [12, 15, 16], and are generally either static or dynamic [1]. We found that none were directly applicable to our parameterization framework.

2.2 Virtualization Approach: Containers

A key enabling technology that permits us to create a consistent, yet realistic, laboratory environment is virtualization. Like its predecessors discussed in the previous section, Labtainers also uses virtualization, but we wanted it to be easier for both instructors and students to use. Linux Docker [3] provides a convenient abstraction layer that builds upon Linux containers [24].

Our use of Docker containers allows the lab designer to specify an appropriate execution environment for each lab in terms of installed programs, library versions, and configuration settings typically found in the Linux `/etc/` directory, e.g., networking configurations. Thus, the execution environment within a lab container can vary significantly from that provided by the host OS without modifying the student's host platform. Similarly, the student's Linux host platform, (e.g., a VM) can be any Linux distribution that supports Dockers, and need not be a particular version or configuration. The advantages of the Linux Docker [4] environment and its impact on the Labtainer framework are listed below [3]

"File system isolation: *each process container runs in a completely separate root file system."* This means that each laboratory exercise can be provisioned and packaged with all of the files, including startup and parameterization scripts, required for students to start the exercise.

"Resource isolation: *system resources like CPU and memory can be allocated differently to each process container, using cgroups."* Problems associated with exercises will not spill out of the container: neither instructors nor students will experience a negative impact on the other work they are conducting on their systems. Through well designed Docker resource allocation, it may be possible to create exercises in which the need for synchronization primitives and transactional properties are clearly illustrated.

"Network isolation: *each process container runs in its own network namespace, with a virtual interface and IP address of its own.* For laboratory exercises, it is possible to create networked containers, thus allowing exercises on topics in network security.

"Copy-on-write: *root file systems are created using copy-on-write, which makes deployment extremely fast, memory-cheap and disk-cheap."* This is attractive for students and instructors who may have resource-constrained platforms.

"Change management: *changes to a container's file system can be committed into a new image and re-used to create more containers."* Lab designers and instructors can easily create variations of laboratory exercises, thus the exercises can be both extensible and evolvable.

"Interactive shell: *docker can allocate a pseudo-tty and attach to the standard input of any container, for example to run a throwaway interactive shell."* This feature supports a variety of exploratory activities by students.

The worst case software stack for an instructor or student using Labtainers will layer a Type II virtual machine monitor, e.g. Virtual Box, on a commodity operating system. Linux will run in a VM, with the Docker engine and the containers (applications) it supports comprising the uppermost layers as shown in Figure 1. An individual running a Linux system will have a shorter stack as shown in Figure 2.

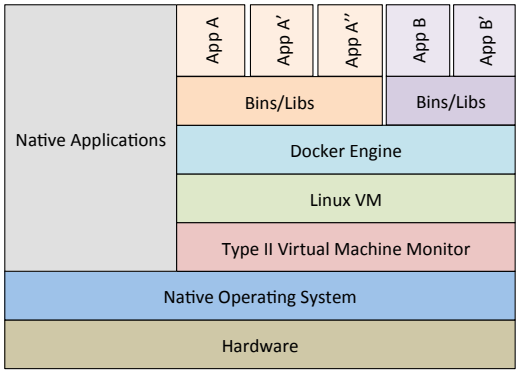

Figure 1: Worst case software stack

3 LABTAINER FRAMEWORK

The envisioned student workflow and the use of the Labtainers framework to define, parameterize, and automate assessment of labs is described below.

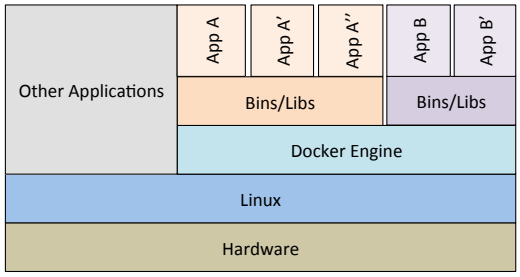

Figure 2: Best case software stack

3.1 Overview of Student Environment and Workflow

Labtainers support laboratory exercises designed for Linux environments, ranging from interaction with individual programs to labs that include what appear to be multiple components and networks. Students see and interact with Linux environments, primarily via *bash* shell commands. In general, the Labtainer framework implementation is not visible to the student, and the Linux environment as seen by the student is not augmented or modified to support the framework. The Linux execution environments presented by Docker containers do not readily include GUIs. As a result, Labtainer are not intended for use with GUIs.

Labtainers are intended for use on individual student computers. The computer utilized by a student must include the Linux operating system, e.g., as a single VM. This Linux operating system can be any distribution and version that supports Dockers, and it must have Dockers installed. In addition to installing Dockers on the Linux host, the student must obtain and expand a tarball, which contains the Labtainer workspace utilities. (This tarball may someday be replaced by standard Linux distribution packages, e.g., Debian and/or RPM packages.) Students initiate any and all labs from a single workspace directory on the Linux host.

To perform a specific Labtainer exercise, the student runs a "start" command from the Labtainer workspace, naming the lab exercise. This starts up one or more containers, along with corresponding virtual terminals through which the student will interact with the containers. These virtual terminals typically present a *bash* shell. Each container appears to the student as a separate computer, and these computers may appear to be connected via one or more networks.

When a student starts a given exercise for the first time, the framework fetches Docker images from a centralized repository, and creates Docker containers parameterized for the individual student using a cryptographic seed.

After the student performs the lab exercise, artifacts from the container environments are automatically collected into a zip file that appears on the student's Linux host. The student forwards this zip file to the instructor, e.g., via email or a Learning Management System (LMS). The instructor collects student zip files into a common directory on his or her own Linux host, and then issues a command that starts the instructor container(s) for that lab. This results in automated assessment of student lab activity, (if the lab is designed for that), and creation of an environment in which the instructor can review the work of each student.

The instructor does not take any additional actions to set up or facilitate automatic assessment or parameterization of student labs. The framework automates those functions based on configuration values defined by the lab designer as described in the following section.

3.2 Defining New Labs

The most challenging and critical part of creating a new cyber security lab is the design of the lab itself, i.e., identifying learning objectives and organizing exercises to achieve those objectives. The Labtainer framework does not specifically address any of that. Rather, the framework is intended to allow the lab designer to focus more time on the design of the lab and less time on mitigating and explaining system administration burdens placed on students and instructors. The framework does not require lab designers to program or create scripts. The lab designer primarily interacts with the framework by editing configuration files that affect the student's execution environment and the optional automated assessment of student activity.

Editing a set of configuration files is how lab designers perform the three primary steps to create a new lab.

Define the lab execution environment

A given lab typically requires some set of software packages, and some system configuration, e.g., network settings. Identifying an expected environment is not unique to this framework, rather, it is typically part of any lab design. The framework captures most configuration details within a standard Dockerfile.[2] Additionally, the lab designer identities the set of lab-specific files, e.g., vulnerable programs, that are to reside in the student's home directory within the container.

For each lab, there will be a template Dockerfile for student containers and one for instructor containers. These use standard Docker file syntax. Simple labs can use the default Dockerfiles created by the Labtainers script.

Lab Parameterization

The lab designer identifies specific properties of the lab that are to be individualized for each student. Parameterization is achieved by defining symbols within source code or data files. During container initialization, the framework will replace these symbols with randomized, student-specific values. A configuration file identifies the files, and the symbols within those files that are to be replaced by the computed values. An example might be a constant controlling a buffer size in the source code of a program that the student is to interact with. A given student's instance of the lab will have a cryptographically generated seed that is unique to that student for that lab. The configuration file entry for the example buffer size value would be expressed as a random value within a given range. When the student first starts the lab, functions within the framework will replace the indicated constant in the source code file prior to code compilation.

Automated assessment of student labs

This section describes how to configure a lab for automated assessment of student work. It is worth noting that the framework does not require that labs include automated assessment, e.g., the

[2] https://docs.docker.com/engine/reference/builder/

"results" of a lab may consist entirely of a written report submitted by the student.

The goal of automated assessment is to provide instructors with some confidence that students performed the lab, and to give instructors insight into which parts of a lab students may be having difficulty with. It is possible to create exercises in which the student must execute a specific sequence of operations, and the lab designer can easily create assessments for such labs. It is more interesting to design labs that encourage experimentation, but such experimentation does not lend itself to highly prescriptive assessment techniques. The automated assessment functions we envision are not intended to standardize each student's approach to a lab, rather the goal is to permit *ad hoc* exploration by students. Therefore, the framework gives the lab designer a means to identify evidence that steps of a lab were performed rather than trying to identify everything a student may have done in the course of the lab.

The framework's automated assessment functions generally assume the student will interact with one or more programs or system utilities. Each time the student invokes a selected program or utility, the framework captures copies of standard input and standard output (*stdin* and *stdout*) into timestamped file sets. This is transparent to the student. For example, one artifact might be the value that follows the string, "The result is:" within the *stdout* file. The lab designer then references these tagged values within a configuration file which defines the expected results of the lab. The expression of expected results can include indirect references to values that are specific to the student's instance of the lab, i.e., are a function of the lab-specific seed described earlier.

These timestamped file sets, and everything else relative to the exercise in the student's container-based home directory, are automatically packaged when the student completes the lab. These packages of artifacts are then transferred to the instructor, (e.g., via email or a LMS), and ingested into the instructor's system where lab assessment occurs.

Thus, for assessments, the framework requires the lab designer to create configuration files identifying what has been parameterized for each lab, and what artifact values constitute success. This extra work by the lab designer greatly simplifies the instructor's task of assessing each student's performance of the labs. While many labs may still include an essay component in which the student is asked to describe the lab activity, the automated evaluation of artifacts from individualized labs gives the instructor confidence that the student essays reflect work performed by the student.

Example Lab Definition for parameterization and automated assessment

Consider a simple buffer overflow lab in which a vulnerable program is exploited by consuming a student-defined data file crafted to overflow a buffer and transfer execution to shellcode embedded in the data file. The lab might be parameterized by changing the size of the overflowed buffer for each student, thereby altering the location of the overwritten return address relative to the start of the buffer. The lab designer inserts a symbolic name into the source code of a vulnerable program where the buffer size is defined, and the designer adds a corresponding configuration file entry directing that symbolic name to be replaced with a random value within a specified range. An example configuration file entry is:

```
rand1:RAND_REPLACE:/home/ubuntu/stack.c:BUF_SZ:100:500
```

which will replace the symbol `BUF_SZ` in the file *stack.c* with a random value between 100 and 500. A different configuration file entry would then cause the individualized *stack.c* program to be compiled the first time the student starts the lab. Thus, the lab designer augments the source code, and defines two configuration file entries to parameterize this lab.

Automated assessment of the lab could be as simple as confirming the student displayed the content of a predefined file after gaining a shell from the *stack.c* program. This can be achieved by entries in two configuration files, the first, in *results.config* identifies the result within the artifacts, e.g.,

```
file_string = stack.stdout : CONTAINS : the cheese
```

The symbol "file_string" will be true if the stdout of any invocation of stack program contains the string *the cheese.* An entry in a second, "goals.config" configuration file defines success:

```
viewed_file = is_true : file_string
```

The goal of "viewed_file" is met if the symbol file_string is true. Goal assessment operations also include comparison of symbols to specific values and to parameterized values. And it supports temporal comparisons and boolean expressions.

When the instructor container is started, the assessment is performed automatically and the instructor is provided with a summary of each student's achievement of the "viewed_file" goal. This simple example can be extended such that file displayed by the student contains an individualized value, an example of which is within one of our "bufoverflow" Labtainer example lab.

4 SUMMARY AND FUTURE WORK

Built around standard Linux Docker-supported containers, the Labtainer framework is targeted for use with labs designed for Linux environments. Deploying cyber security labs using this framework provides simplicity, consistency, per-student parameterization, and assessment support.

- The use of containers provides the benefit of a consistent execution environment without requiring an individual VM per lab, and without requiring all labs to be adapted for a common Linux execution environment. The lab execution environment is controlled and consistent across all students' computers regardless of the Linux distribution, version, and configuration. This allows each lab designer to control which software packages are present, the versions of libraries and configuration settings, e.g., `/etc` values. These configurations may vary between labs.
- Labs may be automatically parameterized for each student so that students cannot simply copy results from another student or from internet repositories.
- Automated assessment of student lab activity is supported through detailed logging of student activity and a set of

configuration files that identify expected results. When combined with appropriate tools, these capabilities can relieve lab instructors from having to individually review detailed lab results. The assessment configurations support per-student parameterization.

An initial implementation of the framework has been completed and we have adapted a subset of the existing labs from the Syracuse University SEED Labs project as a proof of concept. We are also moving some of our existing and new computer forensics labs to the Labtainer environment. At this point Labtainers are sufficiently robust to support creation and adoption of new labs focused on the security of industrial control systems. We expect these labs will be attractive to instructors of ICS courses who have modest resources for or lack physical ICS lab equipment or for distance learning courses.

As we progress, the laboratory exercises will be evaluated in several ways. First each will bench tested for functionality. Exercise instructions will be reviewed and tested by individuals outside of the exercise development group, thus ensuring that some tacit knowledge of what the directions are *supposed* to mean is not brought into the tests. In our classes, student use of the exercises will be monitored and the exercises will be refined to ensure clarity and easy identification of expected student progress.

We will pursue tools to help instructors evaluate student work with selected levels of granularity. For example, it may be helpful to enable instructors to ask questions related to student progress within a lab exercise and the duration of student activity.

REFERENCES

[1] K. M. Ala-Mutka. 2005. A survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education* 15, 2 (June 2005), 83–102.

[2] Amazon.com. 2011. About AWS. https://aws.amazon.com/about-aws/ Retrieved 8 May 2017. (September 2011).

[3] Abel Arvam. 2013. Docker: Automated and Consistent Software Deployments. https://www.infoq.com/news/2013/03/Docker. (27 March 2013).

[4] Docker.com. 2017. Docker. https://www.docker.com. (2017).

[5] John L. Donaldson. 1987. Teaching Operating Systems in a Virtual Machine Environment. In *Proceedings of the Eighteenth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '87)*. ACM, New York, NY, USA, 206–211. DOI: http://dx.doi.org/10.1145/31820.31759

[6] Wenliang Du. 2011. SEED: Hands-On Lab Exercises for Computer Security Education. *IEEE Security and Privacy Magazine* 9, 5 (Sept. 2011), 70–73. DOI: http://dx.doi.org/10.1109/MSP.2011.139

[7] W. Du, K. Jayaraman, and N. B. Gaubatz. 2010. Enhancing Security Education with Hands-on Laboratory Exercises.. In *Proceedings 5th Annual Sympoisum on Inforamtion Assurance (ASIA'10)*.

[8] Wenliang Du and Ronghua Wang. 2008. SEED: A Suite of Instructional Laboratories for Computer Security Education. *J. Educ. Resour. Comput.* 8, 1, Article 3 (March 2008), 24 pages. DOI: http://dx.doi.org/10.1145/1348713.1348716

[9] Janet Eyler. 2009. The Power of Experiential Education. *Liberal Education* 95, 4 (2009).

[10] Niclaus A. Giacobe and Ryan Kohler. 2016. Development of Polymorphic Homework and Laboratory Assignments in Cyber Security with PolyLab. https://www.fbcinc.com/e/nice/ncec/presentations/2016/Track_C_ShawneeMission/C-8_Giacobe_and_Kohler_(2016)_Development_of_Polymorphic_Homework_and_Laboratory_Assignments_in_Cyber_Security.pdf. In *NICE (National Initiative for Cyber Education) Conference 2016*. Kansas City, MO.

[11] Brian Hay, Ronald Dodge, and Kara Nance. 2008. Using Virtualization to Create and Deploy Computer Security Lab Exercises. In *Proceedings of The IFIP Tc 11 23rd International Information Security Conference: IFIP 20th World Computer Congress, IFIP SEC'08, September 7-10, 2008, Milano, Italy*, Sushil Jajodia, Pierangela Samarati, and Stelvio Cimato (Eds.). Springer US, Boston, MA, 621–635. DOI: http://dx.doi.org/10.1007/978-0-387-09699-5_40

[12] Petri Ihantola, Tuukka Ahoniemi, Ville Karavirta, and Otto Seppälä. 2010. Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10)*. ACM, New York, NY, USA, 86–93. DOI: http://dx.doi.org/10.1145/1930464.1930480

[13] Jelena Mirkovic and Terry Benzel. 2012. Teaching Cybersecurity with DeterLab. *IEEE Security and Privacy* 10, 1 (Jan. 2012), 73–76. DOI: http://dx.doi.org/10.1109/MSP.2012.23

[14] Kara Nance, Blair Taylor, Ronald Dodge, and Brian Hay. 2011. Creating Shareable Security Modules. In *7th World Conference on Information Security Education (IFIP Advances in Information and Communication Technology)*, Ronald C. Dodge and Lynn Futcher (Eds.), Vol. 406. 156–163.

[15] R. S. Pettit, J. D. Homer, K. M. Holcomb, N. Simone, and S. A. Mengel. 2015. Are Automated Assessment Tools Helpful in Programming Courses?. In *122nd ASEE Annual Conference & Exposition*. American Society for Engineering Education.

[16] Vreda Pieterse. 2013. Automated Assessment of Programming Assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research (CSERC '13)*. Open Universiteit, Heerlen, Open Univ., Heerlen, The Netherlands, The Netherlands, Article 4, 12 pages. http://dl.acm.org/citation.cfm?id=2541917.2541921

[17] Amit Shah. 2016. Ten Years of KVM. https://lwn.net/Articles/705160/. (02 November 2016).

[18] VMware. 2017. vSphere and vSphere with Operations Management. http://www.vmware.com/products/vsphere.html. (April 2017).

[19] Xinli Wang, Yan Bai, and Guy C. Hembroff. 2015. Hands-on Exercises for IT Security Education. In *Proceedings of the 16th Annual Conference on Information Technology Education (SIGITE '15)*. ACM, New York, NY, USA, 161–166. DOI: http://dx.doi.org/10.1145/2808006.2808023

[20] Richard Weiss, Jens Mache, and Michael Locasto. 2014. EDURange: Hands-on Cybersecurity Exercises in the Cloud. *J. Comput. Sci. Coll.* 30, 1 (Oct. 2014), 178–180. http://dl.acm.org/citation.cfm?id=2667369.2667402

[21] Richard Weiss, Jens Mache, and Erik Nilsen. 2013. Top 10 Hands-on Cybersecurity Exercises. *J. Comput. Sci. Coll.* 29, 1 (Oct. 2013), 140–147. http://dl.acm.org/citation.cfm?id=2527148.2527180

[22] Richard Weiss, Vincent Nestler, Michael E. Locasto, Jens Mache, and Brian Hay. 2013. Hands-on Cybersecurity Exercises and the RAVE Virtual Environment (Abstract Only). In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education (SIGCSE '13)*. ACM, New York, NY, USA, 759–759. DOI: http://dx.doi.org/10.1145/2445196.2445505

[23] C Willems and C Meinel. 2012. Online assessment for hands-on cyber security training in a virtual lab. In *Global Engineering Education Conference (EDUCON)*. IEEE. DOI: http://dx.doi.org/10.1109/EDUCON.2012.6201149

[24] Y. Yu. 2007. *OS-level Virtualization and its Applications*. Ph.D. Dissertation. State University of New York, Stony Brook, Stony Brook, NY.