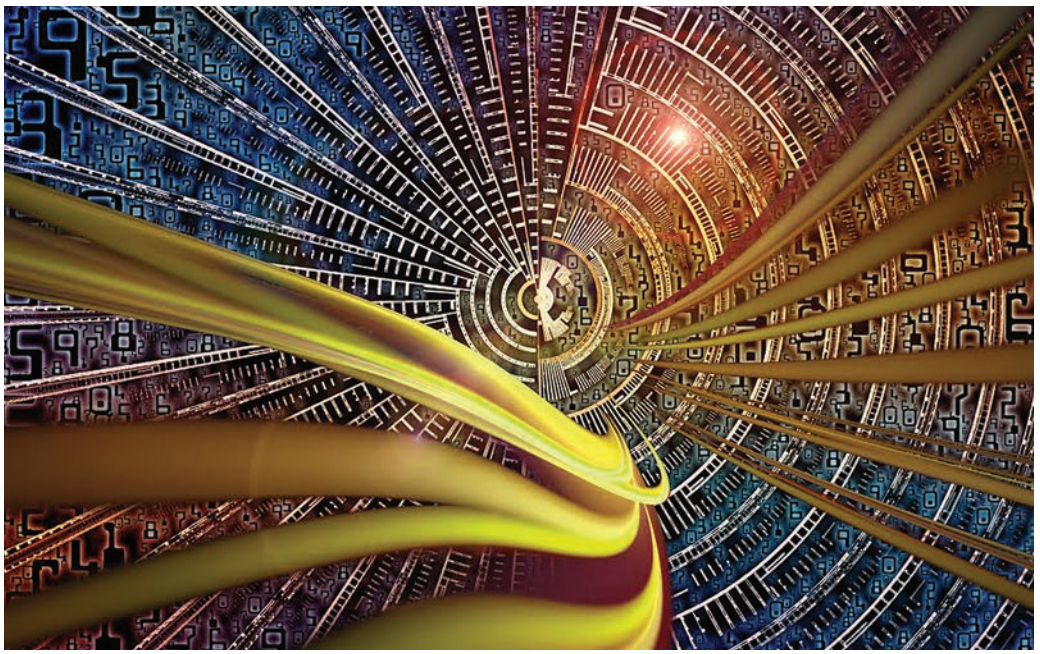

COMPUTING
edge
> The Cyber-Physical Systems Revolution
> Automotive Innovations
> Computing Education
> Smart Cities
> Cloud Computing
MAY 2018
www.computer.org
IEEE
IEEE computer society

Editors: Melissa Dark, dark@purdue.edu | Jelena Mirkovic, mirkovic@isi.edu | Bill Newhouse, william.newhouse@nist.gov

Individualizing Cybersecurity Lab Exercises with Labtainers

Michael F. Thompson and Cynthia E. Irvine | Naval Postgraduate School



Hands-on laboratory exercises help students internalize knowledge so that it can be applied in new contexts. Hence, cybersecurity educators try to create lab exercises that allow students to explore systems, yet provide sufficient guidance so that students achieve the desired learning objectives without becoming lost in minutiae. Challenges associated with developing such exercises include creating and supporting each lab, ensuring students do their own work, and grading exploratory activities.

In many cases, providing labs is difficult because access to physical lab computers—or remote access to institutionally or other centrally provisioned and managed resources—is not practical. Binding students to centralized servers can make self-paced, intermittent activity more difficult, yet, lacking institutional IT equipment and staff, instructors may not be able to present easily managed and deployed fine-tuned lab environments. However, if students run lab exercises directly on their own computers, other problems arise: the results produced may vary from student to student depending on software installed on the computer used, and all the tools required for an exercise may not even execute on certain platforms. The solution is tailored cybersecurity lab environments that eliminate divergent results caused by software differences. This can be achieved by providing students with virtual machine (VM) images containing lab-related software.[1] Students then run VMs on their personal computers or on institutionally provided computers. Through use of VMs, variations in the results among students can be largely limited to hardware performance differences.

But, use of VMs on student computers has several drawbacks. First, exercises involving two or more networked computers require multiple VMs, the hosting of which is beyond the performance capabilities of many student computers. Also, different labs may rely on mutually incompatible configurations or software packages, thus requiring students to either perform complex provisioning steps or to install separate VMs for each lab. The provisioning and administration of the execution environments required by different labs can become a significant distraction and source of frustration for both students and instructors.

Another challenge is that, regardless of how they are provisioned, cybersecurity lab exercises are often susceptible to students' sharing solutions and cribbing from each other's lab reports. Use of VM images on individual student computers complicates schemes designed to verify student performance of their own lab exercises, for example, logging

2469-7087/18/$33.00 © 2018 IEEE Published by the IEEE Computer Society

and audit features that might be part of a remotely accessed cyber range. Student actions on VMs can be logged; however, use of a single VM for multiple labs would require some method to distinguish the artifacts of different labs. The alternative of allocating each lab to a distinct VM image can be prohibitive in terms of network bandwidth and disk storage on the student computer.

The last challenge is encouraging students to explore the lab environment while providing instructors with a simple way to determine that students have achieved expected milestones. How can students "show their work"? How can instructors observe what students have done and provide advice if they are stuck, yet not have to stand over the students while they complete the entire exercise?

Labtainers: A Practical Solution Using Docker Containers

Labtainers is a framework for developing and deploying Linux-based labs involving multicomponent network topologies all hosted entirely on modestly provisioned student computers. Our initial emphasis is on cybersecurity. Docker containers[2] are used to standardize complete lab execution environments, thereby reducing lab setup and configuration distractions. By using containers, labs can incorporate complex topologies without suffering the overhead of running multiple VMs. The Labtainer framework supports automated assessment of student work and allows lab exercises to be individualized for each student, thus discouraging the appropriation of others' work.

The use of Docker containers simplifies the Labtainer approach to individualizing student labs and recording student activity for later assessment by instructors. The framework automatically collects artifacts from a student lab environment into an archive file that the student forwards to her instructor. Here we describe strategies for ensuring that the artifacts in the archive file are the result of that student's efforts. We present these strategies in the context of two example Labtainer exercises. The first provides an introduction to network traffic analysis using *tshark*, and the second employs the *nmap* utility to locate a selected network service.

The Labtainer framework supports three types of users. *Lab designers* are responsible for creating laboratory exercises so that they meet intended learning objectives. Each lab designer determines if and how the lab is parameterized and whether automated assessment will be supported. *Instructors* assign labs to students and assess their work. Instructors may or may not work with lab designers to create exercises. *Students* perform the laboratory exercises. They are oblivious to the underlying framework that configures and individualizes their labs and that gathers artifacts required for assessment.

Target Lab Context and Automated Assessment

Students start Labtainer exercises by executing a Python script on a Linux host, typically a VM. The script augments the Linux host environment with one or more Docker containers and a set of virtual terminals. Students use the virtual terminals to interact with the containers, which from the students' vantage point appear to be independent computers. The execution environment within each container is prescribed by the designer of the lab. In the degenerate case where the lab designer provides only a name for the lab, the environment seen by the student will be a *bash* shell on what appears to the student to be an Ubuntu Linux system. The Labtainer framework allows the lab designer to select from a variety of Linux distributions for each container and to include software packages and configuration settings as appropriate for the lab. Designers define virtual networks and the connections among containers. The student sees the resulting network topology and has virtual terminals connected to only those containers indicated by the designer.

After the student performs the lab exercise, she runs another Python script that terminates the lab on the Linux host. This results in the collection of artifacts from her lab activity. She then provides the resulting archive to the instructor. The instructor can review these artifacts on similarly provisioned Docker containers. The framework includes tools for the lab designer to specify expected attributes of the student artifacts, which are then automatically assessed and summarized for the instructor. Labtainer support for consistent execution environment provisioning and automated assessment of student work is described in detail elsewhere.[3,4]

Attribution through Lab Individualization

Several approaches to ensuring the individuality of student work are possible in the Labtainer framework: *watermarks*, *per-student artifacts*, and *per-student solutions*. Within a particular lab exercise, these can be used separately or in combination.

Per-Student Watermarks

When a student starts a lab, the Labtainer framework incorporates a student-supplied email address into a seed for generation of pseudorandom values. A watermark file is automatically created for each student lab, and this file becomes one of the artifacts in the student's archive. The watermark value is validated as part of the assessment process initiated by the instructor.

This simple strategy ensures (albeit weakly) that the archive provided by the student originated with the student who started the lab. As will be seen in subsequent sections, the Labtainer framework allows lab designers to improve on the assurances provided by the watermarks.

The simple watermark check inherent in all Labtainer exercises is readily bypassed by replacing a single file in the archive, perhaps by an automated script shared among students, effective on all Labtainer exercises. Variations on the watermark strategy can be defeated by correspondingly advanced automations, for instance, scripts that replace individual artifacts such as the output of a program invocation. These automations become specific to the individual labs and thus require more effort by the benevolent cheater to build and maintain. A fundamental limitation on the robustness of the watermarking strategies is that the Labtainer framework does not keep secrets from the student environment. Our design explicitly avoided the large step in complexity inherent in maintaining secrets.

Additional assurances of the originality of student work rely on choices made by the lab designer. In the Labtainer framework, these schemes typically have one of two purposes. The first, per-student artifacts, provides further evidence that someone performed the exercise on a Labtainer instance that was initiated using the student's email address. This strategy causes selected artifacts generated by lab exercise steps to be individualized for each student. The second approach, per-student solutions, seeks to ensure that whoever performed the exercise did not simply reproduce dictated actions. In the Labtainer framework, these per-student solutions, when practical for a given lab, provide more assurance that students performed their own work.

Per-Student Artifacts

Introduction of per-student artifacts makes development of cheating automations more challenging, because individual artifacts themselves are tailored to the individual student. A simple example is an exercise that requires the student to display to standard output the content of a student-specific file on a server once a remote shell on the server is obtained. This would defeat an automation that simply inserts unmodified artifacts into the student's archive. While one can imagine correspondingly sophisticated automations that are informed by the particulars of the lab exercise, at some point, it becomes easier for enterprising students to publish and re-perform exact keystrokes necessary to create the desired artifacts. For some labs, the keystrokes problem can be addressed by per-student solutions.

A less trivial example of per-student artifacts is found in the Labtainier *pcap analysis* lab, in which students are introduced to basic network traffic analysis techniques using the tshark utility. Each student's Labtainer environment for this lab includes a *pcap* file tailored to that student. The pcap file is individualized by truncating a random quantity of "filler" packets from the start of a baseline pcap file. This results in packet numbers that are unique to the student, while the content of the non-filler traffic remains constant for all students. The student is required to display the single packet of a specific invalid login attempt. Hence, the output of the corresponding tshark command will include a student-specific packet number that can be deterministically reproduced by the assessment function in the instructor's environment.

Lab designers individualize labs using parameterization configuration files containing commands that cause the framework to replace symbols in selected files with random values derived from the student email address. For the pcap analysis lab, the target of replacement is a parameter passed to a utility that truncates the start of the pcap file. This *parameterization* utility is invoked the first time a student runs the lab, resulting in truncation of the pcap file seen by the student. (Note: students do not have to complete a lab in one sitting, and if the student wishes, the framework allows her to restart a lab from the beginning, with complete reinitialization and consistent personalization.) The configuration file entry shown in Figure 1 causes the symbol "START_FRAME" in the file "fixlocal.sh" to be replaced by a random value between 1 and 100.

```
FIRST_FRAME : RAND_REPLACE: .local/
bin/fixlocal.sh : START_FRAME : 1 :
100
```

Figure 1. Extract from parameterization configuration file.

Lab designers define automated assessment in assessment configuration files that identify student artifacts and their expected attributes. The pcap analysis lab assessment configuration files identify the standard input of the tshark command as an artifact of interest. The configuration file includes a directive to extract the "frame.number" filter argument provided to tshark. Labtainer assessment configuration file directives allow the designer to symbolically reference symbols named in parameterization configuration files. The expected value of the frame number is derived by subtracting the random value used during parameterization from the value of the frame number as it existed before the pcap file was truncated. The configuration file entry in Figure 2 subtracts from 190 the

```
view_frame = matchany : integer_
equal : (190-frame_num) : parameter.
FIRST_FRAME
```

Figure 2. Extract from assessment configuration file.

"frame_num" value found in a student's artifact and compares this to the random value that resulted from the directive in Figure 1.

Per-Student Solutions

The example pcap analysis lab is susceptible to one student providing another with the precise keystrokes needed to complete the lab—a problem associated with all labs that rely only on per-student results. Per-student solutions defeat rote repetition of keystrokes. An example is the *nmap-discovery* lab, which presents the student with a fictional scenario in which he is told that he has an account on an SSH server but is given only the server name (not the network address) and his password. The student uses nmap to locate the server IP address and to discover the SSH port number that the IT department had set to an arbitrary value. This exercise is individualized by assigning a random port number, within a range, to each student. Thus, rote keystroke repetition fails to complete the lab.

The parameterization configuration file for the nmap-discovery lab names symbols in Linux system services configuration files. These system files were modified by the lab designer to contain symbolic names in place of the SSH port numbers. These symbolic names are replaced during parameterization. As a result, system networking services on the configured container will listen to the individualized port number for SSH traffic.

In this particular lab, there is no need for assessment configuration directives to reference symbols in parameterization configuration files, because the student could not have SSH'd to the server unless the port number was discovered. This simplifies automated assessment and is in contrast to the previous example in which the assessment configuration files referenced the pcap truncation parameter.

The nmap-discovery lab automated assessment reveals whether the student was able to use SSH to connect to the target server and the number of times the student invoked the nmap command. The assessment configuration file identifies standard output from the SSH command as an artifact of interest—specifically any output that contains a constant string within a file present on the target SSH server. If this string appears in the artifacts, the student is assumed to have discovered the SSH port number.

Beyond the primary motivation of not rewarding rote replays of lab steps, per-student solutions have an advantage from the perspective of the lab designer. Because the parameterization need not be reproduced in the assessment step, the expected results as represented in the student artifacts can be confirmed without reference to any specific parameterized values generated for that student. Although making a tie between a parameterized value and the expected results in the assessment configuration files is often relatively straightforward, it can become tedious and error prone for some exercises. Consider a forensics-oriented lab that requires students to recover a deleted file from a virtual file system. A simple way to individualize the lab is to add a randomly determined number of filler files into the file system prior to creating the files of interest. The filler files force file offsets and inode numbers of the target files to be a function of the student's email address. Assessing per-student results for this lab (for instance, comparing student-provided inode values with expected values) is challenging because the effects of filler files on inodes and offsets depend on file system implementation vagaries, and these are not easily predicted. The assessment step for this forensics lab need not be concerned with what the values are for any given student; rather it can rely only on whether the deleted file was in fact recovered. Thus, a per-student solution assumes the student could not have recovered the file content without discovering the offset or inode specific to that student.

Status and Availability

More than 35 labs are available to students and instructors in the Labtainer framework (https://my.nps.edu/web/c3o/labtainers). Each includes a student lab manual, and most include automated assessment and perform per-student individualization. The website also includes a developer package for use by lab designers when creating new labs or transitioning existing labs into the Labtainer framework. The containers themselves are hosted on the public Docker hub (https://hub.docker.com) and are transparently loaded onto a student's computer when the corresponding lab is first started. The "Labtainer Lab Designer User Guide"[4] describes how lab designers can publish their own Labtainers-based labs on the Docker hub, thus making them available to their students.

Future work we hope to pursue on Labtainers includes GUI-based, integrated lab-authoring tools as well as additional features to support instructors. These include HTML-based reporting of student assessment results and integration into learning management systems. We would also like to convert our development and publishing work flow to a collaborative environment to simplify the integration of

This article originally appeared in
IEEE Security & Privacy, *vol. 16, no. 2, 2018.*

contributions by a community of lab designers. ■

Acknowledgments

This work was supported by NSF grant DUE-1438893. 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 US Government.

References

1. W. Du, "SEED: Hands-On Lab Exercises for Computer Security Education," *IEEE Security & Privacy*, vol. 9, no. 5, 2011, pp. 70–73.
2. A. Arvam, "Docker: Automated and Consistent Software Deployments," 27 Mar. 2013; https://www.infoq.com/news/2013/03/Docker.
3. C.E. Irvine et al., "Labtainers: A Docker-Based Framework for Cybersecurity Labs," *Proc. 2017 USENIX Workshop on Advances in Security Education*, August 2017.
4. M.F. Thompson, "Labtainer Lab Designer User Guide," 27 Oct. 2017; http://my.nps.edu/documents/107523844/109121513/labdesigner.pdf.

Michael F. Thompson is with Naval Postgraduate School. Contact him at mfthomps@nps.edu.

Cynthia E. Irvine is with Naval Postgraduate School. Contact her at irvine@nps.edu.