Online Assessment for Hands-On Cyber Security Training in a Virtual Lab

Christian Willems and Christoph Meinel
Internet Technologies and Systems Group
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany
{christian.willems, meinel}@hpi.uni-potsdam.de

Abstract—**Online (self) assessment is an important functionality e-learning courseware, especially if the system is intended for use in distant learning courses. Precisely for hands-on exercises, the implementation of effective and cheating-proof assessment tests poses a great challenge. That is because of the static characteristics of exercise scenarios in the laboratories: adopting the environment for the provision of a "unique" hands-on experience for every student in a manual manner is connected with enormous maintenance efforts and thus not scalable to a large number of students.**

This work presents a software solution for the assessment of practical exercises in an online lab based on virtual machine technology. The basic idea is to formally parameterize the exercise scenarios and implement a toolkit for the dynamic reconfiguration of virtual machines in order to adopt the defined parameters for the training environment. The actual values of these parameters come to use again in the dynamic generation of multiple-choice or free-text answer tests for a web-based e-assessment environment.

Keywords-Virtual Machine, Remote Laboratory, Online Assessment, Self Assessment, Cyber Security Training

I. INTRODUCTION

Traditional techniques of teaching (i.e. lectures or literature) have turned out to be not sufficient for cyber security training, because the trainee cannot apply the principles from the academic approach to a realistic environment during the course. In security training, gaining hands-on experience through exercises is indispensable for consolidating the knowledge. Besides this, only practical training is suitable to efficiently illustrate the importance of details: a tiny flaw in a service or firewall configuration may ruin all efforts to secure a system or network.

Precisely the allocation of an environment for these practical training sessions poses a challenge not only for lecturers, but also for research and development. That is, since students need privileged access rights (root/administrator-account) on the training system to perform most of the imaginable security exercises. With these privileges, students might easily destroy a training system or even use it for unintended, illegal attacks on other hosts on the campus network or the Internet world.

Traditional means for practical cyber security training are dedicated, isolated computer labs for security training. These labs are expensive and demanding concerning creation and maintenance. Especially for the training of network security topics, a single student needs a workplace with three networked computers (e.g. to experience a Man-in-the-Middle attack, compare section III). Due to the drawbacks there is a trend towards the provision of such laboratories using virtual machine technology [3].

Figure 1. Tele-Lab: Web-based Training System and Remote Desktop of Virtual Machine for Hands-On Training

A comprehensive implementation of a computing laboratory based on virtual machines is the Tele-Lab platform (described in detail in section II). Tele-Lab platform combines the virtual lab with a web-based training system and allows remote lab access on the Internet, making the lab not only suitable for local classes but also for self- and distance learning approaches. Self-assessment is a crucial part as well as for local students as for autonomous distance learners, but the implementation of a suitable assessment framework is a difficult task when it comes to hands-on exercises.

A very basic learning unit in Tele-Lab on Attacks on Accounts and Passwords asks the student to experience how

Christian Willems, Christoph Meinel:
"Online Assessment for Hands-On Cybersecurity Training in a Virtual Lab"
in Proceedings of the 3rd IEEE Global Engineering Education Conference (EDUCON 2012),
IEEE Press, Marrakesh, Morocco, 2012. (to be published)

fast weak passwords can be cracked. On the training machine (Windows XP) the user must dump the passwords to a file using *PwDump*, and then reveal the plaintext passwords with the well-known *John-the-Ripper* password recovery tool. In an online assessment session on this learning unit the student is asked, which passwords he could reveal on the training machine. The knowledge of these passwords proves that the student has completed the hands-on exercise successfully.

The basic problem with existing virtual and physical laboratories and (automatic) e-assessment of the practical training is the missing versatility of the exercise scenarios: the virtual machines for the exercises are usually configured once during the design phase of the exercise scenario, then deployed to the (virtual) lab and recurrently used by students. Applied to the context of the above example this means that every user cracks the same passwords any time she performs the exercise. While this way of assessment can be sufficient in one-shot tests for autonomous learners (e.g. in a distant learning course), there are a number of limitations for assessment in lab classes or for the homework assignments in a university course.

In particular, the static characteristic of the assessment makes the system prone to cheating attempts: students in lab class could spy out the results – i.e. the cracked passwords – from their bench neighbors, homework assignments would have to be solved only by a single student who could then distribute the solution amongst all other course members. The original purpose of the assignment – getting hands-on experience – would be completely undermined. Additionally, the uniform end result of each learning unit cuts down the motivation of (remote) students to repeat a once completed assignment, even if they should practice the scenario repeatedly.

The paper at hand presents an approach to overcome these limitations with an enhancement to the Tele-Lab platform. Section IV introduces a concept for a dynamic assessment module for virtual labs, including a generic parameterization scheme for the description of versatile exercise scenarios: which values inside a virtual machine are seen as changeable parameters, what is the range of parameter values, and how can these values be deployed inside the virtual machine prior to a training session?

Section V describes the implementation of the concept for the Tele-Lab platform and presents a number of exercise scenarios that have been parameterized.

Section VI summarizes the results and gives an outlook on future work.

II. Related Work

Laboratories for security experiments and cybersecurity training exist in various manifestations. The traditional approach is a dedicated computer lab for IT security training. Such labs are exposed to a number of drawbacks: they can only be accessed on campus, are expensive to purchase and maintain and must be isolated from all other networks on the site. Of course, students are not allowed to have Internet access on the lab computers. Hands-on exercises on network security topics demand to provide more than one machine to each student, which have to be interconnected. An example for this kind of isolated, dedicated lab is the "Network and Systems Security Lab" at the Rochester Institute for Technology [14]. A benefit of such labs is the ability to provide experience with real hardware and experiments going down to the physical layer.

The *Emulab/PlanetLab* project at the University of Utah [10] exposes a networking test bed as a remote laboratory. Real hardware can be dynamically allocated, configured and managed to provide remote access to a number of networking experiments. This approach keeps up the benefit of dealing with actual hardware, while automatic management and allocation reduce the maintenance cost and remote access allows more flexible usage scenarios compared to the traditional approach.

There is a number of approaches describing the usage of virtual machine technology in networking and systems security courses, such as in [1], [2], [7] or [8]. The authors describe slightly different implementations of the use of virtual machine technology in local and remote lab classes. Considering security education, all of the approaches differ from Tele-Lab in the focus on a system administrators view on security training. While these virtual labs provide experiments where students should configure operating systems, networks and security-relevant software packages such as firewalls or intrusion detection systems, cyber security training in Tele-Lab highlights the importance of experiencing the capabilities of attackers, i.e. getting in touch with hacking tools and performing live attacks.

Work on concepts for self assessment and (semi-)automatic e-assessment in the context of hands-on exercise labs is still quite rare. Among the related projects mentioned so far, only the authors of [7] implemented a service that allows the students to evaluate their practical work by means of a scripted test procedure: if the assigned task was about configuring a firewall to restrict the access on certain ports, the test script would run a port scan against the student's lab VM and then parse the port scanner's output for open and closed ports. The student can trigger the test script from a web interface and is notified, if the task was solved properly. The authors of [8] follow a similar approach: the students configure intrusion detection systems during the lab session and can trigger the execution of attacks on their training system in order to evaluate their work. While this procedure is suitable for assignments following a defensive teaching methodology, there are security considerations (discussed in section IV) that do not allow implementing this concept for Tele-Lab.

III. Tele-Lab – A Remote Virtual Cybersecurity Laboratory

The Tele-Lab platform (accessible at http://www.tele-lab.org/, see fig. 1) was initially proposed as a standalone system [4], later enhanced to a live DVD system introducing virtual machines for the hands-on training [5], and then emerged to a server system [6]. The Tele-Lab server basically consists of a web-based tutoring system and a training environment built of virtual machines. The tutoring system presents learning units that do not only offer information in form of text or multimedia, but also practical exercises. Students perform

those exercises on virtual machines (VM) on the server, which they operate via remote desktop access. Virtual machine technology allows easy deployment and recovery in case of failure. Tele-Lab uses this feature to revert the virtual machines to the original state after each usage.

With the release of the current iteration of Tele-Lab, the platform was enhanced with dynamic assignment of more than one virtual machine to a single user at the same time. Those machines are connected within a virtual network (known as team, see also in [1]) providing the possibility to perform basic network attacks such as interaction with a virtual victim (e.g. port scanning). A victim is the combination of a suitably configured virtual machine running all needed services and applications and a collection of scripts that simulate user behavior or react to the attacker's actions (see also exemplary description of a learning unit below). A short overview of the architecture of the Tele- Lab platform is given later in this section.

A. Learning Units in Tele-Lab – an exemplary Walkthrough

Learning units follow a straightforward didactic path beginning with general information on a security issue, getting more concrete with the description of useful security tools (also for attacking and exploiting) and culminating in a hands-on assignment, where the student has to apply the learned concepts in practice. Every learning unit concludes with hints on how to prevent the just conducted attacks.

An exemplary Tele-Lab learning unit on eavesdropping (described in more detail in [13]) starts off with academic knowledge such as information on technologies for local area networks (LAN), the difference between switches and hubs or wireless networking. After that, various existing tools for packet sniffing are presented, such as *tcpdump* or the well-known *Wireshark* network protocol analyzer.

Following an offensive teaching approach, the user is asked to take the attacker's perspective – and hence is able to lively experience possible threats to his personal security objectives. The closing exercise for this learning unit is about eavesdropping on network traffic between two virtual communication partners, reveal credentials (username and password) for services from the captured messages and use these to steal private data from an FTP server.

Since the laboratory machines are connected on a virtual hub-like device, the student is able to capture all messages on the network – including the traffic between the two virtual victims Alice and Bob. Bob runs a server with HTTP and FTP services; Alice uses those services. The student has to use *wireshark* and inspect the captured packets for the login data. After that, he can log into Bob's servers using Alice's username and password. On the FTP server is a file containing secret information (a *flag*) that has to be stolen by the student. The knowledge of this secret information proves the successful completion of the assignment.

Such an exercise example underlines the need for the Tele-Lab user to be provided with a team of interconnected virtual machines: one machine is needed for attacking (all necessary tools installed), one machine for Bobs services and a third one for the client (Alice) that runs a set of scripts simulating access to Bob's server. Remote desktop access is only possible to the attackers VM.

Other learning units are also available on, e.g., attacks on accounts and passwords, wireless security, reconnaissance, remote exploitation, malware, Man-in-the-Middle attacks etc.

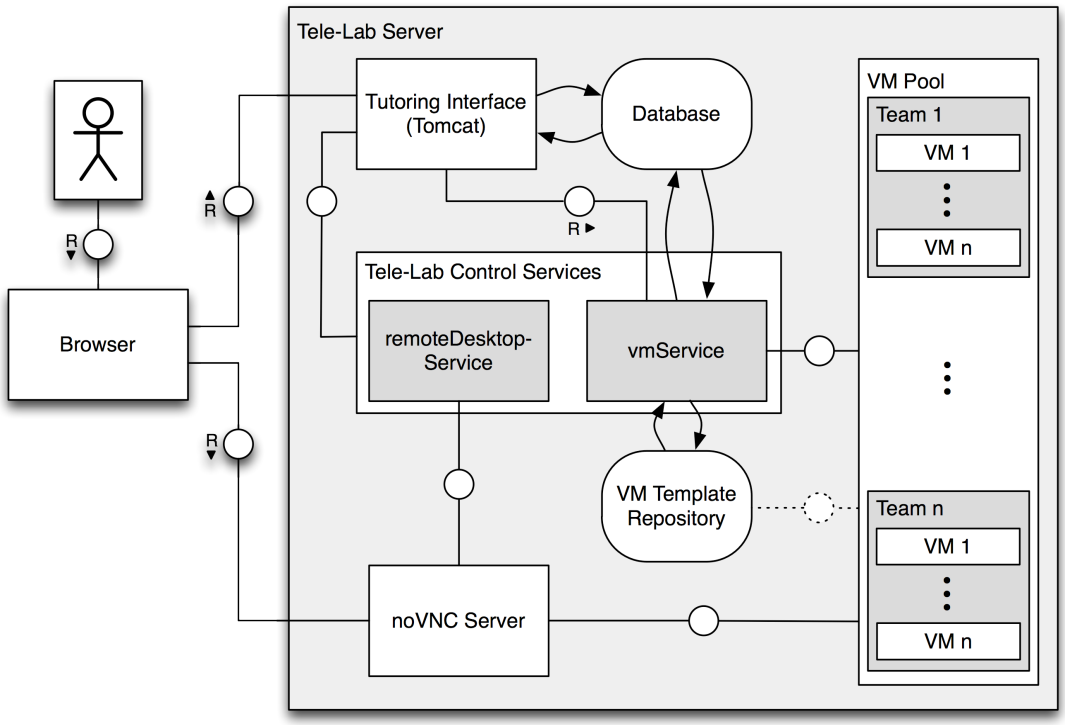

Figure 2. Overview – Architecture of the Tele-Lab Platform

The system can easily be enhanced with new content and exercise scenarios.

B. *Architecture of the Tele-Lab Platform*

The current architecture of the Tele-Lab server is a refactored enhancement to the infrastructure presented in [11]. Basically it consists of the components illustrated in Fig. 2. The following overview just explains the relevant components for the proposed enhancement.

Virtual Machine Pool: The server is preconfigured with a set of different virtual machines, which are needed for the exercise scenarios – the VM pool. Those VMs are constantly up and running to minimize the time, a user has to wait for a training environment. The resources of the physical server limit the maximum total number of VMs in the pool. In practice, a few (3-5) machines of every kind are started up. If all teams for a certain exercise scenario are in use, new instances can be launched dynamically (again depending on the resources and current load of the physical host). Those machines are dynamically connected to teams and bound to a user on request.

The current hypervisor solution used for providing the virtual machines is *KVM/Qemu* [15], [16]. The *libvirt* package [17] is used as a wrapper for the virtual machine control. *LVM* (Linux Logical Volume Management) provides virtual hard discs that are capable of copy-on-write-like differential storage. Differential storage is important to save space on the physical hard disc, because the Tele-Lab server holds so called VM templates as master images (depicted in Fig. 2 as *VM Template Repository*) and clones multiple instances of each template for use within the exercise environment. VM templates also contain configuration files defining hardware parameters like memory, number of CPUs, and network interfaces.

For the *network connections* within the teams, Tele-Lab uses the *Virtual Distributed Ethernet (VDE)* package [18]. VDE emulates all physical aspects of Ethernet LANs in software. The Tele-Lab Control Services launch virtual switches or hubs for each virtual network defined for a team of VMs and connect the machines to the appropriate network infrastructure. For the distribution of IP addresses in the virtual networks, a DHCP server is attached to every network. After sending out all leases, the DHCP server is killed due to security considerations [13].

Tutoring Interface: A web application (implemented in Grails) is the core of the Tele-Lab user interface for students. It provides learning units available to a user that consists of text, images and multimedia clips (screen casts, clips from recorded lectures). The application guides students through the content of a learning unit (presented as a sequence of *chapters*) including the practical exercise assignments. These assignment web pages (see Fig. 3) explain the students tasks (*challenge*), render a control to request VM(s) for the hands-on training and implement a questionnaire to verify the students *response* to the challenge. The structure, the actual content of a learning unit and the questionnaires are stored in the central database and can be maintained from a custom *content management system* (not depicted in Fig. 2). This CMS (actually called Tele-Lab Web Admin Interface) also allows management and monitoring of VM templates and pools.

Figure 3. Screenshot of an Assignment Web Page

The web application for tutoring also keeps track of the users' progress: it stores page visits and times the user spends on each part of a learning unit, VMs that are requested and assigned as well as questionnaire results.

Tele-Lab Control Services: Purpose of the central Tele-Lab control services is bringing all the components illustrated in Fig. 2 together. To realize an abstraction layer for the encapsulation of the virtual machine monitor (or hypervisor) and the remote desktop proxy, the system implements a suite of lightweight XML-RPC web services: the *vmService* and the *remoteDesktopService*. The *vmService* is used to control virtual machines – start, stop or recover them, grouping teams or assigning machines or teams to a user.

The *remoteDesktopService* is used to initialize, start, monitor, and terminate remote desktop connections to machines, which are assigned to students when they perform exercises. The above-mentioned Grails applications (tutoring environment and web admin) let the user and administrators control the whole system using the *Tele-Lab Control* web services suite.

Remote Desktop Access is implemented as a proxy based on the open-source project *noVNC*, a client for the Virtual Network Computing protocol based on *HTML5 Canvas* and *WebSockets* [19]. On the client side, the user only needs a state-of-the-art web browser. Actually, the current implementation of the noVNC client does not even need an HTML5-capable browser: for older browsers, HTML5 Canvas and/or the WebSockets are emulated using Adobe Flash.

C. *Virtual Machine Lifecycle*

The virtual machines used for the Tele-Lab training environment follow a circular lifecycle (see Fig. 4). When creating a new learning unit, the author sets up one or more fresh virtual machine templates (or clones existing ones), then installs tools, services or scripts for the emulation of victim activities (such as network communication or reaction or users' activities) and finally connects the new VM templates to a team template by defining virtual networking parameters (1).

For this new VM team template – representing a specific exercise experiment – the author limits a minimum number of instances running in the VM pool. The system fires up the specified number of VM teams (2). Finally, the author must connect the exercise scenario to the assignment section of the new learning unit.

Once a student requests for a VM team from the pool for a hands-on session, the Tele-Lab Control Services check for a free team instantiated from the specified template, assigns it to the user (3) and starts the Remote Desktop session. The student performs the exercise tasks and closes the browser window for the VNC client when finished. The control services shut down the virtual machines from that team (4), rolls them back to the original state (5) and starts them up again (2). The VM team is now ready to be used in the next training session.

This VM lifecycle is not specific for the Tele-Lab platform. Similar lifecycles are implemented in other computing labs based on VM technology, with variance in the persistence of VM states (not every lab is designed a roll back the experiment after execution) and in the automation of the lifecycle management (some labs platforms require user or administrator actions to trigger the next phase of the lifecycle.

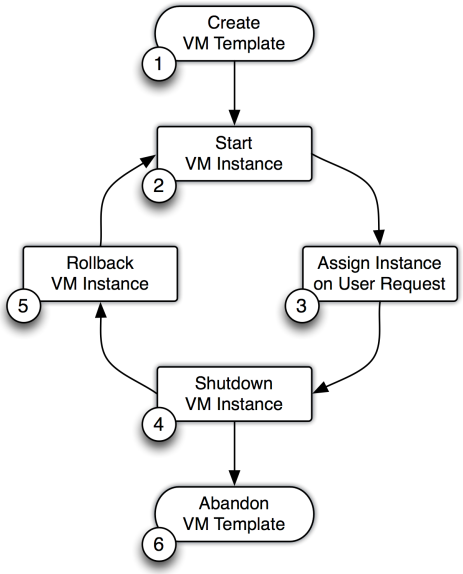

Figure 4. Virtual Machine Lifecycle in Tele-Lab

IV. Automatic Assessment for Parameterized Hands-On Exercises

A general model for assessment in hands-on exercises builds on the assertion of pre-conditions and the check of post-conditions regarding the students exercise session (see Fig. 5). This applies not only in the context of automatic assessment in virtual laboratories, but also for traditional manual assessment conducted by human tutors.

Let an exemplary assignment be the configuration of a firewall in order to prevent access to certain ports. The tutor sets up an experiment environment, i.e. by starting some services that should be protected and removing any existing firewall rules on the training system (assertion of pre-conditions). After the student finished the hands-on session, the tutor manually checks, if the student has completed his tasks, i.e. configured a firewall properly (check of post-conditions).

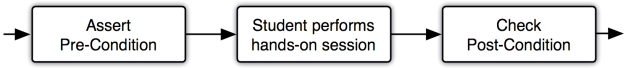

Figure 5. Pre- and Post-Conditions for Hands-On Assessment

The semi-automatic assessment mode described in [7] and already introduced in section II also matches this simple scheme. The administrators of the system manually assert that suitable virtual machines for hands-on sessions are running, which would be the pre-condition. The test script that runs the port scanner and parses it's results automatically checks the post-condition.

Concerning the assertion of pre-conditions, the automatic lifecycle management in Tele-Lab ensures that every user gets access to a fresh, unused VM team. If the author of the respective learning unit assured the proper implementation of the pre-conditions during the design of the experiment (creation of the VM template), no further action is necessary on the allocation of a concrete instance of the experiment. It has already been mentioned in sections I and II that Tele-Lab takes a different assessment approach when testing post-conditions: the assignments are designed as *challenges* that require the student to gain specific knowledge by performing hands-on tasks inside the training environment. The student can respond to the challenge by taking a quiz in the web-based tutoring environment (multiple choice or free-text question answering). If the student provides the correct answers, the post-condition check is regarded as passed.

The assessment in Tele-Lab is – among other reasons – realized in this manner due to security considerations. Comparing to the approach presented by the authors of [7] and [8], Tele-Lab does neither need any additional virtual machines on the virtual network used for the student's hands-on session, nor does it need any shared resources (networking or shared folders) between the lab machines and the physical host. Since Tele-Lab requires the students to use tools for attacking that he could also use to attack the physical host or virtual machines assigned to other users, it is an important security constraint to isolate the virtual network for a certain experiment from all other (physical and virtual) resources [12].

When revising the VM lifecycle from Fig. 4, the major limitation of this approach becomes obvious: since the virtual machines are always being rolled back to the original state, the secret knowledge to be gained to solve the assessment quiz is the same for every student, who takes the challenge (and for any repetition of an exercise). To transform this static characteristic to a dynamic behavior, we propose an enhancement to the presented lifecycle (and the underlying architecture) that consists of the parameterization of exercise scenarios and experiment setups, reconfiguration of virtual machines implementing a concrete exercise and the dynamic generation of quizzes for the student responses.

A. Parameterization of Exercise Scenarios

To be able to realize the deployment of exercise scenarios in a dynamic manner, the variable parameters of a scenario have to be defined by the author of a learning unit. For assessment purposes, we basically consider the information to be gained in the challenge as variable: these values should be unique for each instance of the exercise scenario and thus for each student and for any repetitive execution of the scenario.

For flexibility, we choose a straightforward and most generic data structure to define the parameterization of a scenario. Let a *parameter definition* be a data structure containing an *identifier (id)* for the scenario to be parameterized and a list of *parameters*. Each of the values in the *parameters* list is again defined as data structure containing

- the *name* of the parameter
- a description of possible parameter *values* (can be a range of numbers, a list of strings or list of key-value pairs)
- an amount of parameter instances to be set dynamically (*value count*)
- optional: a total amount of response options (valid and invalid) to be used to generate a multiple choice quiz (*response count*).

The dynamic parameters for the exercise scenario from the learning unit on “Attacks on Accounts and Passwords” (see section I) obviously are the user accounts and passwords inside the virtual machine. A simple parameter definition (in human readable XML format) for this scenario is defined as in Fig. 6 below:

```
<parameter-definition id="Password Security">
   <parameter name="useraccount">
      <values>
         <value key="user1">secret</value>
         <value key="user2">123456</value>
         <value key="user3">princess</value>
         <value key="user4">kitten</value>
         ...
         <value key="user100">password</value>
      </values>
      <value-count>4</value-count>
      <response-count>10</response-count>
   </param>
</parameter-definition>
```

Figure 6. Example Parameter Definition

The main part of this data structure is the list of *value* elements containing key-value pairs with arbitrary usernames and weak passwords chosen from a password list. The *value-count* of 4 determines, that four random values will become username and password of accounts inside the virtual machine. Setting the *response-count* to 10 causes the system picking another six random values that will *not* be used for accounts in the virtual machine. These six values will serve as invalid answers during the assessment.

The presented data structure is flexible enough to cover a variety of different parameters. Examples for parameters include but are not limited to:

- directives in Linux configuration files
- values of Windows registry keys
- network connections and IP addresses
- filenames and file contents
- running services
- commands to run applications or call programs on a shell (bash, cmd.exe, Windows Power Shell)

A virtual laboratory implementing this parameterization technique for experiments or exercise scenarios must provide storage for the parameterization data. This can be a database scheme or XML-based file storage.

For the convenience of learning unit authors, the system should provide a user interface for parameter creation and maintenance.

B. On-the-fly Reconfiguration of Virtual Machines

The second necessary component of the proposed dynamic e-assessment enhancement to the virtual lab is an extensible toolkit for the (re-)configuration of the virtual machines and networks that build learning environment. The solution must allow triggering reconfiguration activities for the guest operating systems in the VMs from the outside (physical host). Since the parameterization data is stored on the physical host, the enhancement must also allow passing the parameter values into the VM.

Additionally, the toolkit should be as independent as possible from the hypervisor implementation (i.e. avoiding the use of hypervisor API functionality) as well as from the guest operating systems (platform independence). Therefore, we suggest implementing the reconfiguration toolkit as a client-server architecture, where the servers run inside the virtual machines and the physical host uses a matching client to initiate the reconfiguration.

The server component (*parameterization server*) of the toolkit should be implemented modular and allow easy extension with additional modules for different configuration

tasks. For security reasons, the parameterization server should destruct itself and remove all traces after the reconfiguration has been carried out. Modules for specific reconfiguration tasks are identified by the *name* of the parameter in the *parameter definition* (see Fig. 6).

An action sequence leading to the reconfiguration is performed as follows:

1. The virtual lab management software randomly picks a set of *value-count* parameter *values*.
2. The management software uses the *parameterization client* to call the *parameterization service* on the virtual machine that should be reconfigured and passes the *values* as well as the *parameter name*.
3. The *parameterization server* in the virtual machine calls the appropriate module (identified by the parameter *name*) and passes on the *values*.
4. The module implements functionality (specific to the configuration task) and thereby configures the virtual machine according to the *values*.
5. If there is more than one *parameter* in the *parameter definition*, the previous steps are executed repeatedly.
6. After the successful application of all parameter values to the virtual machine, the *parameterization server* calls its self-destruct method to shut down, remove itself and the traces.
7. The virtual lab management software on the physical host monitors the network connection to the *parameterization server*. The connection breaks down when the self-destruction is triggered. The host system knows, that the procedure has been finished.

Continuing the example exercise scenario on "Password Security" from Fig. 6, the lab management system picks four key-value pairs representing usernames and passwords and passes those to the *parameterization server*. The server activates a module identified as "useraccount" (*name* of the exemplary parameter), which implements the creation of user accounts and setting the passwords specific to the operation system of the virtual machine.

A concrete implementation of a *parameterization server* and a corresponding *client* as well as of several modules is described in section V.

C. Generation of Dynamic Multiple Choice Tests

The final missing component is the actual assessment environment. Since the assignments are designed as challenge-response tests, this component is a tool generating questionnaires in form of multiple-choice or free-text answering quizzes. Revising the *parameter definition*, the realization of these dynamic quizzes is straightforward.

The challenge part is actually the assignment created by the author of a learning unit and must not be generated in a dynamic manner. Again referring to the "Password Security" example, the challenge is phrased as "Which passwords could you reveal with the password cracker?". The author must also define the type of the quiz for the response (multiple-choice or free-text answering).

The answers of the response are the dynamic part of the assessment. As the student must gain the necessary knowledge through the hands-on tasks in the virtual machine, the quiz must deal with the same data as the virtual machine does. Speaking in terms of the "Password Security" exercise, the quiz must recognize the passwords set in the virtual machine as correct solutions.

The general procedure for the construction of quizzes can be realized as a web application (from now on called *quiz service*) that renders and delivers the questionnaires, checks answers and gives feedback (e.g. scores). Quizzes can be multi-staged: if an exercise scenario has more than one dynamic parameter in the *parameter definition*, the assessment can cover of more than one challenge. An example for an exercise scenario with a multi-staged quiz is given later in section V with the learning unit on "Reconnaissance".

The virtual lab management software calls the *quiz service*, when the parameterization of an exercise scenario is finished (connection to *parameterization server* terminates). For each *parameter* element from the *parameter definition*, the lab management passes the following arguments to the *quiz service*, that are necessary to generate the quiz:

- the *value-count* number of valid answers (i.e. the *value* elements that have been passed to the *parameterization server* and thus have been applied in a virtual machine)
- a *(response-count – value-count)* number of invalid answers (*value* elements that have *not* been applied to the virtual machine)

Additionally, the *quiz service* receives a unique *session identifier* (i.e. the hash value of the VM identifier concatenated with a timestamp). This identifier is needed to control the web workflow of the assessment.

The *quiz service* provides three basic methods:

- *generateQuiz(id, challenges[])* is called to initiate a quiz session identified by *id*, receiving the valid and invalid answers for each challenge of a multi-staged quiz in the *challenges* array.
- *challenge(id, num)* renders and returns the HTML code for the *num*-th question (in a multi-staged quiz) that is associated to the session labeled with *id*.
- *response(id, num, answerValues[])* checks the students response on the *num*-th challenge (submitted in the *answerValues* array) and renders HTML code for the feedback.

The interaction between the lab management software and the *quiz service* is meant to be realized as web service call (e.g. XML-RPC request), while the integration of the HTML code for questions and feedback (calls to *challenge()* and *response()* methods) should be integrated with the assessment web page via AJAX requests.

D. *Adopted VM Lifecycle with Parameterized Exercises and Quiz Service Integration*

Summing up the proposed enhancements shows an extended lifecycle for the training VMs with two additional stages compared to the lifecycle presented in Fig. 4. The author of a learning unit must provide a suitable *parameter definition* after the creation of a VM template (1). In some cases, the author must also implement a new module for the parameterization service, if the intended reconfiguration can not be realized with existing modules.

When a student requests for a virtual machine (3), the assigned VM must be dynamically reconfigured before the lab management system can provide remote access. At the same time, the system must trigger the generation of the assessment environment, i.e. call the *quiz service* and pass the parameter values. The adopted lifecycle is illustrated in Fig. 7.

V. Implementation of Automatic Assessment with Parameterized Exercise Scenarios for Tele-Lab

Since the Tele-Lab management system is already implemented as a service-based architecture (*vmService* and *remoteDesktopService*) with XML-RPC web services, it was the obvious solution to also implement the parameterization system and the *quiz service* using this technology. The *quizService* has actually been implemented in two parts: the *generateQuiz()* XML-RPC method is a Python-based service that writes the challenges (i.e. valid and invalid answers) into the Tele-Lab database. The *challenge()* and *response()* methods have been integrated into the Tele-Lab tutoring application as Grails actions. The tutoring application can also access the database.

The means for authors to provide and maintain the *parameter definition* are implemented in the *Tele-Lab Web Administration Interface*, which the author also uses for learning unit content creation and the design of new VM templates. The *parameter definitions* are also stored in the central database.

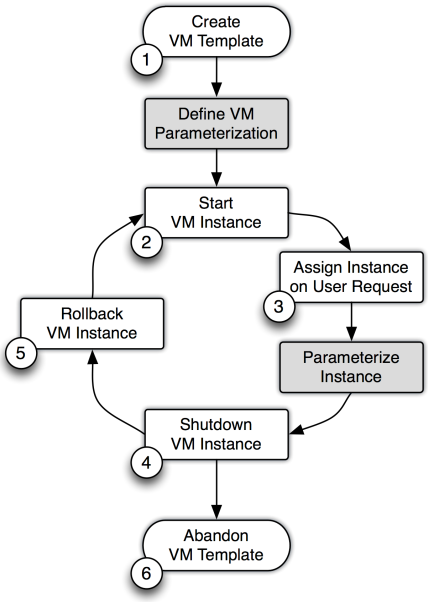

Figure 7. Adopted Virtual Machine Lifecycle

The *parameterization client* has been integrated into the *vmService*. When the student starts a request for a virtual machine, the *vmService* randomly picks the parameter values for the valid quiz answers as well as the invalid answers from the database and calls the *parameterization server* on the virtual machine that will be assigned to the requesting user. When the *parameterization server* finishes the reconfiguration tasks, the *vmService* calls the *quizService* and triggers the generation of the challenge.

The *parameterization server* is also based on Python's XML-RPC server. The server implements a plugin infrastructure for easy enhancement with configuration modules. To avoid the need of a Python environment on the virtual machines that shall be dynamically reconfigured, the core *parameterization server* is packed with the *PyInstaller* tool, which generates stand-alone executables out of Python scripts. Specific modules (plugins) for the reconfiguration tasks are loaded into the *parameterization server* at runtime. The server is provided for Linux and Windows operating systems. The *parameterization server* must not be integrated into virtual machines at the design stage. The Tele-Lab system automatically injects the server into the hard disk images of the virtual machines prior to the first startup.

The modules for reconfiguration usually implement a sequence of system calls. Coming back to the "Password Security" example, the reconfiguration tasks are generating user accounts and setting the corresponding passwords. A module for performing these tasks in a Linux environment is implemented as follows:

```
@export
def create_user(user, pass)
   cmd = "useradd '%s'" % user
   error = shell_with_error_handling(cmd)
   if(!error)
      error = set_password(pass)
   return error
create_user.supported_os("Linux")
```

Figure 8. Module for Adding a System User in Linux VMs

The equivalent module for setting the user's password (called as *set_password* in Fig. 8) is implemented using the *Pexpect* library, since the Linux *passwd*-command requires user interaction.

The same module for Windows is implemented analogous, but since the Windows net command does not require interaction, the password is set directly in the module:

```
@export
def create_user(user, passwd)
   cmd = "net user '%s' '%s'" % (user, passwd)
   return shell_with_error_handling(cmd)
create_user.supported_os("Windows")
```

Figure 9. Module for Adding a System User in Windows VMs

Similar modules have been implemented for e.g.

- deleting users accounts
- adding and removing groups
- changing group membership of user accounts
- starting and stopping services
- changing IP and netmask addresses of network interfaces
- creating files with specified content
- adding and removing lines from (configuration) files
- etc.

Continuing the example, the *vmService* connects to the *parameterization server* of the virtual machine and subsequently calls the *create_user()* method for each *value* element given in the *parameter definition* in order to add new user accounts and set the passwords. Finally, it calls the *destroy_self()* method to cause the Python service to shutdown and remove itself.

To illustrate the usage of the implementation of the proposed system in the Tele-Lab platform, the following sections present two additional learning units with parameterized exercise scenarios.

A. Use Case: (Online) Reconnaissance Scenario

A learning unit on "(Online) Reconnaissance" is basically about host discovery and service discovery (port scanning) in the context of attack preparation. The student learns about the general concept of ports and services and relevant techniques in this domain, such as ping sweeping or ARP sweeping for host discovery, a number of port scanning variants and the concept of banner grabbing for service identification.

Subsequently, the learning unit presents a number of tools that implement the introduced reconnaissance techniques, e.g. *thc-rut* [20] for ARP sweeping, the *nmap* security scanner [21] for ping sweeping and port scanning or *telnet* for service identification.

The challenge issued in the exercise section of the learning unit asks the student a) to find all hosts on the local network of the virtual machine and b) to identify all services running on the found machine(s). In fact, there are only two machines on the virtual network: one for the student (Linux, equipped with all necessary tools) and one as target for the scans (Linux, a large number of different services – e.g. Apache, MySQL, ProFTPd, etc. – are installed, but not running).

The quiz for the response to this challenge is multi-staged: the first question asks for the IP addresses of the found hosts (free-text answer), the second question is a multiple-choice test, where the student has to check all found services.

Parameterization for this exercise scenario affects the IP address of the target machine and the running services. When initiating the parameterization, the *vmService* of Tele-Lab selects four services from a list of all available services that are started up via calls to the *parameterization server*. After that, the last block of the target's IP address is set to a random value between 2 and 254 (1 is reserved for the attacker VM). Note, that the change of the IP address must be the last reconfiguration task, because the *parameterization client* will loose the connection during reconfiguration. Changing the IP address implicitly causes self-destruction of the *parameterization server*.

The student has to perform host discovery to be able to provide a valid response to the first stage of the assessment, and port scanning as well as banner grabbing on the identified open ports to be able to select the valid services in the second assessment stage.

B. Use Case: Remote Exploitation Scenario

A learning unit on "Remote Exploitation" introduces the student to the concept of buffer overflows and exploits that utilize those, to provide a remote shell on the victim's computer.

The student learns about memory layout, vulnerabilities and exploits, buffer overflows, shell code and NOP sleds and is finally introduced to the *Metasploit framework* [22] and its usage.

The exercise scenario for this learning unit again consists of two virtual machines on a virtual network. The machine for the student is a *Backtrack* Linux distribution [23] (a penetration testing suite equipped with a large number of scanners and attack tools). The virtual victim is a Windows XP system with Internet Information Server, which is vulnerable to a stack corruption in the SMB service described in CVE-2008-4250 and the IIS FTP Server NLST Response Overflow described in CVE-2009-3023 [24].

The challenge issued for this exercise scenario is to find a file called *secret.txt* on the victim's machine and steal its content. The valid response to this challenge is the stolen data. In order to get access to the secret file, the student must scan the local network for the IP address of the victim's host, and then use a port scanner or vulnerability scanner to find possible vulnerable services and finally use Metasploit to exploit on of the vulnerable services and gain remote access.

The parameter for this exercise scenario is the path and the content of the secret file. The *parameter definition* holds a number possible file names and random strings (as key-value pairs). This secret file is often called a "flag" that has to be captured. The student can only provide the knowledge of the flag during assessment, if she had gained remote access to the victim machine.

Such capture-the-flag challenges can be applied to various exercise scenarios, e.g. in a learning unit on "Eavesdropping on Network Traffic".

VI. Conclusion and Future Work

The paper at hand proposes a system for automatic (self) assessment with hands-on exercises in virtual (remote) computer laboratories. The presented architecture is independent from the underlying hypervisor and from the guest operating systems in the training machines. The system can be

applied for virtual laboratories with automatic lifecycle management and web-based assessment with challenge-response like exercise assignments.

The system has been successfully implemented for the Tele-Lab platform; several exercise scenarios have been adopted for dynamic parameterization.

An important part of future work on this topic will be the extensive evaluation of the system as well as in supervised classroom sessions as in self and distance learning usage. Another future activity aims on detaching the implementation of the assessment and parameterization components from the Tele-Lab platform for easy integration into other virtual laboratories.

REFERENCES

[1] C. Border. "The development and deployment of a multi-user, remote access virtualization system for networking, security, and system administration classes", SIGCSE Bulletin, 39(1): pp. 576–580, 2007.

[2] W. I. Bullers, Jr., S. Burd, A. F. Seazzu. "Virtual machines – an idea whose time has returned: application to network, security, and database courses", Proc. 37th SIGCSE technical symposium on Computer science education, Houston, Texas, USA, 2006.

[3] A. Gaspar, S. Langevin, and W. D. Armitage. "Virtualization technologies in the undergraduate IT curriculum", in IT Professional, vol.9 (4), IEEE Computer Society, 2007, pp. 10–17.

[4] J. Hu, M. Schmitt, C. Willems, and C. Meinel. "A tutoring system for IT-Security", in Proceedings of the 3rd World Conference in Information Security Education, p. 51–60, Monterey, USA, 2003.

[5] J. Hu and C. Meinel. "Tele-Lab IT-Security on CD: Portable, reliable and safe IT security training", Computers & Security, 23:282–289, 2004.

[6] J. Hu, D. Cordel, and C. Meinel. "A Virtual Machine Architecture for Creating IT-Security Laboratories", Technical report, Hasso-Plattner-Institut, 2006.

[7] J. Keller and R. Naues. "A Collaborative Virtual Computer Security Lab", in Proc. 2nd IEEE International Conference on e-Sciece and Grid Computing (e-Science '06), IEEE Computer Society, Amsterdam, Netherlands, 2006.

[8] H. A. Lahoud and X. Tang. "Information Security Labs in IDS/IPS for Distance Education", in Proc. 7th Conference on Information Technology Education (SIGITE '06), ACM Press, pp. 47–52, 2006.

[9] S. Roschke, C. Willems, and C. Meinel. "A Security Laboratory for CTF Scenarios and Teaching IDS", Proc. 2nd Intl. Conference on Education Technology and Computer (ICETC 2010), IEEE Press, Shanghai, China (2010), pp. 433-437.

[10] K. Webb, M. Hibler, R. Ricci, A. Clements, J. Lepreau. "Implementing the Emulab-PlanetLab Portal: Experience and Lessons Learned" in Workshop on Real, Large Distributed Systems (WORLDS), 2004.

[11] C. Willems and C. Meinel. "Tele-Lab IT Security: an Architecture for an Online Virtual IT Security Lab", in International Journal on Online Engineering (iJOE), Vol. 4 No. 2 (2008), pp. 31-37.

[12] C. Willems, T. Klingbeil, W. Dawoud, and C. Meinel. "Security in Tele-Lab – Protecting an Online Virtual Lab for Security Training", in Proc. 2009 ELS workshop (in conjunction with 4th ICITST) on E-Learning Security (ELS 2009), IEEE Press, London, UK, pp. 1-7, 2009.

[13] C. Willems and C. Meinel. "Practical Network Security Teaching in an Online Virtual Laboratory", Proc. 2011 Intl. Conference on Security & Management (SAM 2011), CSREA Press, Las Vegas, Nevada, USA, 2011.

[14] Golisano College of Computing and Information Sciences. (2007) Networking and Systems Security Laboratory website [Online]. Avaliable: http://nssa.rit.edu/?q=node/52, accessed: 2011-11-18

[15] F. Bellard. (2011) QEMU – Open Source Processor Emulater homepage. [Online]. Available: http://www.qemu.org/, accessed: 2011-11-18

[16] Red Hat, Inc. (2011) Kernel-based Virtual Machine (KVM) homepage. [Online]. Available: http://www.linux-kvm.org/, accessed: 2011-11-18

[17] The Libvirt Developers. (2011) libvirt – The virtualization API homepage. [Online]. Available: http://libvirt.org/, accessed: 2011-11-18

[18] R. Davoli. (2011) Virtual Distributed Ethernet homepage. [Online]. Available: http://vde.sourceforge.com/, accessed: 2011-11-18

[19] J. Martin. (2011) noVNC project website. [Online]. Available: http://kanaka.github.com/noVNC/, accessed: 2011-11-18

[20] The Hackers Choice. (2003) thc-rut website. [Online]. Available: http://www.thc.org/thc-rut/, accessed: 2011-11-20

[21] Lyon, Gordon. (2011) Nmap security scanner website [Online]. Available: http://insecure.org/nmap/, accessed: 2011-11-20

[22] Rapid7. (2011) Metasploit Penetration Testing Software [Online]. Available: http://metasploit.com/, accessed: 2011-11-20

[23] The BackTrack Developers. (2011) BackTrack Linux – Penetration Testing Distribution [Online]. Available: http://www.backtrack-linux.org/, accessed: 2011-11-20

[24] Mitre Corporation. (1999) Common Vulnerabilities and Exposures website [Online]. Available: http://cve.mitre.org/, accessed: 2011-11-20