The final version was published in: M. Frank, M. Leitner and T. Pahi, "Design Considerations for Cyber Security Testbeds: A Case Study on a Cyber Security Testbed for Education," 2017 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech), Orlando, FL, 2017, pp. 38-46. doi: 10.1109/DASC-PICom-DataCom-CyberSciTec.2017.23

Design Considerations for Cyber Security Testbeds: A Case Study on a Cyber Security Testbed for Education

Maximilian Frank, Maria Leitner, Timea Pahi
AIT Austrian Institute of Technology
Center for Digital Safety & Security
Vienna, Austria
firstname.lastname@ait.ac.at

***Abstract*—Educational testbeds have been developed for many years. Within the past ten years, the development of cloud-based storage architectures as well as the facilitation of memory and storage technology allowed for the building of small to medium-sized testbeds at low or medium cost. These developments provide the foundation for the development of educational testbeds that can be used for cyber security training and exercise of various target groups (e.g., students, IT professionals, engineers) in many domains (e.g., cyber security, IoT, Industry 4.0). Testbeds have been well established within the information security community (e.g., malware analysis, cyber security experimentation, etc.). However, these testbeds often require a certain level of maintenance or resources and were therefore not often used in non-expert communities. However, it is essential that testbeds gain a wider audience in order to enable many different groups cyber security skills and competencies. In this paper, we analyze how an educational testbed could be designed by (1) examining established testbeds in research and education and (2) analyzing how typical testbeds are designed. Based on this, we propose a design life cycle, i.e. a methodology to facilitate the development of cyber security testbeds. We demonstrate our findings in a case study. In the study, we designed and implemented a cyber security testbed for educational purposes using open source technology. The results and reviewed literature validate the design life cycle and show dependencies between the underlying technology of the testbed and the designed challenges. These findings contribute to the overall development of testbeds and can be used as basis for future work. We plan to further extend this testbed in order to develop an automated and flexible cyber security testbed.**

***Keywords*-cyber range; cyber security; education; testbed; training;**

I. Introduction

Educational testbeds have become an essential part in cyber security education, training and exercises. New technologies such as cloud computing enabled the facilitation and implementation of testbeds as well as the ability to scale quickly up or release resources.

Particularly in cyber security education and exercises, testbeds are used to support the objectives of training or exercise. For example, to establish a testbed for SCADA systems [1], [2], to teach and train cyber security incidents [3] or for cyber security experimentation and test [4], [5]. Furthermore, testbeds can be used to develop skills and competencies for the use and interpretation of common operating pictures [6]. Although testbeds have been well-developed, there is only few information online on the design and implementation of testbeds for cyber security education, training and exercise in order to understand the architectural decisions and requirements to develop these testbeds. Developments within the academic and research often provide the source code but it is often not easy to understand documentation and setup (e.g., for beginners). In addition, setting up these systems can become time consuming and cumbersome particularly for non-experts. Another reason is that many cyber security testbeds have been developed in the military domain (e.g., [8], [9]) often referred to as cyber range or national range.

In order to fill this gap and to discuss more design considerations for cyber security testbeds, we aim to assess typical functionality and development methodologies for cyber security testbeds. In this paper, we assume that cyber security challenges, i.e. a task or activity to be solved by a participant, can be deployed within a testbed. Then, a participant, further called challenger, will be able to work on the challenge in order to solve it (e.g., in typical capture the flag (CTF) contests or training sessions on IT security). It is challenging to provide a testbed that enables challenges not only for beginners but also for advanced learners. In addition, new technologies such as cloud computing provide lower cost to setup and maintain testbeds than 10 years ago. Hence, nowadays it is easier to setup and remove testbeds but the challenge creating adequate and sophisticated content still remains.

In order to develop challenges on top of a testbed, we assess current literature and derive a methodology, a *design life cycle model*, to design and implement cyber security testbeds. To the best of our knowledge, current literature does not provide a methodology to design cyber security testbeds. To demonstrate the validity of our life cycle model, we conducted a case study in summer 2016. We used open source technology to design a virtual environment that automatically deploys challenges for challengers. Specifically, we used OpenStack technology to design and implement the cyber security testbed as well as the software (e.g., OS,

Applications) for the challenges. The results of the case study show that it requires many steps to actually establish a testbed and create feasible and deployable challenges within. This research will contribute to a set of ongoing research activities focusing on cyber security education using testbeds. For future work, we aim to establish a flexible testbed that is open source and can be used by various target groups.

The rest of this paper is structured as follows. Section II outlines related work. Section IV defines the life cycle for setting up testbeds and challenges. Section V demonstrates the findings within a case study. Section VI concludes the paper.

II. Background

In the past 10 years, testbeds, experimentation environments or others have been established in cyber security. Each testbed has often an individual design and architecture. To demonstrate the variety of testbeds, we will summarize selected cyber security testbeds in Table I. For example, [3] proposes a testbed for network security education using live exercises for graduate students. This testbed has been further developed throughout the years and used for the UCSB iCTF[1] contest [10]. Other testbeds exist that are used for capture the flag (CTF) contests[2] such as DEFCON CTF[3] or Google CTF[4]. Another example is the cyber security framework proposed in [11]. Furthermore, the DETER project developed a testbed for experimental research in cyber security [4], [5], [12], [13]. Furthermore, smart grid testbeds have emerged that focus on testing power grid systems (e.g., SCADA) such as [14]–[16]. These testbeds often combine physical devices with virtual systems in order to set up experiments (e.g., virtual power systems [2], [17]). Moreover, national cyber ranges are testbeds that often incorporate command and control functionality as well as experiments (e.g., [8], [18]).

Table I
Selected cyber security testbeds for research and education

Ref#	Purpose
[3], [10]	Hands-on network security exercises, iCTF
[15], [16]	Smart grid testbed
[4], [5], [12], [13]	Experimental research in cyber security
[14]	SCADA testbed
[2], [17]	Virtual power system testbed
[8], [18]	National cyber range testbed
[11]	Cyber security training framework

Most of this related work does not fully outline the design considerations for testbeds; mostly they describe the architecture of the testbed. To the best of our knowledge, current literature does not provide a methodology for setting up or comparing cyber security testbeds. While we will describe a testbed design life cycle, we will leave discussion on comparing testbeds for future work.

[1]UC Santa Barbara International Capture The Flag: https://ictf.cs.ucsb.edu/

[2]An overview of CTFs is outlined at https://ctftime.org/

[3]https://www.defcon.org/html/links/dc-ctf.html

[4]https://capturetheflag.withgoogle.com/

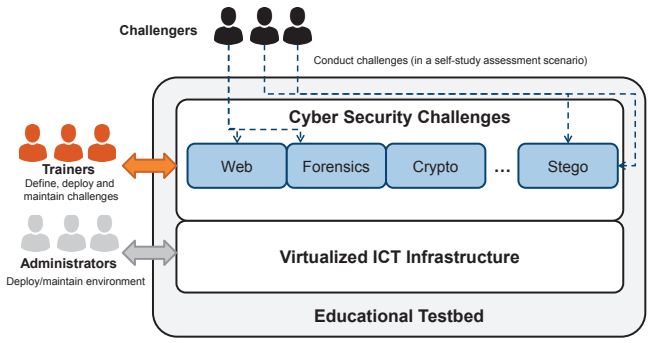

Figure 1. Example setup of a cyber security testbed

III. Typical Design Considerations for Cyber Security Testbeds

This section describes typical design considerations for testbeds based on related work and ongoing research projects. Figure 1 shows a schematic example of a layout of a cyber security testbed. On top of virtual ICT infrastructure, security challenges of different categories are provided. The challenges can have different contents (e.g., web, stego or crypto, see Section III-A1) depending on the curriculum or course goals. Challenges are typically defined and implemented in a collaboration between administrative staff and trainers (e.g., lecturers in an university or professional trainers in a cooperation or training facility). Challengers may conduct security challenges. In a self-assessment case study, for example, challengers would be able to start and conduct security challenges themselves (as shown in Figure 1) while in supervised trainings, trainers might start the security challenges for challengers.

A. Security Challenges

The analysis of software systems for security vulnerabilities or the examination of data or programs is a large part of cyber security (or IT security). Security challenges try to take parts of these tasks into a gamified context and turn them into something similar to a puzzle. The goal of these puzzles are to teach specific principles or security issues in a playful and practical way. Most of the time a challenger must find a secret flag as the goal of a challenge. This is a very useful learning tool for security professionals. They can learn about common problems, by exploiting them their selves and building an understanding on how and why they occur. Without the need to endanger or examine the complexity of a real system.

1) Challenge Types: IT security spans over many different areas due to this, there also exist different types of security challenges. The most common types are described below.

- **Web:** As the name suggest Web challenges are about websites and webapps. In these challenges a challenger has to find vulnerabilities on or in a given website or webapp to gain access to a secret flag or the server hosting the service itself. This helps in understanding security concepts and practices in the context of the web.
- **Forensics:** This type of challenge is about data. A challenger is tasked with finding certain information that is hiding in a given data set like log files, network traffic or memory dumps. Skills learned in this category can be utilized in incident response scenarios.
- **Crypto:** Cryptography is an important part of IT security. In crypto challenges a challenger learns important concepts and protocols of cryptography. Challenges of this type often require a challenger, to break primitive cryptographic protocols or incorrect implementations, to decrypt secret messages. This gives a challenger insight into and a better understanding of the complex world of cryptography.
- **Reversing:** Reverse Engineering is an important skill for many security analysts that can be very useful when examining malicious programs. Reversing challenges require the challenger to work out what a binary file is doing to reveal a secret flag.
- **Exploitation:** This type is somewhat similar to reversing, but here the focus is more on finding vulnerabilities in a given binary or application, often with known code, and actively exploiting them. A common type of exploitation challenges are "set user id"-binaries with buffer overflows where the challenger has to execute arbitrary commands with elevated privileges. Skills and information learned in this category improve a challengers defensive and offensive security skills.
- **Stego:** Stenography is the art of hiding data. In some situations the mere existence of data, even if it is encrypted, can be dangerous for a person. For example, in suppressive regimes activists might have information on their electronics that is deemed illegal by their government. Encrypting such data would not protect them from prosecution as the government will most likely force them to decrypt all encrypted information. Therefore they have to hide it so it looks like there is no data at all. Stego challenges require a challenger to find such data. Challenges often use weak or incorrectly applied stenography techniques, as well designed stenography requires enormous resources to break.

2) Providing security challenges: Now that there is some clarity on the definition of security challenges there is a need to define the requirements of a system providing such challenges. Looking at the different types of challenges available one can see that they often require some sort of server, a specific operating system or access to specific files. For example web challenges always require one or more web servers to run (at least) a vulnerable application. These are basically the same requirements small to medium sized businesses have for hosting their services or internal systems. This is not all too surprising as most challenges try to simulate such environments just on a smaller scale. Now if one wants to provide security challenges to a user base may it be the public or paying customers one also has to provide the appropriate infrastructure. There are multiple different approaches, with which this can be achieved.

The most straight forward approach would be simply replicating the servers and providing challengers access to them. This is not really cost efficient and might also compromise servers hosted in the same domain, because of the fact that security challenge servers are deliberately insecure. Due to the nature of some challenges multiple challengers might not be able to use a single server at the same time. Therefore user and server isolation needs to be implemented. Servers might also have to be reset to a previous state after a challenger solved the challenge. All these concerns and requirements allow for different infrastructure designs and concepts. Some possibilities and their underlying technologies are explained below.

Security challenges as a local application: To solve the problem of user isolation, it is possible to provide security challenges locally utilizing a challengers own system. This can be done through desktop virtualization technology. In such a system security challenges requiring servers or a specific operating systems can be provided as virtual machine (VM) images. A system like that would be a three layered architecture. Providing a system based on local virtualization has the advantage of requiring very little resources on the provider site apart from hosting copies of the interface and images. This reduces costs incurred by the provider, but challenges requiring more complex and resource intensive server setups might be too much for the hardware of challengers.

Security challenges as a service: In contrast to providing security challenges using a local application there is also the possibility to provide challenges as a service. Using such an approach a single challenge would be packaged as a service that can be started on demand. This lets the provider dynamically allocate resources as demand for challenges fluctuates. Additionally the challenger does not need to provide a strong personal computer to run challenge servers on a VM.

These initial design considerations show that it is important to specify what kind of challenges are selected and which technology can support the use of these challenges adequately.

IV. Design Life Cycle for Testbeds

This section describes the overall setup process derived from the case study and compared and harmonized with literature (see Section II). The overall process of the design and setup of a cyber security testbed is shown in Figure 2. It can be seen from the figure that this setup life cycle is divided into two aspects: setting up the environment (e.g., ICT infrastructure such as networks, hard- and software) and configuring specific challenges that should be conducted on top of the environment. A challenge is a task, activity or experiment (or similar) that is conducted by a challenger (in the testbed).

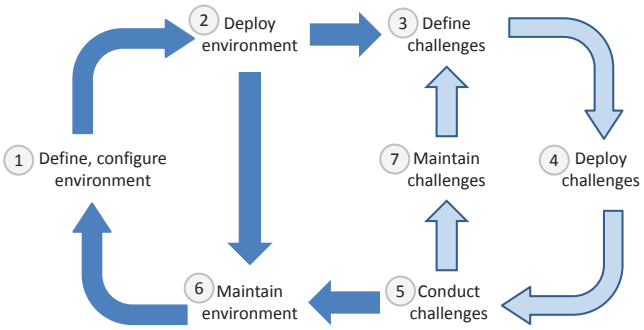

Figure 2. Design life cycle for testbeds

It can be seen from Figure 2 that the cycle is divided into two parts. The first part (colored in dark blue) contains all activities around setting up the main environment (e.g., defining the hardware, virtual infrastructure and networks). One of the main architectural questions is to define whether to use local applications or a cloud environment for setting up the virtual environment (see previous section). The second part in the figure (colored in light blue) focuses on setting up and maintaining the challenges on top of this infrastructure such as OS, software or applications (cmp. Figure 1). Main questions here are concerned with the use of open source or commercial software, the scalability or complexity of the challenge. Another important note is to prepare challenges or tasks in such a way so that e.g., experiments go as expected or in CTFs only one vulnerability can be identified (and no others) within a challenge.

It can be seen from Figure 2 that it can become very complex and cumbersome to setup and maintain the environment and challenges as there are many aspects to consider such as software maintenance, increasing threats and new vulnerabilities.

In the following, we will demonstrate the validity of this life cycle and describe what choices we made in the case study in order to set up a cyber security testbed for education.

V. Case Study: A self-study Testbed for Education

This case study describes how a cyber security testbed can be defined from scratch. Due to page limitations, we will only describe the first five steps outlined in Figure 2: (1) define, configure environment, (2) deploy environment, (3) define challenges, (4) deploy challenges and (5) conduct challenges. Maintaining challenges and environment should be straightforward and are therefore out of scope of this section.

A. Define, Configure Environment

Our motivation to set up a cyber security testbed was to develop an environment that supported these requirements:

- **Automated deployment of environment and challenges:** The virtual machines and challenges can be automatically deployed and challenges are already injected into the environment. This allows the challenger to start immediately and not have to start a script that prepares the injects for current challenges.
- **Reuse of challenges:** Challenges for students should be easy to start and restart in order to take challenges again and increase the learning effects.
- **Maintain low cost of platform:** The utilization of an open source technology allows challengers and educational institutions to keep the license cost to use the platform to a minimum. However, other license models such as academic licenses might exist that could also maintain a relatively low cost.
- **Keep high availability and of platform:** Enable a platform for cyber security education that is available and is further developed within a transparent and open community.
- **Enable *challenge as a service*:** A self-study platform should enable (1) trainers to select challenges for challengers but also (2) challengers to select the challenges they want to participate in case of self-study assessments.

With these requirements, we've selected Open Stack (https://www.openstack.org/) as an open source technology to enable private and public clouds. Furthermore, we've drafted several challenges that we aim to implement in our testbed. For demonstration purposes we will describe the challenge "SQL Injection Challenge" (see Section V-C).

B. Deploy Environment

Based on the requirements outlined in the previous section, we set up a test environment that is outlined in Figure 3. The test environment consists of three machines two of which are virtual machines.

- **VirtualBox Host:** The VirtualBox host machine is the only physical machine in the test setup. It is used to run two virtual machines, one is used for development

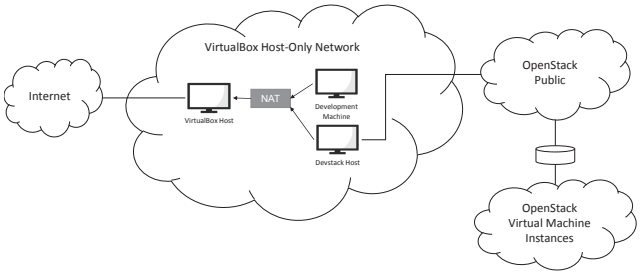

Figure 3. Testbed Environment (Example)

of challenges and the other one is used to host the Devstack. VirtualBox is configured to enable internet access for both virtual machines through a NAT adapter. There is also a Host-Only adapter configured allowing communication between the three machines.

- **Development Machine:** In this test scenario a linux mint virtual machine was used for development. OpenStack virtual machines also use the development machine as Git server to download challenge sources.
- **Devstack Host:** For Devstack, an Ubuntu server virtual machine is created in VirtualBox. It was configured to forward all traffic to allow access to public OpenStack instances from within the VirtualBox Host-Only subnet. Using the OpenStack platform it is possible to host multiple virtual machines on the Devstack host. In a default Devstack installation there are two subnets for virtual machines public and private. OpenStack virtual machines can be accessed from within the VirtualBox Host-Only subnet if they are either spawned in the public subnet or are assigned an IP from it.

C. Define Challenges

The designed testbed should support different types of challenges as outlined in Section III-A1. Furthermore, each challenge can be classified into categories such as easy, medium or hard. Challengers can have the same background information or training. For example in schools or universities the students are taught exactly the same things before taking a test about a specific subject. Based on this common background it is possible to say that one test is hard and another is easy. For security challenges this common background (unless its in a cyber security curriculum) is often difficult to grasp. Information technology is an enormously huge field. People engaging in this field have vastly different backgrounds, specializations and interests. The same is also true for the field of IT security which is only a subset of information technology. Some people specialize in organizational security aspects, which are less technical, some specialize in malware analysis, which is highly technical. While categorizing challenges by difficulty is hard, it is still something that should be done. Providing challengers with a rough estimation of a challenges difficulty not only helps them decide which challenges they might want to tackle it also encourages them when they solve challenges of high difficulty.

In this paper, we provide a running example, a web challenge, to demonstrate the steps (3) define challenges, (4) deploy challenges and (5) conduct challenges (see Figure 2).

1) SQL Injection Challenge (Web): Injection attacks try to inject context specific data, which results in unintended system behavior, into input vectors. SQL (Structured Query Language) Injection attacks try to inject SQL code into database queries to invoke attacker controlled database operations [19]. In real world scenarios an attacker might try to gain access to the system or just information stored in the database. For this example challenge a challenger has to find a secret code or flag. We prepared three different variations based on the same context and principles. Each variation has its own secret code and relative difficulty level (basic, easy and medium). In the context of this paper we will only describe the basic SQL injection example.

Basic SQL Injection: The challenge is based on the context of a private microblogging service. Microblogging services allow users to post short message to public or private boards. The microblogging service in the challenge is completely private and only allows registered users to view published posts. Challengers are not able to register an account on the microblogging platform as this functionality is disabled. They are also not supplied with login credentials. A challengers task is to successfully log into the microblogging platform and retrieve the secret posted by the user *AIT*.

On the provider side the microblogging service used in the challenge has to be developed. This section will describe key components of the developed application along side snippets of their code.

The microblogging service is built with *python flask,* using *flask-sqlalchemy* as its database driver and *sqlite3* as its engine. Using flask allows us to simply define web page views, paths and a database model without doing much extra work.

For this challenge a challenger is supposed to log into the microblogging service without a password by use of a SQL injection attack. To make this possible the login function of the service is deliberately designed to be vulnerable to SQL injection. The actual login functionality can be seen in Listing 1.

For the login the username (email) and the password supplied by the challenger are directly inserted in a database query. The query searches for a registered user with the given username and password. A login is regarded as successful as long as the query returns a row. Due to the design of the login check and the SQL injection vulnerability it is possible to login without a password by making the query return at least one row. This can be done by making the *SQL WHERE* clause of the query a tautology.

```
def login():
    """For GET requests, display the login form.
    For POSTS, login the current user
    by processing the form."""
    print db
    form = LoginForm()
    if form.validate_on_submit():
        with db.engine.connect() as con:
            statement = "SELECT email AS
    user_email, password AS user_password,
    authenticated AS user_authenticated FROM user
    WHERE email='"+form.email.data+"' AND
    password='"+form.password.data+"'"
            rs = con.execute(statement)
            user = rs.first()
            if user:
                print user
                email,password,authenticated =
    user
                user = User.query.get(email)
                user.authenticated = True
                db.session.add(user)
                db.session.commit()
                login_user(user, remember=True)
                return redirect(url_for("index"))
    return render_template("login.html", form=form
    )
```

Listing 1. Login function

D. Deploy Challenges

The sections below discuss how deployment of a security challenge into a OpenStack implementation can be automated. A proof of concept application that can deploy the challenges is described below.

1) Configure Environment with OpenStack Features: This sections describes features of OpenStack that are used when automating instance deployment.

Cloud-Init: When launching virtual machines in OpenStack it is possible to pass an initialization script called cloud-init through the userdata. The script can be any kind of shell script bash,python,sh, etc. or it can also be a cloud-config. Cloud-configs start with *#cloud-config* and are YAML based (for details on cloud-config see[5]). Using cloud-init it is possible to automate initialization and installation of applications. For the proof of concept challenge specific cloud-configs are passed to new virtual machines. Using the supplied cloud-config Cloud-Init can install and start a challenge.

Images: Virtual machines are saved on images, booting an image starts the virtual machine. In OpenStack images are managed by Glance. Glance assigns an ID to every image. These IDs are then used by Nova to start instances of the virtual machine images.

Flavors: In addition to images OpenStack virtual machines can also be started using different flavors. In OpenStack a flavor is a virtual machine configuration defining the resources of an instance. Flavors define available disk space, memory and other resources.

2) Proof of Concept Application: In order to be able to provide security challenges as a service using OpenStack the deployment of a challenges has to be automated. As implementing a complete system, that can be used by security challenge providers, is beyond the scope of this document we created a proof of concept application. The proof of concept only implements the functionality to start challenges all other functions like user management, adding new challenges, etc. where not implemented.

Some challenges require that a virtual machine instance is started, others only require a file. In the context of an OpenStack based platform this means that access to Nova and Swift services is required. Nova and Swift are both accessible via REST APIs and there are many different SDKs making using these APIs easier. For our proof of concept application we decided to use the Python *apache-libcloud* library. Using *apache-libcloud* we created a Python application that reads challenge information from a database and depending on the challenge either starts a virtual machine instance or downloads a file from a Swift container.

Database Model: Because some challenges require virtual machines and some only a file download we had to create a database model which allowed our application to differentiate between these two types.

For this we used Pythons *SQLAlchemy* library which allows us to define ORM (Object Relationship Mapping) with Python classes. The database model uses a basic challenge class which only has an id and a name. This base class is then extended by two classes, *python ComputeChallenge* and *python StorageChallenge* . This relationship is shown in Figure 4.

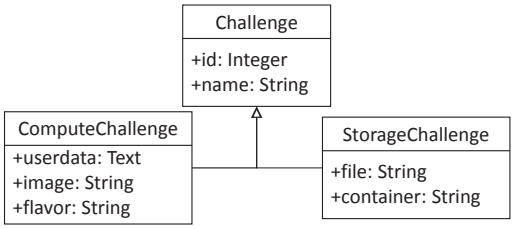

Figure 4. Class Hierarchy

ComputeChallenge represents challenges that require virtual machine instances. It has three fields:

- userdata (see Section V-D1)
- image (see Section V-D1)
- flavor (see Section V-D1)

StorageChallenge is used for challenges that only require file download. It has two fields.

- file: The name of the file that needs to be downloaded.
- container: The name of the swift container it is stored in.

[5]http://cloudinit.readthedocs.io

```
    '''
        Initialize and connect to the OpenStack
    Platforms Nova and Swift service.
    '''
    def __init__(self,name,instance_name):
        __name__ = name
        self.instance_name = instance_name
        auth_username = 'admin'
        auth_password = 'nomoresecret'
        auth_url = 'http://X.X.X.X/identity/v2.0/
    tokens'
        project_name = 'demo'
        region_name = 'RegionOne'
        #connect to nova
        self.provider = get_driver(Provider.
    OPENSTACK)
        self.nova = self.provider(auth_username,
          auth_password,
            ex_force_auth_url=auth_url,
            ex_force_auth_version='2.0_password',
            ex_tenant_name=project_name,
            ex_force_service_region=region_name)
        #conncet to swift
        self.swift_provider = get_storage_driver(
    StorageProvider.OPENSTACK_SWIFT)
        self.swift = self.swift_provider(
    auth_username,
            auth_password,
            ex_force_auth_url=auth_url,
            ex_force_auth_version='2.0_password',
            ex_tenant_name=project_name,
            region=region_name,
            ex_force_service_type='object-store',
            ex_force_service_name='swift')
```

Listing 2. Connecting to the services

Application Code Description: This section explains some important parts of the proof of concept application.

Connecting to Nova and Swift

In order for our application to be able to access Nova or Swift functions it first has to connect to both services. Using *apache-libcloud* this is done by first getting the correct driver for the service and then connecting through the driver (see Listing 2).

Start Compute Challenge Instance To start a compute challenge the proof of concept application has to create a new virtual machine instance with the correct base image, flavor and configuration (see Listing 3).

Download Storage Challenge File For storage challenges the proof of concept application needs to retrieve the challenge file from swift. The *apache-libcloud* API gives us the option to download files directly (see Listing 4) or as a stream (see Listing 5). A swift file stream can be used as HTTP response in case of a web based testbed interface.

E. Conduct Challenges

The challenge *Basic SQL Injection* (cmp. Section V-C1) is based on the context of a private microblogging service. A challengers task is to successfully log into the microblogging platform and retrieve the secret posted by the user *AIT*.

```
    '''
        Starts a instance with the given
    parameters
    '''
    def start_instance(self,instance_name,image,
    flavor,keypair_name,userdata,security_group):
        # check if instance already exists
        print('Checking for existing instance...')
        instance_exists = False
        for instance in self.nova.list_nodes():
            if instance.name == instance_name:
                new_instance = instance
                instance_exists = True

        if instance_exists:
            print('Instance ' + new_instance.name
    + ' already exists. Skipping creation.')
        else:
        # start up instance
            new_instance = self.nova.create_node(
    name=instance_name,
                    image=image,
                    size=flavor,
                    ex_userdata=userdata,
                    ex_keyname=keypair_name,
                    ex_security_groups=[
    security_group])
            self.nova.wait_until_running([
    new_instance])
        return new_instance
```

Listing 3. Creating Compute Challenge Instance

```
    '''
        Downloads an object from a swift container
    '''
    def download_object(self,container,obj):
        obj = self.swift.get_object(container_name
    =container,object_name=obj)
        filename = os.path.basename(obj.name)
        path = os.path.join(os.path.expanduser('.'
    ),filename)
        print 'Downloading: %s to %s' % (obj.name,
     path)
        obj.download(destination_path=path)
```

Listing 4. Storage Challenge File Download

```
    '''
        Downloads an object from a swift container
     as stream
    '''
    def download_object_stream(self,container,obj)
    :
        obj = self.swift.get_object(container_name
    =container,object_name=obj)
        return self.swift.
    download_object_as_stream(obj)
```

Listing 5. Storage Challenge Stream Download

Below the steps a challenger has to take to solve the challenge are described alongside screenshots of the microblogging platform.

1) The user is presented with a challenge description and a link to the microblogging service.
2) After reading the challenge description a challenger can proceed to the microblogging site. (see Figure 5)

Figure 5. Login Try

3) When trying to login or visit the ”Home”-page the challenger will be redirected to the login page asking him or her to login. (see Figure 6)

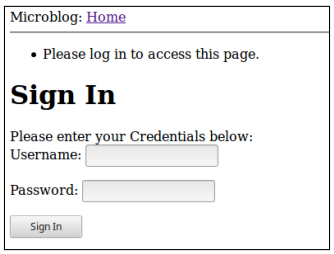

Figure 6. Login Fail

4) Presented with limited option challengers will start to test out the input fields leading them to an error page with valuable information (see Figure 7 and Figure 8).

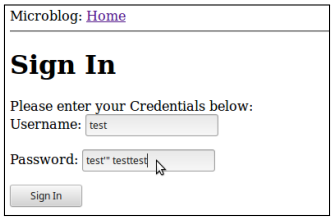

Figure 7. Testing input

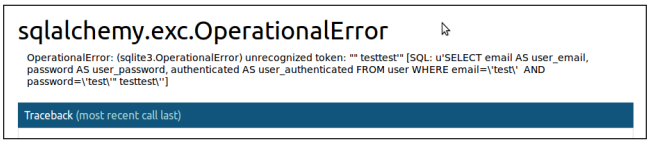

Figure 8. Error Page

5) The error page will reveal that the system uses a *sqlite3* database and user credentials are checked with a SQL query. The challenger can also see that his or her input is within single quotes in this query (see Listing 6).

```
SELECT email AS user_email, password AS
  user_password, authenticated AS
  user_authenticated FROM user WHERE email='
  test' AND password='test' testtest'
```

Listing 6. SQL input 1

```
SELECT email AS user_email, password AS
  user_password, authenticated AS
  user_authenticated FROM user WHERE email='
  test' AND password='test' or '1'='1'
```

Listing 7. SQL input 2

6) Using this information the challenger can alter the queries structure to always return a user even if neither email nor password match any database entry. This is done by adding a tautology like or '1'='1 (the final single quote is supplied by the existing query) to the query (see Listing 7).
7) With such a query the microblogging platform will login the challenger as the first user contained in its database and reveal the secret message (see Figure 9).

Figure 9. Successful login and secret

Creating tautologies using SQL injection is a classic example used in many presentations and security courses about basic IT-Security. Therefore almost anyone in the field of information technology should have seen a variation of such a challenge, at one point in his or her career. Because of this the challenge can be categorized as easy or even very easy.

VI. Conclusion

This paper presented a life cycle for the design of testbeds in education. The life cycle consists of 7 steps: (1) design, configure environment, (2) deploy environment, (3) define challenges, (4) deploy challenges, (5) conduct challenges, (6) maintain environment and (7) maintain challenges. Within the paper, we demonstrated in a case study the phases (1) through (5) in order to show the multiple aspects that are required to setup cyber security challenges

for educational testbeds. We selected a open source platform for implementation and demonstrated with a running example of an SQL injection how many steps it takes from definition, specification to implementation and actual execution of challenges. With this, we showed how complex and multifaceted defining and setting up cyber security challenges for education can be in order to enable skills management and development. For future work, we plan to add more cyber security challenges and further develop the testbed in order to establish an automated and flexible cyber security testbed.

ACKNOWLEDGMENT

This study was partly funded by the Austrian FFG research program KIRAS in course of the projects CISA (850199) and ACCSA (860649).

REFERENCES

[1] C. M. Davis, J. E. Tate, H. Okhravi, C. Grier, T. J. Overbye, and D. Nicol, "SCADA Cyber Security Testbed Development," in *2006 38th North American Power Symposium*, Sep. 2006, pp. 483–488.

[2] H. Christiansson and E. Luiijf, "Creating a European SCADA Security Testbed," in *Critical Infrastructure Protection*. Springer, Boston, MA, Mar. 2007, pp. 237–247.

[3] G. Vigna, "Teaching hands-on network security: Testbeds and live exercises," *Journal of Information Warfare*, vol. 3, no. 2, pp. 8–25, 2003.

[4] J. Mirkovic, T. V. Benzel, T. Faber, R. Braden, J. T. Wroclawski, and S. Schwab, "The DETER project: Advancing the science of cyber security experimentation and test," in *2010 IEEE HST*, Nov. 2010, pp. 1–7.

[5] T. Benzel, "The Science of Cyber Security Experimentation: The DETER Project," in *Proceedings of the 27th Annual Computer Security Applications Conference*, ser. ACSAC '11. ACM, 2011, pp. 137–148.

[6] T. Pahi, M. Leitner, and F. Skopik, "Data Exploitation at Large: Your Way to Adequate Cyber Common Operating Pictures," in *Proceedings of the 16th European Conference on Cyber Warfare and Security*. Reading, UK: Academic Conferences and Publishing International Limited, Jun. 2017, pp. 307–315.

[7] L. Pridmore, P. Lardieri, and R. Hollister, "National Cyber Range (NCR) automated test tools: Implications and application to network-centric support tools," in *2010 IEEE AUTOTESTCON*, Sep. 2010, pp. 1–4.

[8] B. Ferguson, A. Tall, and D. Olsen, "National Cyber Range Overview," in *2014 IEEE Military Communications Conference*, Oct. 2014, pp. 123–128.

[9] G. Vigna, K. Borgolte, J. Corbetta, A. Doupé, Y. Fratantonio, L. Invernizzi, D. Kirat, and Y. Shoshitaishvili, "Ten Years of iCTF: The Good, The Bad, and The Ugly," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. USENIX Association, 2014.

[10] R. Beuran, C. Pham, D. Tang, K. Chinen, Y. Tan, and Y. Shinoda, "Cytrone: An integrated cybersecurity training framework," in *ICISSP 2017*, 2017, pp. 157–166.

[11] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab, "Experience with DETER: a testbed for security research," in *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006.*, 2006, pp. 10 pp.–388.

[12] T. Benzel, R. Braden, D. Kim, A. D. Joseph, B. C. Neuman, R. Ostrenga, S. Schwab, and K. Sklower, "Design, Deployment, and Use of the DETER Testbed." in *DETER*, 2007.

[13] S. Jung, J.-g. Song, and S. Kim, "Design on SCADA test-bed and security device," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 3, no. 4, pp. 75–86, 2008.

[14] A. Hahn, B. Kregel, M. Govindarasu, J. Fitzpatrick, R. Adnan, S. Sridhar, and M. Higdon, "Development of the PowerCyber SCADA Security Testbed," in *CSIIRW '10*. ACM, 2010, pp. 21:1–21:4.

[15] A. Hahn, A. Ashok, S. Sridhar, and M. Govindarasu, "Cyber-Physical Security Testbeds: Architecture, Application, and Evaluation for Smart Grid," *IEEE Transactions on Smart Grid*, vol. 4, no. 2, pp. 847–855, Jun. 2013.

[16] D. C. Bergman, D. Jin, D. M. Nicol, and T. Yardley, "The Virtual Power System Testbed and Inter-testbed Integration," in *CSET'09*. USENIX Association, 2009, pp. 5–5.

[17] M. Rosenstein and F. Corvese, "A Secure Architecture for the Range-Level Command and Control System of a National Cyber Range Testbed," in *CSET*. USENIX, 2012.

[18] W. G. Halfond, J. Viegas, and A. Orso, "A classification of SQL-injection attacks and countermeasures," in *Proceedings of the IEEE International Symposium on Secure Software Engineering*, vol. 1. IEEE, Mar. 2006, pp. 12–23.