EncryptEdge Labs

Cybersecurity Analyst Internship Task Report

atalmamun@gmail.com

Task No: 23

Table of Contents

1.0 EncryptEdge Labs Internship Task Report

1.1 Introduction

In today's rapidly evolving cybersecurity landscape, organizations face an increasing volume of threats and incidents that require swift and accurate responses. Traditional manual methods of managing security operations are often time-consuming and prone to human error, leading to potential vulnerabilities and slower incident handling. Security automation addresses these challenges by using scripts and automated workflows to streamline repetitive tasks, enhance threat detection, and accelerate incident response. By integrating automation into cybersecurity processes, organizations can achieve greater efficiency, improve response times, and strengthen their overall security posture. This report explores the foundational concepts of security automation, provides hands-on experience with scripting languages, and demonstrates the practical application of automation in a sample security operations environment.

1.2 Objective

The primary objectives of this task are:

- To understand the significance and impact of security automation in modern cybersecurity operations.
- To learn and apply basic scripting skills to automate routine security tasks.
- To set up a functional environment for testing and developing automation scripts.
- To gain practical experience by creating and validating scripts that improve the efficiency of security processes such as log monitoring, alerting, and vulnerability scanning.

- To complete hands-on labs focused on Python and Bash scripting, enhancing technical proficiency in automating cybersecurity tasks.

1.3 Requirements

The successful completion of this task requires the following:

- Research and documentation on the importance and benefits of security automation.
- Selection and justification of a suitable scripting language (e.g., Python or PowerShell) for security tasks.
- Setup of a secure and isolated lab environment for script development and testing.
- Development of automation scripts to perform selected routine security tasks.
- Validation and testing of scripts to ensure correct functionality.
- Completion of the TryHackMe labs: "Python Basics" and "Bash Scripting," along with submission of relevant screenshots as evidence.

2.0 Security Automation Overview

2.1 Importance of Security Automation

Security automation has become a critical component in modern cybersecurity strategies. With the growing volume and complexity of cyber threats, manual responses are no longer sufficient to protect organizational assets effectively. Automation addresses these challenges by enabling faster, more consistent, and more accurate handling of security events.

Key benefits of security automation include:

- **Improved Efficiency:** Automation significantly reduces the time required to perform routine security tasks, allowing cybersecurity teams to focus on more complex and strategic activities.

- **Faster Incident Response:** Automated systems can detect, analyze, and respond to threats in real time, minimizing the potential damage from security incidents.

- **Consistency and Accuracy:** Automated processes follow predefined workflows without human error, ensuring that security policies are applied uniformly.

- **Scalability:** Automation makes it possible to manage security operations at a scale that would be impossible through manual efforts alone, particularly in large and dynamic IT environments.

- **Cost Reduction:** By reducing the manpower needed for repetitive tasks, organizations can lower operational costs while improving their security posture.

In summary, security automation is essential for enhancing the speed, accuracy, and effectiveness of cybersecurity operations in today's threat environment.

2.2 Use Cases for Security Automation

Security automation can be applied to a wide range of use cases across the cybersecurity domain. Some notable examples include:

2.2.1 Incident Response

Automation can streamline incident response by automatically detecting suspicious activities, generating alerts, collecting relevant data for analysis, and even executing predefined containment actions, such as blocking malicious IP addresses or isolating compromised endpoints.

2.2.2 Log Analysis

Manually analyzing logs from various systems is tedious and error-prone. Automation scripts can be used to collect, parse, and analyze logs to identify anomalies, patterns, or signs of malicious behavior, greatly accelerating the detection of security incidents.

2.2.3 Threat Intelligence Integration

Automation enables continuous integration of threat intelligence feeds into security systems. This helps organizations stay updated with the latest threat indicators and apply defenses dynamically without manual intervention.

2.2.4 Vulnerability Scanning and Patch Management

Automated tools can regularly scan systems for vulnerabilities, generate reports, and even initiate patching processes. This reduces the window of exposure and ensures that systems remain secure against known threats.

2.2.5 Alerting and Notification

Automation can be used to configure real-time alerts and notifications for specific security events, ensuring that security teams are promptly informed of critical incidents without the need for constant manual monitoring.

3.0 Choosing a Scripting Language for Security Automation

3.1 Language Selection

For the purpose of this task, **Python** was selected as the scripting language for developing security automation scripts.
Python is widely recognized in the cybersecurity community for its simplicity, versatility, and powerful capabilities in automating a variety of security tasks. Its extensive library ecosystem, easy-to-read syntax, and strong community support make it a preferred choice for both beginners and experienced cybersecurity professionals.

Key reasons for choosing Python include:

- **Ease of Learning and Readability:** Python's clear syntax reduces the learning curve, allowing faster development and easier maintenance of scripts.
- **Extensive Libraries:** Libraries such as `Scapy`, `Requests`, `Paramiko`, and `Socket` enable rapid development of network, web, and system automation tools.
- **Cross-Platform Compatibility:** Python scripts can be run on multiple operating systems (Linux, Windows, macOS) with minimal adjustments.
- **Strong Community Support:** A large and active community provides extensive documentation, tutorials, and shared scripts relevant to security operations.
- **Integration with Security Tools:** Python easily integrates with tools like SIEM platforms, firewalls, and vulnerability scanners through APIs and SDKs.

3.2 Basic Python Syntax for Automation

To effectively use Python for automation, it is important to understand key language features such as variables, conditions, loops, and functions. Below are some examples relevant to security automation:

3.2.1 Variables

Variables store data that can be used throughout the script.

python

```
log_file_path = "/var/log/auth.log"

alert_threshold = 5
```

3.2.2 Conditional Statements

Conditional logic helps in decision-making within automation scripts.

python

```
if failed_login_attempts > alert_threshold:

    print("Alert: Too many failed login attempts detected!")
```

3.2.3 Loops

Loops allow automation of repetitive tasks, such as scanning multiple log files or IP addresses.

python

```
ip_addresses = ["192.168.1.1", "192.168.1.2", "10.0.0.5"]

for ip in ip_addresses:

    print(f"Scanning IP: {ip}")
```

3.2.4 Functions

Functions organize reusable blocks of code for better structure and maintenance.

python

```python
def scan_ip(ip):

    print(f"Performing security scan on {ip}")

scan_ip("192.168.1.1")
```

Python's capabilities and ease of use make it an excellent choice for cybersecurity automation. Through Python, complex tasks such as log monitoring, alert generation, and vulnerability scanning can be efficiently automated, significantly improving the effectiveness and responsiveness of cybersecurity operations.

4.0 Setting Up the Automation Lab Environment

4.1 Lab Environment Preparation

To develop, test, and validate security automation scripts safely, a controlled lab environment was set up. Creating a virtualized environment ensures that testing does not interfere with live systems and provides the flexibility to simulate various cybersecurity scenarios.

The following steps were undertaken to prepare the lab:

4.1.1 Virtual Environment Setup

A virtual environment was established using **VirtualBox** as the virtualization platform. Two virtual machines (VMs) were created:

- **Kali Linux VM:** Used as the primary system for developing and executing automation scripts.

- **Ubuntu Server VM:** Acted as a target machine generating logs and simulating basic services.

Both VMs were connected through an internal network in VirtualBox to mimic a small organizational network without exposing it to the public internet.

4.1.2 Installation of Necessary Tools

On the Kali Linux VM, the following installations were performed:

- **Python 3:** Already included by default in Kali Linux; verified using `python3 --version`.

- **Python Libraries:** Key libraries for scripting were installed using `pip3`, including:

  - `requests` — for interacting with web APIs.

  - `paramiko` — for automating SSH connections.

  - `scapy` — for network packet manipulation and analysis.

- **Log Files:** Sample log files were created and pre-existing system logs (such as `/var/log/auth.log`) were prepared for monitoring.

On the Ubuntu Server VM:

- Basic services (SSH, Apache web server) were installed and configured to generate logs for testing purposes.

- User login attempts and system activity were enabled to simulate real-world event logging.

4.1.3 Environment Configuration

Additional configurations included:

- **SSH Access:** Enabled between Kali and Ubuntu VMs for remote automation testing.
- **User Accounts:** Multiple user accounts were created on Ubuntu to simulate different login attempts and user activities.
- **Firewall Rules:** Basic firewall rules (using `ufw`) were configured and could be dynamically manipulated via automation scripts.

A simple directory structure was also organized on the Kali VM to manage scripts and results:

swift

```
/home/user/security_automation/

├── scripts/

├── logs/

└── reports/
```

This structure helped maintain a clean and organized workspace for the automation project.

4.2 Summary of the Setup

The automation lab environment successfully mirrored a simplified security operations center (SOC) environment. By using isolated virtual machines, essential tools, and realistic data sources like logs and alerts, a safe and effective setting was established to develop and test security automation scripts.

This setup enabled practical experience in simulating security monitoring, incident response, and basic threat detection workflows — key foundations for further hands-on security automation tasks.

5.0 Automating Routine Security Tasks

5.1 Identifying Routine Tasks for Automation

Several routine tasks were selected for automation based on their relevance to security operations and their potential to improve efficiency. These tasks include:

- **Log Monitoring:** Automatically scanning system logs for suspicious login attempts and alerting if anomalies are detected.

- **SSH Login Alerting:** Monitoring SSH logs to detect failed login attempts and notifying the administrator.

- **Basic Vulnerability Scanning:** Automating the process of scanning open ports on target machines.

These tasks are commonly encountered in security environments and provide practical scenarios for applying automation techniques.

5.2 Development of Automation Scripts

The following scripts were developed using Python to automate the selected tasks.

5.2.1 Script 1: Log Monitoring for Failed SSH Logins

Objective: Parse system authentication logs to identify multiple failed login attempts from the same IP address.

python

```
import re

log_file = "/var/log/auth.log"

failed_attempts = {}

with open(log_file, "r") as file:

    for line in file:

        if "Failed password" in line:

            match = re.search(r"from (\d+\.\d+\.\d+\.\d+)", line)

            if match:

                ip = match.group(1)

                failed_attempts[ip] = failed_attempts.get(ip, 0) + 1


for ip, count in failed_attempts.items():

    if count > 5:

        print(f"Alert: {ip} has {count} failed login attempts!")
```

Functionality:

- Scans the `/var/log/auth.log` file.
- Counts failed SSH login attempts per IP address.

- Triggers an alert if attempts exceed a set threshold.

5.2.2 Script 2: SSH Login Real-Time Monitoring (Using Tail)

Objective: Continuously monitor SSH log entries in real time and detect suspicious behavior.

python

```
import subprocess

process = subprocess.Popen(['tail', '-F', '/var/log/auth.log'],
stdout=subprocess.PIPE)

while True:

    line = process.stdout.readline()

    if b"Failed password" in line:

        print("Failed SSH login attempt detected!")
```

Functionality:

- Uses `tail -F` to follow the auth log live.
- Immediately alerts when a failed login attempt is detected.

5.2.3 Script 3: Basic Network Vulnerability Scanner

Objective: Scan a list of IP addresses for open ports.

python

```
import socket
```

```
def scan_ports(ip):
    open_ports = []
    for port in range(20, 1025):
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(0.5)
        result = sock.connect_ex((ip, port))
        if result == 0:
            open_ports.append(port)
        sock.close()
    return open_ports


target_ips = ["192.168.1.10", "192.168.1.20"]


for ip in target_ips:
    ports = scan_ports(ip)
    print(f"Open ports on {ip}: {ports}")
```

Functionality:

- Scans TCP ports 20-1024 for each target IP address.
- Lists all open ports found.

5.3 Validation and Testing Results

Each script was tested within the prepared lab environment:

- **Log Monitoring Script:** Successfully detected multiple failed login attempts and generated alerts once the threshold was exceeded.

- **Real-Time SSH Monitoring:** Detected and printed alerts immediately when simulated failed login attempts were made.

- **Port Scanner Script:** Accurately identified open ports on the Ubuntu Server VM, including standard services like SSH (port 22) and HTTP (port 80).

5.4 Challenges and Resolutions

During development and testing, several challenges were encountered:

- **File Permission Errors:** Accessing system log files like `/var/log/auth.log` required elevated permissions. This was resolved by running the scripts with `sudo` or adjusting file permissions in the lab environment.

- **False Positives in Log Parsing:** Initially, the log monitoring script captured unrelated entries. Fine-tuning the regular expressions helped target only relevant SSH failure lines.

- **Timeout Issues in Port Scanning:** Scanning many ports can cause delays. Setting socket timeouts to a lower value improved the performance of the port scanning script.

These challenges provided valuable learning experiences, reinforcing practical troubleshooting skills essential in real-world cybersecurity automation work.

6. Hands-on Labs Completion

6.1 TryHackMe Lab: Python Basics

6.1.1 Lab Overview

The **Python Basics** room on TryHackMe provided a structured introduction to Python programming. The lab covered essential concepts such as:

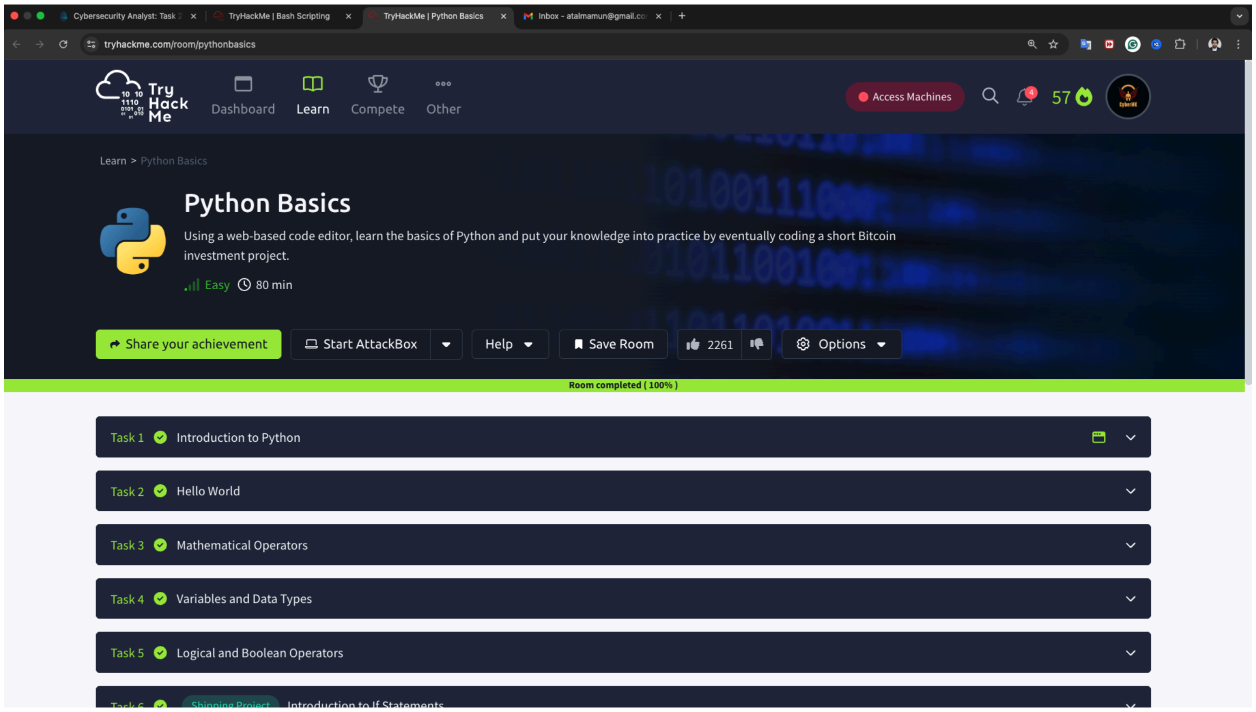
- Variables and Data Types
- Conditional Statements and Loops
- Functions and Modules
- File Handling
- Basic Automation Techniques (e.g., reading files, simple data processing)

These foundational skills are directly applicable to security automation tasks, such as processing logs, analyzing network data, and automating repetitive security procedures.

6.1.2 Key Exercises Completed

- **Created Simple Scripts:** Developed Python scripts to read and process data from files.
- **Automated Tasks:** Used loops and conditions to automate decision-making based on input data.
- **Built Functions:** Modularized code into reusable functions for easier maintenance and readability.

6.1.3 Screenshots


tryhackme.com/room/pythonbasics

Learn > Python Basics

Python Basics

Using a web-based code editor, learn the basics of Python and put your knowledge into practice by eventually coding a short Bitcoin investment project.

Easy 80 min

Room completed (100%)

Task 1 Introduction to Python

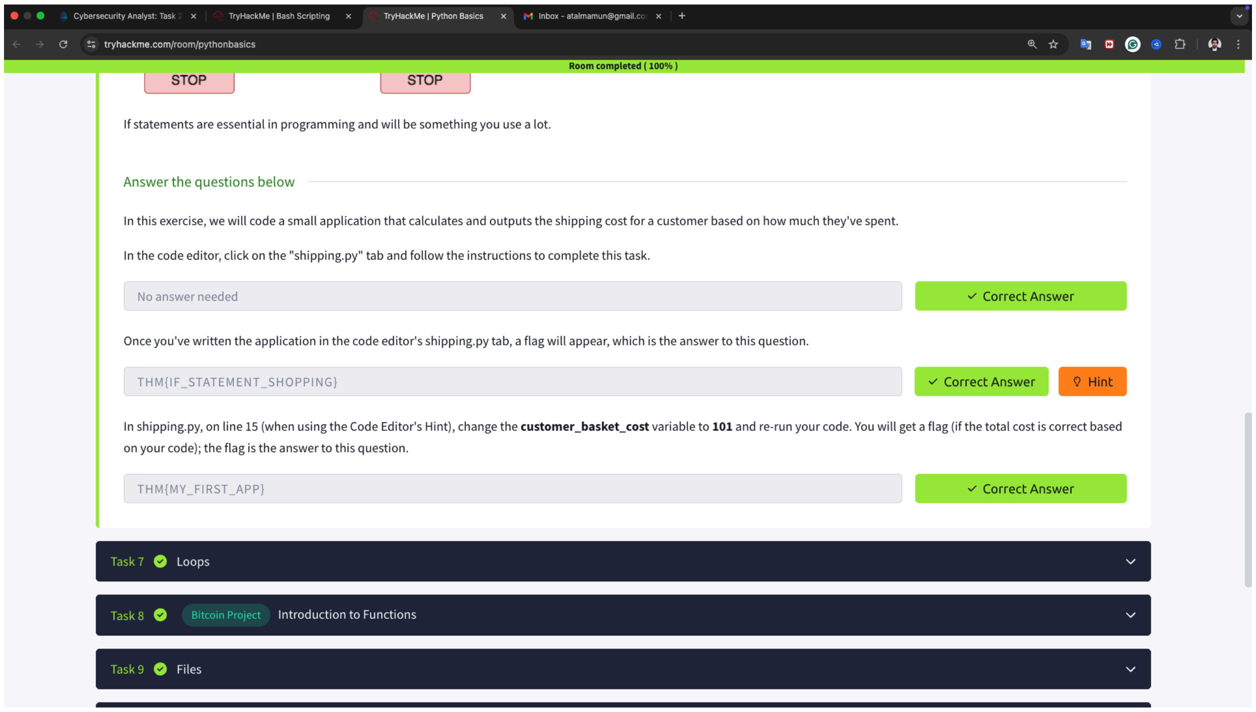
Task 2 Hello World

Task 3 Mathematical Operators

Task 4 Variables and Data Types

Task 5 Logical and Boolean Operators

Task 6 Shipping Project Introduction to If Statements


tryhackme.com/room/pythonbasics

Room completed (100%)

If statements are essential in programming and will be something you use a lot.

Answer the questions below

In this exercise, we will code a small application that calculates and outputs the shipping cost for a customer based on how much they've spent.

In the code editor, click on the "shipping.py" tab and follow the instructions to complete this task.

No answer needed

Correct Answer

Once you've written the application in the code editor's shipping.py tab, a flag will appear, which is the answer to this question.

THM{IF_STATEMENT_SHOPPING}

Correct Answer Hint

In shipping.py, on line 15 (when using the Code Editor's Hint), change the **customer_basket_cost** variable to **101** and re-run your code. You will get a flag (if the total cost is correct based on your code); the flag is the answer to this question.

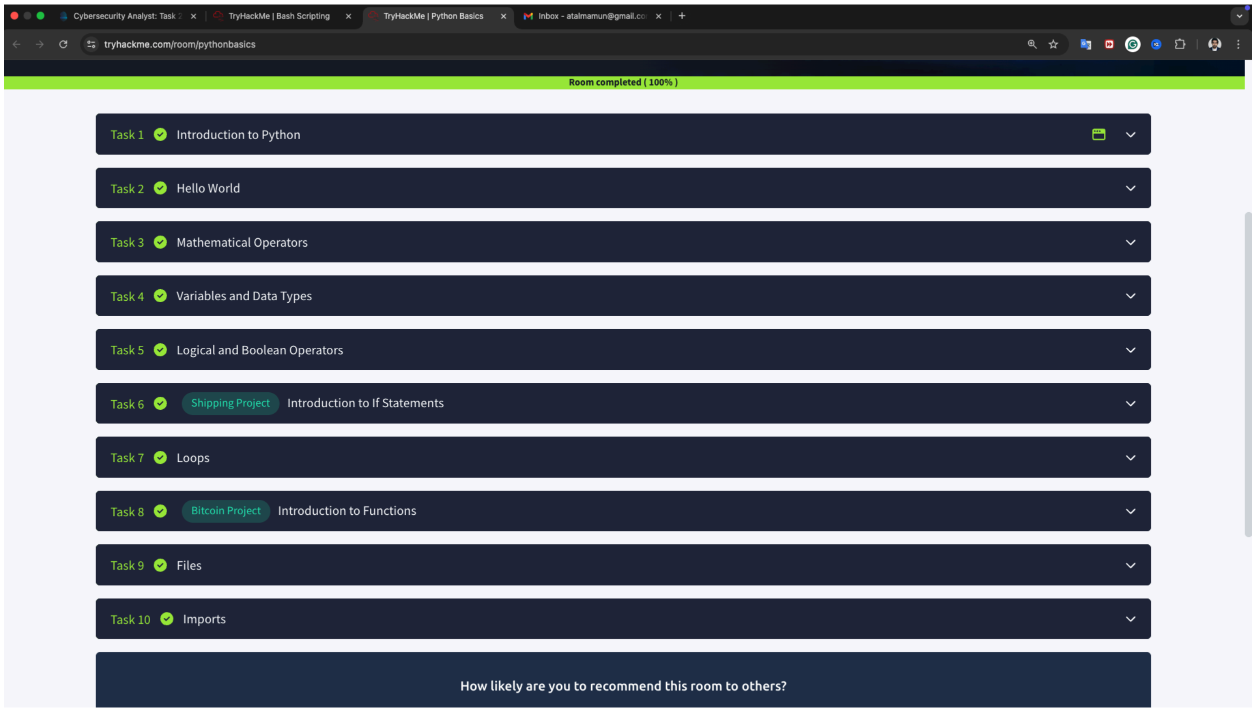
THM{MY_FIRST_APP}

Correct Answer

Task 7 Loops

Task 8 Bitcoin Project Introduction to Functions

Task 9 Files


Cybersecurity Analyst: Task 2
TryHackMe | Bash Scripting
TryHackMe | Python Basics
Inbox - atalmamun@gmail.com
tryhackme.com/room/pythonbasics

Room completed (100%)

Task 1 Introduction to Python

Task 2 Hello World

Task 3 Mathematical Operators

Task 4 Variables and Data Types

Task 5 Logical and Boolean Operators

Task 6 Shipping Project Introduction to If Statements

Task 7 Loops

Task 8 Bitcoin Project Introduction to Functions

Task 9 Files

Task 10 Imports

How likely are you to recommend this room to others?

6.2 TryHackMe Lab: Bash Scripting

6.2.1 Lab Overview

The **Bash Scripting** room introduced key scripting techniques used for automating administrative and security tasks on Linux systems. Key topics included:

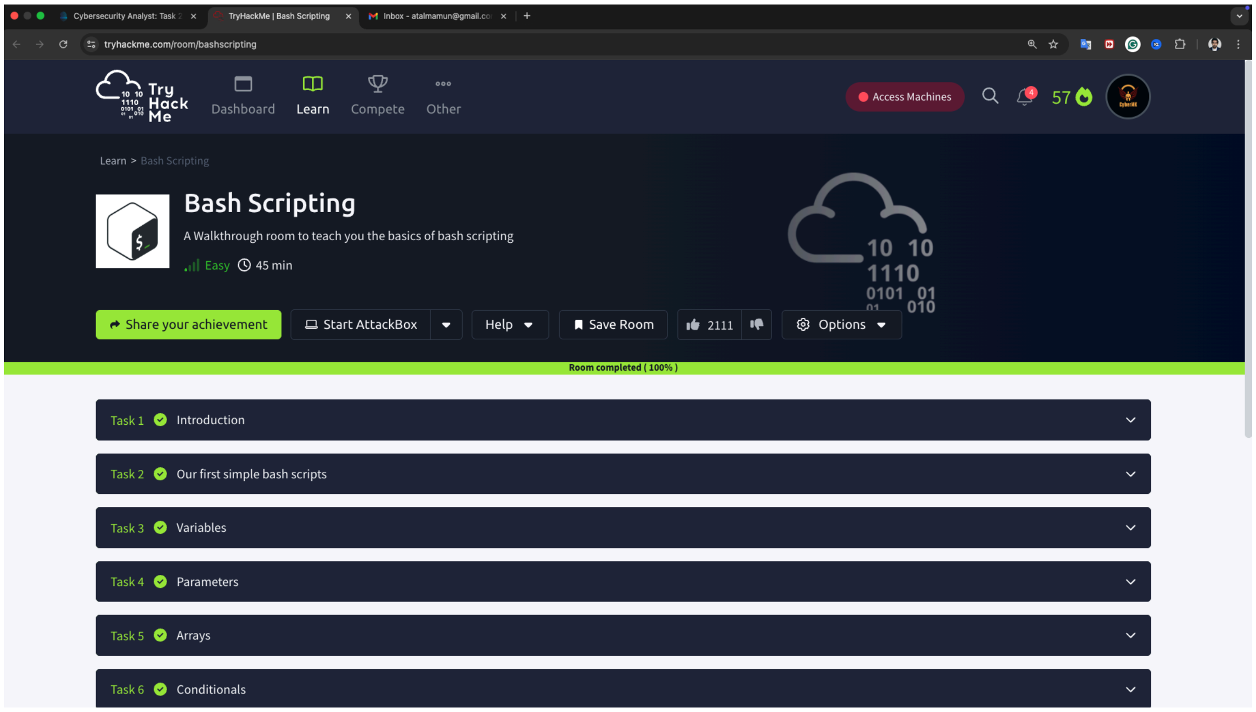
- Writing and executing Bash scripts
- Using loops, conditional statements, and functions in Bash
- Managing files and directories via scripts
- Automating system monitoring and maintenance tasks

These skills are crucial for cybersecurity professionals, especially when managing Linux-based environments where automation can significantly improve system oversight and response times.

6.2.2 Key Exercises Completed

- **Created Monitoring Scripts:** Wrote scripts to automate monitoring of system logs and file changes.
- **Used Conditional Logic:** Implemented conditions to create dynamic and intelligent scripts.
- **Automated Routine Admin Tasks:** Developed scripts to manage file backups and user activity logging.

6.2.3 Screenshots



Learn > Bash Scripting

Bash Scripting

A Walkthrough room to teach you the basics of bash scripting

Easy 45 min

Room completed (100%)

Task 1 Introduction

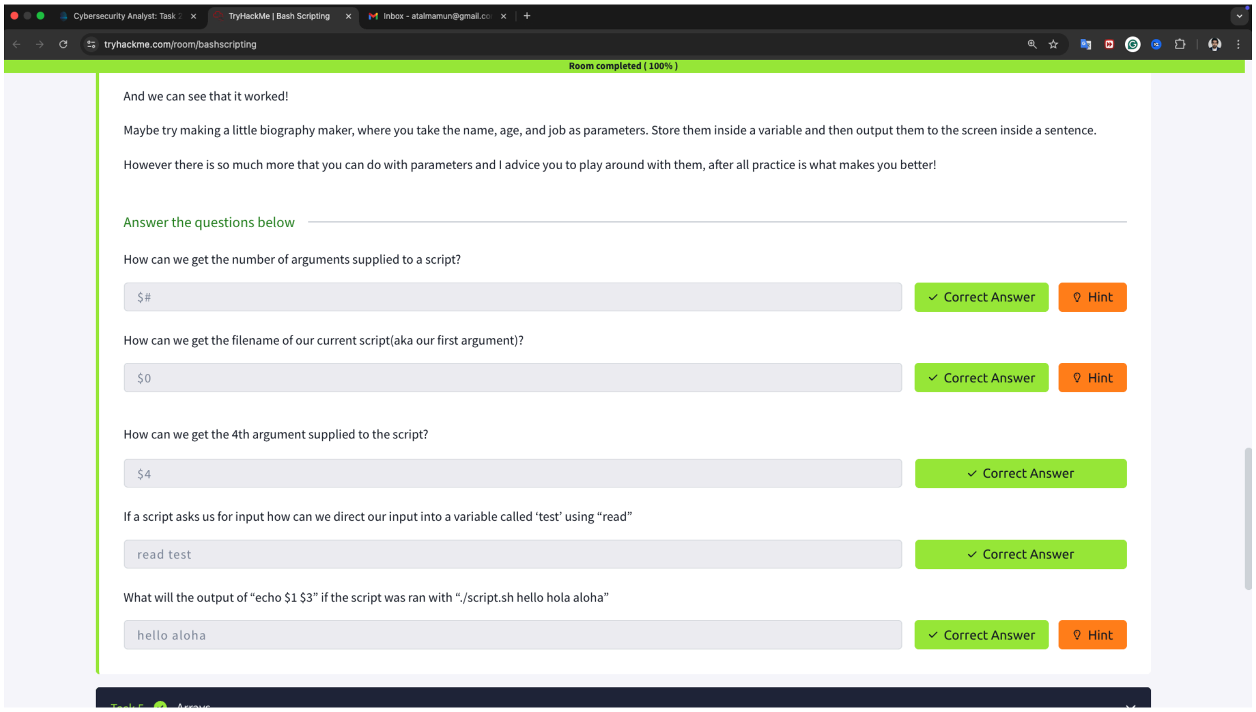
Task 2 Our first simple bash scripts

Task 3 Variables

Task 4 Parameters

Task 5 Arrays

Task 6 Conditionals



tryhackme.com/room/bashscripting

Room completed (100%)

And we can see that it worked!

Maybe try making a little biography maker, where you take the name, age, and job as parameters. Store them inside a variable and then output them to the screen inside a sentence.

However there is so much more that you can do with parameters and I advice you to play around with them, after all practice is what makes you better!

Answer the questions below

How can we get the number of arguments supplied to a script?

`$#` — ✓ Correct Answer

How can we get the filename of our current script(aka our first argument)?

`$0` — ✓ Correct Answer

How can we get the 4th argument supplied to the script?

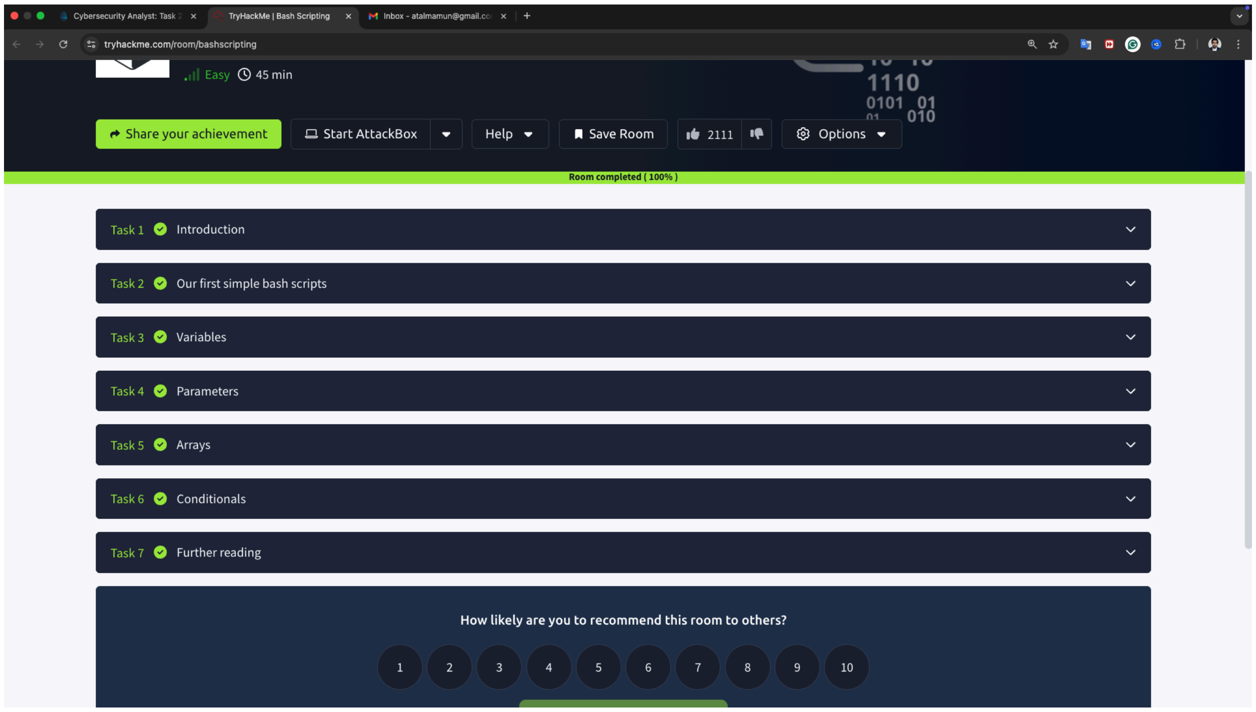
`$4` — ✓ Correct Answer

If a script asks us for input how can we direct our input into a variable called 'test' using "read"

`read test` — ✓ Correct Answer

What will the output of "echo $1 $3" if the script was ran with "./script.sh hello hola aloha"

`hello aloha` — ✓ Correct Answer



tryhackme.com/room/bashscripting

Easy 45 min

Share your achievement | Start AttackBox | Help | Save Room | 2111 | Options

Room completed (100%)

Task 1 ✓ Introduction

Task 2 ✓ Our first simple bash scripts

Task 3 ✓ Variables

Task 4 ✓ Parameters

Task 5 ✓ Arrays

Task 6 ✓ Conditionals

Task 7 ✓ Further reading

How likely are you to recommend this room to others?

1 2 3 4 5 6 7 8 9 10

6.3 Summary

Successfully completing the TryHackMe Python and Bash scripting labs reinforced critical automation concepts. Through hands-on practice, proficiency was gained in two major scripting languages essential for security automation. These labs enhanced practical skills in:

- Automating security monitoring tasks
- Writing efficient, reusable scripts
- Managing files, processes, and network data programmatically

The experience directly supported the development of the scripts and tasks carried out in the automation project.

This Internship Task report was developed on [April, 28, 2025]

By:

atalmamun@gmail.com