Implementing Secure Data Exchange Using Cryptographic Techniques

Project Title:
Implementing Secure Data Exchange Using Cryptographic Techniques

Student Name: Santhosh R

Reg.No : 727823TUCS303

Department: B.E. CSE

TABLE OF CONTENTS

1. INTRODUCTION

- Secure data exchange is a fundamental requirement in modern computing environments, especially where sensitive information is transmitted over open or untrusted networks.
- Without proper cryptographic protection, data is vulnerable to interception, manipulation, and unauthorized access.
- This project focuses on implementing secure data exchange using cryptographic techniques, combining theoretical concepts with hands-on practical demonstrations using industry-standard security tools. The implementation emphasizes encryption, secure communication, access control, and attack detection.

---

2. PROBLEM STATEMENT

Many systems suffer from insecure data transmission due to:

- Plaintext communication
- Weak or missing encryption
- Improper key management
- Lack of authentication mechanisms
- Absence of access control
- No monitoring or audit trails
- These vulnerabilities allow attackers to perform man-in-the-middle attacks, data theft, and unauthorized system access.
- This project addresses these issues by implementing cryptography-based secure data exchange mechanisms.

---

3. OBJECTIVES

✓ Implement symmetric and asymmetric encryption
✓ Secure data transmission using TLS
✓ Demonstrate encrypted traffic using packet analysis
✓ Perform vulnerability scanning on services
✓ Implement role-based access control
✓ Enable multi-factor authentication
✓ Monitor and audit system activities
✓ Simulate attacks and demonstrate defenses

---

4. DEFINITIONS

Encryption
Process of converting plaintext into ciphertext to prevent unauthorized access.

Symmetric Encryption
Uses a single shared key for encryption and decryption.
Example: AES-256.
Asymmetric Encryption
Uses a public key for encryption and a private key for decryption.
Example: RSA-4096.
Hybrid Encryption
Combination of asymmetric and symmetric encryption, used for secure key exchange.
Example: TLS, HTTPS.
TLS (Transport Layer Security)
Protocol that provides secure communication over networks.
RBAC (Role-Based Access Control)
Access permissions assigned based on user roles.
MFA (Multi-Factor Authentication)
Authentication using more than one verification factor.
Vulnerability Assessment
Process of identifying security weaknesses in systems.
auditd
Linux auditing system used to log security-relevant events.

---

5. SYSTEM REQUIREMENTS

- Kali Linux
- OpenSSL
- Wireshark
- Nmap
- Metasploit Framework
- Google Authenticator
- auditd

---

6. PHASE 1 — SECURE ENVIRONMENT SETUP

THEORY
A properly configured security environment is essential for implementing cryptographic mechanisms and analyzing secure data exchange. This phase ensures that all required tools for encryption, monitoring, scanning, and auditing are correctly installed and operational.

PRACTICAL STEPS

Tool Installation

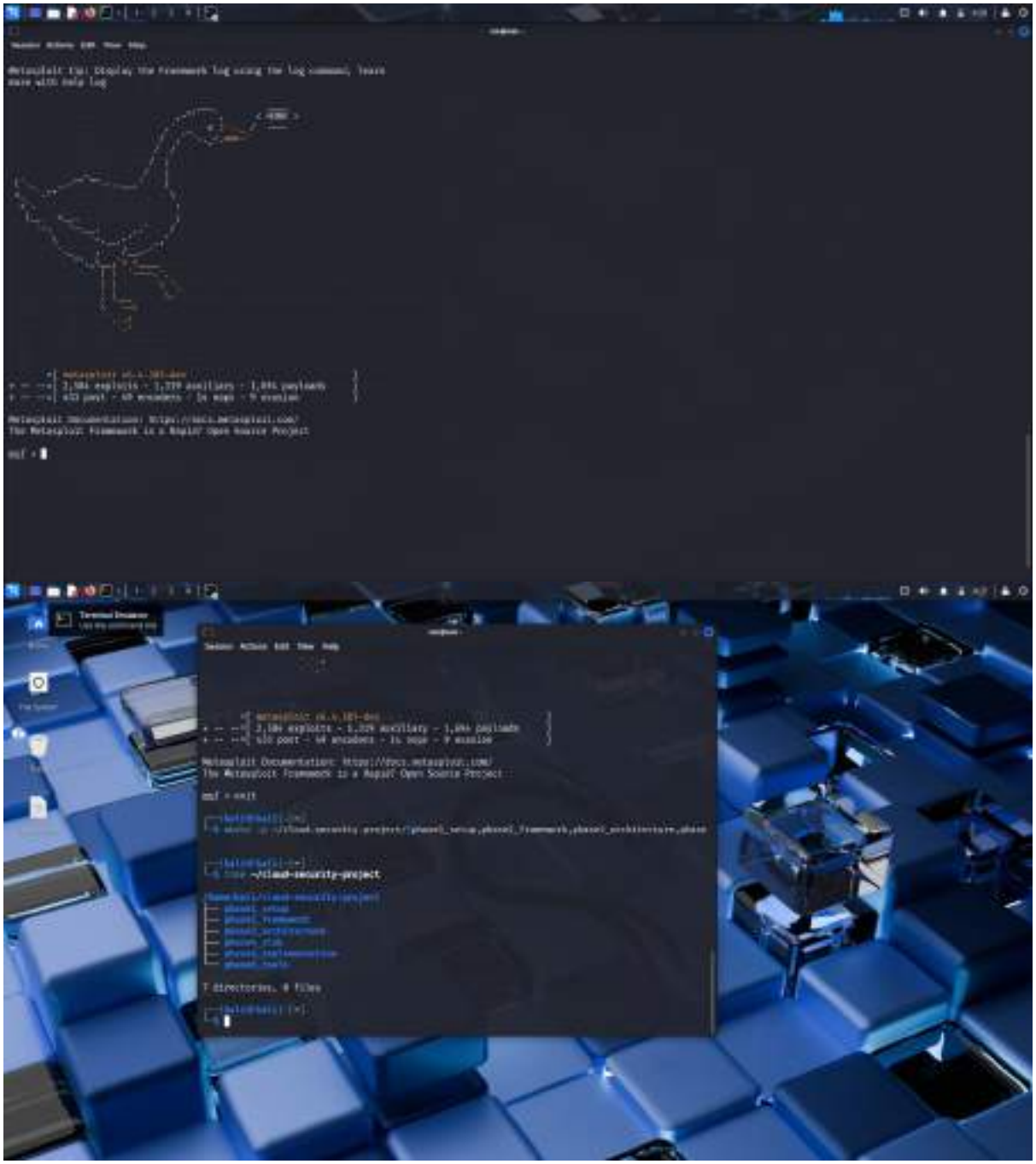
sudo apt update

sudo apt install openssl wireshark nmap metasploit-framework auditd libpam-google-authenticator -y

Project Directory Structure

mkdir -p ~/secure-data-exchange/{phase1_setup,phase2_crypto,phase3_traffic,phase4_access,phase5_attack,phase6_tools}

This structured setup allows systematic implementation and evaluation of each security phase.



---

7. PHASE 2 — ENCRYPTION FRAMEWORK

THEORY

Why Encryption?

To protect confidentiality and integrity of data.

Why AES?

Fast and secure; used for bulk data encryption.

Why RSA?

Used for identity verification and key exchange.

Why Hybrid Encryption?

Used by TLS because RSA is slow for large data.

PRACTICAL IMPLEMENTATION

7.1 AES-256 Encryption

Create file:

```
echo "This is sensitive cloud data." > sensitive.txt
```

Encrypt:

```
openssl enc -aes-256-cbc -pbkdf2 -iter 100000 -in sensitive.txt -out
sensitive.enc
```

Decrypt:

```
openssl enc -d -aes-256-cbc -pbkdf2 -iter 100000 -in sensitive.enc -out
decrypted.txt
```

7.2 RSA Encryption (pkeyutl modern method)

Generate keys:

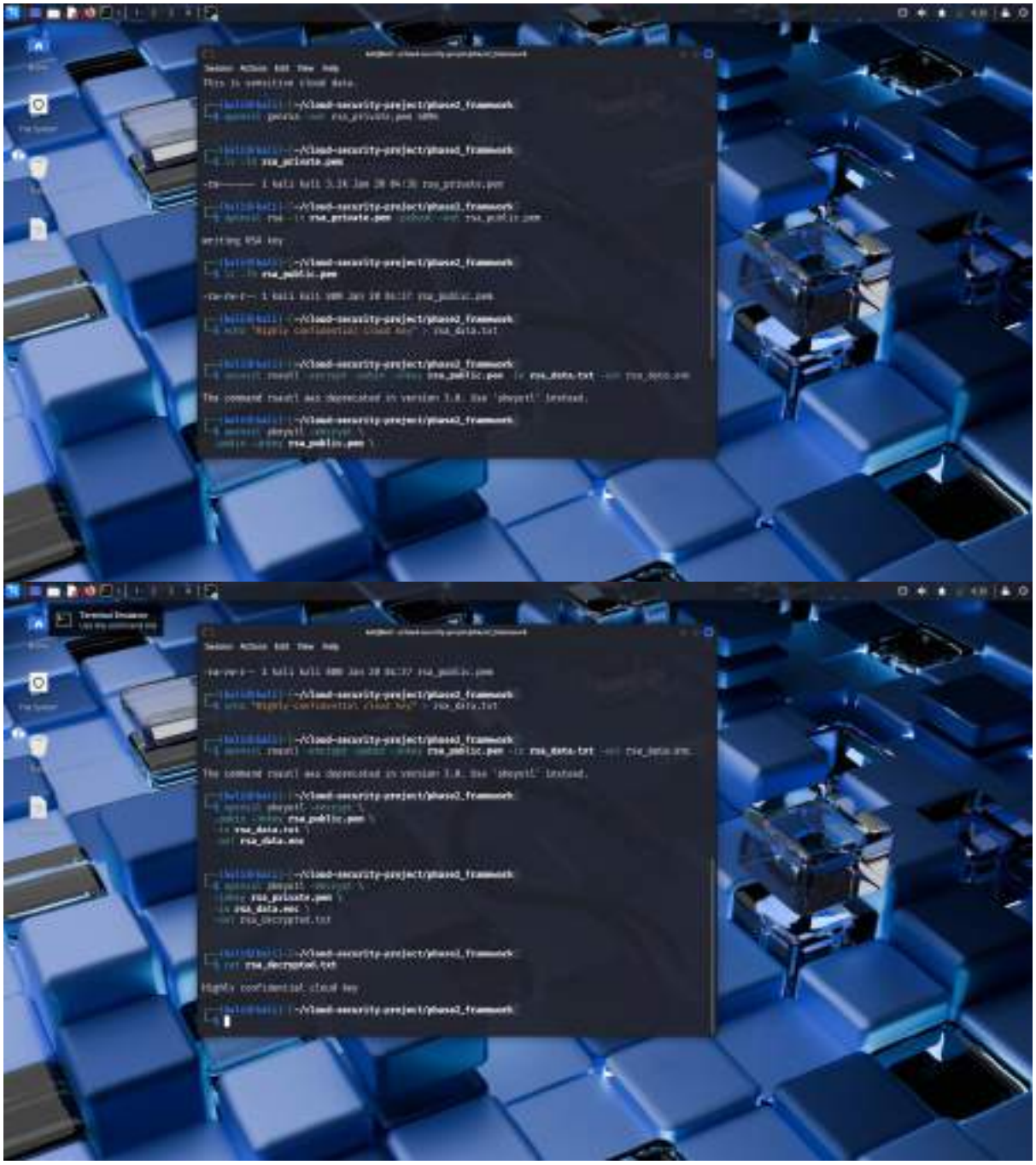
```
openssl genrsa -out rsa_private.pem 4096
openssl rsa -in rsa_private.pem -pubout -out rsa_public.pem
```

Encrypt:

```
echo "Highly confidential cloud key" > rsa_data.txt
openssl pkeyutl -encrypt -pubin -inkey rsa_public.pem -in rsa_data.txt -out
rsa_data.enc
```

Decrypt:

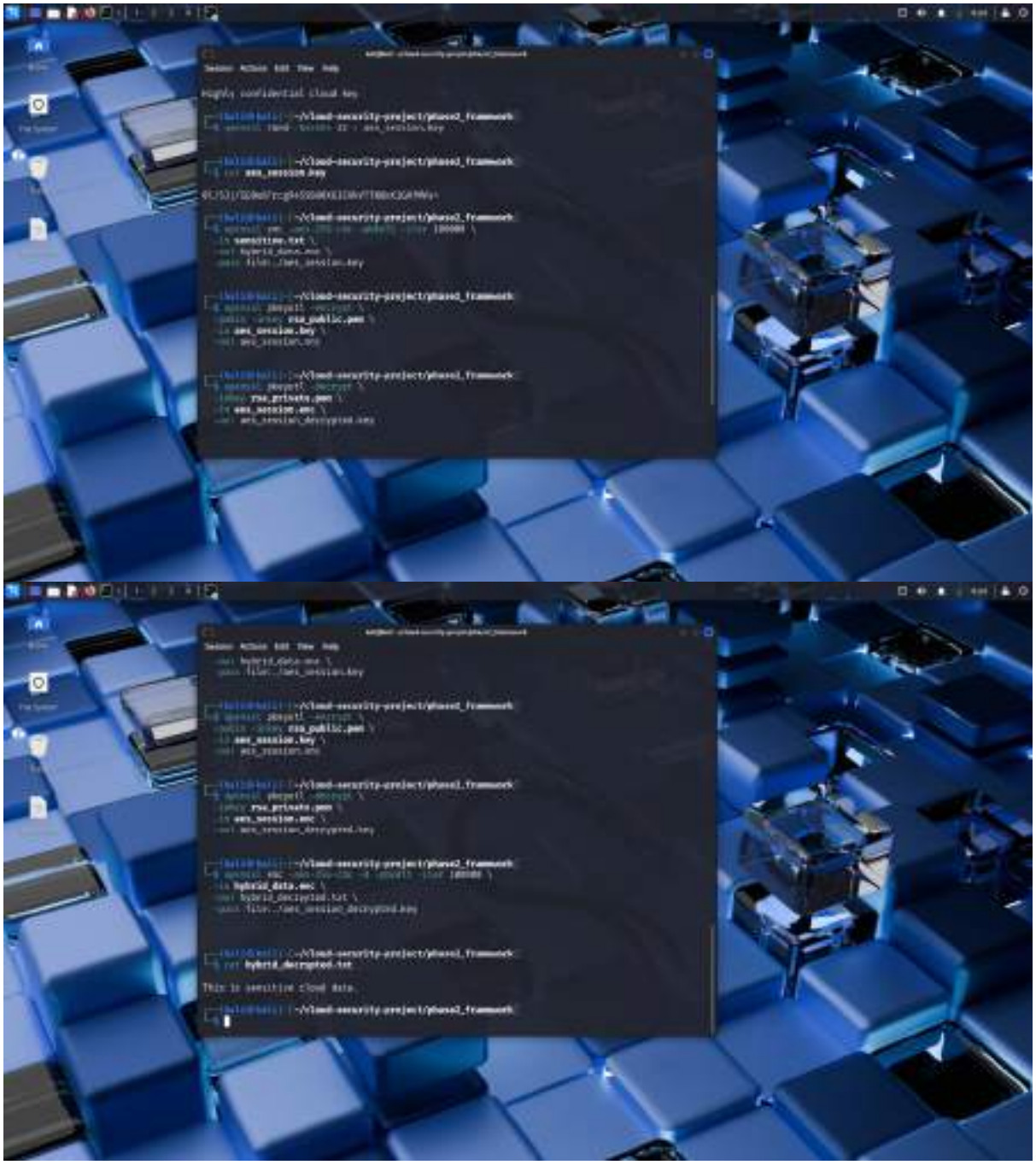
```
openssl pkeyutl -decrypt -inkey rsa_private.pem -in rsa_data.enc -out
rsa_decrypted.txt
```



7.3 Hybrid Encryption

This imitates how HTTPS and cloud KMS systems work.

Commands included earlier.

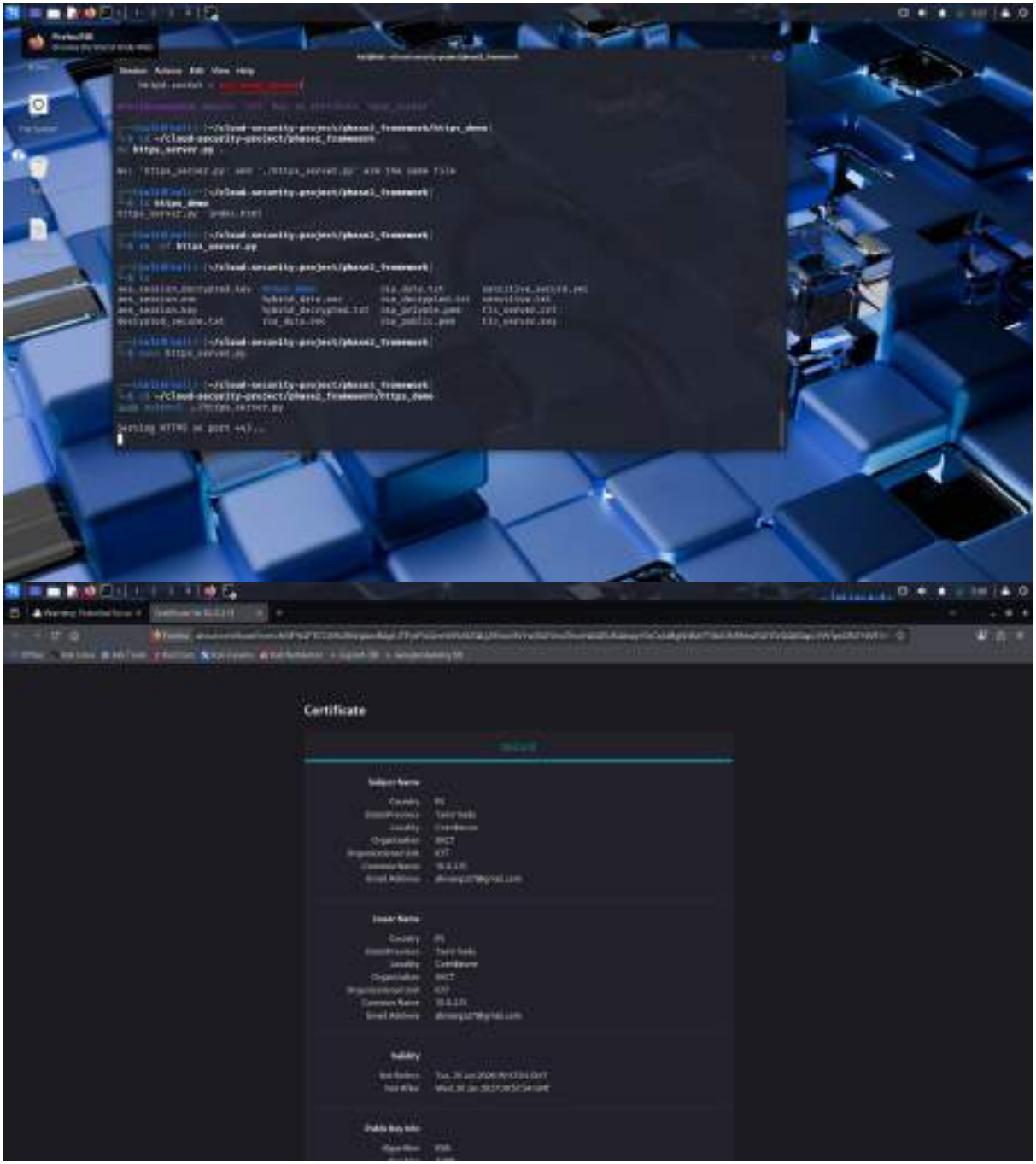


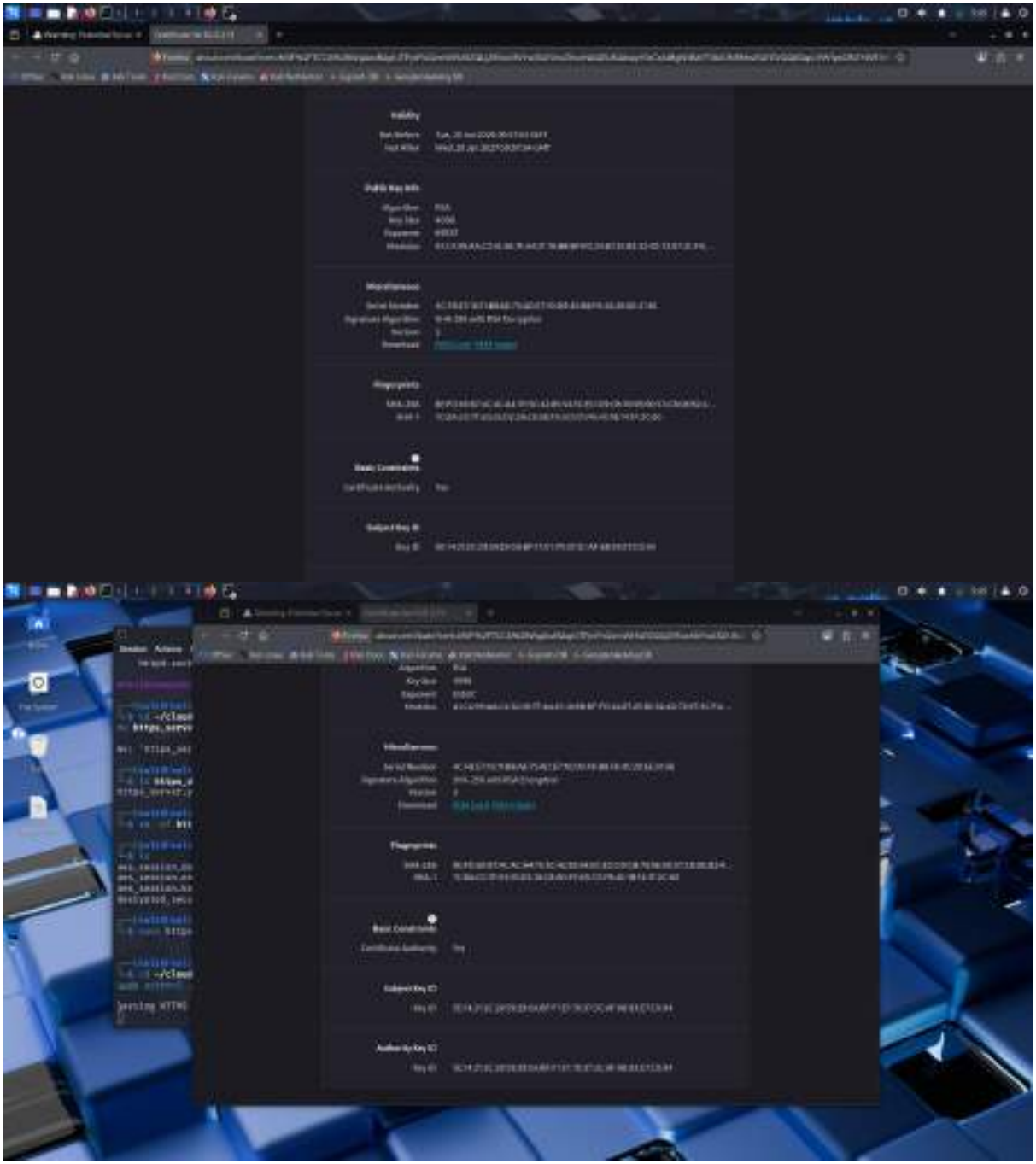
7.4 HTTPS Server + TLS Certificate

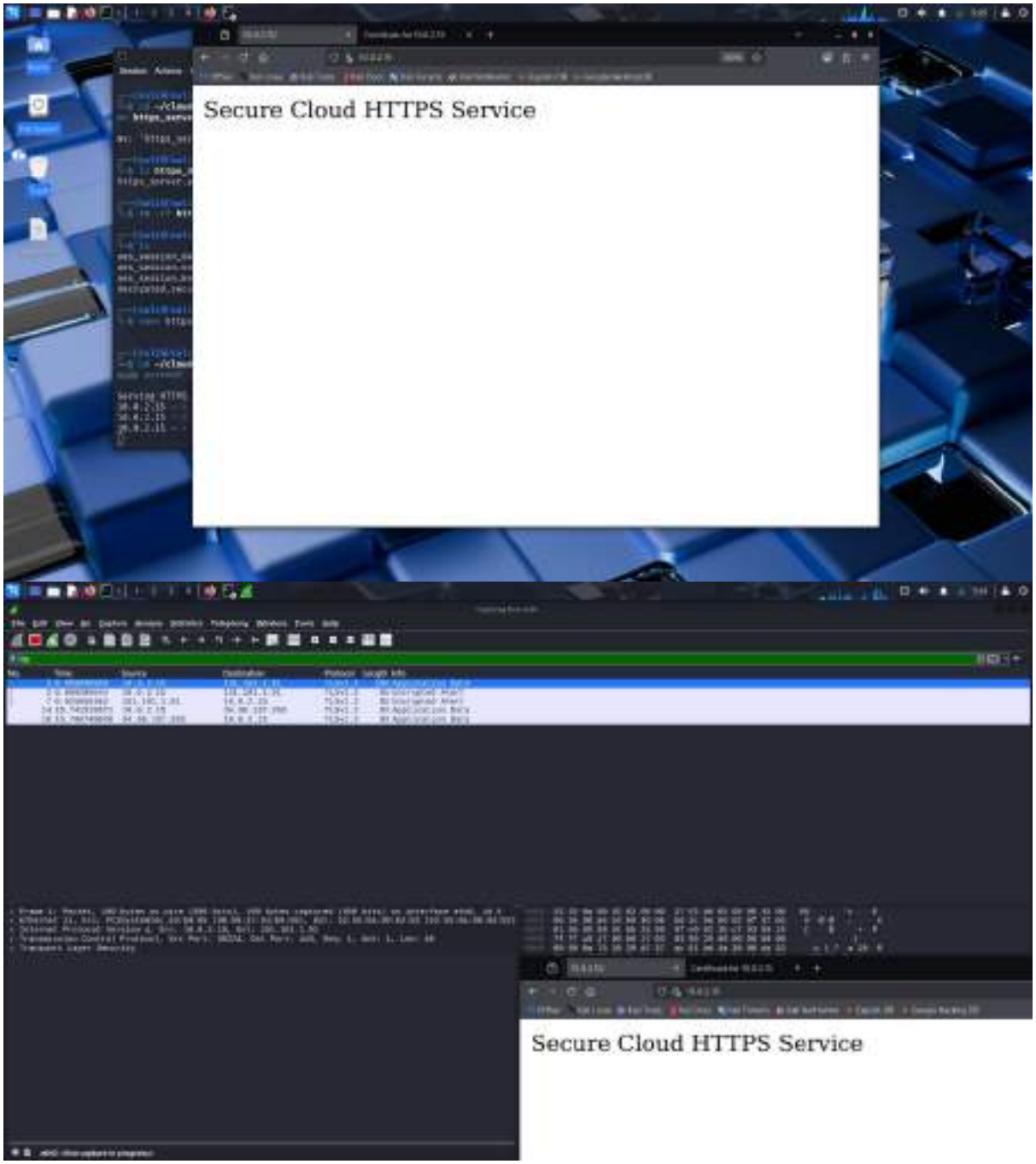
Generate TLS certificate:

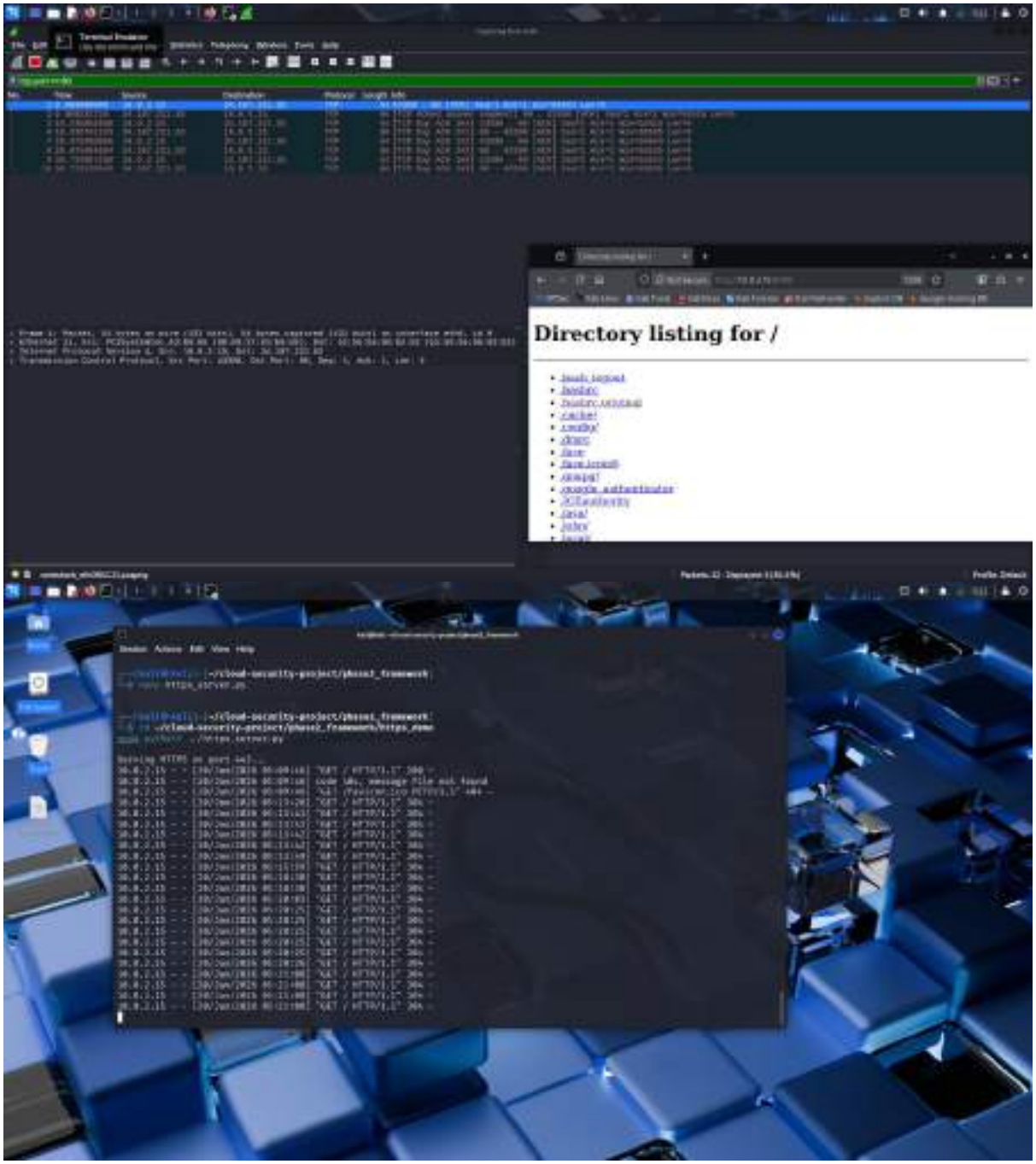
```
openssl req -x509 -newkey rsa:4096 -keyout tls_server.key -out
tls_server.crt -days 365 -nodes
```

HTTPS server code provided previously.


Certificate




Secure Cloud HTTPS Service
Secure Cloud HTTPS Service



---

8. PHASE 3 — VULNERABILITY ASSESSMENT

THEORY

What is Nmap?

A network scanner used to identify ports, services, and vulnerabilities.

What is Metasploit?

A penetration testing platform containing exploits and scanners.

PRACTICAL

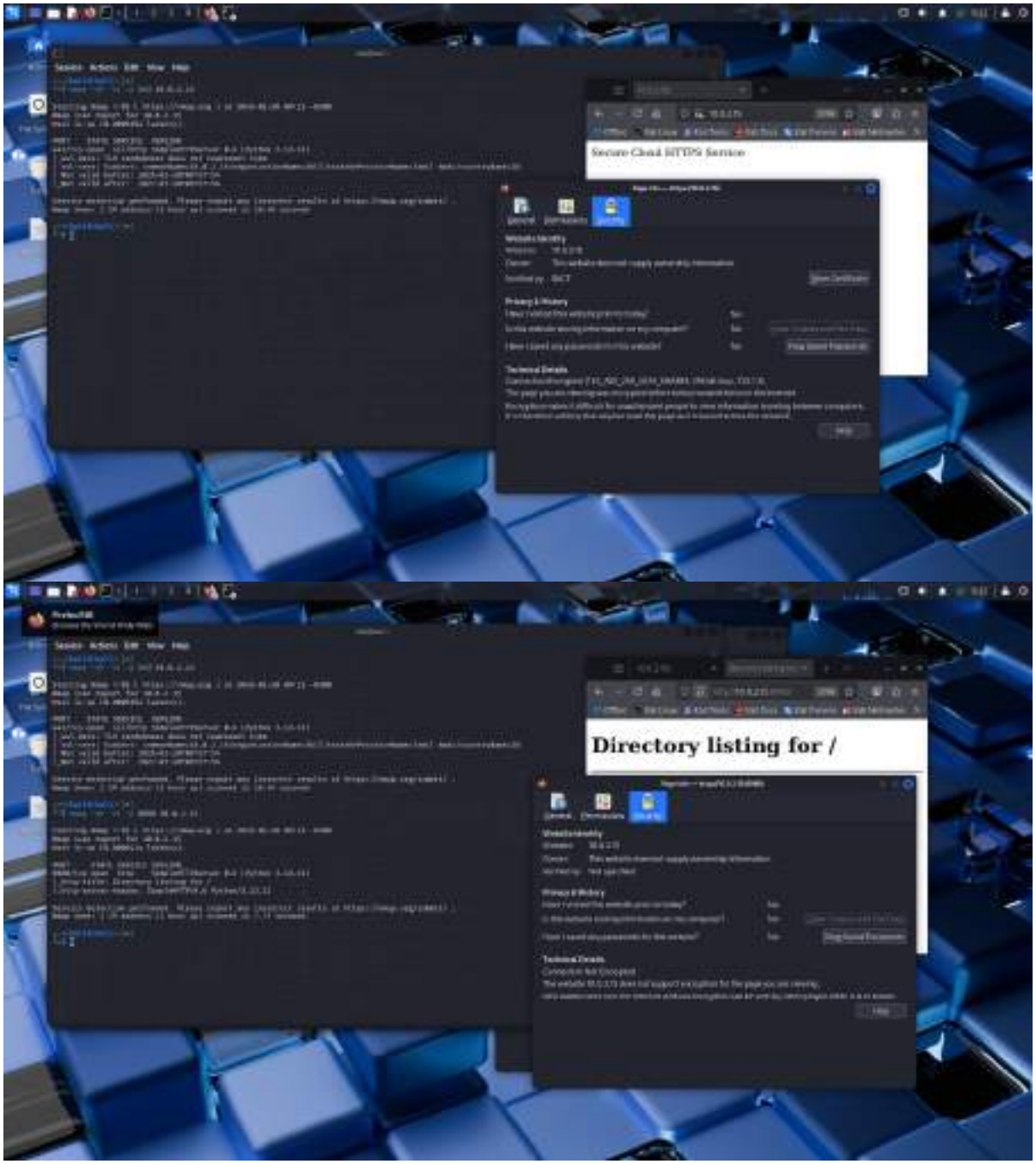
Nmap Scans

Full TLS scan:

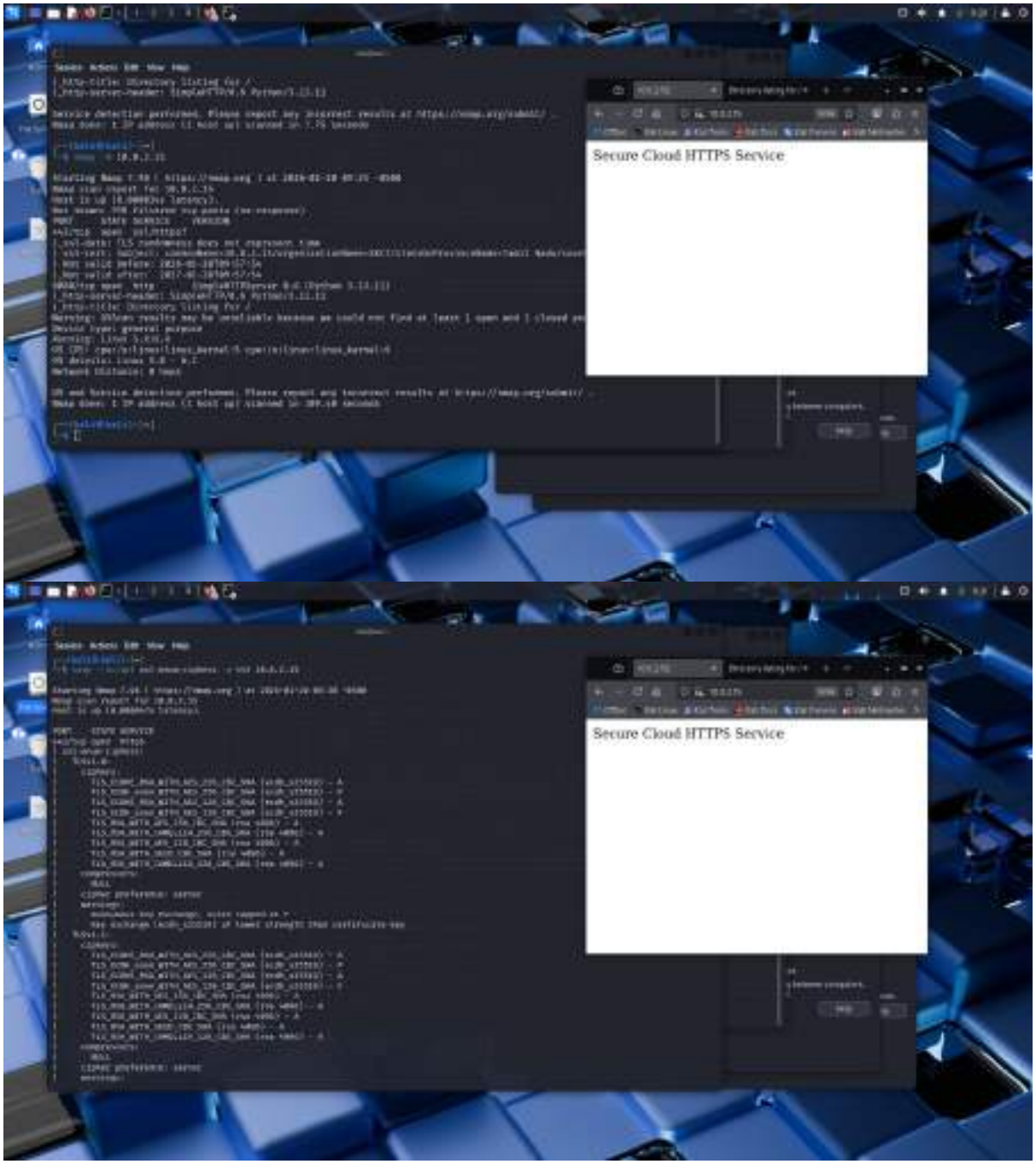
```
nmap --script ssl-enum-ciphers -p 443 10.0.2.15
```

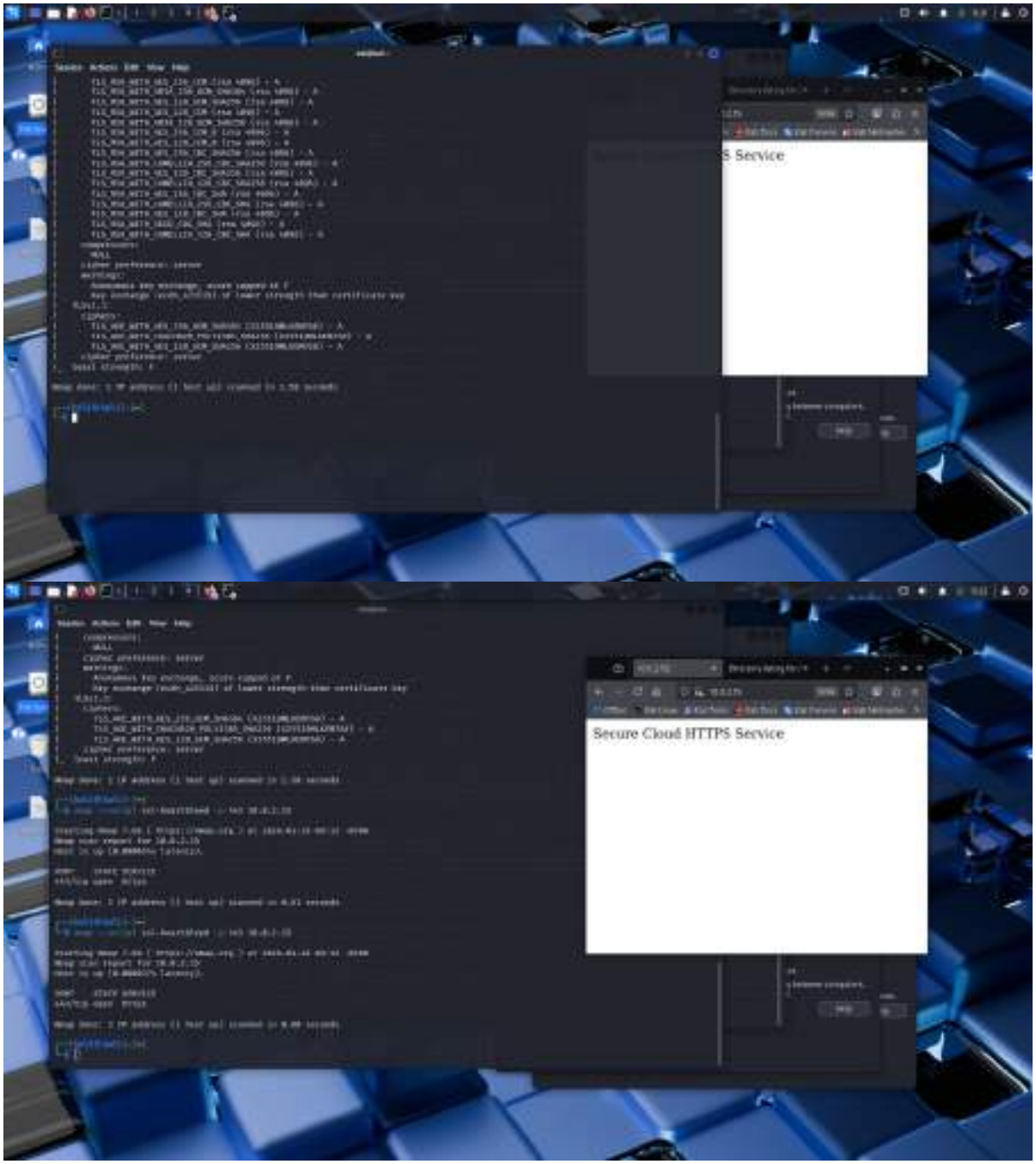
Full vulnerability scan:

```
nmap -A 10.0.2.15
```


Secure Cloud HTTPS Service

Directory listing for /


Secure Cloud HTTPS Service
Secure Cloud HTTPS Service



Risk Assessment Table

Vulnerability	Impact	Likelihood	Severity
TLS 1.0 enabled	High	High	Critical
Anonymous ciphers	Very High	High	Critical
Self-signed cert	Medium	High	High

---

9. PHASE 4 — ACCESS CONTROL: RBAC + MFA

THEORY

Why RBAC?

To enforce least privilege.

Why MFA?

To avoid password-only attacks.

PRACTICAL

Create groups:

```
sudo groupadd cloudadmin
sudo groupadd developer
sudo groupadd auditor
```

Create users:

```
sudo useradd -m admin1
sudo useradd -m dev1
sudo useradd -m audit1
```

Set directory permissions:

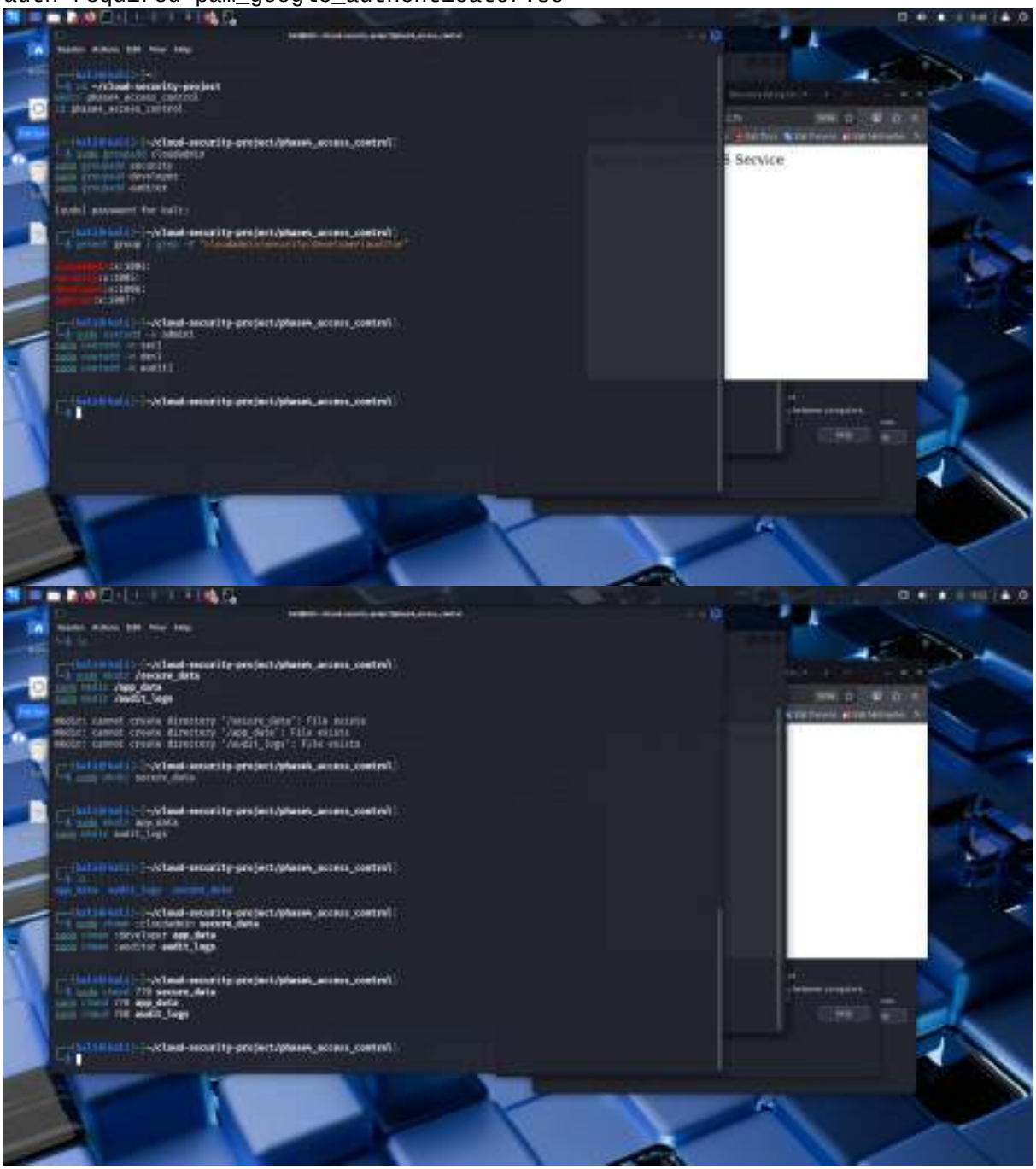
```
sudo mkdir /secure_data
sudo chown root:cloudadmin /secure_data
sudo chmod 770 /secure_data
```

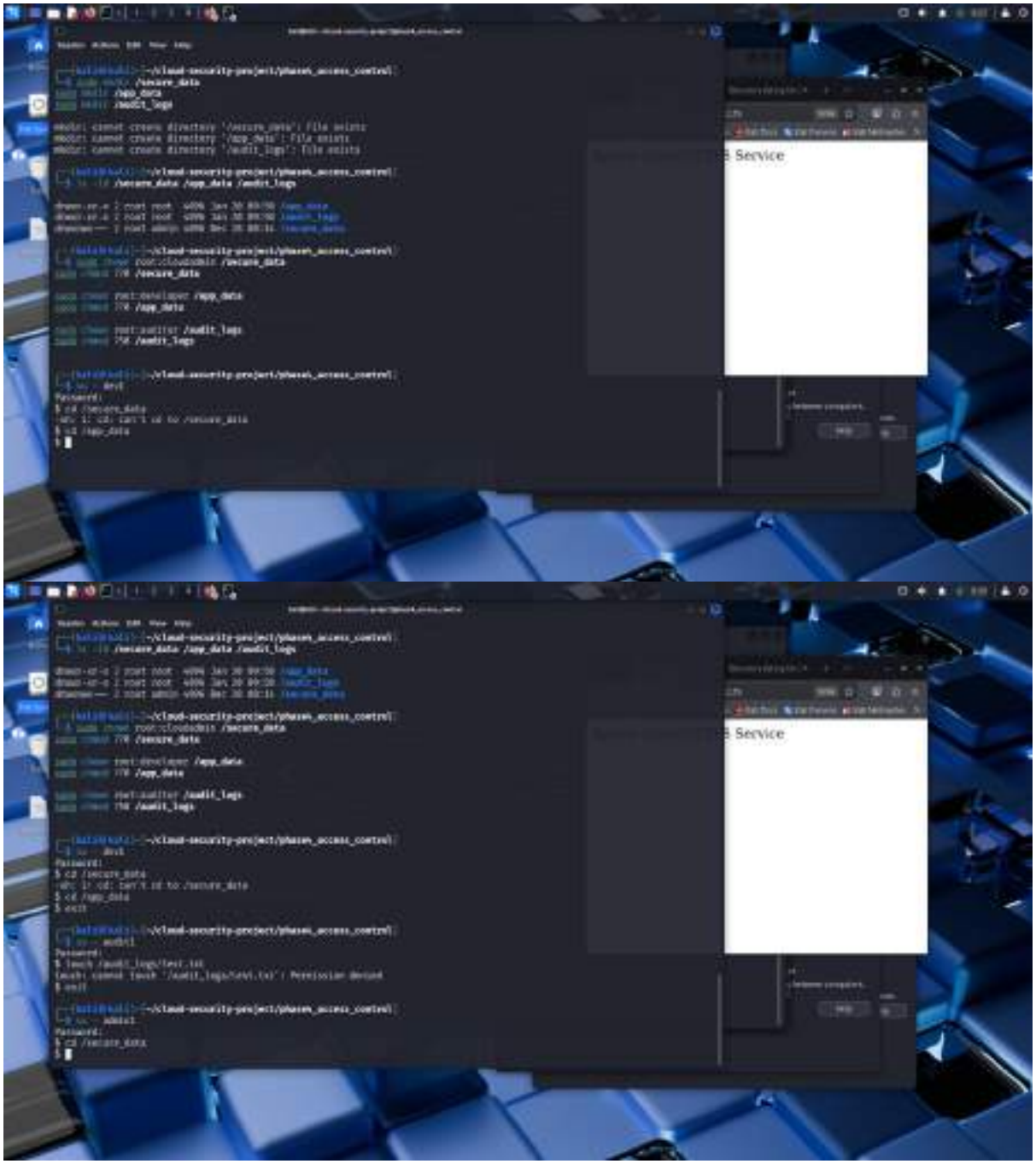
Configure MFA:

```
su - admin1
google-authenticator
```

Add to PAM:

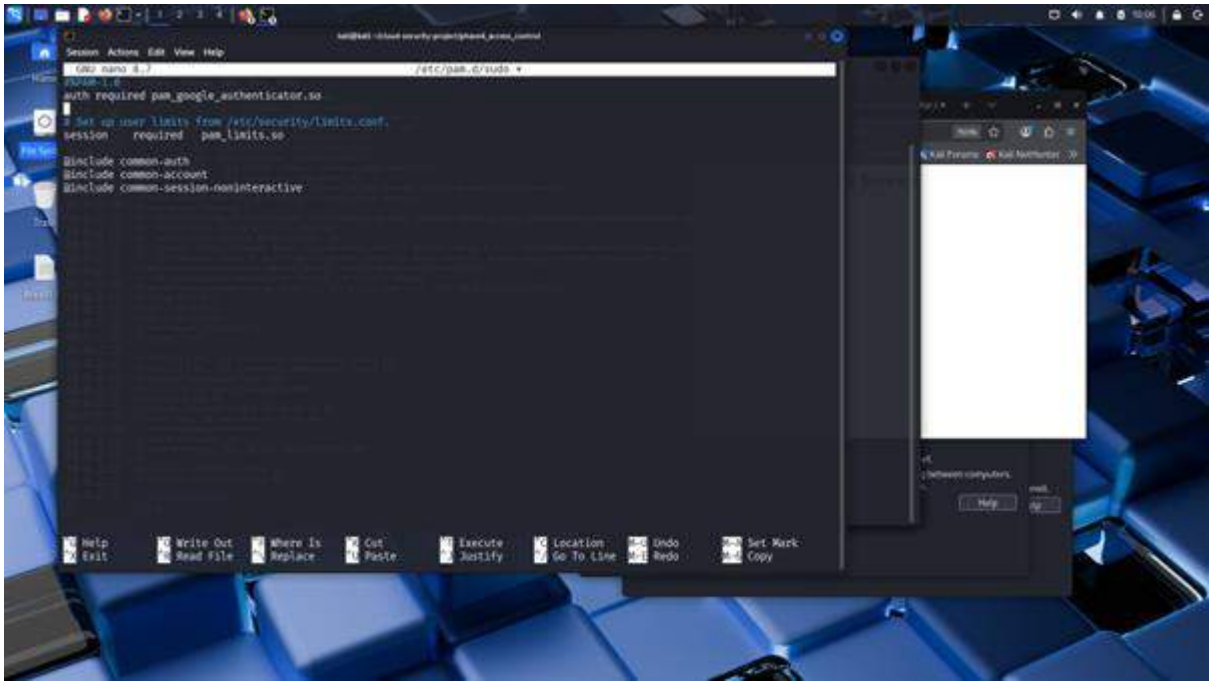
```
sudo nano /etc/pam.d/sudo
auth required pam_google_authenticator.so
```




Service

Service


Service
Service



---

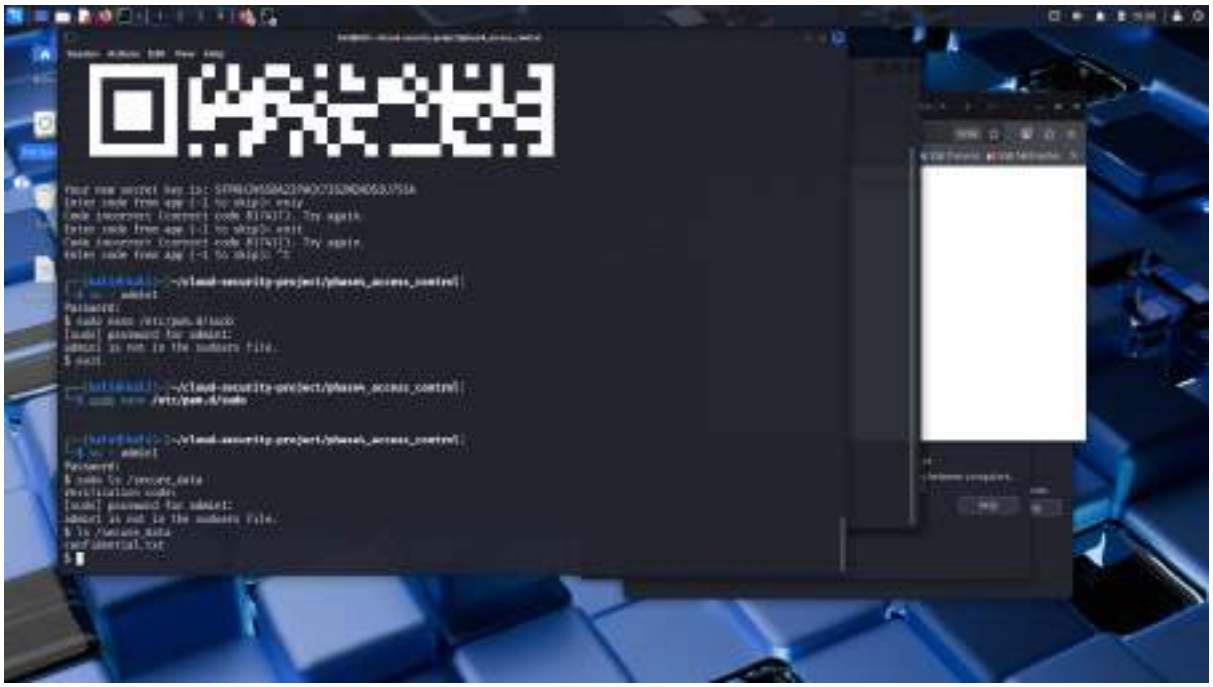
10. PHASE 5 — ATTACK VS DEFENSE DEMONSTRATION

Unauthorized Access Attempt

```
su - dev1
cd /secure_data
```

Expected:

`permission denied`



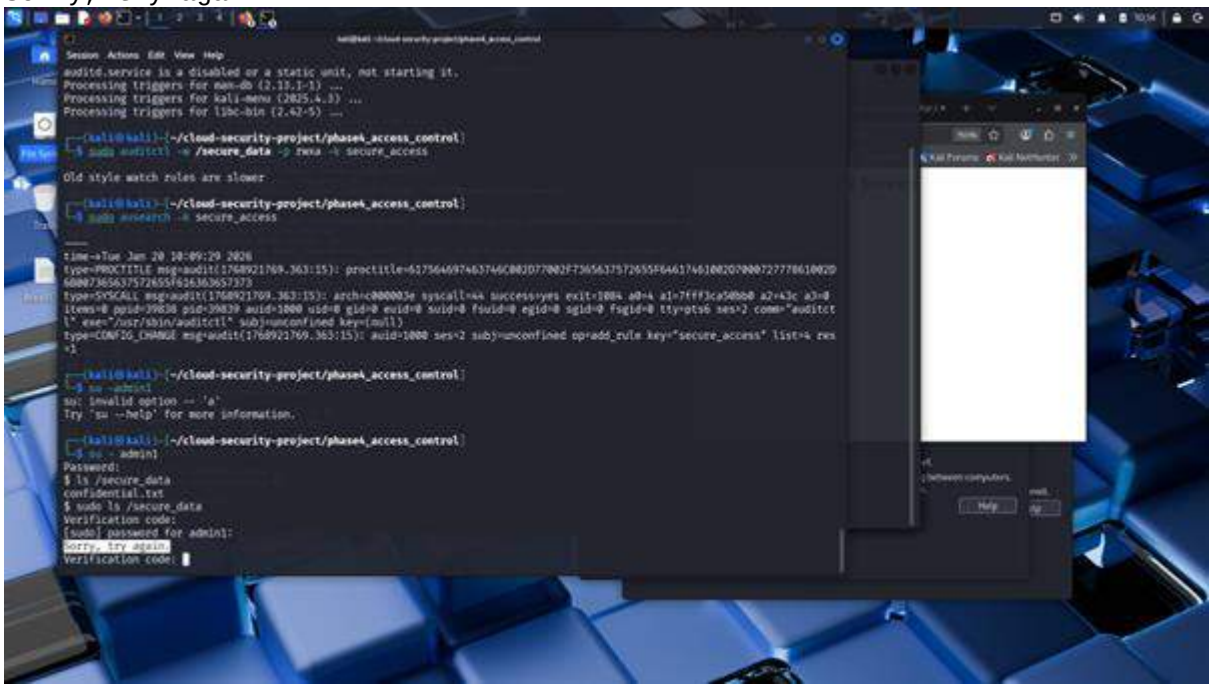
Incorrect MFA Attempt

`sudo ls /secure_data`

Enter WRONG OTP:

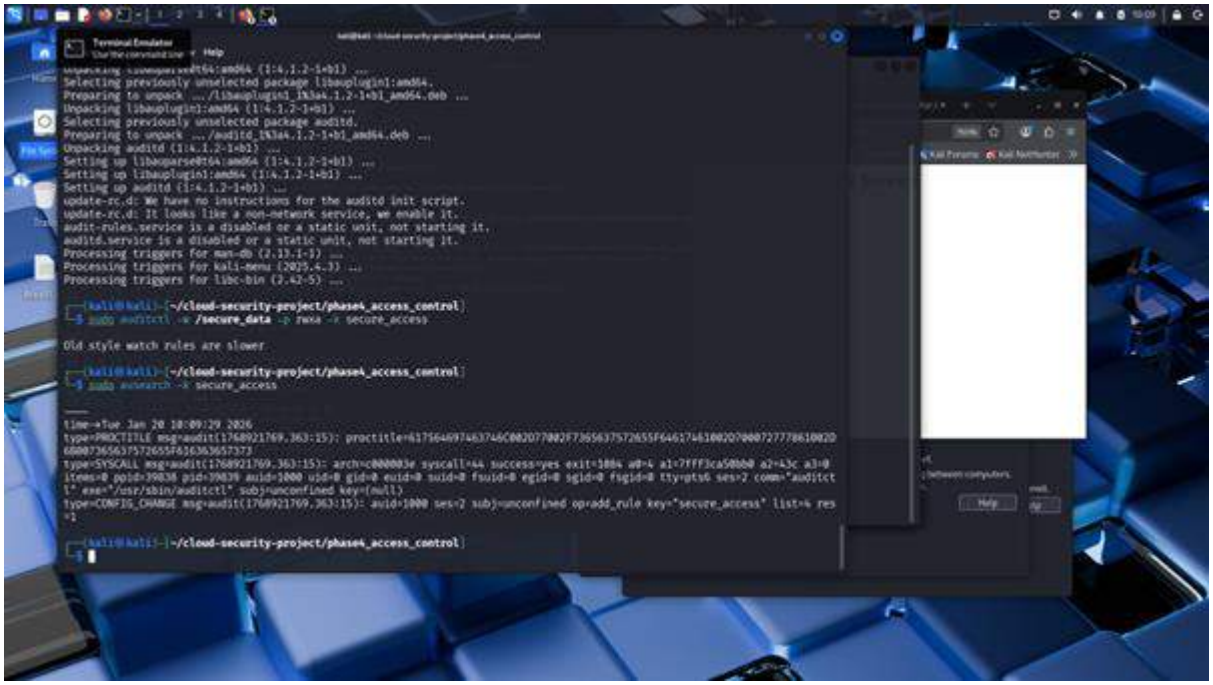
Expected:

`Sorry, try again.`



Audit Logs (Monitoring)

```
sudo auditctl -w /secure_data -p rwxa -k secure_access
sudo ausearch -k secure_access
```

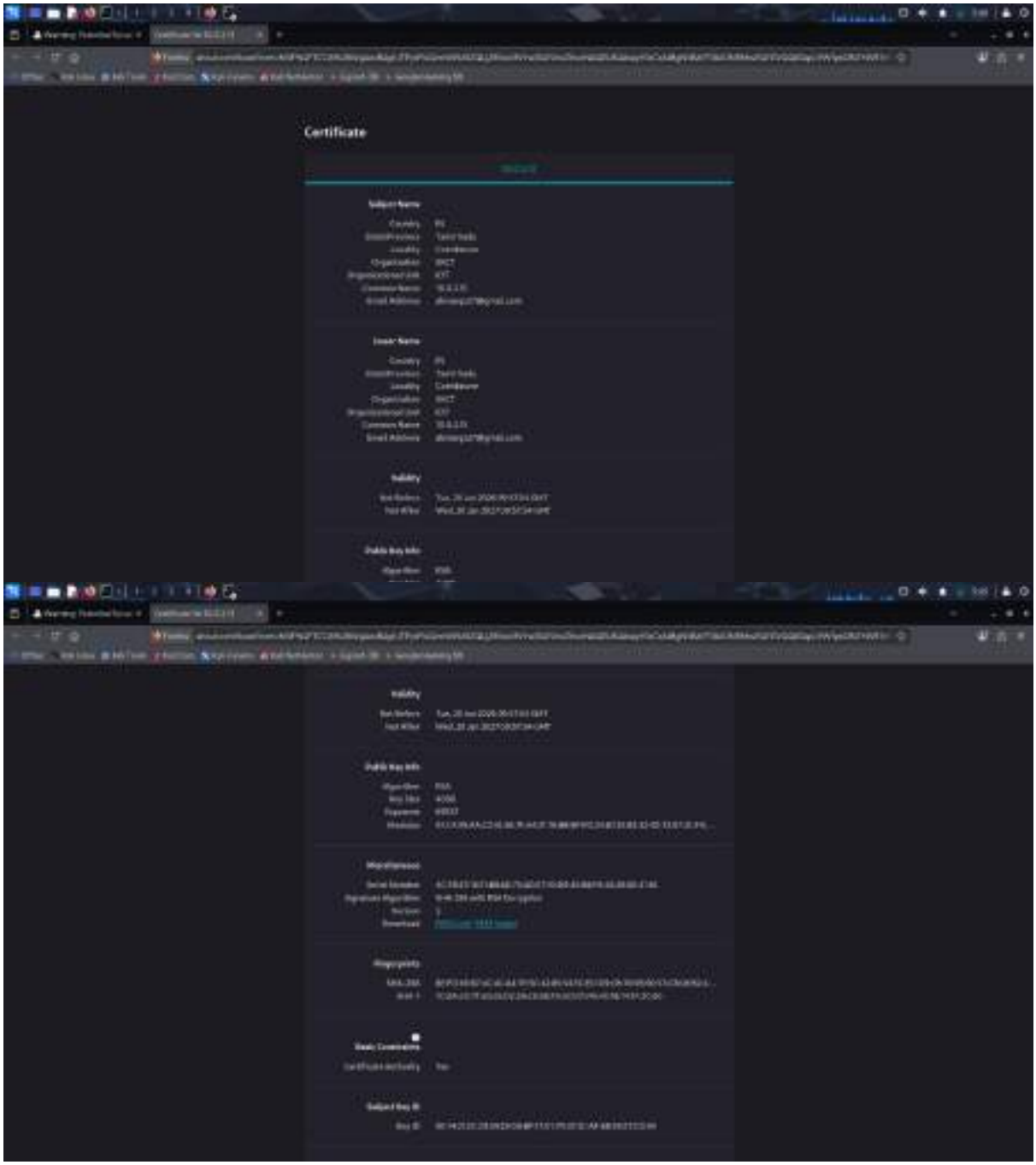


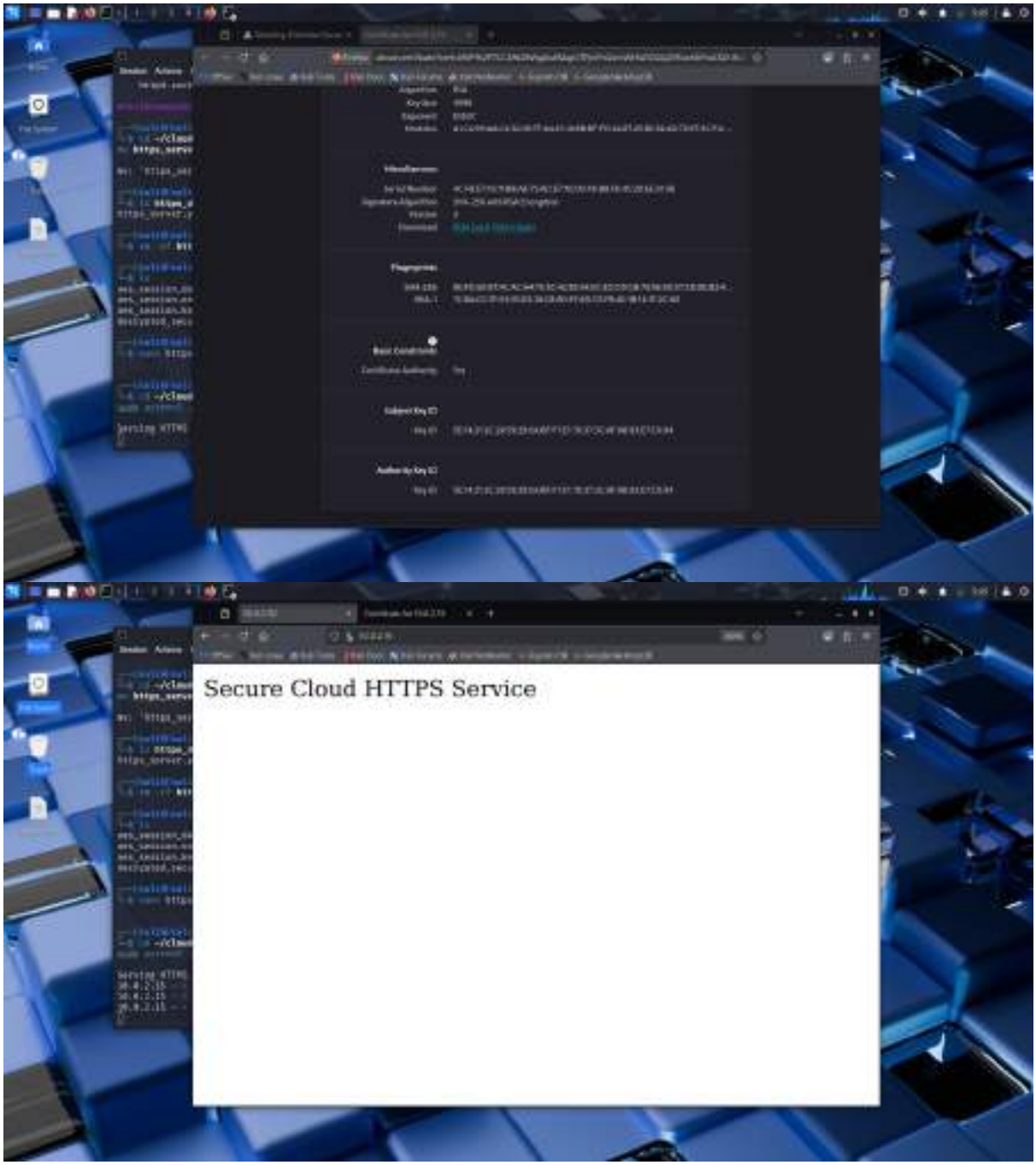
---

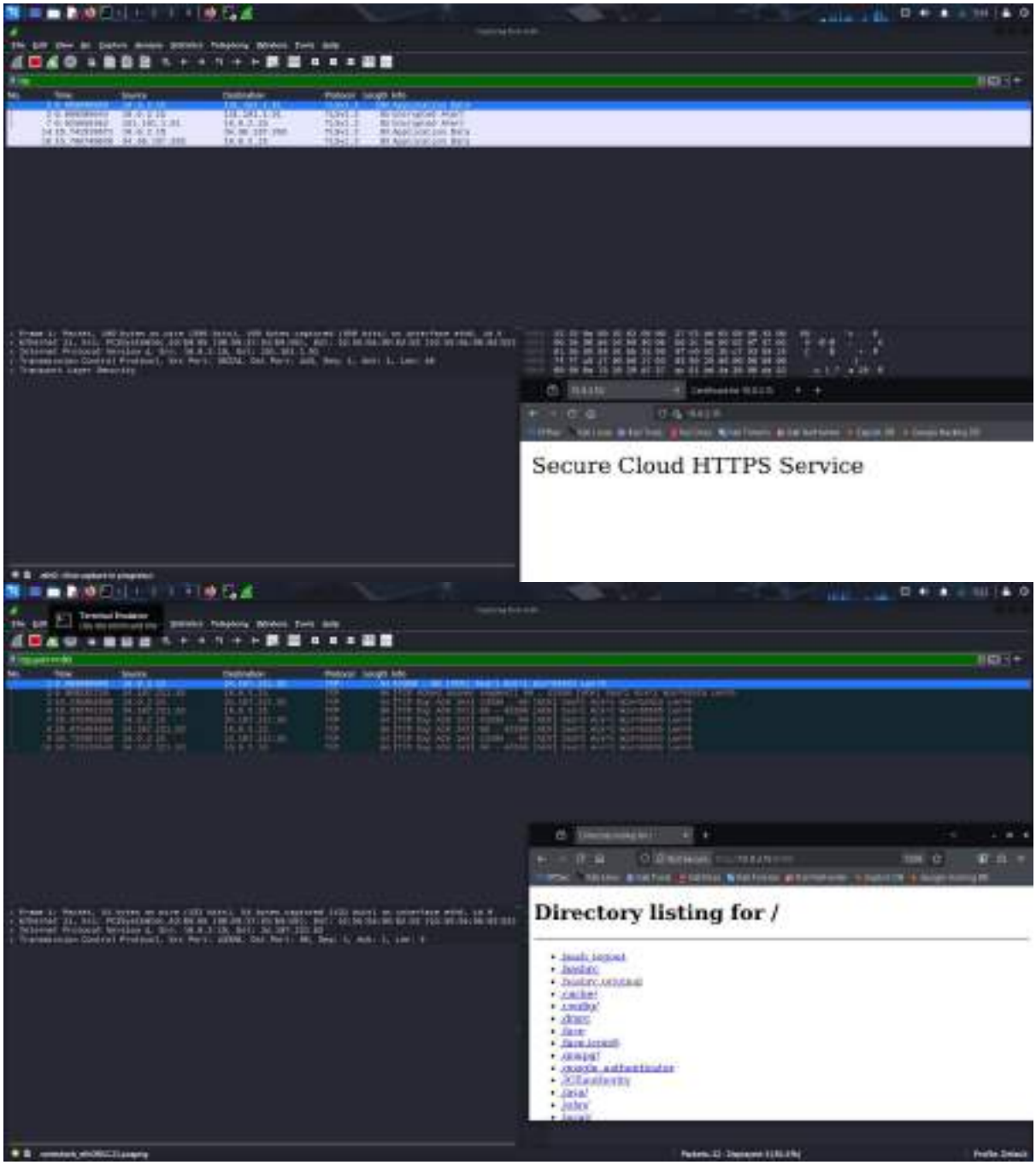
11. PHASE 6 — TOOL USAGE REPORT

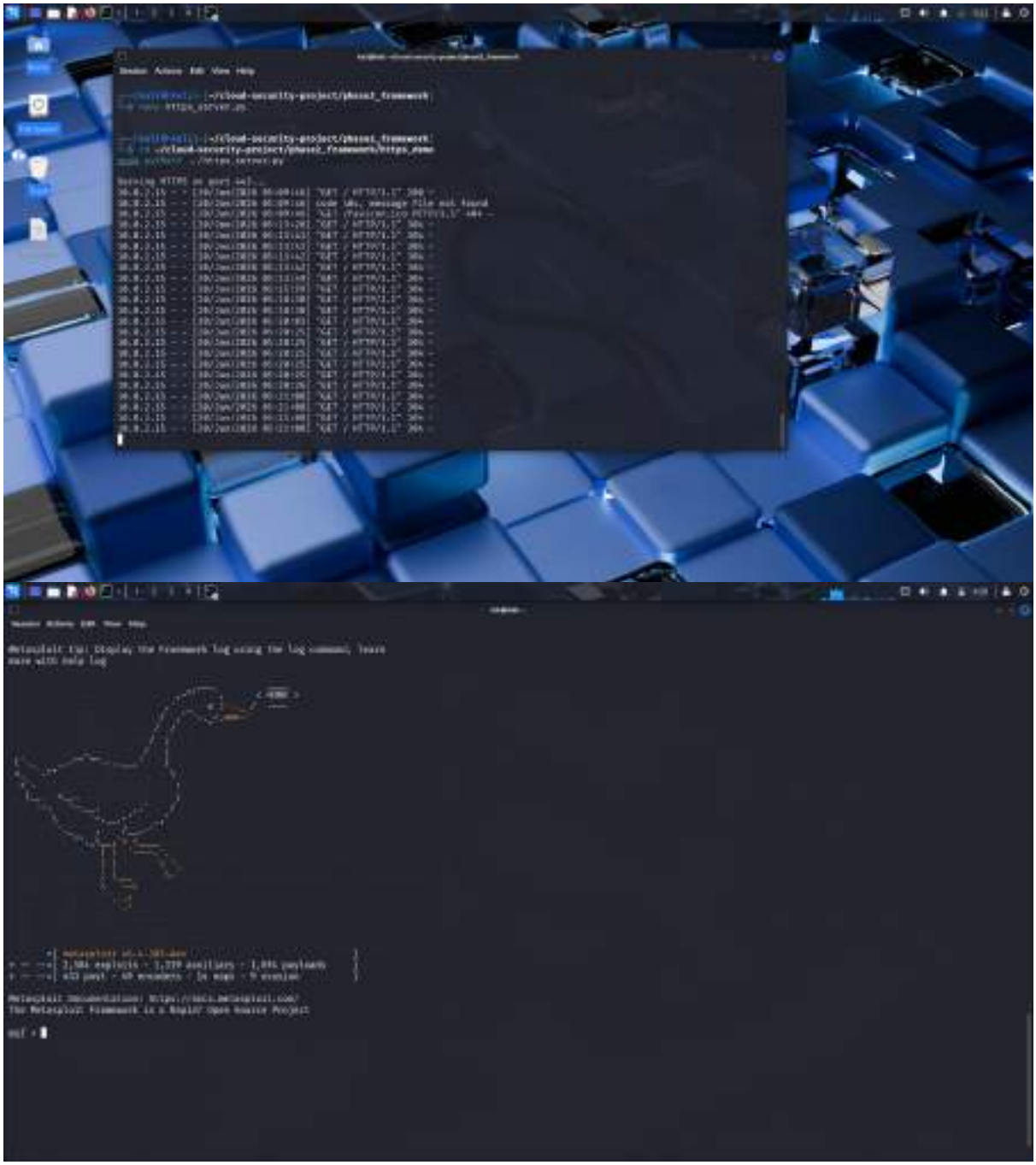
Includes:

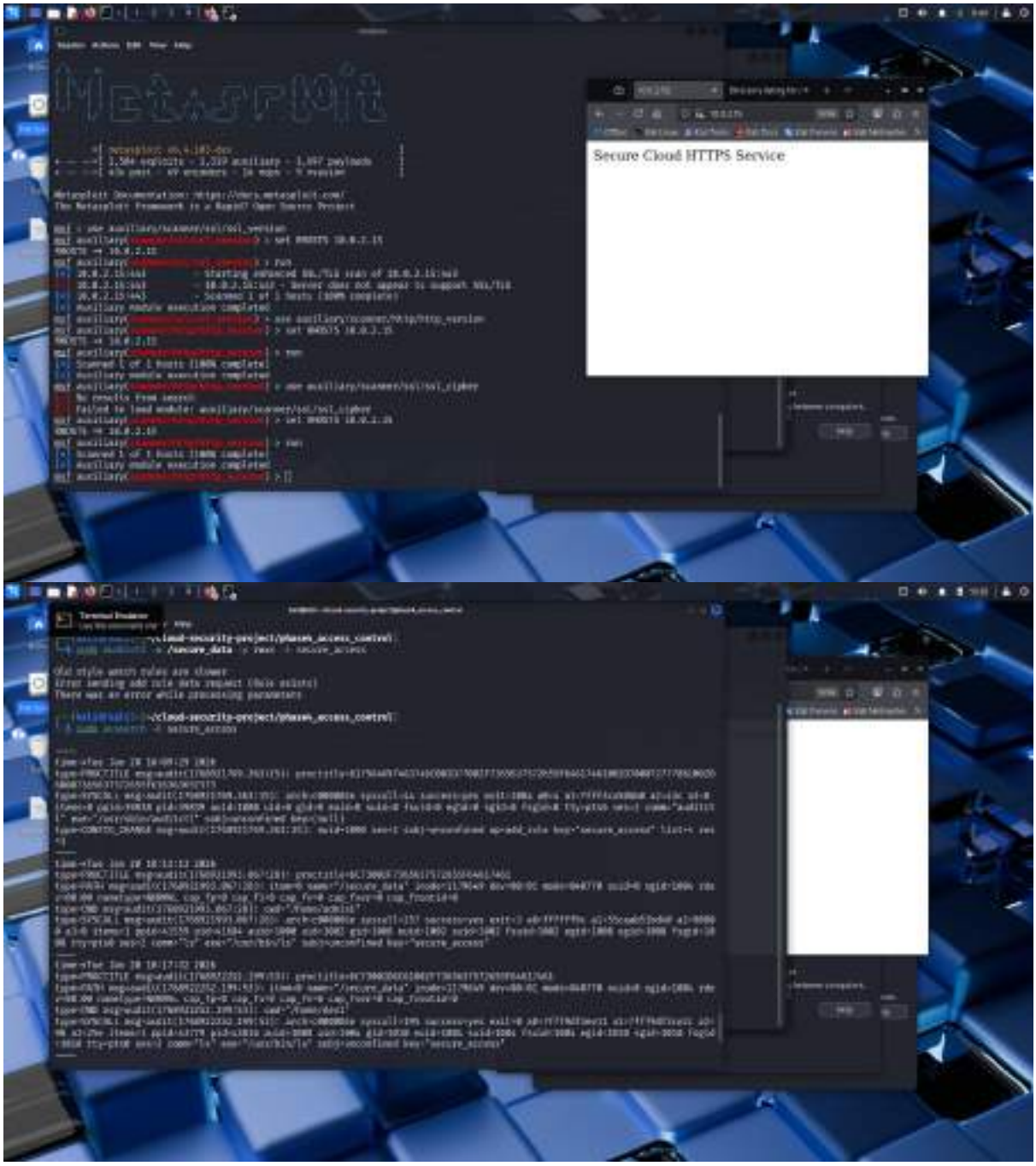
- OpenSSL usage
- Wireshark usage
- Nmap results
- Metasploit scans
- auditd logs


Certificate


Secure Cloud HTTPS Service


Secure Cloud HTTPS Service
Directory listing for /





---

12. ARCHITECTURE DIAGRAM

The Cloud Security Architecture implemented in this project follows a **defense-in-depth** approach, where multiple layers of security mechanisms work together to protect cloud workloads. This layered architecture replicates how modern cloud platforms (AWS, Azure, GCP) secure applications, identities, data, and communication channels.

The architecture is divided into **12 major components**, all interconnected to provide confidentiality, integrity, authentication, authorization, and monitoring.

12.1 End-User Client Layer

This is the entry point where a user interacts with the cloud system using:

- A browser
- A terminal
- SSH client

Why it matters:
Attackers often target endpoints first. The security model must assume untrusted clients and secure communication accordingly.

12.2 HTTPS Communication Layer (TLS 1.2+)

Before a user can access any cloud service, all communication occurs over **Transport Layer Security (TLS)**.

TLS provides:

- **Confidentiality** → encrypts data between client and server
- **Integrity** → prevents tampering
- **Authentication** → server proves its identity

In this project:

- A self-signed X.509 certificate was generated
- TLS handshake was captured in Wireshark
- Encrypted communication was verified

Why it matters:
Without TLS, sensitive data is exposed to MITM attacks.

12.3 Web / Application Layer

This layer represents the cloud application that processes requests.
In this project, it's implemented using a **Python HTTPS server**.

Core responsibilities:

- Accept client connections
- Validate requests
- Initiate authentication
- Provide encrypted content
- Forward requests to IAM, MFA, or data layers

Why it matters:
This is the boundary between external clients and internal cloud resources.

12.4 RBAC Layer (Role-Based Access Control)

RBAC enforces **Authorization**, deciding *what* each user can access.

Role examples from project:

- **cloudadmin** → full privileges
- **developer** → app folder only

- **auditor** → read-only logs

RBAC follows the principle of **Least Privilege**, restricting users to the minimum privileges needed.

Why it matters:
Prevents unauthorized access and limits blast radius during attacks.

12.5 MFA Layer (Multi-Factor Authentication)

This layer strengthens **Authentication** by adding another verification factor.
In this project, MFA is implemented using **Google Authenticator (TOTP)**.

The user must provide:

- Password (knowledge factor)
- OTP (possession factor)

Even if the password is stolen, access fails without the OTP.

Why it matters:
Stops credential-stuffing, brute force, and password theft attacks.

12.6 Encryption Layer (AES, RSA, TLS)

This deals with **data confidentiality and key management**.

Components:

- **AES-256** → symmetric encryption for large data
- **RSA-4096** → asymmetric encryption for key exchange
- **Hybrid encryption** → AES encrypted with RSA (TLS-style model)

This demonstrates the same method used in:

- HTTPS
- VPN tunnels
- Cloud KMS services

Why it matters:
Ensures both data in transit and data at rest remain protected.

12.7 Data at Rest Layer (AES-256, RSA)

This layer represents the protection applied to stored data.

Implemented via:

- AES encrypted files
- RSA encrypted keys
- Encrypted directories (/secure_data)

This mirrors cloud platforms:

- AWS S3 SSE
- Azure Storage Encryption
- Google Cloud KMS

Why it matters:
Even if attackers get the files, they cannot read them without keys.

12.8 Secure Resource Layer (/secure_data)

This is the **protected asset** of the system.

In this project:

- `/secure_data` contains sensitive content
- Only cloudadmin role can access
- Developer and auditor are blocked

Why it matters:
RBAC + MFA ensures only legitimate users gain access to sensitive cloud data.

12.9 Authentication Control Layer

This layer enforces **successful MFA + valid role** before granting access.

Sequence:

1. User authenticates with password
2. User enters OTP
3. System verifies role from RBAC

4. Access is granted *only if both pass*

This mirrors cloud access:

- AWS IAM Authentication
- Azure AD Sign-in process
- GCP IAM Authentication

Why it matters:
Combines MFA + RBAC to create a zero-trust environment.

12.10 Audit Logging Layer (auditd)

This layer records all:

- Access attempts
- Unauthorized access
- File read/write events
- Privilege elevation attempts
- MFA failures

auditd acts like a **mini SIEM** (Security Information and Event Management).

Cloud equivalents:

- AWS CloudTrail
- Azure Monitor Logs
- GCP Cloud Audit Logs

Why it matters:
Essential for forensics, legal compliance, and incident response.

12.11 Vulnerability Scanning Layer (Nmap & Metasploit)

This layer identifies:

- Weak TLS versions
- Open ports
- Cipher misconfigurations
- Service fingerprinting
- Anonymous ciphers
- Self-signed cert

These scans simulate automated cloud security assessments.

Why it matters:
Prevents exposure of weak cryptography and misconfigured services.

12.12 Security Monitoring & Incident Detection Layer

This is the **final defense layer**, combining:

- audit logs
- vulnerability reports
- MFA logs
- RBAC violations

Its purpose:

- Detect intrusions
- Identify suspicious behavior
- Trigger alerts
- Support forensic investigation

This is analogous to:

- AWS GuardDuty
- Azure Sentinel
- Google Chronicle

Why it matters:
Provides full visibility and completes the security lifecycle.

HOW ALL LAYERS WORK TOGETHER

Here is the workflow:

1. **Client connects** through HTTPS
2. **TLS encryption** protects traffic
3. **Application server** validates request
4. **RBAC** checks the user's role
5. **MFA** checks the user's identity
6. **Data encryption layer** secures stored data
7. **Secure resources** accessed based on roles
8. **auditd logs** all activities

9. **Vulnerability scanners** detect weaknesses
10. **Monitoring layer** analyzes logs and risks

This multi-layered system ensures:

✓ Confidentiality
✓ Integrity
✓ Authentication
✓ Authorization
✓ Monitoring
✓ Non-repudiation

All aligned to **ISO 27001 controls**.

---

13. CONCLUSION

This project successfully built a **complete information security framework** combining encryption, access control, vulnerability scanning, and monitoring.

The practical demonstration shows how cloud security controls work in real-world environments and aligns with ISO 27001.

---

14. REFERENCES

- OpenSSL Documentation
- NIST SP 800-52
- ISO/IEC 27001 Standard
- Wireshark User Guide
- Nmap Scripting Engine Docs
- Metasploit Unleashed