Automated security assessment of Amazon Web Services accounts using CIS Benchmark and Python 3

Oleksandr Volotovskyi[1,†], Roman Banakh[1,*,†], Andrian Piskozub[1,†] and Zoreslava Brzhevska[2,†]

[1] Lviv Polytechnic National University, 12 Stepana Bandery str., 79013 Lviv, Ukraine
[2] Borys Grinchenko Kyiv Metropolitan University, 18/2 Bulvarno-Kudryavska str., 04053 Kyiv, Ukraine

Abstract

This paper focuses on the security assessment of Amazon Web Services (AWS) accounts using the Center for Internet Security (CIS) benchmarks. Considering the rapid growth of digital technologies and the increasing reliance on cloud services for business and personal use, ensuring the security of data and accounts is paramount. The study aims to analyze and assess the security posture of AWS accounts, emphasizing automating this process through Python 3 while also exploring the application of CIS benchmarks specific to the platform. A thorough examination of existing security evaluation methods and tools is conducted, including practical tests to ensure that AWS accounts comply with CIS benchmark security standards. The paper highlights the benefits of streamlining and enhancing the process to improve overall efficiency by automating the security assessment. The findings offer valuable insights for businesses and individual AWS users, providing practical recommendations to strengthen data security and ensure high confidentiality, integrity, and availability. These recommendations can be a foundation for developing and implementing effective security strategies in cloud environments.

Keywords

AWS, CIS benchmarks, cloud security, automated security assessment, compliance, account security

1. Introduction

In today's digital world, where virtual infrastructure is becoming integral to business and personal life, data and account security is critical. This is especially true for cloud platforms such as Amazon Web Services (AWS), which offer a wide range of data storage, processing, and ans. In this context, the issue of assessing the security of AWS accounts becomes increasingly relevant. Although tools and methods for security assessment, such as the CIS Benchmark for AWS, play a crucial role in enhancing information security, it is equally important to consider comprehensive frameworks like ISO/IEC 27001:2022 and approaches such as Secure as Code [1] to address configuration management more effectively, as the lack of such comprehensive approaches could lead to significant and potentially irreversible losses.

Assessing the security of Amazon Web Services accounts using the Center for Internet Security (CIS) benchmarks [2] and automating this process [3] allows for effective monitoring and enhancement of security measures. By utilizing existing methods and tools for security evaluation, studying the CIS benchmark recommendations for AWS, conducting practical tests, and verifying account compliance with security standards, the reliability of cloud environments can be significantly improved [4].

The CIS Benchmark recommendations cover the configuration of various AWS services, such as Amazon S3 [5], Amazon EC2 [6], Amazon RDS [7], and others. These guidelines help configure access permissions, ensure effective monitoring and logging of events, and provide automated tools to verify compliance with security standards. Continuous updates in response to new threats and changes in the AWS environment ensure the relevance and effectiveness of security measures.

Assessing AWS account security with CIS Benchmark is a powerful tool for organizations looking to protect their data and services in the cloud [8, 9]. Using such tools mitigates risks and builds trust with customers and partners, enhancing the organization's reputation in the market. Implementing the AWS CIS Benchmark is thus a strategic step for any organization that aims to ensure the highest level of security for its cloud resources.

2. Measures and tools to improve security in AWS

The AWS CIS Benchmark is a set of recommendations and guidelines for setting up security in an Amazon Web

CPITS-II 2024: Workshop on Cybersecurity Providing in Information and Telecommunication Systems II, October 26, 2024, Kyiv, Ukraine
* Corresponding author.
† These authors contributed equally.
oleksandr.volotovskyi.kb.2020@lpnu.ua (O. Volotovskyi);
roman.i.banakh@lpnu.ua (R. Banakh);
andrian.z.piskozub@lpnu.ua (A. Piskozub);
z.brzhevska@kubg.edu.ua (Z. Brzhevska)

0009-0003-3102-3694 (O. Volotovskyi);
0000-0001-6897-8206 (R. Banakh);
0000-0002-3582-2835 (A. Piskozub);
0000-0002-7029-9525 (Z. Brzhevska)

Services (AWS) environment. CIS (Center for Internet Security) is a non-profit organization specializing in developing standards and methods for ensuring information technology security.

The AWS CIS Benchmark consists of recommendations and guidelines to help organizations ensure a high level of security for their accounts, resources, and services in the AWS environment. This set includes recommendations for configuring various AWS services, setting up access rights, monitoring, logging, and other security aspects.

Key features of the AWS CIS Benchmark:

- Security Standards: The recommendations define security standards for various AWS services, including Amazon S3, Amazon EC2, Amazon RDS, and others.
- Security Recommendations: The CIS Benchmark provides detailed recommendations for securely configuring AWS services and resources.
- Automated testing: The recommendations can be used for automated security testing of an AWS environment to detect security breaches and compliance.
- Updates: CIS regularly updates its recommendations to reflect changes in the AWS environment and evolving security threats.

Openness and community: CIS Benchmark is an open standard, and all its recommendations are available for the community and third-party developers.

3. Compliance achievement with AWS Services

Although AWS was created as a platform for providing virtual machine services, today, this provider offers hundreds of different services. Since there are many services and the account owner can add many users to this account, monitoring user activity is a natural need. Therefore, AWS pays great attention to services for the security of user accounts and their monitoring. In this discussion, we pay attention to such services.

3.1. Using AWS services to improve accounts' security

There are a couple of essential services that allow owners to keep accounts safe. If you neglect them, you can lose access to the account, which in turn can lead to reputational and financial losses.

3.1.1. Using the IAM service to improve accounts' security

There are several ways to provide unlimited and long-term access to AWS S3 storage.

The first way is to set access rules to the data in the storage. Also, the number of people with access to S3 storage, even for senior management, should be limited if there is no critical need for this.

The second way is to use the least privilege rule. The Identity and Access Management (IAM) service allows you to restrict access to S3 storage with the proper settings. Thus, users and programs are granted only the minimum permissions necessary to perform their work. This approach allows you to control permissions and reduces risks.

The third way is temporary access through IAM roles. The policy may be customized by adding conditions such as IP addresses to define a secure process between the application and S3 storage through IAM roles. This ensures that access to data is temporary and limited.

To prevent inappropriate permissions and privileges in AWS, it is essential to proactively manage identity and access rights by configuring user permissions according to their roles and responsibilities.

It is worth using an identity and access management (IAM) provider that allows you to assign permissions to each user or group of users. To increase the effectiveness of permissions management, it is necessary to regularly review all users with higher privileges and update their permissions to match their current roles and responsibilities. This will help avoid unauthorized use of permissions and ensure compliance with the principle of least privilege.

3.1.2. Using MFA and AWS secrets manager to improve accounts' security

To protect yourself from losing your AWS account, you should use multi-factor authentication [10] to log in to your account. This will provide an additional layer of security and make it harder for an attacker to take over your accounts, even if the data is compromised.

It is also important to constantly monitor attempts to log in to your accounts to detect possible intrusion attempts in time.

For more reliable control and security of credentials, you can use AWS Secrets Manager, which provides the ability to rotate credentials and store them in a stable environment. This method will limit the risk of credential theft and misuse of the infrastructure.

3.2. Using AWS services for monitoring and logging

In this section, we will use CloudTrail [11] to monitor and audit account activity, AWS Config [12] for automated configuration management and compliance, and AWS GuardDuty to detect potential threats to the infrastructure. Using these services allows you to maintain a high level of security and respond to possible security threats on time.

3.2.1. Using the CloudTrail service for monitoring and logging

Enabling container access logging can prevent undetected S3 storage request events. It is important to note that S3 storage does not create logs by default, so it is essential to enable this feature. From then on, the S3 bucket will log all types of requests they receive. In addition, they will log the time of each request.

Using access logs speeds up the process of detecting and responding to unexpected activity.

In addition, you should use Amazon CloudTrail. This service allows you to track and log every API call to your AWS account.

Logs contain essential information such as IP addresses, request execution time, and types of interactions.

Monitoring logs allow for the detection of dangerous or unusual activities in time.

This detection process is essential for preventing cyber threats and security breaches. CloudTrail makes it easy to receive notifications of security events, such as root logins, and receiving these notifications speeds up the response to potential risks.

3.2.2. Use AWS Config for configuration management and compliance

AWS Config allows you to evaluate, verify, and control the configuration of resources in the AWS infrastructure [13]. It also allows you to perform actions such as change logging, compliance assessment, configuration tracking, and change history.

AWS Config logs every resource configuration change, including access and security policies. This allows you to respond to any changes quickly and helps identify possible security issues.

Moreover, AWS Config allows you to create rules that automatically evaluate resource configurations against defined security policies and standards. These rules can include checking data encryption, configuration settings, and more.

Configuration history shows all the changes to resources over a particular time. Thus, configuration history allows you to analyze the causes of possible configuration problems or failures.

Also, notifications through Amazon SNS [14] allow you to receive information about configuration changes and inconsistencies in real-time, allowing you to respond quickly to potential problems.

3.2.3. Using AWS GuardDuty for continuous threat monitoring

AWS GuardDuty is a service designed to analyze event logs, network [15] traffic, and other data sources hosted by AWS to detect unusual or suspicious activity. In addition, GuardDuty uses machine learning and artificial intelligence algorithms to identify potential security threats.

The system can analyze numerous activities, such as unusual external traffic, suspicious intrusion attempts, changes in security system configuration, etc., to identify potential threats. Once such threats are detected, GuardDuty sends alerts and event reports, allowing security operators to respond immediately to potential problems.

3.3. Using AWS services to protect traffic and resources

This section will cover the use of Web Application Firewall (WAF [16]) and Network Access Control Lists (NACL) to filter traffic, protect against distributed denial of service (DDoS [17]) attacks with AWS Shield, and the role of AWS Security Hub [18] in centralized security management.

3.3.1. Use WAF and NACL to filter traffic and improve security

It would help if users used a Web Application Firewall (WAF) to protect AWS from unfiltered traffic from untrusted resources. WAF effectively filters traffic, preventing attacks and prohibited access to AWS resources.

However, it's important to remember that installing a WAF alone doesn't guarantee complete protection. To be more effective, you should combine WAF with other security measures, such as user identification and authentication, network security measures, regular security audits, and staff training on the latest threats and security practices. You should also keep your WAF rules up to date and analyze traffic to identify new threats and attacks.

Additional security can be provided through network access to control lists that manage the entry and exit of site visitors from the subnet. For example, setting up security rules in a NACL denies access to specific ports or IP addresses. Thus, by frequently checking and updating the rules, you can avoid threats and have a higher level of protection.

3.3.2. Using AWS Shield to protect against DDoS attacks

AWS Shield is an integral part of the infrastructure for protecting [19] against DDoS attacks. AWS Shield helps ensure the stability of applications and websites in the AWS environment. The main focus of AWS Shield is to protect against various types of DDoS attacks, including parser attacks at Layer 7 and attacks at Layers 3 and 4.

This service automatically detects attacks, responds quickly, and mitigates their impact on systems. In addition, AWS Shield integrates with other AWS security services, including AWS WAF or Web Application Firewall, to provide an advanced level of protection.

In addition to the standard level of protection, there is an extended version: AWS Shield Advanced. This paid plan provides additional features, such as protection against sophisticated and large-scale attacks.

3.3.3. Use AWS Security Hub for centralized security management

AWS Security Hub is a centralized service for security control and monitoring of a customer's AWS infrastructure. It offers security incident detection, automated notification processing, and integration with other security tools.

AWS Security Hub processes data from many sources, including AWS CloudTrail, AWS Config, Amazon GuardDuty, and many others, and then provides a single view of an AWS user account's security status.

Using the AWS Security Hub, you can notice potential security threats, such as unusual or suspicious activity, non-compliance with security requirements, and many other vulnerabilities. When such incidents are detected, Security Hub can send alerts and provide recommendations on how to resolve them.

AWS Security Hub centralizes and automates AWS security management, enabling you to identify and respond to potential security threats quickly. This service helps ensure high security for infrastructure and data in the AWS cloud environment.

4. Security issues in Amazon Web Services

Poor security in Amazon Web Services (AWS) is a widespread problem that exposes companies and enterprises to high risks. Issues that undermine the integrity, confidentiality, and availability of data and resources hosted in an AWS environment can mainly result in this. Incorrect configurations are often the cause of AWS security breaches. Configuration errors related to various AWS services, such as security groups or Simple Storage Service (S3) storage, can easily lead to the leakage of confidential information or unauthorized access that was not intended in any way. These mistakes can result from oversight, incompetence, or failure to follow the security rules set by AWS.

Another reason is that we need more visibility into security in the AWS environment. Monitoring all assets in large infrastructures around the clock to capture such incidents is difficult. Hackers can only go undetected with adequate monitoring and logging systems once they cause damage.

5. Threats to AWS services

This section describes the security threats associated with using Amazon S3 and AWS. Particularly, it discusses the issues of unlimited and long-term access to S3 buckets, which can lead to data leakage. Undetected request events to S3 buckets make it challenging to detect unauthorized access.

5.1. Unlimited and long-lasting access to S3 buckets

Unlimited and prolonged access to S3 buckets can create vulnerabilities. S3 (Simple Storage Service) allows you to store data that is easy and secure to access. The data is uploaded to several data centers in a selected region and stored with backups. C3 buckets can be vulnerable if they provide uncontrolled access to all users. Attackers can use read/write accounts to encrypt essential documents, change settings, or install malware. Therefore, it is crucial to manage permissions for access to buckets. Permissions can include editing, viewing, uploading/deleting, and list viewing. Reviewing permissions helps reduce AWS security risks. Using temporary access through IAM Roles is recommended by creating particular policies with conditions, such as IP addresses. This allows you to ensure a secure interaction process between your application and S3 buckets.

5.2. Unprotected request events to S3 Buckets

S3 Buckets can be a target for data theft because they process objects and store application files. Cyberattacks that lead to data breaches consist of countless requests to access the data in these buckets. Without logs of these requests, they go undetected until it's too late.

S3 Buckets do not generate logs by default, so this feature must be enabled manually. Once enabled, S3 Buckets will create access logs for any request made to them, with details such as the type of request, the resource used for the request, and date and time stamps. Having access logs helps you assess AWS security risks by tracking requests and recognizing the type of requests made. Access logs enable you to assess AWS security risks by monitoring requests and recognizing the type of requests made.

An AWS security audit would be a great approach to identify such misconfigurations.

5.3. Unfiltered traffic from unreliable sources

When traffic to the AWS instances or load balancers is unrestricted, attackers can obtain information about the application to attack. To avoid this, you must restrict access to instances and control traffic.

DDoS attacks are possible without proper network configuration and can quickly overwhelm the system. Restricting traffic from suspicious sources will reduce risks and reduce the attack surface.

Security groups that function as a firewall allow only authorized traffic. They only allow access from specific IP addresses or ranges. A Network Access Control List (NACL) provides an additional layer of security for subnets. Users must ensure that the NACL does not allow access from all IP addresses or ports and creates new restrictive rules.

6. Automated assessment of compliance with CIS Benchmark controls

6.1. Identity and access management section

The code is implemented in Python to check and collect information about the security of accounts in AWS Identity and Access Management (IAM). It uses the boto3 [20] and pytz libraries to interact with AWS services and work with data. It checks various aspects by the CIS benchmark controls of the Identity and Access Management section. The results of the checks are saved in a JSON file.

First, we need to import a few important libraries that will be used in our script:

- boto3: This is the core AWS SDK library for Python that allows you to interact with AWS services, specifically S3.
- json: Used to work with JSON data, in which we will store the results of the check.
- subprocess: Allows you to execute system commands through the shell, which is necessary for some specific queries.
- xml.etree.ElementTree: A standard library for processing XML. (In our case, it is not used directly, but may be needed for future integrations.)

The first check we will perform is to evaluate the encryption of data in the S3 bucket. According to the CIS Benchmark, all buckets must be encrypted using the AES-256 algorithm. To do this, we use the check_s3_bucket_encryption() function. It calls the S3 API and checks whether encryption is enabled and whether AES-256 is used.

```
import boto3
from botocore.exceptions import ClientError

def check_s3_bucket_encryption(bucket_name):
    s3_client = boto3.client('s3')
    try:
        encryption_response =
s3_client.get_bucket_encryption(Bucket=bucket_name)
        encryption_configuration =
encryption_response.get('ServerSideEncryptionConfiguration', {})

        sse_algorithm = encryption_configuration.get('Rules',
[{}])[0]\
            .get('ApplyServerSideEncryptionByDefault', {})\
            .get('SSEAlgorithm', '')

        return sse_algorithm in ['AES256', 'aws:kms']

    except s3_client.exceptions.NoSuchBucketEncryption:
        print(f"Bucket '{bucket_name}' does not have
encryption configured.")
        return False
    except ClientError as e:
        print(f"Error checking S3 bucket encryption for
{bucket_name}: {e}")
        return False
```

The function makes a request to the S3 API to get the encryption configuration of the bucket. If the bucket is encrypted with AES-256, the function returns True. Otherwise, it returns False. In case of an error (for example, if encryption is not configured or the batch does not exist), a corresponding message is displayed.

The next step is to make sure that all traffic to the S3 bucket is transmitted over a secure connection (SecureTransport) [21]. To do this, we use a system command through the subprocess library that searches the bucket policy [22] for the requirement to use HTTPS.

```
import boto3
import json
from botocore.exceptions import ClientError

def check_secure_transport(bucket_name):
    s3 = boto3.client('s3')
    try:
        response =
s3.get_bucket_policy(Bucket=bucket_name)
        policy = json.loads(response['Policy'])

'aws:SecureTransport'
        for statement in policy.get("Statement", []):
            if "Condition" in statement and "Bool" in
statement["Condition"]:
                if "aws:SecureTransport" in
statement["Condition"]["Bool"]:
                    if
statement["Condition"]["Bool"]["aws:SecureTransport"] ==
"true":
                        return True
        return False

    except ClientError as e:
        print(f"Error checking secure transport for bucket
{bucket_name}: {e}")
        return False
```

The aws s3api get-bucket-policy command is used to retrieve an S3 bucket policy that is checked for the presence of a SecureTransport key. If the policy contains a requirement to use only a secure connection, the function returns True, otherwise, it returns False.

Another important recommendation is to enable versioning of the batch and additional protection with MFA (Multi-Factor Authentication) [23]. Versioning helps to save all changes made to files, and MFA protects against accidental or malicious deletions.

```
import boto3
from botocore.exceptions import ClientError

def check_bucket_versioning_mfa(bucket_name):
    s3_client = boto3.client('s3')
    try:
        versioning_response =
s3_client.get_bucket_versioning(Bucket=bucket_name)

        versioning_status = versioning_response.get('Status',
'Disabled')
        if versioning_status == 'Enabled':
            mfa_delete_status =
versioning_response.get('MFADelete', 'Disabled')
            return mfa_delete_status == 'Enabled'
        else:
            return False

    except ClientError as e:
        print(f"Error checking S3 bucket versioning and
MFADelete: {e}")
        return False
```

The function checks whether versioning is enabled for a particular batch. If versioning is enabled, it also checks whether MFA Delete is enabled. Returns True if both features are enabled, or False otherwise.

The last check concerns the public access blocking settings. It is important to ensure that S3 buckets are not publicly accessible unless it is a conscious choice. To do this, we use the check_public_access_block() function.

```
import boto3
from botocore.exceptions import ClientError

def check_s3_public_access_block(bucket_name):
    s3_client = boto3.client('s3')
    try:
        access_block_response =
s3_client.get_public_access_block(Bucket=bucket_name)
        config =
access_block_response.get('PublicAccessBlockConfiguration', {})

        block_public_acls = config.get('BlockPublicAcls',
False)
        ignore_public_acls = config.get('IgnorePublicAcls',
False)
        block_public_policy = config.get('BlockPublicPolicy',
False)
        restrict_public_buckets =
config.get('RestrictPublicBuckets', False)

        return block_public_acls and ignore_public_acls and
block_public_policy and restrict_public_buckets

    except ClientError as e:
        print(f"Error checking public access block for bucket
{bucket_name}: {e}")
        return False
```

The function checks the Public Access Block configuration to ensure that all policies that block public access are enabled. Returns True if all these options are enabled.

6.2. Elastic Compute Cloud (EC2) section

This code snippet implements checking the default encryption settings for EBS (Elastic Block Store) [24] objects in different AWS regions.

Using the AWS API for each EC2 region, it checks whether the default encryption for EBS is set in each of them. This allows you to ensure that security settings are consistent across all regions where AWS infrastructure is used.

This function checks whether EBS encryption is enabled by default in the specified region.

```
import boto3
from botocore.exceptions import ClientError

def check_ebs_encryption_by_default(region):
    try:
        ec2_client = boto3.client('ec2', region_name=region)
        response =
ec2_client.get_ebs_encryption_by_default()
        return response.get('EbsEncryptionByDefault', False)
    except ClientError as e:
        print(f"Error checking EBS encryption by default for
region {region}: {e}")
        return False
```

The nascent function collects all AWS regions, checks the default EBS encryption status in each region, and saves the results.

```
import boto3
from botocore.exceptions import ClientError

def check_ebs_encryption_by_default(region):
    try:
        ec2_client = boto3.client('ec2', region_name=region)
        response =
ec2_client.get_ebs_encryption_by_default()
        return response.get('EbsEncryptionByDefault', False)
    except ClientError as e:
        print(f"Error checking EBS encryption by default for
region {region}: {e}")
        return False

def write_results_to_file(results):
    with open('ebs_encryption_results.txt', 'w') as file:
        for region, is_encrypted in results.items():
            file.write(f"{region}: {'Enabled' if is_encrypted else
'Disabled'}\n")

def main():
    try:
        ec2_client = boto3.client('ec2')
        ec2_regions = [region['RegionName'] for region in
ec2_client.describe_regions()['Regions']]
        results = {}
        for region in ec2_regions:
            result = check_ebs_encryption_by_default(region)
            results[region] = result
        write_results_to_file(results)
    except ClientError as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    main()
```

6.3. Relational database service section

This code snippet implements the verification of some security [25] aspects of the RDS (Relational Database Service) database in AWS in different regions. It checks whether the data storage is encrypted, whether automatic updates of minor versions of RDS are enabled, and whether the databases are available for public access.

The function checks whether encryption is enabled for each database across all AWS regions.

```
import boto3
from botocore.exceptions import ClientError

def check_rds_storage_encryption():
    results = {}

    try:
        ec2_client = boto3.client('ec2')
        regions = [region['RegionName'] for region in
ec2_client.describe_regions()['Regions']]

        for region in regions:
            rds_client = boto3.client('rds', region_name=region)
            try:
                db_instances =
rds_client.describe_db_instances()['DBInstances']

                for db_instance in db_instances:
                    db_instance_identifier =
db_instance['DBInstanceIdentifier']
                    storage_encrypted =
db_instance.get('StorageEncrypted', False)

                    if region not in results:
                        results[region] = {}
                    results[region][db_instance_identifier] =
{"StorageEncrypted": storage_encrypted}
            except ClientError as e:
                print(f"Error describing DB instances in region
{region}: {e}")

    except ClientError as e:
        print(f"Error describing regions: {e}")

    return results
```

The following function checks whether the database is publicly available.

```
import boto3
from botocore.exceptions import ClientError

def check_rds_publicly_accessible():
    results = {}

    try:
        ec2_client = boto3.client('ec2')
        regions = [region['RegionName'] for region in
ec2_client.describe_regions()['Regions']]

        for region_name in regions:
            rds_client = boto3.client('rds',
region_name=region_name)
            try:

                db_instances =
rds_client.describe_db_instances()['DBInstances']

                for db_instance in db_instances:
                    db_instance_identifier =
db_instance['DBInstanceIdentifier']
                    publicly_accessible =
```

```
db_instance.get('PubliclyAccessible', False)

                if region_name not in results:
                    results[region_name] = {}
                results[region_name][db_instance_identifier]
= {"PubliclyAccessible": publicly_accessible}
            except ClientError as e:
                print(f"Error describing DB instances in region
{region_name}: {e}")

    except ClientError as e:
        print(f"Error describing regions: {e}")

    return results
```

6.4. Logging section

This code snippet implements the verification of compliance with various security requirements and settings in the CloudTrail service, which provides event logging in AWS. It checks the presence and status of various components, such as event logging, the inclusion of various types of events, the time of the last log delivery to CloudWatch, the status of the configuration logger, encryption and KMS [26] key settings, KMS key rotation, and others.

The function determines whether logging is enabled for each CloudTrail route.

```
import boto3
from botocore.exceptions import ClientError

def describe_trails():
    cloudtrail_client = boto3.client('cloudtrail')
    try:
        response = cloudtrail_client.describe_trails()

        return response.get('trailList', [])
    except ClientError as e:
        print(f"Error describing trails: {e}")
        return []

if __name__ == "__main__":
    trails = describe_trails()
    if trails:
        for trail in trails:
            print(trail)
    else:
        print("No trails found.")
```

The script checks whether CloudTrail uses event logging on S3, which allows event auditing.

```
import boto3
from botocore.exceptions import ClientError

def check_cloudtrail_s3_logging():
    cloudtrail_client = boto3.client('cloudtrail')
    try:

        response = cloudtrail_client.describe_trails()
        trails = response.get('trailList', [])


        for trail in trails:
            logging_s3_enabled = trail.get('S3BucketName') is
not None
            if logging_s3_enabled:
                return True
        return False

    except ClientError as e:
        print(f"Error describing trails: {e}")
        return False
```

The function checks if event log encryption is enabled using AWS KMS.

```
import boto3
from botocore.exceptions import ClientError

def check_cloudtrail_sse_kms():
    cloudtrail_client = boto3.client('cloudtrail')
    try:

        response = cloudtrail_client.describe_trails()
        trails = response.get('trailList', [])

        for trail in trails:
            kms_key_id = trail.get('KmsKeyId')
            if kms_key_id:
                return True
        return False

    except ClientError as e:
        print(f"Error describing trails: {e}")
        return False
```

The script checks whether automatic encryption key rotation is enabled to improve security.

```
import boto3
from botocore.exceptions import ClientError

def get_kms_key_id():

    return 'your-kms-key-id'

def check_kms_key_rotation():
    kms_client = boto3.client('kms')
    key_id = get_kms_key_id()

    try:
        response =
kms_client.get_key_rotation_status(KeyId=key_id)
        rotation_enabled = response.get('KeyRotationEnabled',
False)
        return rotation_enabled

    except ClientError as e:
        print(f"Error getting key rotation status: {e}")
        return False
```

6.5. Networking section

This code snippet implements checking compliance with various security aspects in the AWS environment. It checks access to the network access control lists (ACLs) [27] and security groups (SGs) [28] for the corresponding ports (22, 3389) from any IP address, checks access to security groups for IPv6, checks for restrictions in the default offline security group, and checks the routing tables for routing rules for the special subnet.

This feature checks whether the network ACLs in the specified region allow unrestricted access to ports 22 (SSH) and 3389 (RDP).

```
import boto3
from botocore.exceptions import ClientError

def check_network_acl_access(region):
    ec2 = boto3.client('ec2', region_name=region)
    try:
        response = ec2.describe_network_acls()
        for acl in response.get('NetworkAcls', []):
            for entry in acl.get('Entries', []):
                if ('PortRange' in entry and
                    entry.get('CidrBlock') == '0.0.0.0/0' and
                    entry.get('PortRange', {}).get('From') in [22,
3389] and
                    entry.get('RuleAction') == 'allow'):
                    return False
        return True

    except ClientError as e:
        print(f"Error describing network ACLs in region
{region}: {e}")
        return False
```

This feature checks to see if the default security group for a VPC [29] in the region has open rules.

```
import boto3
from botocore.exceptions import ClientError

def check_default_security_group(region, vpc_id):
    ec2 = boto3.client('ec2', region_name=region)
    try:
        response = ec2.describe_security_groups(
            Filters=[
                {'Name': 'vpc-id', 'Values': [vpc_id]},
                {'Name': 'group-name', 'Values': ['default']}
            ]
        )

        for group in response.get('SecurityGroups', []):
            if group.get('IpPermissions'):
                return False
        return True

    except ClientError as e:
        print(f"Error describing security groups in region
{region}: {e}")
        return False
```

This feature verifies that the VPC routing tables are properly configured, including peering connections.

```
import boto3
from botocore.exceptions import ClientError

def check_route_tables(region, vpc_id,
peering_connection_id, desired_cidr_block):
    ec2 = boto3.client('ec2', region_name=region)
    try:
        response = ec2.describe_route_tables(Filters=[{'Name':
'vpc-id', 'Values': [vpc_id]}])
        for route_table in response.get('RouteTables', []):
            for route in route_table.get('Routes', []):
                if route.get('GatewayId') ==
peering_connection_id and
route.get('DestinationCidrBlock') == desired_cidr_block:
                    return False
        return True

    except ClientError as e:
        print(f"Error describing route tables in region {region}:
{e}")
        return False
```

7. Conclusions

The following conclusions and results were reached from analyzing and assessing the security of Amazon Web Services (AWS) accounts using the CIS (Center for Internet Security) benchmark standards.

First, we reviewed the existing methods and tools for assessing AWS account security in detail. Studying tools such as AWS Config, AWS Security Hub, and specialized solutions from third-party vendors allowed us to form a holistic view of the capabilities and limitations of different approaches to ensuring security in cloud environments.

Secondly, an in-depth study of the CIS Benchmark recommendations for AWS revealed critical security settings for various AWS services, including Amazon S3, Amazon EC2, Amazon RDS, and others. The recommendations cover a wide range of settings, such as access control, event monitoring, and logging of user actions, allowing for comprehensive cloud resource security.

Practical tests and an assessment of AWS accounts' compliance with CIS Benchmark security standards have confirmed the effectiveness of implementing these recommendations. In particular, automating the security assessment process using tools that integrate with AWS has significantly increased the efficiency and speed of identifying and fixing potential vulnerabilities.

References

[1] O. Vakhula, et al., Security as Code Concept for Fulfilling ISO/IEC 27001: 2022 Requirements, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3654 (2024) 59–72.

[2] CIS AWS Benchmark v1.5.0. URL: https://www.scribd.com/document/624550364/CIS-Amazon-Web-Services-Foundations-Benchmark-v1-5-0

[3] Automated Approach to Evaluation and Security of AWS Services using Python and "CIS Benchmark", in: 2nd International Scientific Conference (2024) 141–142.

[4] V. Shapoval, et al., Automation of Data Management Processes in Cloud Storage, in: Workshop on Cybersecurity Providing in Information and Telecommunication Systems, CPITS, vol. 3654 (2024) 410–418.

[5] Amazon S3. URL: https://aws.amazon.com/s3/

[6] Amazon EC2. URL: https://aws.amazon.com/ec2/

[7] Amazon Relational Database Service. URL: https://aws.amazon.com/rds/

[8] Practical Aspects of Using Fully Homomorphic Encryption Systems to Protect Cloud Computing | P. Anakhov, et al., Evaluation Method of the Physical Compatibility of Equipment in a Hybrid Information Transmission Network, Journal of Theoretical and Applied Information Technology 100(22) (2022) 6635–6644.

[9] V. Zhebka, et al., Optimization of Machine Learning Method to Improve the Management Efficiency of Heterogeneous Telecommunication Network, in: Workshop on Cybersecurity Providing in Information

and Telecommunication Systems, vol. 3288 (2022) 149–155.
[10] D. Shevchuk, et al., Designing Secured Services for Authentication, Authorization, and Accounting of Users, in: Cybersecurity Providing in Information and Telecommunication Systems II, vol. 3550 (2023) 217–225.
[11] CloudTrail. URL: https://aws.amazon.com/cloudtrail/
[12] AWS Config. URL: https://aws.amazon.com/config/
[13] V. Khoma, et al., Comprehensive Approach for Developing an Enterprise Cloud Infrastructure, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3654 (2024) 201–215.
[14] Amazon Simple Notification Service. URL: https://aws.amazon.com/sns/
[15] R. Banakh, A. Piskozub, Y. Stefinko, External Elements of Honeypot for Wireless Network, in: 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET) (2016) 480–482. doi: 10.1109/TCSET.2016.7452093.
[16] AWS WAF. URL: https://aws.amazon.com/waf/
[17] DDoS Attack. URL: https://aws.amazon.com/shield/ddos-attack-protection/
[18] AWS Security Hub. URL: https://aws.amazon.com/security-hub/
[19] P. Anakhov, et al., Protecting Objects of Critical Information Infrastructure from Wartime Cyber Attacks by Decentralizing the Telecommunications Network, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3550 (2023) 240–245.
[20] Python Bibliotheca Boto3. URL: https://aws.amazon.com/sdk-for-python/
[21] SecureTransport End-User API v1.4 Documentation. URL: https://www.postman.com/api-evangelist/axway/documentation/x04b0lo/securetransport-end-user-api-v1-4
[22] O. Deineka, et al., Designing Data Classification and Secure Store Policy According to SOC 2 Type II, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3654 (2024) 398–409.
[23] Multi-Factor Authentication (MFA). URL: https://aws.amazon.com/iam/features/mfa/
[24] Amazon EBS Documentation. URL: https://docs.aws.amazon.com/ebs/
[25] Y. Martseniuk, et al., Automated Conformity Verification Concept for Cloud Security, in: Cybersecurity Providing in Information and Telecommunication Systems, vol. 3654 (2024) 25–37.
[26] Getting Started with AWS Key Management Service. URL: https://aws.amazon.com/kms/getting-started/
[27] Access Control List (ACL) Overview. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/acl-overview.html
[28] Find Security Group (SG) IDs, AMS. URL: https://docs.aws.amazon.com/managedservices/latest/userguide/find-SGs.html
[29] Amazon Virtual Private Cloud (VPC). URL: https://docs.aws.amazon.com/toolkit-for-visual-studio/latest/user-guide/vpc-tkv.html