NASA/PTES—024-0730 [@juhanTM]

[NOT FOR DISTRIBUTION,DRAFT,UNOFFICIAL]

PTES - TEST PLAN

ISS Communication System - NASA

I Introduction

- The ISS Communication System is responsible for managing the flow of data between the ISS and Earth.
- The purpose of this penetration test is to identify and address potential security vulnerabilities in the ISS Communication System.
- PTES version used: v1.2

II. Test Objectives

- Identify and exploit potential security vulnerabilities in the ISS Communication System
- Ensure the confidentiality, integrity, and availability of the system
- Test objectives include:
    - **Authentication and authorization**
    - Input validation and error handling
    - **Data encryption and transmission**
    - **Access control and permissions**
    - System configuration and hardening

III. Test Scope

- The SUT includes the ISS Communication System software, hardware, and network components
- Testing will cover the SCaN Testbed software and hardware components, as well as the testbed's ability to simulate space communication networks
- Out-of-scope areas include:
    - Physical security of the ISS and ground stations
    - Personnel security and training
    - Third-party systems and services

IV. Test Approach

- Testing will be performed using a combination of manual and automated techniques
- Manual testing will be performed using the provided manual regression test suite
- Testing will be conducted in a controlled environment, with the SCaN Testbed configured and set up according to the manufacturer's instructions
- Test data will be generated and stored in a secure and accessible location, and will be reviewed and approved by the testing team before use

V. Security Test Cases

- Network testing:
    - TCP/IP protocol testing
    - UDP protocol testing
    - HTTP protocol testing
    - FTP protocol testing
    - Space Packet Protocol (SPP) testing
    - Proximity-1 (Prox-1) Protocol testing
    - Space Link Extension (SLE) Protocol testing
    - Error handling and recovery testing
    - Performance testing
    - Security testing
    - Interoperability testing
    - Fault tolerance testing
    - Resource utilization testing
    - Compatibility testing
    - Error handling testing
    - Configuration testing
    - Upgrade testing
- **Application testing:**
    - **Authentication and authorization testing**
    - Input validation and error handling testing
    - Data encryption and transmission testing
    - Access control and permissions testing
    - System configuration and hardening testing

VI. Test Schedule

- Testing will be performed in a series of iterations, with each iteration focusing on a specific scenario or set of scenarios
- The testing schedule will be determined based on the availability of the SCaN Testbed and the testing team

VII. Test Risks and Assumptions

- The SCaN Testbed is assumed to be functioning correctly and available for testing
- The testing team has the necessary expertise and resources to perform the testing
- The testing schedule may be affected by changes to the SCaN Testbed or the testing team's availability

VIII. Test Metrics and Criteria

- Test coverage: percentage of test cases executed
- Test effectiveness: percentage of defects found and reported
- Test efficiency: ratio of test cases executed to test cases planned
- Test quality: rating of test case quality and relevance

IX. Test Environment Setup

- The SCaN Testbed hardware and software will be set up and configured according to the manufacturer's instructions
- The testing team will ensure that the test environment is stable and consistent throughout the testing process

X. Test Data Management

- Test data will be generated and stored in a secure and accessible location
- Test data will be reviewed and approved by the testing team before use
- Test data will be updated and revised as necessary during the testing process

XI. Test Case Execution

- Test cases will be executed in a controlled and systematic manner
- Test cases will be executed in a specific order, as determined by the testing team
- Test cases will be repeated as necessary to ensure consistent results

XII. Test Result Analysis

- Test results will be analyzed and reported in a timely and accurate manner
- Test results will be compared to expected outcomes to determine pass or fail status
- Test results will be used to identify defects and issues in the SCaN Testbed

XIII. Defect Reporting and Tracking

[Authentication and authorization testing]

Authentication and Authorization Testing Suite

I. Test Objectives

- Verify the authentication and authorization mechanisms of the ISS Communication System
- Ensure that only authorized users can access the system and its resources
- Identify and report any defects or issues related to authentication and authorization

II. Test Scope

- The SCaN Testbed software and hardware components
- The testbed's ability to authenticate and authorize users
- The testbed's ability to enforce access control and permissions

III. Test Environment

- The SCaN Testbed hardware and software setup
- The testbed's configuration and settings
- The test data and inputs used for testing

IV. Test Cases

1. **Test Case: Successful Authentication**
    - Objective: Verify that a valid user can authenticate to the SCaN Testbed
    - Input: A valid username and password
    - Expected Output: The user is successfully authenticated and granted access to the system
2. **Test Case: Invalid Authentication**
    - Objective: Verify that an invalid username or password is rejected by the SCaN Testbed
    - Input: An invalid username or password
    - Expected Output: The user is not authenticated and access to the system is denied
3. **Test Case: Password Complexity**
    - Objective: Verify that the SCaN Testbed enforces password complexity requirements
    - Input: A password that does not meet the complexity requirements
    - Expected Output: The password is rejected and the user is prompted to enter a new password that meets the requirements
4. **Test Case: Password Expiration**
    - Objective: Verify that the SCaN Testbed enforces password expiration policies
    - Input: A password that has expired

- Expected Output: The user is prompted to enter a new password

5. **Test Case: Account Lockout**
    - Objective: Verify that the SCaN Testbed enforces account lockout policies
    - Input: Multiple failed authentication attempts
    - Expected Output: The account is locked out and access to the system is denied
6. **Test Case: Role-Based Access Control**
    - Objective: Verify that the SCaN Testbed enforces role-based access control policies
    - Input: A user with a specific role
    - Expected Output: The user is granted access to the system and its resources based on their role
7. **Test Case: Permission-Based Access Control**
    - Objective: Verify that the SCaN Testbed enforces permission-based access control policies
    - Input: A user with specific permissions
    - Expected Output: The user is granted access to the system and its resources based on their permissions
8. **Test Case: Session Timeout**
    - Objective: Verify that the SCaN Testbed enforces session timeout policies
    - Input: A user with an active session
    - Expected Output: The session is terminated after a specified period of inactivity
9. **Test Case: Logout**
    - Objective: Verify that the SCaN Testbed allows users to log out of the system
    - Input: A user who is logged in to the system

- Expected Output: The user is logged out of the system and access to the system is denied

V. Test Deliverables

- Test plan document
- Test cases and scripts
- Test data and inputs
- Test results and reports
- Defect reports and issue tracking

VI. Test Schedule

- Testing will be performed in a series of iterations, with each iteration focusing on a specific scenario or set of scenarios
- The testing schedule will be determined based on the availability of the SCaN Testbed and the testing team

VII. Test Risks and Assumptions

- The SCaN Testbed is assumed to be functioning correctly and available for testing
- The testing team has the necessary expertise and resources to perform the testing
- The testing schedule may be affected by changes to the SCaN Testbed or the testing team's availability

VIII. Test Metrics and Criteria

- Test coverage: percentage of test cases executed
- Test effectiveness: percentage of defects found and reported
- Test efficiency: ratio of test cases executed to test cases planned
- Test quality: rating of test case quality and relevance

IX. Test Environment Setup

- The SCaN Testbed hardware and software will be set up and configured according to the manufacturer'

Authentication Tests

```
// iss-auth.test.js
const request = require('supertest');
const app = require('../iss-communication-system');

describe('ISS Communication System Authentication', () => {
  it('should return 401 for unauthorized requests to telemetry
endpoint', async () => {
    const response = await request(app).get('/telemetry');
    expect(response.status).toBe(401);
  });

  it('should return 200 for authorized requests to telemetry endpoint
with valid NASA credentials', async () => {
    const token = 'valid-nasa-token';
    const response = await
request(app).get('/telemetry').set("Authorization", `Bearer
${token}`);
    expect(response.status).toBe(200);
  });
  it('should return 401 for invalid NASA credentials', async () => {
    const token = 'invalid-nasa-token';
    const response = await
request(app).get('/telemetry').set("Authorization", `Bearer
${token}`);
    expect(response.status).toBe(401);
  });

  it('should return 401 for expired NASA credentials', async () => {
    const token = 'expired-nasa-token';
    const response = await
request(app).get('/telemetry').set("Authorization", `Bearer
${token}`);
    expect(response.status).toBe(401);
  });
});

describe('ISS Communication System Authentication for Command and
Control', () => {
  it('should return 401 for unauthorized requests to command
endpoint', async () => {
    const response = await request(app).post('/command');
    expect(response.status).toBe(401);
  });

  it('should return 200 for authorized requests to command endpoint
with valid NASA credentials', async () => {
    const token = 'valid-nasa-token';
    const response = await
request(app).post('/command').set("Authorization", `Bearer
${token}`);
    expect(response.status).toBe(200);
  });

  it('should return 401 for invalid NASA credentials', async () => {
    const token = 'invalid-nasa-token';
    const response = await
request(app).post('/command').set("Authorization", `Bearer
${token}`);
```

```
    expect(response.status).toBe(401);
  });

  it('should return 401 for expired NASA credentials', async () => {
    const token = 'expired-nasa-token';
    const response = await
request(app).post('/command').set("Authorization", `Bearer
${token}`);
    expect(response.status).toBe(401);
  });
});


Authorization Tests

// iss-authz.test.js
const request = require('supertest');
const app = require('../iss-communication-system');

describe('ISS Communication System Authorization', () => {
  it('should return 403 for unauthorized users attempting to access
restricted telemetry data', async () => {
    const token = 'valid-nasa-token';
    const response = await
request(app).get('/restricted-telemetry').set("Authorization",
`Bearer ${token}`);
    expect(response.status).toBe(403);
  });

  it('should return 200 for authorized users accessing restricted
telemetry data', async () => {
    const token = 'authorized-nasa-token';
    const response = await
request(app).get('/restricted-telemetry').set("Authorization",
`Bearer ${token}`);
    expect(response.status).toBe(200);
  });

  it('should return 403 for users with insufficient permissions
attempting to access restricted command functionality', async () => {
    const token = 'valid-nasa-token';
    const response = await
request(app).post('/restricted-command').set("Authorization", `Bearer
${token}`);
    expect(response.status).toBe(403);
  });

  it('should return 200 for authorized users accessing restricted
command functionality', async () => {
    const token = 'authorized-nasa-token';
    const response = await
request(app).post('/restricted-command').set("Authorization", `Bearer
${token}`);
    expect(response.status).toBe(200);
  });
});
Test Setup and Teardown

// setup.test.js
const app = require('../iss-communication-system');

beforeAll(async () => {
  // Set up test database and fixtures for ISS Communication System
});

afterAll(async () => {
  // Tear down test database and fixtures for ISS Communication
System
});

module.exports = app;
```

This test suite uses Jest as the test runner and Supertest for making HTTP requests to the ISS Communication System application. The tests cover various authentication and authorization scenarios specific to the ISS Communication System, including:

- Unauthorized requests to telemetry and command endpoints
- Authorized requests with valid NASA credentials
- Invalid NASA credentials
- Expired NASA credentials
- Unauthorized users attempting to access restricted telemetry data and command functionality
- Authorized users accessing restricted telemetry data and command functionality

```
Ddos testing

const http = require('http');
const async = require('async');

// Set the NASA ISS Communication System endpoint
const TARGET_URL = 'https://api.nasa.gov/iss/v1';
const TARGET_PORT = 443; // HTTPS port

// Set the API key (if required)
const API_KEY = 'YOUR_API_KEY_HERE';

// Function to send a single request
function sendRequest(endpoint, callback) {
  const options = {
    method: 'GET',
    hostname: TARGET_URL,
    port: TARGET_PORT,
    path: endpoint,
    headers: {
      'User-Agent': 'ISS Communication System Test Script',
      'Authorization': `Bearer ${API_KEY}` // Add API key if required
    }
  };

  const req = http.request(options, (res) => {
    res.on('data', (chunk) => {
      // Do nothing with the response data
    });
    res.on('end', () => {
      callback();
    });
  });

  req.on('error', (error) => {
    console.error(`Error sending request: ${error}`);
    callback();
  });

  req.end();
}

// Function to simulate the test plan
function simulateTestPlan(numRequests, endpoints, callback) {
  async.times(numRequests, (n, next) => {
    const endpoint = endpoints[Math.floor(Math.random() *
endpoints.length)];
    sendRequest(endpoint, next);
  }, callback);
}

// Define the endpoints to test
const endpoints = [
  '/location', // ISS location
  '/pass_times', // ISS pass times
  '/people_in_space' // People in space
];

// Run the test plan
const numRequests = process.argv[2] || 100;
simulateTestPlan(numRequests, endpoints, () => {
  console.log(`Sent ${numRequests} requests to ${endpoints.length}
endpoints`);
});
```

Data Encryption Suite

1. **Data Encryption**
   - Objective: Verify that the ISS Communication System encrypts data using a secure algorithm
   - Input: A sample data set
   - Expected Output: The data is encrypted using a secure algorithm (e.g. AES-256)
2. **Data Decryption**
   - Objective: Verify that the ISS Communication System decrypts data correctly
   - Input: An encrypted data set
   - Expected Output: The data is decrypted correctly and matches the original input
3. **Encryption Key Rotation**
   - Objective: Verify that the ISS Communication System rotates encryption keys securely
   - Input: A sample encryption key
   - Expected Output: The ISS Communication System rotates the encryption key securely
4. **Encryption Algorithm Validation**
   - Objective: Verify that the ISS Communication System uses a validated encryption algorithm
   - Input: A sample data set
   - Expected Output: The ISS Communication System uses a validated encryption algorithm
5. **Data Encryption at Rest**
   - Objective: Verify that the ISS Communication System encrypts data at rest
   - Input: A sample data set
   - Expected Output: The data is encrypted at rest
6. **Data Encryption in Transit**
   - Objective: Verify that the ISS Communication System encrypts data in transit
   - Input: A sample data set
   - Expected Output: The data is encrypted in transit

Transmission Security Suite

1. **Transmission Security**
    - Objective: Verify that the ISS Communication System transmits encrypted data securely
    - Input: An encrypted data set
    - Expected Output: The data is transmitted securely and without errors
2. **Secure Sockets Layer/Transport Layer Security (SSL/TLS)**
    - Objective: Verify that the ISS Communication System uses SSL/TLS for secure communication
    - Input: A sample data set
    - Expected Output: The ISS Communication System uses SSL/TLS for secure communication
3. **Certificate Validation**
    - Objective: Verify that the ISS Communication System validates certificates correctly
    - Input: A sample certificate
    - Expected Output: The ISS Communication System validates the certificate correctly
4. **Certificate Pinning**
    - Objective: Verify that the ISS Communication System uses certificate pinning to prevent man-in-the-middle attacks
    - Input: A sample certificate
    - Expected Output: The ISS Communication System uses certificate pinning to prevent man-in-the-middle attacks
5. **Transmission Error Handling**
    - Objective: Verify that the ISS Communication System handles transmission errors correctly
    - Input: A sample data set with intentional errors
    - Expected Output: The ISS Communication System handles transmission errors correctly

Man-in-the-Middle (MitM) Attack Suite

1. MitM Attack Detection
    - Objective: Verify that the ISS Communication System detects MitM attacks
    - Input: A simulated MitM attack
    - Expected Output: The ISS Communication System detects the MitM attack
2. **MitM Attack Prevention**

- Objective: Verify that the ISS Communication System prevents MitM attacks
- Input: A simulated MitM attack
- Expected Output: The ISS Communication System prevents the MitM attack

3. **SSL/TLS Striping Attack**
    - Objective: Verify that the ISS Communication System prevents SSL/TLS striping attacks
    - Input: A simulated SSL/TLS striping attack
    - Expected Output: The ISS Communication System prevents the SSL/TLS striping attack
4. **Certificate Impersonation Attack**
    - Objective: Verify that the ISS Communication System prevents certificate impersonation attacks
    - Input: A simulated certificate impersonation attack
    - Expected Output: The ISS Communication System prevents the certificate impersonation attack

Data Integrity Suite

1. **Data Integrity**
    - Objective: Verify that the ISS Communication System ensures data integrity during transmission
    - Input: A sample data set with intentional errors
    - Expected Output: The ISS Communication System detects and corrects errors during transmission
2. **Checksum Validation**
    - Objective: Verify that the ISS Communication System validates checksums correctly
    - Input: A sample data set with intentional errors
    - Expected Output

Automation test suite for Data Encryption and Transmission security

1. Encryption Algorithm Test

```
const crypto = require('crypto');
const app = require('../iss-communication-system');

describe('Encryption Algorithm Test', () => {
  it('should use AES-256 encryption algorithm', async () => {
    const data = 'Hello, World!';
    const encryptedData = await app.encrypt(data);
    expect(encryptedData).toBeInstanceOf(Buffer);
    expect(crypto.createCipheriv('aes-256-cbc', 'secret-key',
'initialization-vector').update(data).toString('hex')).toBe(encrypted
Data.toString('hex'));
  });

  it('should throw an error if the encryption algorithm is not
AES-256', async () => {
    const data = 'Hello, World!';
    const invalidAlgorithm = 'aes-128-cbc';
    expect(() => app.encrypt(data,
invalidAlgorithm)).toThrowError('Invalid encryption algorithm');
  });

  it('should encrypt data with a different key', async () => {
    const data = 'Hello, World!';
    const key1 = 'secret-key-1';
    const key2 = 'secret-key-2';
    const encryptedData1 = await app.encrypt(data, key1);
    const encryptedData2 = await app.encrypt(data, key2);
    expect(encryptedData1).not.toBe(encryptedData2);
  });

  it('should encrypt data with a different initialization vector',
async () => {
    const data = 'Hello, World!';
    const iv1 = 'initialization-vector-1';
    const iv2 = 'initialization-vector-2';
    const encryptedData1 = await app.encrypt(data, 'secret-key',
iv1);
    const encryptedData2 = await app.encrypt(data, 'secret-key',
iv2);
    expect(encryptedData1).not.toBe(encryptedData2);
  });
});
```

2. **Encryption Key Management Test**

```
const crypto = require('crypto');
const app = require('../iss-communication-system');

describe('Encryption Key Management Test', () => {
  it('should generate a new encryption key', async () => {
    const key = await app.generateEncryptionKey();
    expect(key).toBeInstanceOf(Buffer);
    expect(key.length).toBe(32); // 256-bit key
  });

  it('should store the encryption key securely', async () => {
    const key = await app.generateEncryptionKey();
    expect(await app.storeEncryptionKey(key)).toBe(true);
  });

  it('should retrieve the stored encryption key', async () => {
    const key = await app.generateEncryptionKey();
    await app.storeEncryptionKey(key);
    const retrievedKey = await app.getStoredEncryptionKey();
    expect(retrievedKey).toBe(key);
  });

  it('should throw an error if the encryption key is not stored',
async () => {
    expect(() =>
app.getStoredEncryptionKey()).toThrowError('Encryption key not
found');
  });
});
```

Transmission Security

1. SSL/TLS Version Test

```
const tls = require('tls');
const app = require('../iss-communication-system');

describe('SSL/TLS Version Test', () => {
  it('should use TLS 1.2 or higher', async () => {
    const socket = tls.connect({ port: 443, host: 'api.nasa.gov' });
    expect(socket.encrypted).toBe(true);
    expect(socket.getProtocol()).toBe('TLSv1.2' || 'TLSv1.3');
  });

  it('should not use SSLv3', async () => {
    const socket = tls.connect({ port: 443, host: 'api.nasa.gov' });
    expect(socket.encrypted).toBe(true);
    expect(socket.getProtocol()).not.toBe('SSLv3');
  });

  it('should not use TLS 1.0 or 1.1', async () => {
    const socket = tls.connect({ port: 443, host: 'api.nasa.gov' });
    expect(socket.encrypted).toBe(true);
    expect(socket.getProtocol()).not.toBe('TLSv1.0' || 'TLSv1.1');
  });
});
```

2. **Certificate Validation Test**

```
const tls = require('tls');
const app = require('../iss-communication-system');

describe('Certificate Validation Test', () => {
  it('should validate the SSL/TLS certificate', async () => {
    const socket = tls.connect({ port: 443, host: 'api.nasa.gov' });
    expect(socket.encrypted).toBe(true);

expect(socket.getPeerCertificate().subject.CN).toBe('api.nasa.gov');
  });

  it('should throw an error if the certificate is invalid', async ()
=> {
    const invalidCert = { ...socket.getPeerCertificate(), subject: {
CN: 'invalid.example.com' } };
    expect(() => tls.connect({ port: 443, host: '
```

Certificate Validation Test

```
const tls = require('tls');
const app = require('../iss-communication-system');

describe('Certificate Validation Test', () => {
  it('should validate the SSL/TLS certificate', async () => {
    const socket = tls.connect({ port: 443, host: 'api.nasa.gov' });
    expect(socket.encrypted).toBe(true);

expect(socket.getPeerCertificate().subject.CN).toBe('api.nasa.gov');
  });

  it('should throw an error if the certificate is invalid', async ()
=> {
    const invalidCert = { ...socket.getPeerCertificate(), subject: {
CN: 'invalid.example.com' } };
    expect(() => tls.connect({ port: 443, host: 'invalid.example.com'
})).toThrowError('Invalid certificate');
```

```
  });

  it('should throw an error if the certificate is expired', async ()
=> {
    const expiredCert = { ...socket.getPeerCertificate(), notAfter:
new Date('2020-01-01T00:00:00.000Z') };
    expect(() => tls.connect({ port: 443, host: 'api.nasa.gov'
})).toThrowError('Certificate has expired');
  });

  it('should throw an error if the certificate is not trusted', async
() => {
    const untrustedCert = { ...socket.getPeerCertificate(), issuer: {
CN: 'Untrusted CA' } };
    expect(() => tls.connect({ port: 443, host: 'api.nasa.gov'
})).toThrowError('Certificate is not trusted');
  });
});
```

3. Data Integrity Test

```
const crypto = require('crypto');
const app = require('../iss-communication-system');

describe('Data Integrity Test', () => {
  it('should ensure data integrity using digital signatures', async
() => {
    const data = 'Hello, World!';
    const signature = await app.signData(data);
    expect(signature).toBeInstanceOf(Buffer);

expect(crypto.createVerify('RSA-SHA256').update(data).verify(signatur
e)).toBe(true);
  });
  it('should throw an error if the data is tampered with', async ()
=> {
    const data = 'Hello, World!';
    const signature = await app.signData(data);
    const tamperedData = 'Hello, Universe!';
    expect(() =>
crypto.createVerify('RSA-SHA256').update(tamperedData).verify(signatu
re)).toThrowError('Data has been tampered with');
  });

  it('should throw an error if the signature is invalid', async () =>
{
    const data = 'Hello, World!';
    const invalidSignature = Buffer.from('invalid signature');
    expect(() =>
crypto.createVerify('RSA-SHA256').update(data).verify(invalidSignatur
e)).toThrowError('Invalid signature');
  });
});
```

4. **Man-in-the-Middle (MitM) Attack Test**

```
const tls = require('tls');
const app = require('../iss-communication-system');

describe('MitM Attack Test', () => {
  it('should detect a MitM attack', async () => {
    const socket = tls.connect({ port: 443, host: 'api.nasa.gov' });
    expect(socket.encrypted).toBe(true);
    // Simulate a MitM attack by modifying the certificate
    const maliciousCert = { ...socket.getPeerCertificate(), subject:
{ CN: 'malicious.example.com' } };
    expect(socket.getPeerCertificate()).not.toBe(maliciousCert);
  });

  it('should throw an error if the MitM attack is detected', async ()
=> {
    const socket = tls.connect({ port: 443, host: 'api.nasa.gov' });
    expect(socket.encrypted).toBe(true);
    // Simulate a MitM attack by modifying the certificate
    const maliciousCert = { ...socket.getPeerCertificate(), subject:
{ CN: 'malicious.example.com' } };
    expect(() => socket.getPeerCertificate()).toThrowError('MitM
attack detected');
  });
});
```

Access control and permissions

Test Case 1: Unauthorized Access

- Scenario 1.1: Test that an unauthorized user cannot access the ISS Communication System with a valid username and invalid password
- Scenario 1.2: Test that an unauthorized user cannot access the ISS Communication System with an invalid username and valid password
- Scenario 1.3: Test that an unauthorized user cannot access the ISS Communication System with an invalid username and invalid password
- Scenario 1.4: Test that an unauthorized user cannot access the ISS Communication System with a blank username and password
- Scenario 1.5: Test that an unauthorized user cannot access the ISS Communication System with a username and password that is not in the correct format

Test Case 2: Authorized Access

- Scenario 2.1: Test that an authorized user can access the ISS Communication System with a valid username and password
- Scenario 2.2: Test that an authorized user can access the ISS Communication System with a valid username and password that has been changed recently

- Scenario 2.3: Test that an authorized user can access the ISS Communication System with a valid username and password that has been reset
- Scenario 2.4: Test that an authorized user can access the ISS Communication System with a valid username and password that has been locked out due to multiple incorrect login attempts
- Scenario 2.5: Test that an authorized user can access the ISS Communication System with a valid username and password that has been unlocked after a lockout period

Test Case 3: Role-Based Access Control

- Scenario 3.1: Test that a user with a "read-only" role can only view data and cannot modify it
- Scenario 3.2: Test that a user with a "read-write" role can view and modify data
- Scenario 3.3: Test that a user with an "admin" role can view, modify, and delete data
- Scenario 3.4: Test that a user with a "guest" role can only access public data and not sensitive data
- Scenario 3.5: Test that a user with a "custom" role can access only the specific resources and functionality assigned to that role

Test Case 4: Permission-Based Access Control

- Scenario 4.1: Test that a user with "view telemetry" permission can view telemetry data
- Scenario 4.2: Test that a user with "edit telemetry" permission can edit telemetry data
- Scenario 4.3: Test that a user with "delete telemetry" permission can delete telemetry data
- Scenario 4.4: Test that a user with "view command and control" permission can view command and control data
- Scenario 4.5: Test that a user with "edit command and control" permission can edit command and control data

Test Case 5: Access Control for Sensitive Data

- Scenario 5.1: Test that a user without "view sensitive data" permission cannot access sensitive data
- Scenario 5.2: Test that a user with "view sensitive data" permission can access sensitive data
- Scenario 5.3: Test that a user without "edit sensitive data" permission cannot edit sensitive data
- Scenario 5.4: Test that a user with "edit sensitive data" permission can edit sensitive data
- Scenario 5.5: Test that a user without "delete sensitive data" permission cannot delete sensitive data

Test Case 6: Access Control for System Configuration

- Scenario 6.1: Test that a user without "configure system" permission cannot modify system configuration
- Scenario 6.2: Test that a user with "configure system" permission can modify system configuration
- Scenario 6.3: Test that a user without "view system configuration" permission cannot view system configuration
- Scenario 6.4: Test that a user with "view system configuration" permission can view system configuration
- Scenario 6.5: Test that a user without "edit system configuration" permission cannot edit system configuration

Test Case 7: Access Control for User Management

- Scenario 7.1: Test that a user without "create user" permission cannot create a new user
- Scenario 7.2: Test that a user with "create user" permission can create a new user
- Scenario 7.3: Test that a user without "edit user" permission cannot edit an existing user
- Scenario 7.4: Test that a user with "edit user" permission can edit an existing user
- Scenario 7.5: Test that a user without "delete user" permission cannot delete a user

Test Case 8: Access Control for Audit Logs

- Scenario 8.3: Test that a user without "edit audit logs" permission cannot edit audit logs
- Scenario 8.4: Test that a user with "edit audit logs" permission can edit audit logs
- Scenario 8.5: Test that a user without "delete audit logs" permission cannot delete audit logs

Test Case 9: Access Control for System Monitoring

- Scenario 9.1: Test that a user without "view system monitoring" permission cannot view system monitoring data
- Scenario 9.2: Test that a user with "view system monitoring" permission can view system monitoring data
- Scenario 9.3: Test that a user without "edit system monitoring" permission cannot edit system monitoring settings
- Scenario 9.4: Test that a user with "edit system monitoring" permission can edit system monitoring settings
- Scenario 9.5: Test that a user without "configure system monitoring" permission cannot configure system monitoring

Test Case 10: Access Control for Emergency Procedures

- Scenario 10.1: Test that a user without "execute emergency shutdown" permission cannot execute an emergency shutdown
- Scenario 10.2: Test that a user with "execute emergency shutdown" permission can execute an emergency shutdown
- Scenario 10.3: Test that a user without "execute emergency restart" permission cannot execute an emergency restart
- Scenario 10.4: Test that a user with "execute emergency restart" permission can execute an emergency restart
- Scenario 10.5: Test that a user without "access emergency procedures" permission cannot access emergency procedures

```
/ iss-auth.test.js
const request = require('supertest');
const app = require('../iss-communication-system');

describe('ISS Communication System Authentication', () => {
  it('should return 401 for unauthorized requests to telemetry
endpoint', async () => {
    const response = await request(app).get('/telemetry');
    expect(response.status).toBe(401);
  });

  it('should return 200 for authorized requests to telemetry endpoint
with valid NASA credentials', async () => {
    const token = 'valid-nasa-token';
    const response = await
request(app).get('/telemetry').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(200);
  });

  it('should return 401 for invalid NASA credentials', async () => {
    const token = 'invalid-nasa-token';
    const response = await
request(app).get('/telemetry').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(401);
  });

  it('should return 401 for expired NASA credentials', async () => {
    const token = 'expired-nasa-token';
    const response = await
request(app).get('/telemetry').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(401);
  });
});

describe('ISS Communication System Authorization', () => {
  it('should return 403 for unauthorized users attempting to access
restricted telemetry data', async () => {
    const token = 'valid-nasa-token';
    const response = await
request(app).get('/restricted-telemetry').set('Authorization',
`Bearer ${token}`);
    expect(response.status).toBe(403);
  });

  it('should return 200 for authorized users accessing restricted
telemetry data', async () => {
    const token = 'authorized-nasa-token';
    const response = await
request(app).get('/restricted-telemetry').set('Authorization',
`Bearer ${token}`);
    expect(response.status).toBe(200);
  });

  it('should return 403 for users with insufficient permissions
attempting to access restricted command functionality', async () => {
    const token = 'valid-nasa-token';
    const response = await
request(app).post('/restricted-command').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(403);
  });

  it('should return 200 for authorized users accessing restricted
command functionality', async () => {
    const token = 'authorized-nasa-token';
    const response = await
request(app).post('/restricted-command').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(200);
  });
});

// iss-authz.test.js
const request = require('supertest');
const app = require('../iss-communication-system');
```

```
describe('ISS Communication System Role-Based Access Control', () =>
{
  it('should return 403 for users with read-only role attempting to
edit telemetry data', async () => {
    const token = 'read-only-token';
    const response = await
request(app).post('/telemetry').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(403);
  });

  it('should return 200 for users with read-write role editing
telemetry data', async () => {
    const token = 'read-write-token';
    const response = await
request(app).post('/telemetry').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(200);
  });

  it('should return 403 for users with admin role attempting to
delete telemetry data', async () => {
    const token = 'admin-token';
    const response = await
request(app).delete('/telemetry').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(403);
  });

  it('should return 200 for users with admin role deleting telemetry
data', async () => {
    const token = 'admin-token';
    const response = await
request(app).delete('/telemetry').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(200);
  });
});
```

```
// iss-permissions.test.js
const request = require('supertest');
const app = require('../iss-communication-system');

describe('ISS Communication System Permission-Based Access Control',
() => {
  it('should return 403 for users without view telemetry permission
attempting to access telemetry data', async () => {
    const token = 'no-view-telemetry-token';
    const response = await
request(app).get('/telemetry').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(403);
  });

  it('should return 200 for users with view telemetry permission
accessing telemetry data', async () => {
    const token = 'view-telemetry-token';
    const response = await
request(app).get('/telemetry').set('Authorization', `Bearer
${token}`);
    expect(response.status).toBe(200);
  });

  it('
```

Old Vulnerabilities

1. Test case for checking if the system is vulnerable to weak passwords:

```
const nmap = require('nmap');

const target = '192.168.1.1'; // replace with the target IP
const port = 22; // SSH port
const script = 'auth/cracklib-checks'; // NSE script for checking
password strength

const nmapScan = new nmap.NmapScan(target, port, script);

nmapScan.on('complete', (data) => {
  if (data.scan.scripts[script].vulns.length > 0) {
    console.log(`Target ${target} is vulnerable to weak passwords`);
  } else {
    console.log(`Target ${target} is not vulnerable to weak
passwords`);
  }
});

nmapScan.run();
```

2. **Test case for checking if the system has any vulnerable CGI or dynamic web server files:**

```
const nmap = require('nmap');

const target = '192.168.1.1'; // replace with the target IP
const port = 80; // HTTP port
const script = 'http-vuln-cgi'; // NSE script for checking CGI
vulnerabilities

const nmapScan = new nmap.NmapScan(target, port, script);

nmapScan.on('complete', (data) => {
  if (data.scan.scripts[script].vulns.length > 0) {
    console.log(`Target ${target} has vulnerable CGI or dynamic web
server files`);
  } else {
    console.log(`Target ${target} does not have vulnerable CGI or
dynamic web server files`);
```

```
  }
});
```

nmapScan.run();

3. **Test case for checking if the system has any poorly configured or implemented services:**

```
const nmap = require('nmap');

const target = '192.168.1.1'; // replace with the target IP
const port = 80; // HTTP port
const script = 'http-title'; // NSE script for checking HTTP title

const nmapScan = new nmap.NmapScan(target, port, script);

nmapScan.on('complete', (data) => {
  if
(data.scan.scripts[script].output.match(/<title>[^<]+<\/title>/)) {
    console.log(`Target ${target} has properly configured and
implemented services`);
  } else {
    console.log(`Target ${target} has poorly configured or
implemented services`);
  }
});

nmapScan.run();
```

4. **Test case for checking if the system has any outdated or unpatched operating systems or network applications:**

```
const nmap = require('nmap');
const target = '192.168.1.1'; // replace with the target IP
const script = 'os-detection'; // NSE script for OS detection

const nmapScan = new nmap.NmapScan(target, script);

nmapScan.on('complete', (data) => {
  const os = data.scan.os[0];
  if (os.accuracy === 100 && os.cpe.length > 0) {
    const cpe = os.cpe[0];
    if (cpe.vendor === 'Microsoft' && cpe.product === 'Windows 7' &&
cpe.version.startsWith('6.1')) {
      console.log(`Target ${target} has outdated or unpatched
operating system`);
    } else if (cpe.vendor === 'Apache' && cpe.product === 'HTTP
Server' && cpe.version.startsWith('2.2')) {
      console.log(`Target ${target} has outdated or unpatched network
application`);
    } else {
      console.log(`Target ${target} has up-to-date operating system
and network applications`);
    }
  } else {
    console.log(`Could not determine the OS of target ${target}`);
  }
});

nmapScan.run();
const nmap = require('nmap');

const target = '192.168.1.1'; // replace with the target IP address

const nmapScan = new nmap.NmapScan(target, '1-1000');

nmapScan.on('complete', (data) => {
  console.log('Open ports and services on target system:');
  data.scan.tcp.forEach((port) => {
    console.log(`Port ${port.portid} - ${port.service.name}`);
  });
});

nmapScan.run();
```