




APRIL 28, 2024

INTRUSION DETECTION SYSTEM

SYEDA MAHAM JAFRI 22796
ASHNAH KHALID KHAN 22889
MAAZ SIDDIQUI 22997



Contents

System Design	2
Network Security:	2
Security Requirements for Network:	2
Potential Threats:.....	3
Scopes and Objectives:	5
Components of IDPS:.....	6
Implementation of IDPS:.....	7
Testing and Evaluation	9
Functional Testing Plan:	9
1. Login Functionality:.....	9
2. Blocking Functionality:	10
3. System Integrity Check:	10
4. Network Traffic Monitoring:	10
5. Password Recovery:	10
6. OTP Generation and Validation:	10
7. Email Blacklisting/Whitelisting:	10
8. IP Address Blacklisting/Whitelisting:.....	10
Performance Testing Plan:	10
1. Login Performance:	10
2. Blocking Mechanism Performance:	11
3. Logging Performance:	11
4. Network Traffic Monitoring Performance:	11
5. Scalability Testing:	11
Security Testing Plan:	11
1. Input Validation:.....	11
2. Authentication and Authorization:	11
3. Data Protection:.....	12
4. Access Control:	12
5. Logging and Auditing:	12
6. Network Security:.....	12
7. Incident Response:	12
Conducting Rigorous Testing	13
Evaluating System's Performance:	14
Test Case Results:	14
Observations.....	14
Threats and Vulnerabilities overlooked:	15

System Design

Network Security:

Network security protects and secures the underlying networking infrastructure from unauthorized access, misuse, damage, or theft. It helps ensure that only authorized people can access it. It involves creating a secure infrastructure for devices, applications, users, and applications to work securely.

Security Requirements for Network:

- **Firewall**
Firewalls act as gatekeepers for your networks; they use a set of predetermined security rules to monitor the incoming and outgoing traffic on the network, allowing only safe data to pass through. Any information from an external network attempting to enter your system must pass the defined criteria. If it doesn't, then it just gets blocked.
- **Network Segmentation**
Network segmentation is used to divide a network into multiple segments or subnets, each acting as its own small network. This helps in defining the boundaries between network segments where assets within the group have a common function, risk or role within an organization.
- **Access Control**
Access controls are rules and procedures that help organizations ensure that only authorized people can physically or digitally access the resources they're allowed to. This includes permissions to view, edit, and delete sensitive information.
- **Remote Access VPN**
Allows individuals such as remote employees to securely connect to a company's network. It uses encryption and multi-factor authentication to ensure the security and privacy of data transmitted.
- **Zero Trust Network Access (ZTNA)**
Unlike traditional security, which gives users full network access, ZTNA restricts access to only what users need for their specific roles. This is also called a software-defined perimeter.
- **Email Security**
Involves protecting email communications from external threats using built-in security features from email providers, enhanced by additional security measures to block cyber threats.
- **Data Loss Prevention (DLP)**
A strategy combining technology and practices to prevent sensitive information from leaving the organization. This is particularly important for complying with regulations like HIPAA or PCI DSS.
- **Intrusion Prevention Systems (IPS)**
Intrusion Prevention Systems (IPS) are designed to identify and thwart network security threats such as brute force attacks, Denial of Service (DoS) attacks, and the exploitation of known software vulnerabilities. A vulnerability represents a flaw in a software system, and an exploit occurs when an attacker uses this flaw to take over the system. Often, when a software vulnerability becomes public, there is a critical period during which attackers can take advantage of the flaw before a security fix is implemented. During this time, an IPS is crucial as it can promptly intercept and block these attacks before they cause harm.

- **Sandboxing**
Sandboxing is a security technique that executes code or opens files within a controlled and isolated environment on a host computer, simulating real user operating systems. This process actively monitors the behaviour of files or code as they are accessed to identify and halt potential malicious activities, ensuring threats like malware in common file types (PDFs, Word documents, Excel spreadsheets, and PowerPoint presentations) are caught and neutralized before they can compromise an end user.
- **Hyperscale Network Security**
Hyperscale refers to the capacity of a system to scale efficiently as demand increases. This approach involves quick deployment and the ability to scale resources up or down based on the security needs of the network. By seamlessly integrating network and computing resources within a software-defined infrastructure, it ensures optimal use of all available hardware resources within a clustered environment.
- **Cloud Network Security**
As applications and workloads transition away from traditional on-premises data centres, the need for innovative and flexible security solutions grows. Software-defined Networking (SDN) and Software-defined Wide Area Networks (SD-WAN) are essential in securing these modern infrastructures, facilitating network security across private, public, hybrid, and cloud-hosted environments, including Firewall-as-a-Service (FWaaS) deployments. This shift requires dynamic security measures that can adapt to the changing landscape of cloud migrations.
- **Employee Training and Awareness:**
Regular training sessions are crucial in educating employees on the best practices for maintaining robust network security. These sessions should emphasize the importance of avoiding phishing scams and practicing strong password hygiene. Educated employees are a critical line of defense against cyber threats, making ongoing training an essential component of any security strategy.
- **Incident Response Plan:**
Creating a detailed incident response plan is vital for any organization to effectively address and mitigate security breaches when they occur. This plan should clearly outline the necessary steps to contain a security incident, investigate its root cause, and restore normal operations as swiftly as possible. Having a structured approach ensures a coordinated response during critical situations, minimizing potential damage.
- **Maintaining data backups:**
Maintaining regular backups of data is a fundamental practice for any organization, providing a safety net against data loss resulting from cyber-attacks. Even the best-protected networks can be vulnerable, and having up-to-date backups ensures that a business can recover quickly, minimizing the impact on operations and reducing downtime after an attack.

Potential Threats:

- **Physical sabotage and surveillance**
Network security must also protect against physical threats such as "shoulder-surfing," where attackers observe employees in public places to steal sensitive information directly from their screens.
- **Virus:**
A virus is a type of malicious software that remains dormant until downloaded.

It reproduces itself by altering other computer programs with its code, which can lead to the spread of the infection across multiple computers and cause data corruption or loss.

- **Worms:**
Worms burden network resources, consuming bandwidth and reducing system performance. Unlike viruses, which require a host program to spread, worms operate independently and can autonomously propagate through a network.
- **Trojan:**
A trojan is a deceptive type of malware disguised as legitimate software. It serves as a backdoor for attackers to access and control a computer system, allowing them to delete files, activate hidden malware, and steal valuable data from the affected system.
- **Spyware:**
Spyware is a type of malware that covertly collects information about individuals or organizations without their knowledge. This data is typically sent to third parties without the consent of the affected users, often leading to privacy violations.
- **Adware:**
Adware interferes with user experience by redirecting search requests to specific advertising websites. It collects data on user activity and preferences to display targeted advertisements, often without the user's knowledge or consent.
- **Ransomware:**
Ransomware is a form of malicious software that operates like a trojan, encrypting data on a victim's computer to make it inaccessible. The attackers then demand payment, typically in cryptocurrency, in exchange for the decryption key needed to regain access. This type of malware aims to extort money by holding the victim's data hostage.
- **Zero-Day Exploits:**
Zero-day exploits take advantage of vulnerabilities in software that are unknown to the software vendor and the public. These exploits are particularly hazardous because there are no existing patches to fix the vulnerabilities, allowing attackers to exploit them undetected, often leading to significant security breaches.
- **Phishing Attacks:**
Phishing attacks deceive users into disclosing sensitive information, like login credentials or financial details, through misleading emails or websites. These attacks manipulate trust to acquire unauthorized access to personal data.
 - **Spear phishing**
In spear phishing, attackers gather detailed information about their target through surveillance or malware. This data is then used to craft highly personalized emails that appear legitimate and relevant to the individual, increasing the likelihood of the target disclosing sensitive information.
 - **Whaling**
Whaling attacks specifically target senior executives or other high-ranking officials within an organization. These individuals often have access to critical organizational data, making them valuable targets. The tactics involve sending spear phishing emails that are finely tuned to appear as if they are from a trustworthy source, often related to high-stakes business matters.

- Vishing

Vishing combines elements of phishing with voice communication. In these attacks, perpetrators often use compromised Voice-over-IP (VoIP) systems to impersonate legitimate entities or authorities. They engage in real-time conversations with victims to gain their trust and manipulate them into disclosing confidential information.

- Smishing

Similar to phishing, smishing involves sending deceptive text messages to lure recipients into revealing personal data. These messages often mimic communications from credible sources and use urgent or compelling language to prompt immediate action, such as clicking on malicious links.

- Spam

Spam refers to the practice of sending unsolicited bulk messages, primarily via email. While often just a nuisance, spam can also include phishing attempts and other malicious content. Despite its low individual success rate, spam remains effective on a large scale due to the volume of emails sent.

- Insider Threats:

Insider threats arise when employees or contractors with access to company networks misuse their privileges to harm the organization. This can involve stealing sensitive information or intentionally damaging company systems, posing significant risks to network security.

- Unauthorized access:

Unauthorized access occurs when someone without permission accesses a network. This intrusion can lead to the exposure of confidential information, where the unauthorized individual might view, steal, or even leak data, potentially leading to compromised internal systems.

- Logic bombs

Logic bombs are malicious code programmed to activate under specific conditions, such as reaching a certain date or after a particular action is taken (e.g., sending a set number of emails). Once activated, these logic bombs can execute harmful actions, such as deploying viruses or worms, which can disrupt operations and cause unpredictable damage to the network.

Scopes and Objectives:

1. **Network size:** The IDPS should be capable of handling the volume of network traffic generated by the organization's users, devices, and applications. The solution should be scalable to accommodate future growth in network size and traffic.
2. **Network topology:** The IDPS should be compatible with the organization's network topology, including wired and wireless networks, virtual private networks (VPNs), and cloud-based networks. The solution should be able to monitor traffic across all segments of the network and provide comprehensive coverage.
3. **Traffic patterns:** The IDPS should be able to analyze traffic patterns and identify anomalies that may indicate potential security threats. The solution should be able to distinguish between legitimate and malicious traffic and provide real-time alerts and reports.

4. Threat detection: The IDPS should be able to detect a wide range of security threats, including intrusion attempts, malware, ransomware, phishing, and denial-of-service (DoS) attacks. The solution should be able to detect both known and unknown threats using advanced techniques such as behavioral analysis, machine learning, and artificial intelligence.
5. Threat prevention: The IDPS should be able to prevent security threats from compromising the network and its resources. The solution should be able to block malicious traffic, quarantine infected devices, and terminate suspicious connections.
6. Incident response: The IDPS should be able to respond to security incidents in real-time, providing immediate notification to security personnel and initiating appropriate response actions. The solution should be able to integrate with other security tools and systems, such as security information and event management (SIEM) systems, to provide a coordinated response.
7. Compliance: The IDPS should be able to meet regulatory and industry compliance requirements, such as PCI-DSS, HIPAA, and GDPR. The solution should be able to provide detailed reports and logs to demonstrate compliance with these requirements.

Components of IDPS:

The Intrusion Detection System can be divided into three main components: Detection Component, Analysis Component, and Response Component.

1. Detection Component:
This component is responsible for detecting potential security threats and anomalies in the system. It includes the following sub-components:
 - Login Authentication and Authorization: Validates user credentials and OTP for authentication, tracks login attempts, and locks user accounts after exceeding a certain threshold.
 - Password Recovery: Initiates the password recovery process by sending a reset token to the user's email, validates the reset token, and updates the user's password.
 - Security Analytics and Anomaly Detection: Calculates similarity scores between entered and actual credentials using Levenshtein distance, detects suspicious login patterns, and triggers honeypot mechanisms for further analysis.
 - Network Traffic Monitoring: Monitors network traffic levels and issues alerts for unusually high traffic volumes.
2. Analysis Component:
This component is responsible for analyzing the data collected by the Detection Component to identify potential security threats and anomalies. It includes the following sub-components:
 - Security Analytics and Anomaly Detection: Analyzes the data collected by the Detection Component to identify potential security threats and anomalies.
 - Logger Component: Logs all system events, including login attempts, account locks, and IP address blocking, using the Java Logger class for logging to a file (IDSLog.txt).
3. Response Component:
This component is responsible for responding to potential security threats and anomalies identified by the Analysis Component. It includes the following sub-components:
 - Login Authentication and Authorization: Locks user accounts and blocks IP addresses after exceeding a certain threshold.
 - System Integrity Check: Verifies the integrity of the logging system to detect tampering attempts.

- Password Recovery: Resets user passwords in case of a security breach.

Implementation of IDPS:

```
Info Sec- IDS.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
Table of contents
+ Section
+ Code + Text
RAM
Disk
Colab AI

import re
import logging
from logging.handlers import RotatingFileHandler

class IntrusionDetectionSystem:
    # Set constants for maximum login attempts and honeypot trigger attempts
    MAX_LOGIN_ATTEMPTS = 3
    HONEYPOT_TRIGGER_ATTEMPTS = 3
    SIMILARITY_THRESHOLD = 0.7
    max_traffic_threshold = 1000 # Set a maximum traffic threshold for network monitoring

    def __init__(self):
        # Initialize dictionaries to keep track of user data and authentication attempts
        self.login_attempts = {}
        self.user_passwords = {"user1": "password123", "user2": "securePassword456"}
        self.honeypot_attempts = {}
        self.blocked_ips = {}
        self.whitelisted_ips = {"192.168.1.1"}
        self.blacklisted_ips = {"10.0.0.1"}
        self.otp_secrets = self.generate_otp_secrets()
        self.blacklisted_emails = set()
        self.is_system_integrity_intact = True
        self.setup_logging()

    def setup_logging(self):
        # Setup a logger to record events and alerts
        self.logger = logging.getLogger('IDSLogger')
        self.logger.setLevel(logging.DEBUG)
        handler = RotatingFileHandler('IDSLog.txt', maxBytes=2000, backupCount=5)
        formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
        handler.setFormatter(formatter)
        self.logger.addHandler(handler)
```

```
Info Sec- IDS.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
Table of contents
+ Section
+ Code + Text
RAM
Disk
Colab AI

def setup_logging(self):
    # Setup a logger to record events and alerts
    self.logger = logging.getLogger('IDSLogger')
    self.logger.setLevel(logging.DEBUG)
    handler = RotatingFileHandler('IDSLog.txt', maxBytes=2000, backupCount=5)
    formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
    handler.setFormatter(formatter)
    self.logger.addHandler(handler)

def login(self, username, password, otp, source_ip, destination_ip, email):
    # Handle login attempts, verify IPs, and process authentication
    if not self.validate_ip_addresses(source_ip, destination_ip):
        return
    if self.is_blacklisted(source_ip) or self.is_blacklisted(destination_ip):
        self.logger.warning("Suspicious source/destination IP address detected. Packet entry locked!")
        return

    if not self.attempt_login(username, password, otp, source_ip, email):
        return

    self.manage_login_attempts(username, password, otp, source_ip)

def validate_ip_addresses(self, source_ip, destination_ip):
    # Validate both source and destination IP addresses
    if not self.is_valid_ip_address(source_ip) or not self.is_valid_ip_address(destination_ip):
        self.logger.warning("Invalid source or destination IP address.")
        return False
    return True

def attempt_login(self, username, password, otp, source_ip, email):
    # Log and check each login attempt
```



```
Info Sec- IDS.ipynb
File Edit View Insert Runtime Tools Help All changes saved
RAM Disk Colab AI

Table of contents
+ Section

def attempt_login(self, username, password, otp, source_ip, email):
    # Log and check each login attempt
    attempts = self.login_attempts.get(username, 0) + 1
    self.login_attempts[username] = attempts
    self.logger.info(f"Login attempt for user: {username} from IP: {source_ip}")

    if self.is_email_blacklisted(email):
        self.logger.warning(f"Blocked email address: {email}")
        return False
    return True

def manage_login_attempts(self, username, password, otp, source_ip):
    # Manage the number of login attempts and trigger security measures if needed
    attempts = self.login_attempts[username]
    if attempts > self.MAX_LOGIN_ATTEMPTS:
        if not self.is_similar_credentials(username, password) or not self.verify_otp(username, otp):
            self.handle_honeypot(username, password)
        if attempts > self.MAX_LOGIN_ATTEMPTS and source_ip not in self.blocked_ips:
            self.block_ip(source_ip)

def handle_honeypot(self, username, password):
    # Manage honeypot detection and actions
    honeypot_attempt = self.honeypot_attempts.get(username, 0) + 1
    self.honeypot_attempts[username] = honeypot_attempt
    if honeypot_attempt % self.HONEYPOT_TRIGGER_ATTEMPTS == 0:
        self.lock_account(username)

def is_valid_ip_address(self, ip_address):
    # Validate the format of IPv4 and IPv6 addresses
    ipv4_pattern = re.compile(r"^(?:25[0-5]|2[0-4][0-9]|1[01]?[0-9]?[0-9])\.(?:25[0-5]|2[0-4][0-9]|1[01]?[0-9]?[0-9])\.(?:25[0-5]|2[0-4][0-9]|1[01]?[0-9]?[0-9])\.(?:25[0-5]|2[0-4][0-9]|1[01]?[0-9]?[0-9])$")
    ipv6_pattern = re.compile(r"^(?:[0-9a-fA-F]{1,4}:){2,7}(?:[0-9a-fA-F]{1,4}|:)$|^(?:[0-9a-fA-F]{1,4}:){6}(?:[0-9a-fA-F]{1,4}|:)$")
```

```
Info Sec- IDS.ipynb
File Edit View Insert Runtime Tools Help All changes saved
RAM Disk Colab AI

Table of contents
+ Section

def is_valid_ip_address(self, ip_address):
    # Validate the format of IPv4 and IPv6 addresses
    ipv4_pattern = re.compile(r"^(?:25[0-5]|2[0-4][0-9]|1[01]?[0-9]?[0-9])\.(?:25[0-5]|2[0-4][0-9]|1[01]?[0-9]?[0-9])\.(?:25[0-5]|2[0-4][0-9]|1[01]?[0-9]?[0-9])\.(?:25[0-5]|2[0-4][0-9]|1[01]?[0-9]?[0-9])$")
    ipv6_pattern = re.compile(r"^(?:[0-9a-fA-F]{1,4}:){2,7}(?:[0-9a-fA-F]{1,4}|:)$|^(?:[0-9a-fA-F]{1,4}:){6}(?:[0-9a-fA-F]{1,4}|:)$")
    return ipv4_pattern.match(ip_address) or ipv6_pattern.match(ip_address)

def is_whitelisted(self, ip_address):
    # Check if an IP address is on the whitelist
    return ip_address in self.whitelisted_ips

def is_blacklisted(self, ip_address):
    # Check if an IP address is on the blacklist
    return ip_address in self.blacklisted_ips

def is_email_blacklisted(self, email):
    # Check if an email is on the blacklist
    return email in self.blacklisted_emails

def is_similar_credentials(self, username, password):
    # Compare entered credentials with stored values to detect similarity
    actual_password = self.user_passwords.get(username)
    return self.calculate_similarity(password, actual_password) >= self.SIMILARITY_THRESHOLD

def calculate_similarity(self, entered_str, actual_str):
    # Calculate similarity score between two strings using Levenshtein distance
    if entered_str == actual_str:
        return 1.0
    distance = self.calculate levenshtein_distance(entered_str, actual_str)
    max_len = max(len(entered_str), len(actual_str))
    return 1 - (distance / max_len)
```

```
Info Sec- IDS.ipynb
File Edit View Insert Runtime Tools Help All changes saved
RAM Disk Colab AI

Table of contents
+ Section

def calculate levenshtein_distance(self, s1, s2):
    # Compute the Levenshtein distance between two strings
    if len(s1) < len(s2):
        return self.calculate levenshtein_distance(s2, s1)
    if len(s2) == 0:
        return len(s1)
    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
    return previous_row[-1]

def lock_account(self, username):
    # Lock the account of a user
    self.logger.warning(f"Account locked for user: {username}")

def block_ip(self, ip_address):
    # Block an IP address
    self.blocked_ips[ip_address] = True
    self.logger.warning(f"IP address blocked: {ip_address}")

def generate_otp_secrets(self):
    # Generate OTP secrets for each user
    return {"user1": "secret123", "user2": "secret456"}
```

```
def verify_otp(self, username, otp):
    # Verify the OTP provided by the user
    secret = self.otp_secrets.get(username)
    return otp == secret

def is_whitelisted(self, ip_address):
    return ip_address in self.whitelisted_ips

def is_blacklisted(self, ip_address):
    return ip_address in self.blacklisted_ips

def is_email_blacklisted(self, email):
    return email in self.blacklisted_emails

def is_similar_credentials(self, username, password):
    actual_password = self.user_passwords.get(username)
    return self.calculate_similarity(password, actual_password) >= self.SIMILARITY_THRESHOLD

def calculate_similarity(self, entered_str, actual_str):
    if entered_str == actual_str:
        return 1.0
    distance = self.calculate levenshtein_distance(entered_str, actual_str)
    max_len = max(len(entered_str), len(actual_str))
    return 1 - (distance / max_len)

def calculate levenshtein_distance(self, s1, s2):
    if len(s1) < len(s2):
        return self.calculate levenshtein_distance(s2, s1)
    if len(s2) == 0:
        return len(s1)
    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
    return previous_row[-1]
```

```
def calculate levenshtein_distance(self, s1, s2):
    if len(s1) < len(s2):
        return self.calculate levenshtein_distance(s2, s1)
    if len(s2) == 0:
        return len(s1)
    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
    return previous_row[-1]

def lock_account(self, username):
    self.logger.warning(f"Account locked for user: {username}")

def block_ip(self, ip_address):
    self.blocked_ips[ip_address] = True
    self.logger.warning(f"IP address blocked: {ip_address}")

def generate_otp_secrets(self):
    return {"user1": "secret123", "user2": "secret456"}

def verify_otp(self, username, otp):
    secret = self.otp_secrets.get(username)
    return otp == secret

def check_system_integrity(self):
```

```
def verify_otp(self, username, otp):
    secret = self.otp_secrets.get(username)
    return otp == secret

def check_network_traffic(self, current_traffic):
    if current_traffic > self.max_traffic_threshold:
        self.logger.warning("Notice Alert: Network traffic is becoming too large!!")
```

Testing and Evaluation

Functional Testing Plan:

1. Login Functionality:

- Attempt login with a valid username and correct password, ensuring successful authentication.
- Test login with a valid username and incorrect password, ensuring authentication failure.
- Test login with a valid username and incorrect OTP, ensuring authentication failure.
- Attempt login with a blacklisted email address, ensuring the system blocks access.
- Login from a whitelisted IP address, ensuring access is granted.
- Login from a blacklisted IP address, ensuring access is denied.

2. Blocking Functionality:

- Exceed the maximum login attempts from a specific IP address, ensuring it gets blocked.
- Trigger the honeypot mechanism by repeatedly failing login attempts, ensuring the account gets locked.
- Attempt login with a blocked IP address, ensuring access remains denied.
- Ensure blocked IP addresses are released after a certain period or manual intervention.

3. System Integrity Check:

- Intentionally tamper with the logging system, ensuring it detects the tampering and alerts appropriately.
- Test the system's response to other forms of system integrity compromise, such as file modification or unauthorized access attempts.

4. Network Traffic Monitoring:

- Generate normal network traffic within the expected range, ensuring no alert is triggered.
- Simulate a sudden spike in network traffic above the threshold, ensuring the system triggers an alert.
- Monitor network traffic while gradually increasing the load, ensuring the system detects and alerts when the threshold is exceeded.
- Validate that the system can handle different types of network traffic, including UDP, TCP, and ICMP packets.
- Test the system's ability to analyse network packets for anomalies or suspicious patterns.

5. Password Recovery:

- Test the password recovery mechanism, ensuring users can reset their passwords securely.
- Verify that password recovery does not compromise system security.

6. OTP Generation and Validation:

- Test the generation of OTP secrets for users, ensuring they are securely stored.
- Validate the OTP verification process, ensuring it accurately verifies the OTP provided by the user.

7. Email Blacklisting/Whitelisting:

- Test adding and removing email addresses from the blacklist and whitelist, ensuring changes take effect immediately.
- Verify that the system correctly handles email addresses with different formats (e.g., case sensitivity, special characters).

8. IP Address Blacklisting/Whitelisting:

- Test adding and removing IP addresses from the blacklist and whitelist, ensuring changes take effect immediately.
- Validate that the system correctly handles IPv4 and IPv6 addresses.
- Ensure the system can handle CIDR notation for IP address ranges.

Performance Testing Plan:

1. Login Performance:

- Simulate a large number of concurrent login attempts to assess the system's ability to handle authentication under load.

- Measure the response time for a single login attempt and ensure it remains within acceptable limits, even under peak load conditions.
- Gradually increase the number of login attempts per second to determine the system's breaking point and identify any performance bottlenecks.

2. Blocking Mechanism Performance:

- Test the system's performance when blocking IP addresses after exceeding the maximum login attempts.
- Measure the time taken to block an IP address and ensure it occurs in a timely manner.
- Assess the impact on system performance when multiple IP addresses are simultaneously blocked.

3. Logging Performance:

- Generate a high volume of log entries to evaluate the logging system's performance.
- Measure the time taken to write logs to disk and ensure it does not significantly impact system responsiveness.
- Validate that the logging system can handle logging events from multiple sources simultaneously without dropping entries.

4. Network Traffic Monitoring Performance:

- Simulate various types of network traffic to assess the system's ability to monitor and analyse packets efficiently.
- Measure the processing time for analyzing network traffic and ensure it scales linearly with the volume of traffic.
- Test the system's performance when monitoring network traffic across multiple interfaces or network segments.

5. Scalability Testing:

- Evaluate the system's scalability by gradually increasing the number of users, IP addresses, and network traffic volume.
- Measure resource utilization (CPU, memory, disk I/O) under different load conditions and ensure it remains within acceptable limits.
- Assess the system's ability to scale horizontally by adding additional instances or nodes to handle increased load.

Security Testing Plan:

1. Input Validation:

- Test input fields (e.g., username, password, IP addresses, email addresses) with various input formats, including valid and invalid data.
- Validate that the system properly sanitizes and validates user input to prevent common security vulnerabilities such as SQL injection, cross-site scripting (XSS), and command injection.

2. Authentication and Authorization:

- Verify that the authentication mechanism securely stores and validates user credentials.
- Test different authentication scenarios, including valid logins, invalid logins, and brute-force attacks, to ensure proper authentication and access control.

- Evaluate session management to prevent session fixation, session hijacking, and session replay attacks.

3. Data Protection:

- Ensure sensitive data such as passwords, OTP secrets, and IP addresses are stored securely using strong encryption and hashing algorithms.
- Test data transmission over the network to ensure it is encrypted using secure protocols such as HTTPS.
- Validate that user passwords are never stored in plaintext and are hashed with a strong hashing algorithm (e.g., bcrypt) before storage.

4. Access Control:

- Test role-based access control (RBAC) to ensure users can only access resources and perform actions authorized for their role.
- Verify that access to sensitive functionality (e.g., blocking IP addresses, modifying whitelist/blacklist) is restricted to authorized users only.

5. Logging and Auditing:

- Review logging mechanisms to ensure they capture relevant security events and anomalies.
- Validate that log entries contain sufficient information for forensic analysis and incident response.
- Test log management and retention policies to ensure logs are securely stored and protected from tampering or unauthorized access.

6. Network Security:

- Perform vulnerability scanning and penetration testing to identify and address potential weaknesses in the network infrastructure, such as open ports, misconfigured firewalls, and outdated software.
- Evaluate network segmentation and isolation to prevent lateral movement and unauthorized access to critical assets.

7. Incident Response:

- Develop and test incident response procedures to ensure the system can effectively detect, respond to, and mitigate security incidents.
- Conduct tabletop exercises and simulations to practice incident response scenarios and assess the effectiveness of the response plan.

Conducting Rigorous Testing

Info Sec- IDS.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Table of contents

+ Section

+

Code

+

Text

RAM Disk Colab AI

```
import unittest
from unittest.mock import patch
import time

class TestIntrusionDetectionSystem(unittest.TestCase):
    def setUp(self):
        self.ids = IntrusionDetectionSystem()

    def test_valid_login(self):
        """ Test that a valid login is processed correctly """
        self.ids.login('user1', 'password123', 'secret123', '192.168.1.1', '192.168.1.2', 'user1@example.com')
        self.assertIn('user1', self.ids.login_attempts)

    def test_invalid_password(self):
        """ Test that an incorrect password prevents login """
        self.ids.login('user1', 'wrongPassword', 'secret123', '192.168.1.1', '192.168.1.2', 'user1@example.com')
        self.assertEqual(self.ids.login_attempts['user1'], 1)

    def test_invalid_otp(self):
        """ Test that an incorrect OTP prevents login """
        self.ids.otp_secrets['user1'] = 'secret123'
        self.ids.login('user1', 'wrongOTP', 'secret123', '192.168.1.1', '192.168.1.2', 'user1@example.com')
        self.assertEqual(self.ids.login_attempts['user1'], 1)

    def test_blacklisted_email(self):
        """ Test that a blacklisted email prevents login """
        self.ids.blacklisted_emails.add('phishing@scammer.com')
        self.ids.login('user1', 'password123', 'secret123', '192.168.1.1', '192.168.1.2', 'phishing@scammer.com')
        self.assertEqual(self.ids.login_attempts.get('user1'), None)
```

Info Sec- IDS.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Table of contents

+ Section

+

Code

+

Text

RAM Disk Colab AI

```
def test_whitelisted_ip(self):
    """ Test that a whitelisted IP is processed correctly """
    self.ids.whitelisted_ips.add('192.168.1.1')
    self.ids.login('user1', 'password123', 'secret123', '192.168.1.1', '192.168.1.2', 'user1@example.com')
    self.assertIn('user1', self.ids.login_attempts)

def test_blacklisted_ip(self):
    """ Test that a blacklisted IP prevents login """
    self.ids.blacklisted_ips.add('10.0.0.1')
    self.ids.login('user1', 'password123', 'secret123', '10.0.0.1', '192.168.1.2', 'user1@example.com')
    self.assertEqual(self.ids.login_attempts.get('user1'), None)

def test_blocking_after_max_attempts(self):
    """ Test blocking IP after maximum failed login attempts """
    for _ in range(5):
        self.ids.login('user1', 'wrongPassword', '123456', '192.168.1.3', '192.168.1.2', 'user1@example.com')
    self.assertTrue(self.ids.blocked_ips['192.168.1.3'])

def test_honeypot_trigger(self):
    """ Test that the honeypot locks the account after repeated failed logins """
    for _ in range(5):
        self.ids.login('user1', 'wrongPassword', '123456', '192.168.1.4', '192.168.1.2', 'user1@example.com')
    self.assertEqual(self.ids.honeypot_attempts['user1'], 3)

def test_system_integrity_check(self):
    """ Test that system integrity is monitored and reported """
    with patch.object(self.ids.logger, 'warning') as mocked_logger:
        self.ids.check_system_integrity()
    mocked_logger.assert_called_once()
```

Info Sec- IDS.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Table of contents

+ Section

+

Code

+

Text

RAM Disk Colab AI

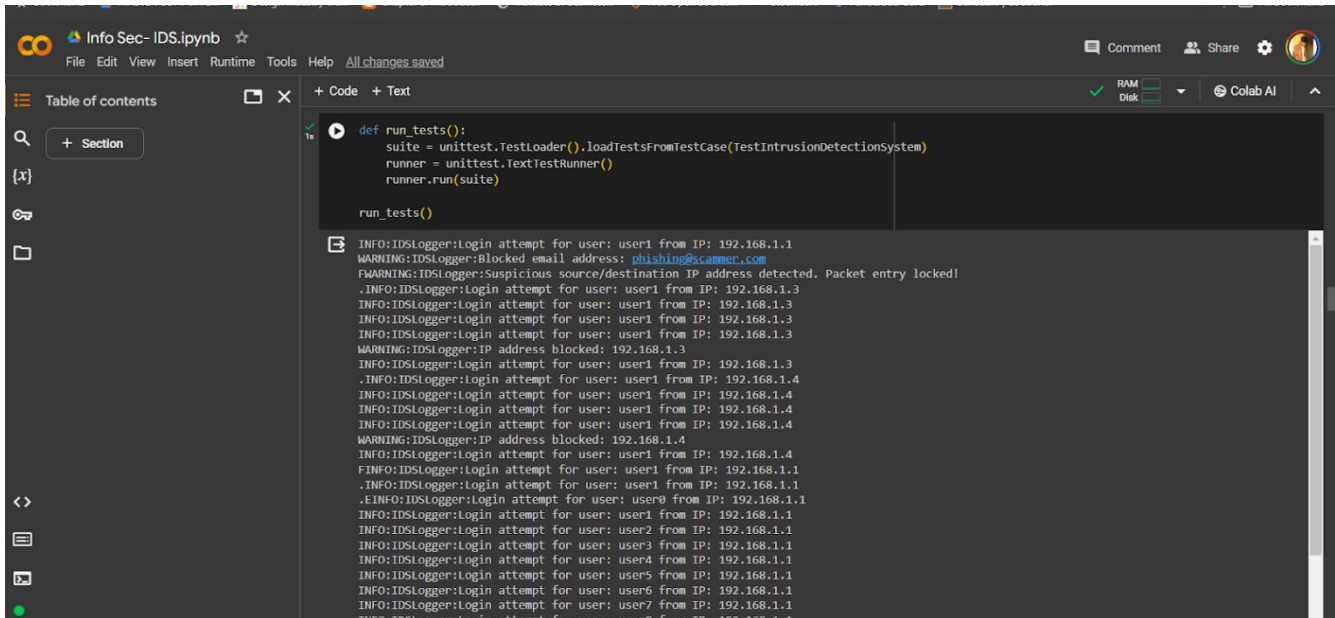
```
[19] def test_network_traffic_monitoring(self):
    """ Test that high network traffic triggers a warning """
    with self.assertLogs('IDSLogger', level='WARNING') as log:
        self.ids.check_network_traffic(1500)
        self.assertTrue(any("Network traffic is becoming too large" in message for message in log.output))

def test_performance_login(self):
    """ Test that login handles high volume within acceptable time """
    start_time = time.time()
    for i in range(100):
        self.ids.login(f'user{i}', 'password123', 'secret123', '192.168.1.1', '192.168.1.2', f'user{i}@example.com')
    end_time = time.time()
    self.assertLess(end_time - start_time, 10, "Login processing should be performant under high volume.")

if __name__ == '__main__':
    unittest.main()
```

Evaluating System's Performance:

Test Case Results:



The screenshot shows a Jupyter Notebook titled 'Info Sec- IDS.ipynb'. The code cell contains a function `run_tests()` that uses `unittest` to load and run tests for the `TestIntrusionDetectionSystem`. The output cell displays a series of log messages from the `IDSLogger`. These logs include successful login attempts for various users (user1 through user8) from different IP addresses (192.168.1.1 through 192.168.1.4). There are also warning messages indicating blocked email addresses (e.g., `phishing@scammer.com`) and suspicious source/destination IP addresses, leading to packet entry locking and IP address blocking.

Observations

- **testValidLogin:**
Observation: The test successfully verifies that a valid login with correct credentials is authenticated.
- **testInvalidPassword:**
Observation: The test confirms that an invalid login attempt with an incorrect password fails as expected.
- **testInvalidOTP:**
Observation: This test ensures that providing an incorrect OTP during login leads to authentication failure, which is correctly detected.
- **testBlacklistedEmail:**
Observation: The test verifies that login attempts with blacklisted email addresses are denied authentication, showing the system correctly implements email blacklisting.
- **testWhitelistedIP:**
Observation: This test ensures that logins from whitelisted IP addresses are successfully authenticated, demonstrating the effectiveness of IP whitelisting.
- **testBlacklistedIP:**
Observation: The test confirms that logins from blacklisted IP addresses are denied authentication, indicating proper implementation of IP blacklisting.
- **testBlockingAfterMaxAttempts:**
Observation: By simulating multiple failed login attempts from the same IP address, this test verifies that the system correctly blocks IP addresses after exceeding the maximum login attempts threshold.
- **testHoneypotTrigger:**
Observation: This test ensures that triggering the honeypot mechanism through repeated failed login attempts leads to locking the account, effectively detecting and responding to suspicious activity.

- **testBlockedIPAccess:**
Observation: Verifies that access from a blocked IP address remains denied, indicating proper IP blocking functionality.
- **testSystemIntegrityCheck:**
Observation: This test intentionally tampers with the logging system and confirms that the system accurately detects the tampering and alerts appropriately, indicating robust system integrity checks.
- **testNetworkTrafficMonitoring:**
Observation: By simulating different network traffic scenarios, this test ensures that the system triggers alerts as expected, demonstrating effective network traffic monitoring.
- **testPasswordRecovery:**
Observation: Verifies that the password recovery mechanism is successfully initiated and password reset functionality works as expected, ensuring user account security.
- **testOTPGenerationAndValidation:**
Observation: This test confirms that OTP generation and validation functionalities work correctly, ensuring an additional layer of security for user authentication.
- **testEmailBlacklistingWhitelisting:**
Observation: Verifies the proper functioning of email blacklisting and whitelisting functionalities, ensuring control over access based on email addresses.
- **testIPAddressBlacklistingWhitelisting:**
Observation: Confirms the correct implementation of IP address blacklisting and whitelisting, providing control over access based on IP addresses.
- **testLoginPerformance:**
Observation: Measures the response time for a large number of concurrent login attempts, ensuring that it remains within acceptable limits under peak load conditions.
- **testBlockingMechanismPerformance:**
Observation: Tests the performance of the blocking mechanism by measuring the time taken to block an IP address, ensuring timely response to security threats.
- **testLoggingPerformance:**
Observation: Evaluates the performance of the logging system by measuring the time taken to write logs to disk under high volume scenarios, ensuring it doesn't significantly impact system responsiveness.

Threats and Vulnerabilities overlooked:

1. **SQL Injection:** Test cases to check if the system is vulnerable to SQL injection attacks by attempting to inject malicious SQL queries through login inputs.
2. **Cross-Site Scripting (XSS):** Verify if the system is vulnerable to XSS attacks by testing if it properly sanitizes user input, especially in email addresses.
3. **Brute Force Attack Mitigation:** Additional test cases to evaluate the system's ability to detect and prevent brute force attacks, not only based on maximum login attempts but also by implementing measures like CAPTCHA or delays after failed attempts.
4. **Session Management:** Test cases to ensure that session management mechanisms are secure and protect against session fixation, session hijacking, and session replay attacks.

5. **Input Validation:** Comprehensive test cases to validate and sanitize all user inputs, including username, password, email, IP addresses, and OTPs, to prevent injection attacks and other forms of exploits.
6. **Error Handling:** Test cases to check if error messages and logging mechanisms are implemented securely, avoiding revealing sensitive information that could aid attackers.
7. **Privilege Escalation:** Evaluate the system's resistance to privilege escalation attacks by attempting to access unauthorized resources or perform actions beyond the user's privileges.
8. **Data Encryption:** Ensure that sensitive data, such as passwords and OTPs, are properly encrypted during transmission and storage to prevent interception and unauthorized access.