EXPERIMENT 1

Case Study of Wireshark Analyse the Local Network Using Wireshark

Aim:

To study the features of Wireshark and analyze network traffic using packet capturing techniques.

Objective:

- To capture and interpret packets on the local network.
- To identify protocols in use and diagnose network issues.

Theory:

Wireshark is a network protocol analyzer that captures packets in real-time, enabling the analysis of network behavior and issues. It supports various protocols like HTTP, TCP/IP, and FTP.

Wireshark is an open-source network protocol analyzer that captures and inspects packets on a network. It is an indispensable tool for network administrators and cybersecurity professionals. By intercepting packets in real time, it helps analyze network performance, troubleshoot issues, and detect malicious activity.

Wireshark works by interfacing with a computer's network adapter to capture raw packet data, which can then be dissected into headers and payloads to reveal detailed protocol information. For example, protocols like TCP/IP, HTTP, DNS, and SSL/TLS can be examined to uncover issues like slow connections, dropped packets, or suspicious activity.

A packet consists of headers and a data payload. Headers store metadata such as source/destination IP addresses, protocol type, and port number. In modern networks, encrypted payloads make it harder to interpret the content but headers remain useful for diagnostics. Wireshark's filtering options allow users to focus on specific packets, such as HTTP GET requests or failed TCP retransmissions.

In addition to diagnostics, Wireshark helps in:

1. **Security auditing:** Detecting attempts like port scans or unauthorized access.
2. **Protocol analysis:** Monitoring specific protocols and compliance.
3. **Application troubleshooting:** Identifying misconfigured applications.

Its intuitive interface enables users to analyze detailed packet data and create visualizations like I/O graphs. However, it requires proper configuration to avoid capturing unnecessary traffic. A powerful feature is its ability to decrypt SSL/TLS traffic when given access to private keys.

Wireshark should be used ethically and only on networks where permission has been granted. Unauthorized use could breach privacy laws or internal policies.

Steps:

1. Install Wireshark.
2. Select the network interface for capturing.
3. Start capturing packets.
4. Analyze captured packets for protocol types, IP addresses, and traffic.

Output:

- Packet capture statistics including protocols, source/destination IPs, and errors (if any).

Conclusion:

Wireshark is an efficient tool for network analysis and troubleshooting. It provides insights into network security and potential vulnerabilities.

Viva Questions:

- What is Wireshark, and what are its primary uses?
- What are the different types of filters in Wireshark?
- How can you capture traffic on a specific network interface?
- Explain the difference between live capture and reading a capture file in Wireshark.
- What is the significance of packet headers in network analysis?
- How does Wireshark handle encrypted traffic?
- What is the importance of using a display filter in Wireshark?
- What protocols can Wireshark analyze, and why is this important for network diagnostics?
- Can Wireshark be used for ethical hacking? If so, how?
- What legal and ethical considerations must be followed when using Wireshark?

EXPERIMENT 2

Implement Caesar Cipher Encryption and Decryption Techniques

Aim:

To implement and understand Caesar cipher encryption and decryption algorithms.

Objective:

- To encrypt plaintext using the Caesar cipher technique.
- To decrypt ciphertext to retrieve the original message.

Theory:

Caesar cipher is a substitution cipher that shifts characters in plaintext by a fixed number (key).

The Caesar cipher is one of the oldest known encryption techniques. It is a simple substitution cipher where each letter in the plaintext is shifted by a fixed number of places down or up the alphabet. For example, with a shift of 3, 'A' becomes 'D', 'B' becomes 'E', and so on. This technique is named after Julius Caesar, who used it for military communications.

The mathematical basis of the Caesar cipher is modular arithmetic. Each character is converted to its numerical equivalent (e.g., 'A' = 0, 'B' = 1) and shifted using the formula:
Encrypted character=(Original character+Shift)mod 26\text{Encrypted character} = (\text{Original character} + \text{Shift}) \mod
26Encrypted character=(Original character+Shift)mod26

Decryption reverses this operation:
Decrypted character=(Encrypted character−Shift)mod 26\text{Decrypted character} = (\text{Encrypted character} - \text{Shift}) \mod
26Decrypted character=(Encrypted character−Shift)mod26

While simple to implement, the Caesar cipher is vulnerable to brute force attacks because it has only 25 possible keys. Frequency analysis can also easily break it by exploiting the predictable distribution of letters in a language.

Applications of the Caesar cipher today are mostly educational, as it serves as an introduction to cryptography. It demonstrates key concepts like encryption, decryption, and substitution, forming the basis for understanding more complex algorithms.

Code (Python):

```
def encrypt(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            shift_base = 65 if char.isupper() else 97
            result += chr((ord(char) - shift_base + shift) % 26 + shift_base)
        else:
            result += char
    return result


def decrypt(text, shift):
    return encrypt(text, -shift)


# Test
plaintext = "HELLO"
shift = 3
ciphertext = encrypt(plaintext, shift)
print("Encrypted:", ciphertext)
print("Decrypted:", decrypt(ciphertext, shift))
```

Output:

- **Encrypted:** KHOOR
- **Decrypted:** HELLO

Conclusion:

Caesar cipher provides a basic level of encryption but is easily vulnerable to brute force attacks.

Viva Questions:

- What is the Caesar cipher, and how does it work?
- Why is Caesar cipher considered insecure for modern use?
- Explain the role of modular arithmetic in the Caesar cipher.
- How would you extend the Caesar cipher to work with numbers or symbols?
- What is the difference between encryption and decryption?
- How can you determine the key used in a Caesar cipher if the plaintext is known?
- What are the main disadvantages of the Caesar cipher?
- In what historical context was the Caesar cipher used effectively?
- Why does the Caesar cipher only have 25 possible keys?

EXPERIMENT 3

Implement Caesar Cipher Brute Force Attack

Aim:

To demonstrate how brute force can decrypt messages encrypted with Caesar cipher.

Objective:

- To decode ciphertext by testing all possible keys.

Theory:

A brute force attack systematically tries all possible keys until the correct one is found. In the context of the Caesar cipher, this means attempting all 25 possible shifts. This method is computationally inexpensive due to the cipher's simplicity and the small size of its key space.

Brute force is particularly effective against encryption methods that lack sufficient complexity. Modern cryptographic algorithms counter brute force by using large key sizes (e.g., 128-bit AES keys), making exhaustive key searches infeasible.

Brute force also demonstrates the importance of incorporating additional layers of security, such as random initialization vectors or salting, in cryptographic systems.

In the case of the Caesar cipher, the attack can be automated by writing a program to decrypt the ciphertext for each key and identifying a meaningful plaintext. The approach highlights the vulnerability of simple substitution ciphers and underscores the need for robust encryption in practical applications.

Code (Python):

```
python
Copy code
def brute_force(ciphertext):
    for shift in range(26):
        print(f"Key {shift}: {decrypt(ciphertext, shift)}")

ciphertext = "KHOOR"
brute_force(ciphertext)
```

Output:

All possible plaintext messages with different keys.

Conclusion:

Brute force can easily break Caesar cipher due to its small key space.

Viva Questions:

- What is a brute force attack?
- Why is brute force effective against the Caesar cipher?
- How does a brute force attack differ from frequency analysis?
- What are the time complexities of a brute force attack on a Caesar cipher?
- How can you automate a brute force attack programmatically?
- What would happen if a Caesar cipher used a larger key space (e.g., 256 shifts)?
- Can brute force attacks be applied to modern encryption algorithms? Why or why not?
- What steps can be taken to defend against brute force attacks?
- How does the language of the plaintext affect the success of brute force attacks?
- How would you identify the correct plaintext among multiple decrypted outputs?

EXPERIMENT 4

Aim:

To apply frequency analysis to a decrypted message and identify the top three most frequent characters.

Objective:

- To demonstrate the use of frequency analysis in cryptography.
- To analyze the pattern of character distribution in a message.

Theory:

Frequency analysis exploits the statistical frequency of characters in a language (e.g., 'E' is the most common letter in English). This method is used in cryptanalysis to break substitution ciphers.

Frequency analysis is a cryptanalytic technique that exploits the statistical properties of a language. For example, in English, letters like 'E', 'T', 'A', and 'O' occur more frequently, while 'Z', 'X', and 'Q' are less common. Substitution ciphers, which replace letters while preserving their frequency, are especially vulnerable to this analysis.

The process involves counting the occurrences of each letter in a given text and comparing these frequencies to known language distributions. For longer texts, the distribution aligns more closely with natural language patterns, increasing the effectiveness of the technique.

In modern cryptography, frequency analysis is largely ineffective against algorithms like AES or RSA, which produce outputs indistinguishable from random data. However, it remains a valuable tool for breaking simpler ciphers, including Caesar, Vigenère, and Playfair ciphers.

Frequency analysis also applies in:

1. **Error detection:** Identifying anomalies in text processing.
2. **Linguistics:** Studying letter distributions in natural languages.
3. **Steganography:** Detecting hidden patterns.

Code (Python):

python
Copy code

```
from collections import Counter

def frequency_analysis(text):
    freq = Counter(filter(str.isalpha, text.upper()))
    return freq.most_common(3)

# Test
decrypted_message = "HELLO WORLD. THIS IS A TEST MESSAGE."
print("Top 3 frequent characters:", frequency_analysis(decrypted_message))
```

Output:

Top 3 frequent characters, e.g., [('S', 6), ('E', 4), ('L', 3)].

Conclusion:

Frequency analysis reveals patterns in encrypted text, making substitution ciphers vulnerable.

Viva Questions:

- What is frequency analysis, and how does it work?
- Why is frequency analysis effective against substitution ciphers?
- What are the limitations of frequency analysis?
- How do you calculate the frequency of letters in a given text?
- Why do certain letters occur more frequently in English text?
- How does the length of the ciphertext affect the accuracy of frequency analysis?
- Can frequency analysis be used for non-alphabetic substitution ciphers?
- How do digraphs and trigraphs enhance cryptanalysis?
- Why is frequency analysis ineffective against modern encryption algorithms?
- How would the distribution of letters differ in other languages, such as French or German?

EXPERIMENT 5

Aim:

To study the role of a firewall in network security and configure basic firewall rules in Windows.

Objective:

- To understand how firewalls block unauthorized access.
- To set inbound/outbound rules in Windows Firewall.

Theory:

A firewall monitors and controls network traffic based on predefined security rules. It can prevent unauthorized access, block malware, and filter content.

A firewall is a security device that monitors and controls incoming and outgoing network traffic based on predetermined rules. Acting as a barrier between a trusted internal network and untrusted external networks, it prevents unauthorized access and data exfiltration.

Firewalls can be hardware- or software-based:

1. **Packet filtering firewalls:** Operate at the network layer and inspect packets based on source/destination IP, port, and protocol.
2. **Stateful inspection firewalls:** Maintain a state table to track active connections, ensuring that incoming packets are part of established sessions.
3. **Application-layer firewalls:** Operate at the application layer, analyzing payloads and user behavior to detect threats.

Windows Firewall, integrated into Microsoft operating systems, provides features like:

1. Inbound/outbound traffic filtering.
2. Predefined and custom rules.
3. Logging and monitoring.

Proper configuration is essential to ensure balance between usability and security. Misconfigured firewalls can inadvertently block legitimate traffic or leave networks exposed to attacks.

Steps:

1. Open Windows Firewall settings.
2. Create inbound/outbound rules (e.g., block/allow specific ports or applications).
3. Test the firewall configuration.

Output:

A log of allowed and blocked connections.

Conclusion:

Firewalls are essential for network security, providing control over data flow and protecting against unauthorized access.

Viva Questions:

- What is a firewall, and why is it important?
- Explain the difference between hardware and software firewalls.
- What is the difference between packet filtering and stateful inspection?
- How does a firewall enforce security policies?
- What are some limitations of firewalls?
- How do you configure inbound and outbound rules in Windows Firewall?
- What is the purpose of logging in a firewall?
- What is a DMZ (Demilitarized Zone), and how is it related to firewalls?
- How can misconfigured firewall rules affect a network?
- What is the significance of NAT (Network Address Translation) in firewalls?

EXPERIMENT 6

Perform Steps to Secure a Web Browser

Aim:

To secure a web browser (e.g., Chrome, Firefox) by implementing security best practices.

Objective:

- To reduce browser vulnerabilities.
- To ensure safe browsing.

Steps:

1. Update the browser to the latest version.
2. Enable "Safe Browsing" or similar features.
3. Use HTTPS Everywhere and block third-party cookies.
4. Install ad blockers and privacy extensions.

Theory:

Web browsers are the primary interface for accessing the internet and are, therefore, a common target for attackers. They process and render dynamic web content, interact with various plugins and extensions, and store sensitive user data like passwords, cookies, and browsing history. This makes them susceptible to threats like malware, phishing attacks, and man-in-the-middle (MITM) attacks.

Common Web Browser Vulnerabilities

1. **Outdated Browser Software:** Older browser versions lack patches for newly discovered vulnerabilities.
2. **Weak Encryption:** Without proper HTTPS enforcement, users are at risk of eavesdropping.
3. **Third-party Cookies:** These can track user activities and compromise privacy.
4. **Malicious Extensions:** Extensions with excessive permissions can collect sensitive data.
5. **Drive-by Downloads:** Automatic downloads of malicious files can exploit unpatched software.
6. **Phishing Websites:** Fraudulent websites impersonating legitimate ones can steal credentials.

Steps to Secure a Browser

1. **Keep the Browser Updated:** Regular updates patch security vulnerabilities and improve performance.
2. **Enable Safe Browsing Features:** These detect and warn against harmful websites and downloads.
3. **Use HTTPS Everywhere:** Force all websites to use encrypted connections.
4. **Disable Unnecessary Extensions:** Remove extensions not actively used and scrutinize their permissions.
5. **Block Third-party Cookies:** Prevent advertisers and trackers from collecting data.
6. **Install Privacy-focused Add-ons:** Tools like uBlock Origin and Privacy Badger block ads and trackers.
7. **Enable Sandbox Mode:** Isolate browser processes to limit the impact of exploits.

Advanced Steps:

- Use DNS over HTTPS (DoH) for secure DNS lookups.
- Opt for browsers that prioritize privacy, such as Mozilla Firefox or Brave.
- Avoid saving passwords in the browser; use a dedicated password manager instead.

By securing browsers, users can significantly reduce their exposure to common cybersecurity threats. However, it is equally important to educate users about safe browsing habits, such as avoiding suspicious links or downloads.

Output:

A secure and privacy-focused browser setup.

Conclusion:

Proper browser security settings mitigate risks like phishing, malware, and data tracking.

Viva Questions:

- Why is securing a web browser important?
- What are the common threats associated with web browsers?
- Explain the significance of HTTPS in secure browsing.
- How do browser extensions pose a security risk?
- What is the role of sandboxing in web browsers?
- How does a privacy-focused browser differ from mainstream browsers?
- What are cookies, and how can they compromise user privacy?
- Explain the importance of regular updates in browser security.

- How can phishing attacks be mitigated at the browser level?
- What is DNS over HTTPS (DoH), and how does it enhance security?

EXPERIMENT 7

Study of Viruses, Malware, and Worms

Aim:

To study the characteristics and impacts of notable viruses, malware, and worms.

Objective:

- To understand how each type spreads and causes damage.
- To analyze real-world case studies like Stuxnet, CryptoLocker, and Zeus.

Theory:

- **Stuxnet:** A worm targeting SCADA systems.
- **CryptoLocker:** Ransomware encrypting user files for ransom.
- **Zeus:** A banking Trojan stealing financial data.

Theory:

Malicious software, or malware, encompasses a variety of programs designed to disrupt, damage, or gain unauthorized access to computer systems. Malware includes viruses, worms, Trojans, ransomware, spyware, and more. Understanding these types helps in devising effective countermeasures.

Key Types of Malware

1. **Viruses:**
   A virus attaches itself to a host file and spreads when the file is executed. Viruses can corrupt, delete data, or damage systems. Example: **ILOVEYOU Virus** (spread via email attachments).
2. **Worms:**
   Worms are self-replicating and spread across networks without user intervention. They exploit vulnerabilities in systems to propagate. Examples: **Morris Worm**, **Blaster Worm**, **Conficker Worm**.
3. **Trojans:**
   A Trojan appears as legitimate software but performs malicious activities when executed. Example: **Zeus Trojan** (used for stealing financial credentials).
4. **Ransomware:**
   Encrypts user data and demands payment for decryption keys. Example: **CryptoLocker** (one of the earliest and most infamous ransomware).

Analysis of Famous Malware

1. **Stuxnet (Worm):** Targeted SCADA systems in nuclear facilities, demonstrating the potential of malware in cyber warfare.
2. **Melissa Virus:** A mass-mailing macro virus spread through infected Word documents.
3. **ILOVEYOU Virus:** Caused billions of dollars in damages by spreading through email with the subject line "I love you."

Defensive Measures:

- Keep software and operating systems updated.
- Use antivirus software and intrusion detection systems (IDS).
- Practice safe email habits and avoid opening suspicious attachments.
- Employ network segmentation to limit the spread of worms.

Steps:

1. Research each malware type.
2. Create a comparative table of their features, attack methods, and impacts.

Output:

A detailed report on each malware's behavior.

Conclusion:

Studying malware helps in understanding modern cybersecurity threats and creating effective countermeasures.

Viva Questions:

- What is malware, and how is it classified?
- Explain the differences between viruses, worms, and Trojans.
- How do ransomware attacks work, and what are their objectives?
- What was the impact of the Stuxnet worm?
- How does a virus spread differently from a worm?
- What defensive measures can prevent malware infections?
- What is the role of antivirus software in detecting malware?
- How does a zero-day vulnerability contribute to malware attacks?
- Why are email attachments a common vector for spreading malware?
- What lessons can be learned from past malware outbreak?

EXPERIMENT 8

Aim:

To identify and analyze security vulnerabilities in email systems.

Objective:

- To evaluate threats like phishing, spoofing, and spam.
- To recommend security measures.

Theory:

Common email vulnerabilities include weak authentication, open relay configuration, and lack of encryption.

Theory:

Email systems are among the most commonly used communication tools, making them prime targets for attackers. Security vulnerabilities in email applications can lead to data breaches, phishing attacks, and malware distribution.

Common Security Threats in Email Applications

1. **Phishing Attacks:** Fraudulent emails attempt to steal user credentials or deploy malware.
2. **Email Spoofing:** Attackers impersonate legitimate senders to deceive recipients.
3. **Open Relay Configuration:** Allows attackers to send spam emails from legitimate servers.
4. **Weak Authentication:** Lax password policies increase susceptibility to brute-force attacks.
5. **Lack of Encryption:** Emails sent without encryption can be intercepted and read.
6. **Malicious Attachments:** Files like PDFs or Word documents can execute malicious code.

Tools for Analyzing Email Security

1. **Header Analysis:** Email headers reveal the sender's IP address, route, and potential spoofing attempts.
2. **SPF, DKIM, and DMARC:** Protocols that authenticate email sources and prevent spoofing.
3. **Sandboxing Attachments:** Testing email attachments in an isolated environment to detect malicious behavior.

Defensive Measures:

- Enable two-factor authentication for email accounts.
- Configure spam filters to block phishing emails.
- Educate users on recognizing suspicious emails.
- Ensure the email application supports end-to-end encryption (e.g., PGP).

Steps:

1. Review email application settings.
2. Analyze headers of phishing emails.
3. Test email encryption (e.g., PGP).

Output:

A list of vulnerabilities and potential solutions.

Conclusion:

Securing email applications is critical to prevent data breaches and unauthorized access.

Viva Questions:

- What are the common security vulnerabilities in email applications?
- Explain the concept of email spoofing.
- How does phishing exploit email vulnerabilities?
- What is the role of SPF, DKIM, and DMARC in email security?
- How can email encryption protect against MITM attacks?
- What are the signs of a compromised email account?
- How does spam filtering work in email applications?
- What is an open relay, and why is it a security risk?
- How can sandboxing help detect malicious email attachments?
- What steps can users take to secure their email accounts?

EXPERIMENT 9

Aim:

To study security vulnerabilities in e-commerce platforms and suggest mitigation strategies.

Objective:

- To analyze threats like SQL injection, cross-site scripting (XSS), and payment fraud.

Theory:

E-commerce platforms are targets for attackers due to sensitive data like payment information.

E-commerce platforms handle sensitive customer data such as personal information, payment details, and transaction records. These platforms are lucrative targets for attackers seeking to exploit vulnerabilities to steal data or disrupt operations.

Common Vulnerabilities in E-Commerce Services

1. **SQL Injection:** Exploits improperly sanitized input fields to execute malicious database queries.
2. **Cross-Site Scripting (XSS):** Injects malicious scripts into web pages viewed by users.
3. **Cross-Site Request Forgery (CSRF):** Tricks users into performing actions without their consent.
4. **Weak Session Management:** Allows attackers to hijack user sessions.
5. **Insecure Payment Gateways:** Compromises credit card and payment data.
6. **Lack of HTTPS:** Transmits sensitive data in plaintext, enabling MITM attacks.

Defensive Strategies:

- Validate and sanitize user inputs to prevent SQL injection.
- Use Content Security Policies (CSP) to mitigate XSS.
- Implement secure session management practices (e.g., using secure cookies).
- Enforce HTTPS with valid SSL/TLS certificates.
- Regularly test for vulnerabilities using penetration testing tools.

Steps:

1. Review the e-commerce platform's architecture.

2. Simulate attacks like XSS or SQL injection (if authorized).
3. Recommend security practices (e.g., secure payment gateways, HTTPS).

Output:

A vulnerability report with mitigation strategies.

Conclusion:

E-commerce security ensures customer trust and protects against financial loss.

Viva Questions:

- What are the most common vulnerabilities in e-commerce platforms?
- Explain how SQL injection works in e-commerce systems.
- How does HTTPS protect e-commerce transactions?
- What is the role of payment gateways in securing online transactions?
- How can session hijacking be prevented?
- What is cross-site scripting (XSS), and how does it affect e-commerce sites?
- How does PCI DSS compliance enhance e-commerce security?
- Why is input validation critical for e-commerce security?
- How does two-factor authentication protect customer accounts?
- What measures can be taken to prevent CSRF attacks?

EXPERIMENT 10

Develop and Test a Two-Factor Authentication System

Aim:

To implement a two-factor authentication (2FA) system for a web or mobile application.

Objective:

- To enhance security by requiring two independent authentication factors.

Theory:

2FA combines "something you know" (password) with "something you have" (e.g., OTP). It provides stronger security than single-factor authentication.

Theory:

Two-factor authentication (2FA) adds an additional layer of security by requiring two forms of verification to access an account:

1. **Something you know:** A password or PIN.
2. **Something you have:** A physical device (e.g., smartphone, token) or biometric (e.g., fingerprint, facial recognition).

Benefits of 2FA:

- Protects against phishing and brute-force attacks.
- Ensures that even if a password is compromised, access cannot be granted without the second factor.

Types of 2FA:

1. **SMS-based OTPs:** One-time passwords sent to a registered mobile number.
2. **App-based Authentication:** Applications like Google Authenticator generate time-based OTPs.
3. **Hardware Tokens:** Devices like YubiKey generate unique authentication codes.
4. **Biometrics:** Uses physical attributes like fingerprints or iris scans.

Implementation Steps:

1. Develop a system that generates and verifies OTPs.
2. Integrate the system with a database of users for verification.

3. Test the system with mock users to ensure reliability.

Code (Python):

python
Copy code
```
import random

def generate_otp():
    return random.randint(100000, 999999)

# Simulate 2FA
user_password = "password123"
input_password = input("Enter password: ")

if input_password == user_password:
    otp = generate_otp()
    print(f"Your OTP is: {otp}")
    input_otp = int(input("Enter OTP: "))
    if input_otp == otp:
        print("Authentication Successful!")
    else:
        print("Invalid OTP.")
else:
    print("Invalid Password.")
```

Output:

- Authentication messages based on correct/incorrect password and OTP.

Conclusion:

2FA significantly enhances application security by adding an additional layer of verification.

Viva Questions:

- What is two-factor authentication (2FA), and how does it work?
- Explain the difference between one-time passwords (OTPs) and biometrics in 2FA.
- What are the advantages of 2FA over single-factor authentication?
- How do time-based OTPs ensure security?
- What role do SMS messages play in 2FA, and what are their vulnerabilities?
- How does app-based 2FA differ from hardware token-based 2FA?

- What is the significance of secure key storage in 2FA systems?
- How would you implement 2FA in a web application?
- What are the challenges in adopting 2FA for large-scale systems?
- Why is user education important when implementing 2FA