CROSS-SITE SCRIPTING ATTACK (DA - 6)

SUBMITTED BY

GAURAV KUMAR SINGH (19BCE2119)

AGARWAL CHIRAG SANJAY (19BCI0202)

PENKE LOHITH SASI ANVESH (19BCE2069)

INTRODUCTION:

Cross-site Scripting (XSS) is a client-side code injection attack that allows an attacker to compromise the interactions that users have with a vulnerable application. The attacker aims to execute malicious scripts in a web browser of the victim by including malicious code in a authorised web page or web application. The actual attack occurs when the victim visits the web page or web application that executes the malicious code. The web page or web application becomes a vehicle to deliver the malicious script to the user's browser.

TOOL INSTALLATION:

We haven't used any tool. Testing is done manually.

VARIABLE	VALUE
OS	Linux/Windows
Browser	Firefox/chrome
Tool used	Burp suite

OBJECTIVES:

- First, we should run malicious JavaScript code in a victim's browser, an attacker must first find a way to inject malicious code into a web page that the victim visits.
- After that, the victim must visit the web page with the malicious code. If the attack is directed at particular victims, the attacker can use social engineering and/or phishing to send a malicious URL to the victim.

Types of XSS:

1. Reflected XSS

2. Stored XSS

3. DOM Based XSS

Reflected XSS:

1. Attacker will craft a URL and will send to the attacker's browser. Attacker will usually use social engineering skills to do this.
2. User will be tricked into clicking that link and will send a HTTP request to the web site.
3. Then the server will response with the malicious code in the URL
4. Then as that response get rendered by the browser, it will execute the malicious code too. In this scenario the URL includes the victim's cookies as a query parameter, which the attacker can extract from the request when it arrives to his server.

Reflected XSS attacks, occur when a malicious script is reflected off of a web application to the victim's browser. The script is activated through a link, which sends a request to a website with a vulnerability that enables execution of malicious scripts. To distribute the malicious link, an attacker typically embeds it into an email or third-party website (e.g., in a comment section or in social media).

Stored XSS:

1) In this step attacker is using a input form in the website to store the malicious link (eg:- in a comment or as a status)
2) Here a user will request the web page which contains the malicious code unknowingly.
3) Server will send the response to the server with the malicious content
4) User's browser will render that page which will make the malicious content execute. After this execution user becomes a victim of XSS. (The **window. Location** object can be used to get the current page address (URL) and to redirect the browser to a new page.)

Stored XSS attacks involve an attacker injecting a script that is permanently stored (persisted) on the target application (e.g.: - database). A classic example is a malicious script inserted by an attacker in a comment field on a blog or in a forum post. When a user opens the affected web page in a browser, the XSS content will be served as part of the web page (just like a normal comment would). This means that the user will unknowingly execute that malicious code and becomes a victim.

DOM XSS

1. The attacker makes a URL containing a malicious string and sends it to the victim.
2. The victim is tricked by the attacker into requesting the URL from the website.
3. The website receives the request, but does not include the malicious string in the response.
4. The victim’s browser executes the legal script inside the response, causing the malicious script to be inserted into the page.
5. The victim’s browser executes the malicious script inserted into the page, sending the victim’s cookies to the attacker’s server.

The server inserts the malicious script into the page, which is then sent in a response to the victim. When the victim's browser receives the response, it assumes the malicious script to be part of the page's original content and automatically executes it during page load as with any other script. In the example of a DOM-based XSS attack, however, there is no malicious script inserted as part of the page; the only script that is automatically executed during page load is an original part of the page.

DEMONSTARTION's :

REFLECTED XSS

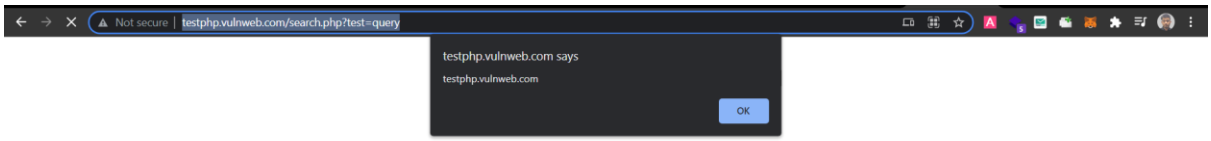
Payload:

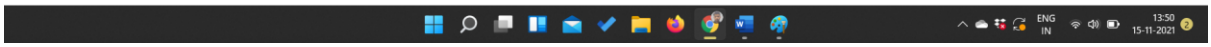
"><script>confirm(document.domain)</script>

Vulnerable location: Search box

Vulnerable url: http://testphp.vulnweb.com/search.php?test=query

Screenshot:





STORED XSS

Payload:

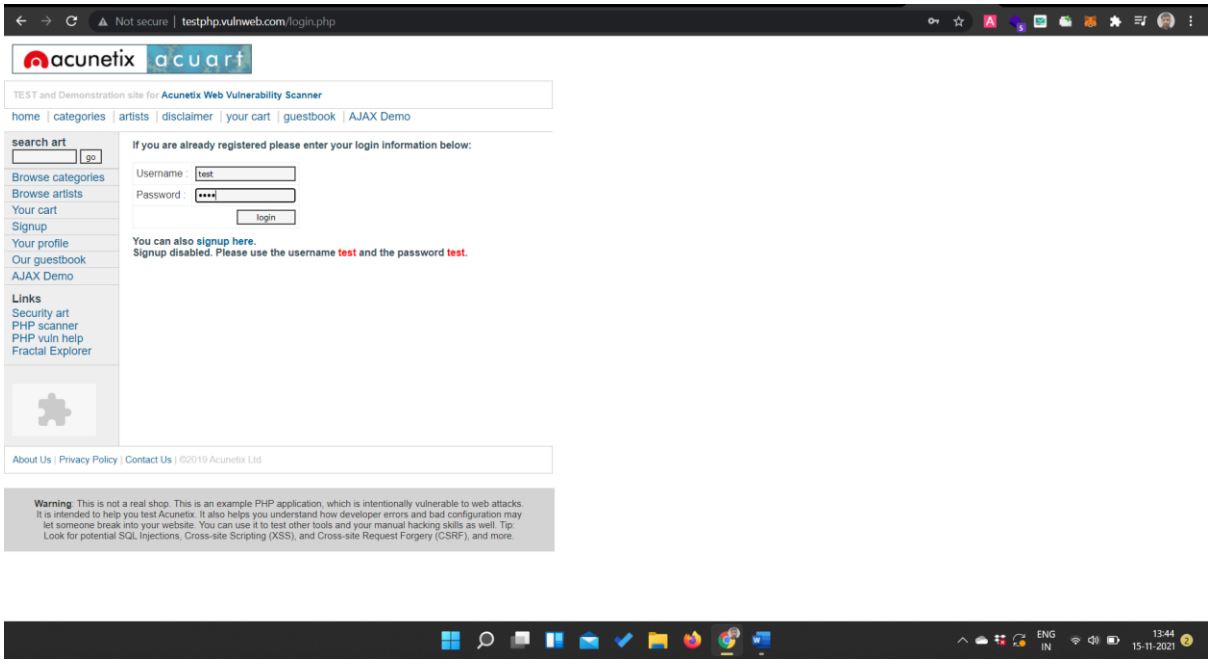
">

"><svg/onload=confirm(STORED_XSS_VIT) />

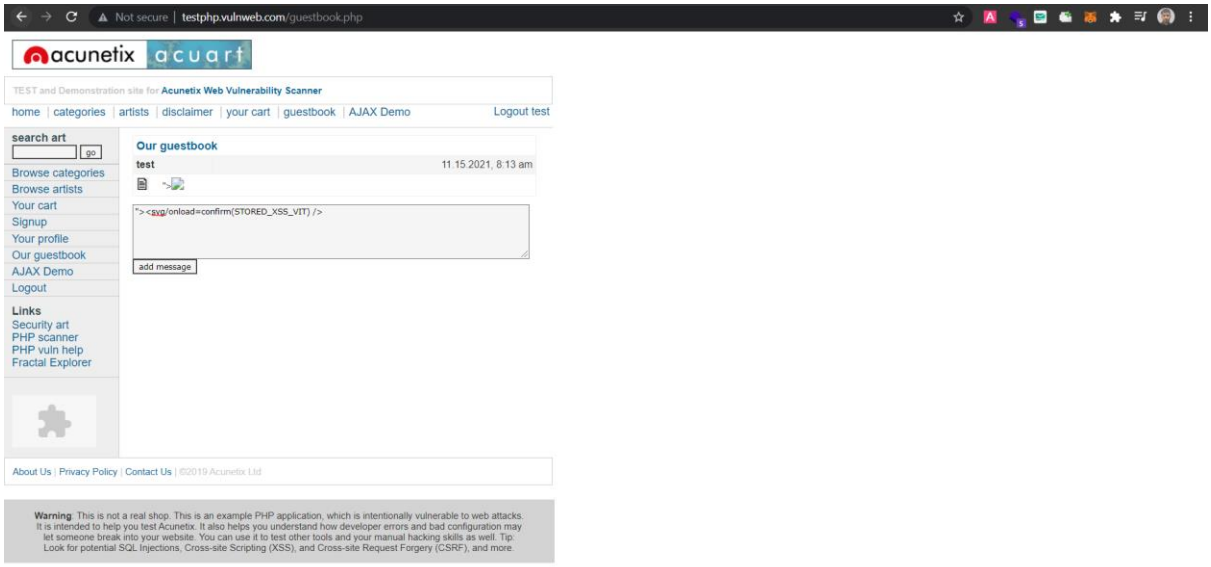
Vulnerable location: Credit card input box

Vulnerable URL: http://testphp.vulnweb.com/userinfo.php

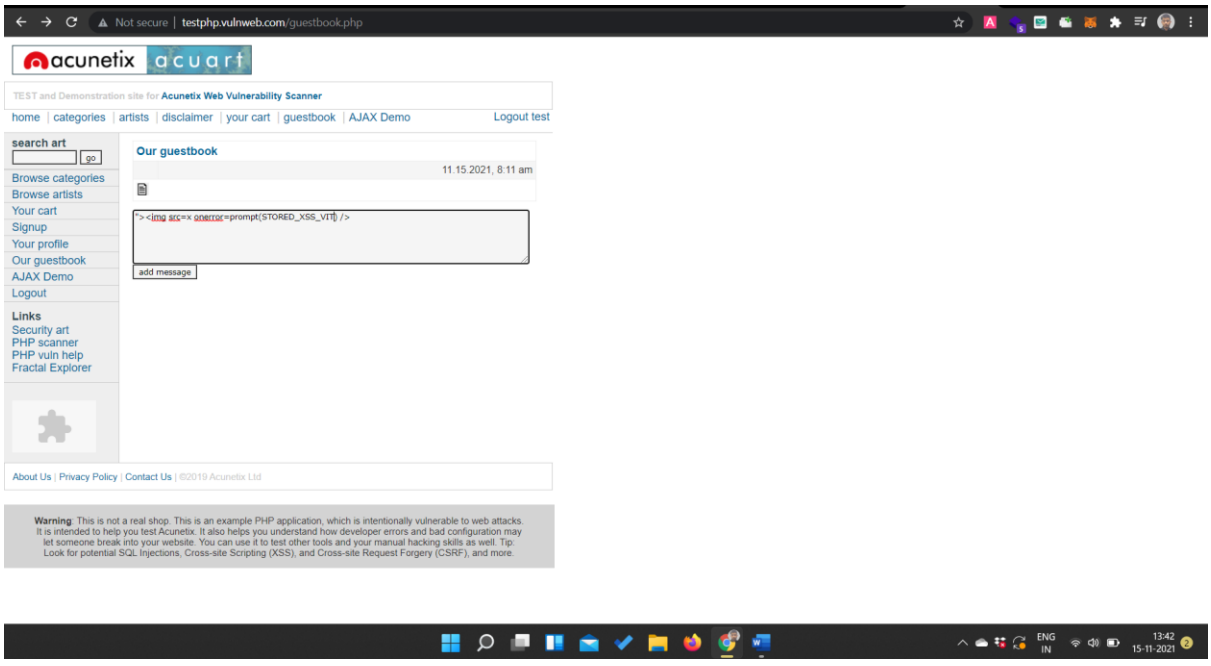
Screenshot of login page



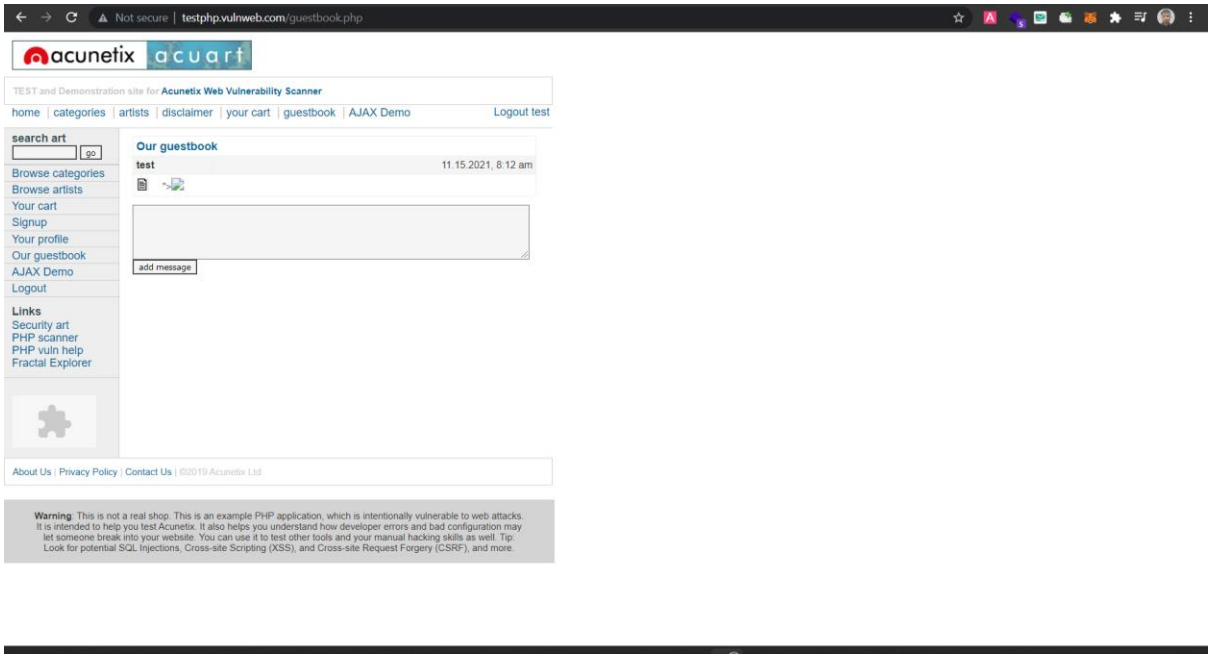
Screenshot of testing page:



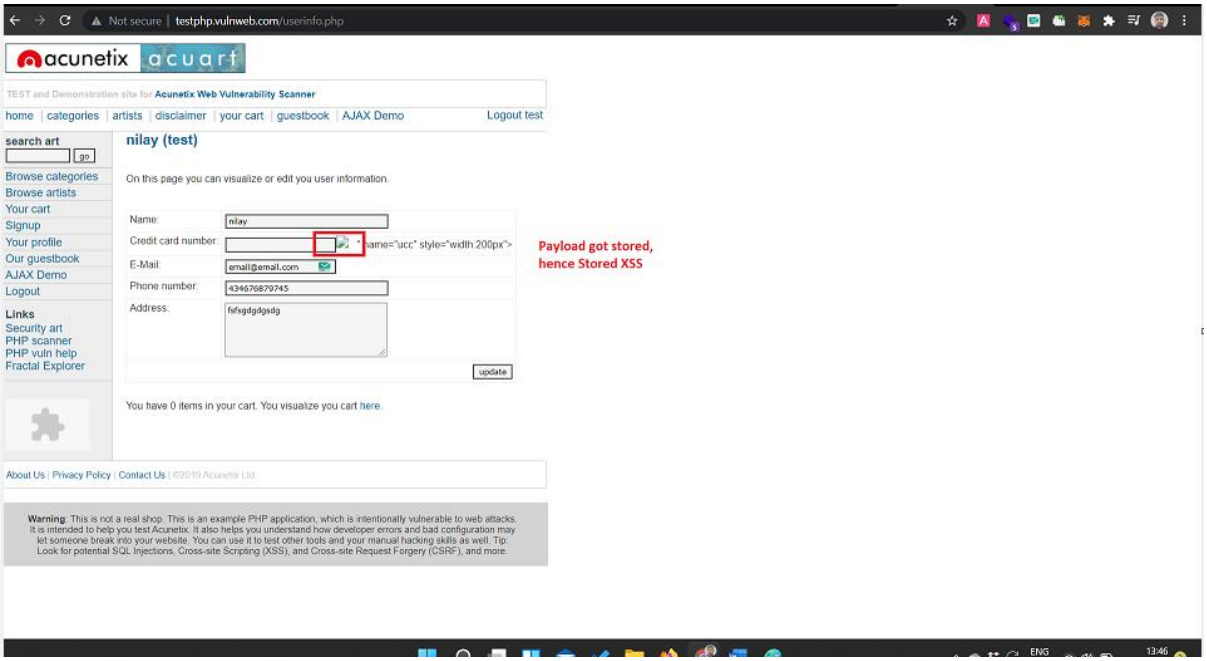
Screenshot of text input page



Screenshot of page after adding the text message



Screenshot of result after tested:



THANK YOU