Assignment 4

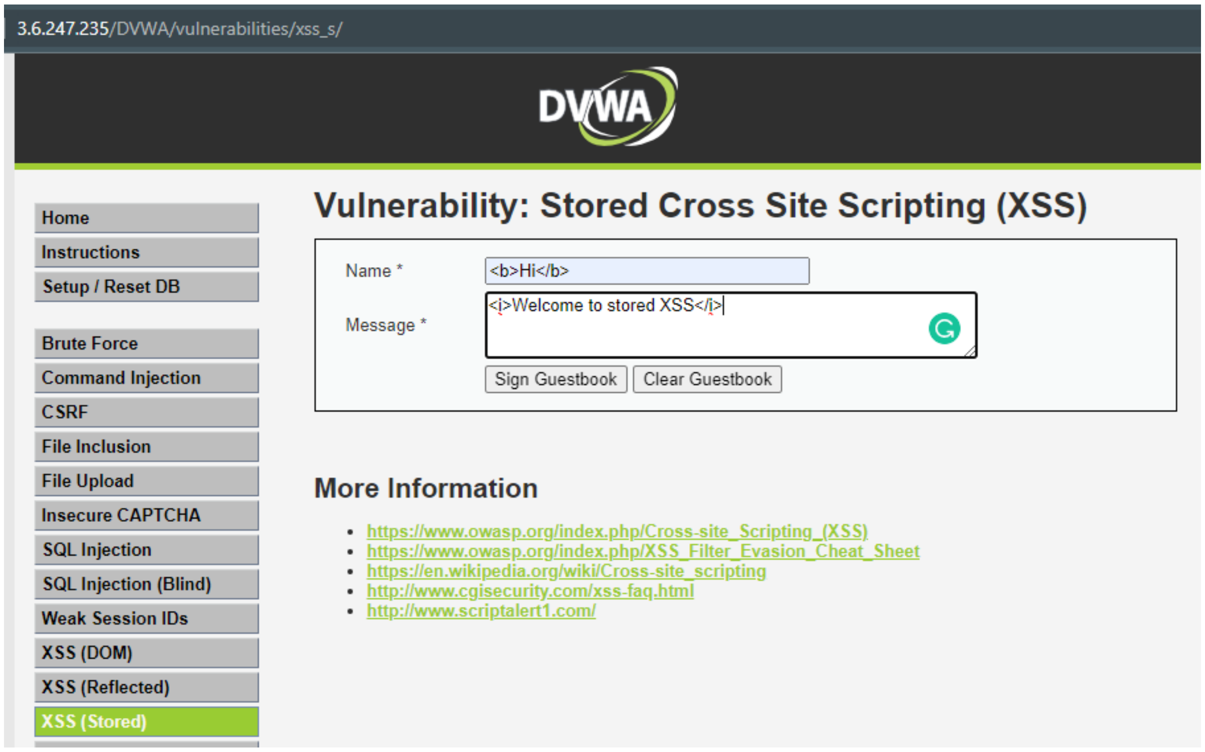
Ruchita Dinesh Entoliya (015331966)
Komal Joshi (014682551)

Stored XSS attack

1. Describe the attack you used. How did it work?

Stored XSS is the most dangerous cross site vulnerability. This type of vulnerability arises whenever a web application stores user supplied data for later use in backend without performing any filter or input sanitization. Since the web application does not apply any filter therefore an attacker can inject some malicious code into this input field.

After navigating to XSS stored tab, on entering html tags in name and message field we see html injection takes place where tags modify the responses.



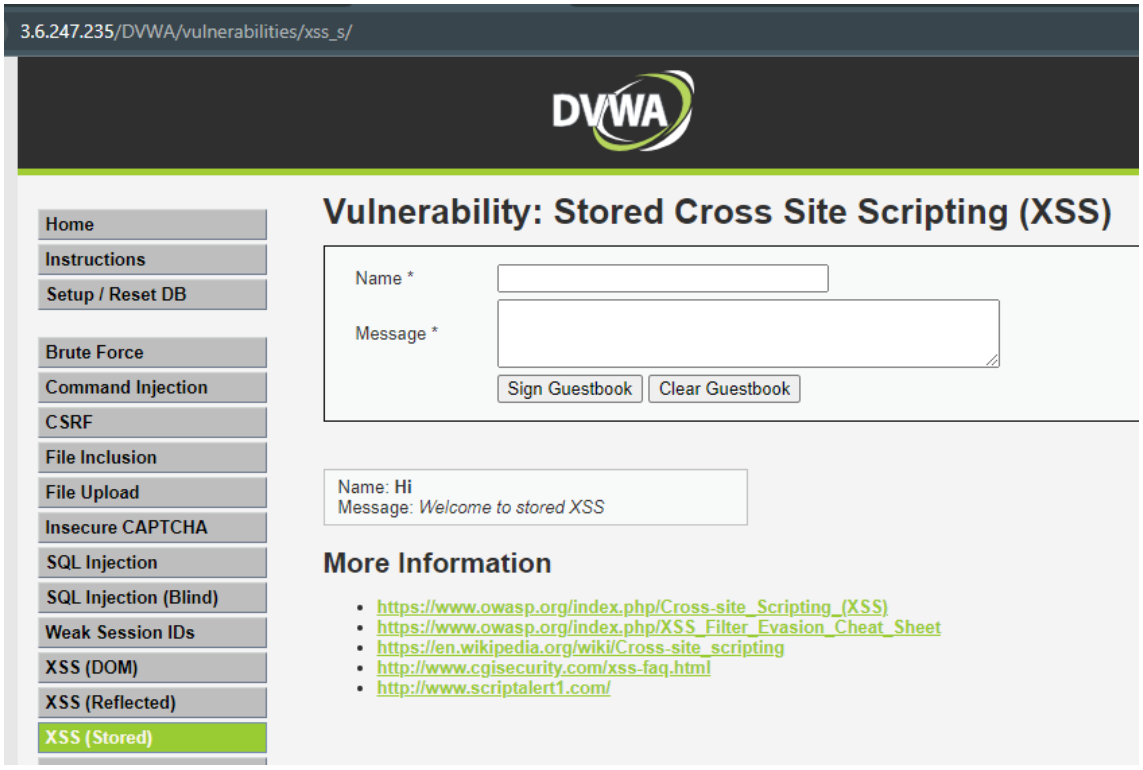
Vulnerability: Stored Cross Site Scripting (XSS)

Name * `<b>Hi</b>`

Message * `<i>Welcome to stored XSS</i>`

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

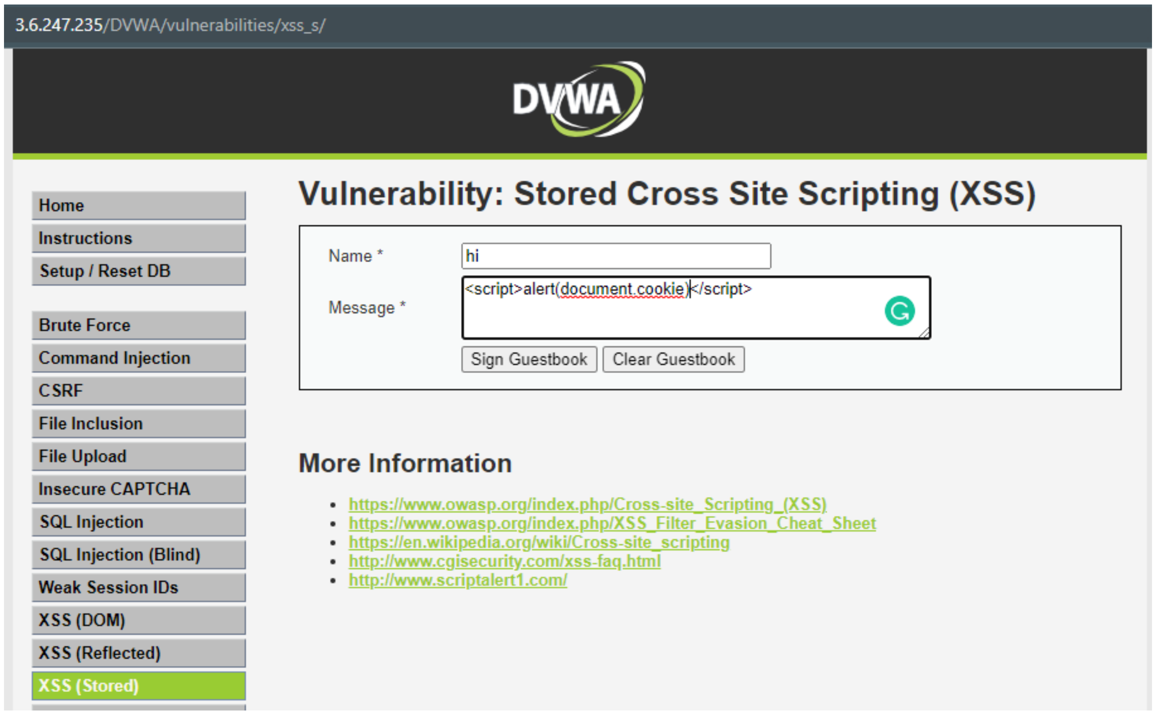
Name: Hi
Message: *Welcome to stored XSS*

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

To check JavaScript injection, entering <script>alert("JavaScript!")</script> in message field causes JavaScript injection. Alert message appears on the screen. On low security, page is vulnerable to XSS.

Next, we ty to get session ID and cookie to display using document.cookie as the alert parameter within the script tag in message field.



Vulnerability: Stored Cross Site Scripting (XSS)

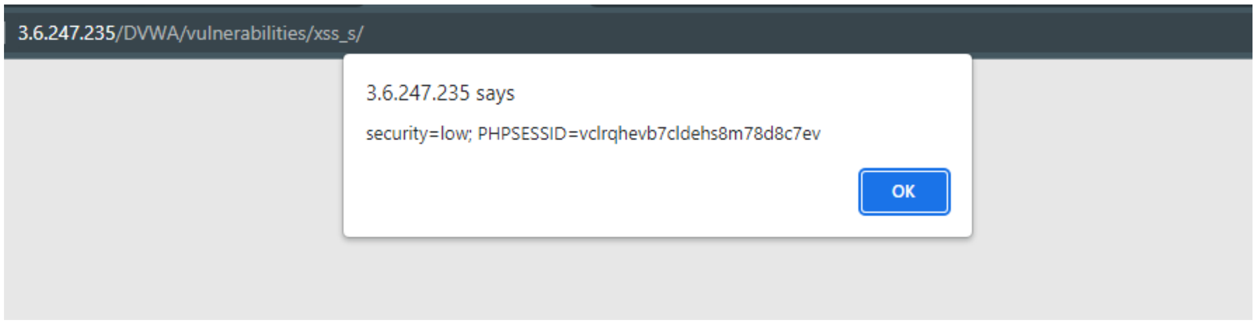
Name * hi

Message * <script>alert(document.cookie)</script>

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

Response

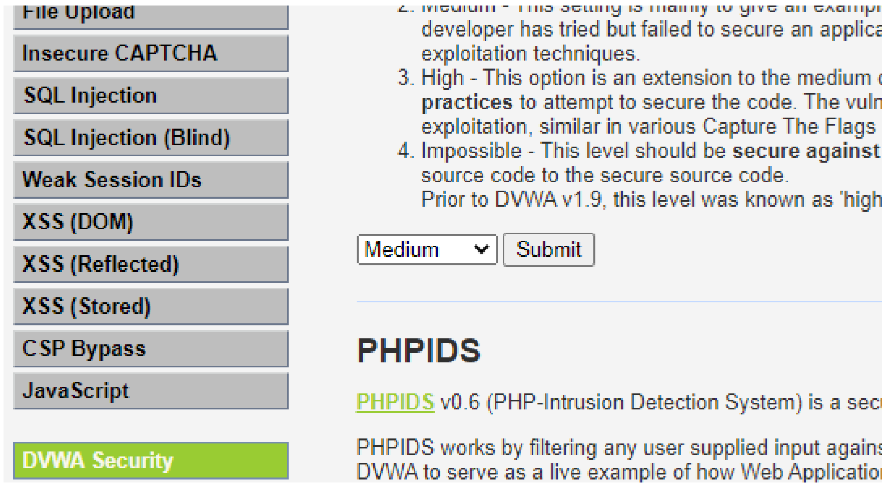

3.6.247.235/DVWA/vulnerabilities/xss_s/

3.6.247.235 says

security=low; PHPSESSID=vclrqhevb7cldehs8m78d8c7ev

This is stored XSS attack as the session id is persistent even after you change tabs and come back to this page. Every time we visit this page same popup is displayed. To clear these details and prevent the popup from re-appearing we need to clear the guestbook.

2. Does your attack work in "Medium" security level?



2. Medium - This setting is mainly to give an exampl
developer has tried but failed to secure an applica
exploitation techniques.
3. High - This option is an extension to the medium d
practices to attempt to secure the code. The vuln
exploitation, similar in various Capture The Flags
4. Impossible - This level should be **secure against**
source code to the secure source code.
Prior to DVWA v1.9, this level was known as 'high

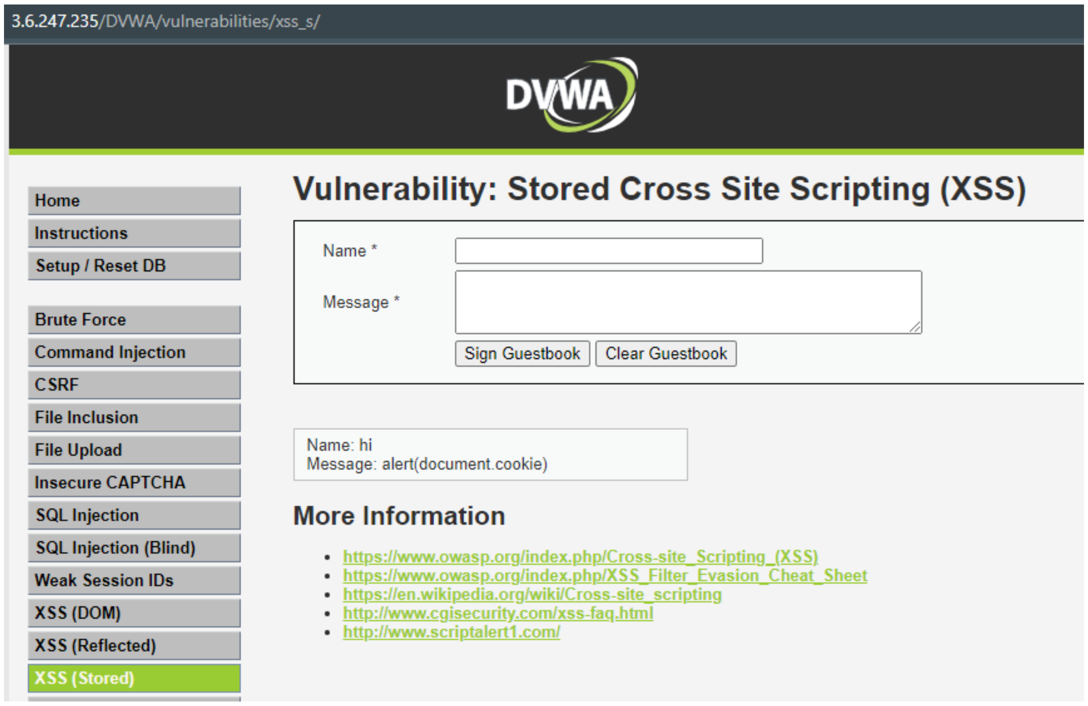
Medium Submit

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a secu

PHPIDS works by filtering any user supplied input agains
DVWA to serve as a live example of how Web Applicatio

No. In medium security level entering the same input does not render the session id.


3.6.247.235/DVWA/vulnerabilities/xss_s/

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

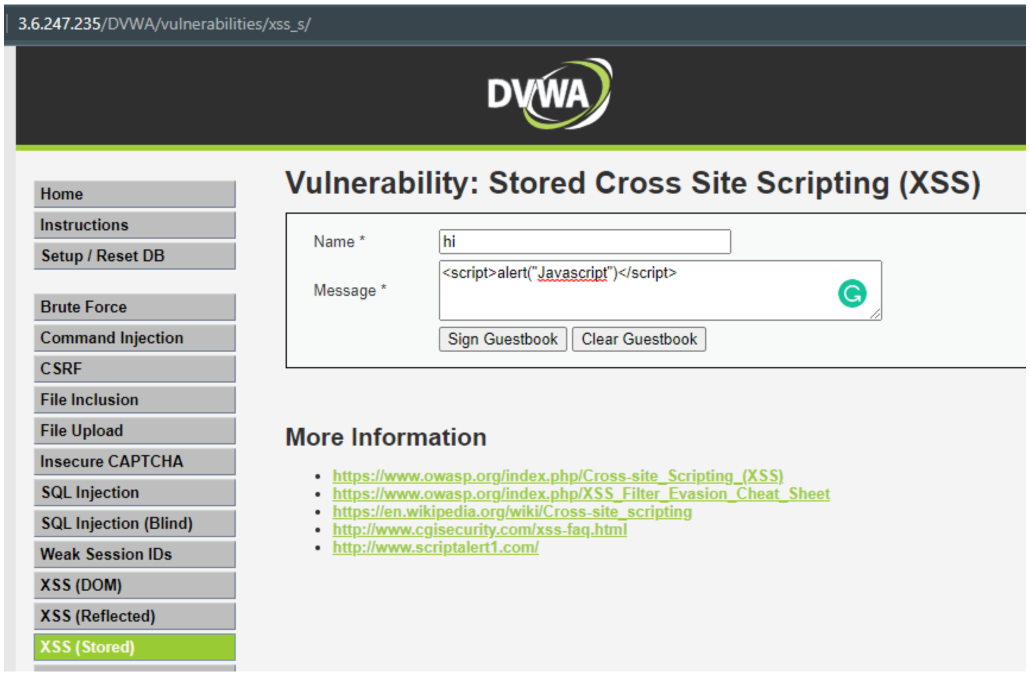
Message *

Name: hi
Message: alert(document.cookie)

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

3. Set the security mode to "Low" and examine the code that is vulnerable, and then set the security mode to "High" and reexamine the same code. What changed? How do the changes prevent attack from succeeding?
In low security level <script>alert("Javascript")</script> renders alert message on the browser as below:


3.6.247.235/DVWA/vulnerabilities/xss_s/

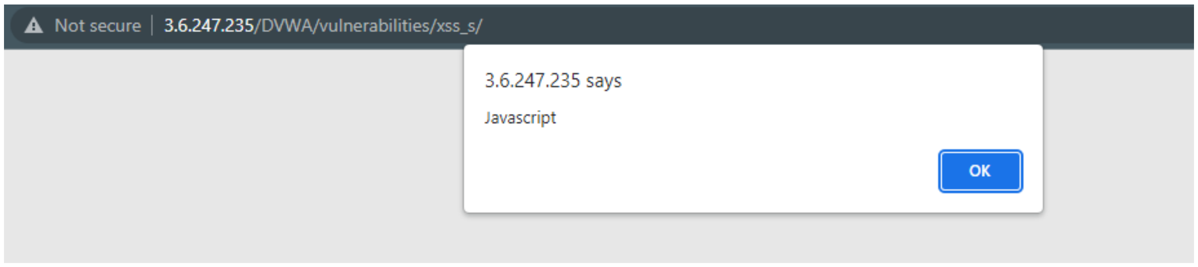
Vulnerability: Stored Cross Site Scripting (XSS)

Name * hi

Message * <script>alert("Javascript")</script>

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

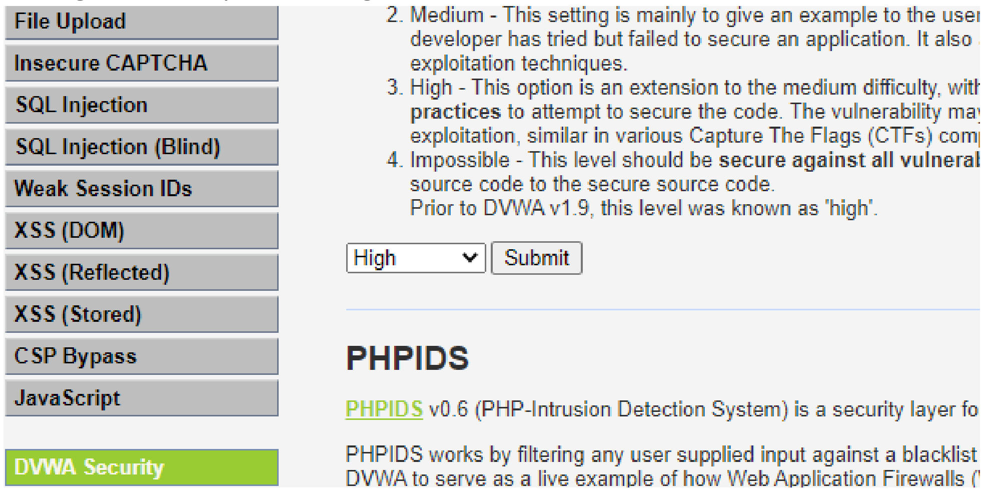


3.6.247.235 says

Javascript

On refreshing the page, same alert message gets displayed because XSS playload is stored in the GuestBook.

Setting the security level to High:



2. Medium - This setting is mainly to give an example to the user developer has tried but failed to secure an application. It also exploitation techniques.
3. High - This option is an extension to the medium difficulty, with **practices** to attempt to secure the code. The vulnerability may exploitation, similar in various Capture The Flags (CTFs) comp
4. Impossible - This level should be **secure against all vulnerat** source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

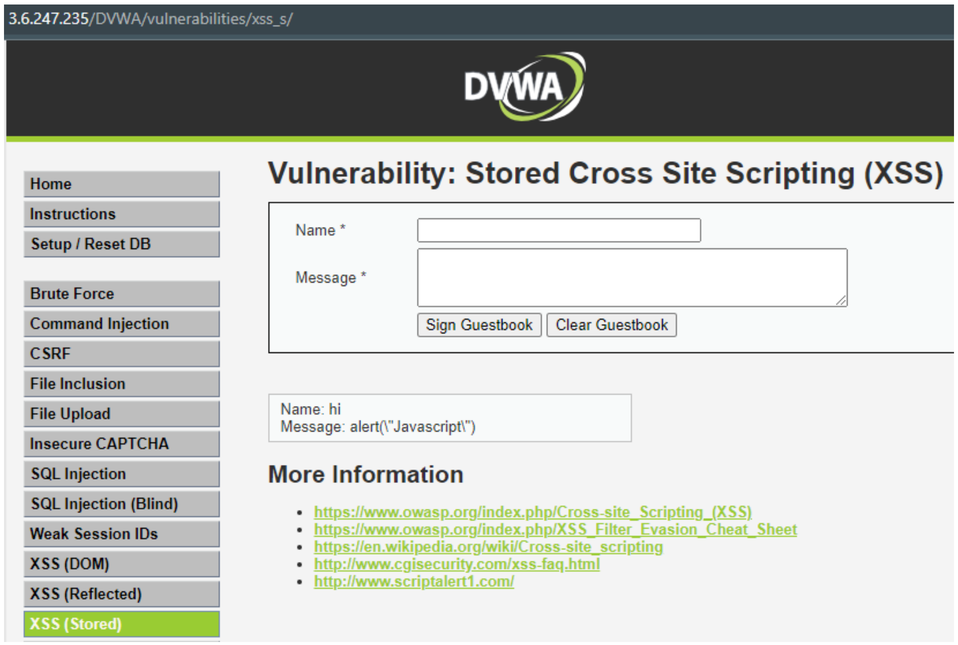
High

PHPIDS

PHPIDS v0.6 (PHP-Intrusion Detection System) is a security layer fo

PHPIDS works by filtering any user supplied input against a blacklist DVWA to serve as a live example of how Web Application Firewalls ('

Entering the same input does not render alert message.



Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: hi
Message: alert(\"Javascript\")

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

There may be chances that the backend code is performing input sanitization or html encoding on message field when it is accepting user’s input. Looking at the server code it is clear that the message field is using two levels of sanitization. Firstly, strip_tags() is being used. This function removes all the HTML tags from the message field and even if some text contains quotes or bad character passes through this function, htmlspecialchars() will definitely encode into equivalent html character. So XSS payload becomes useless when the security level is set to high. Hence, message field is completely secure, and we cannot inject any XSS payload into it. Also, above screenshot shows that the strip_tags() function has removed the script tag from the message and we see only alert() text in the response.