TESTPHP.VULNWEB.COMVAPT REPORT

short line

SANJAY S KUMAR

# 

| **SI NO** | **Vulnerability** |
| --- | --- |
| **1** | **XSS** |
| **2** | **Command Injection** |
| **3** | **SSRF** |
| **4** | **file path manipulation** |
| **5** | **HTTP parameter Pollution** |

# 

# **1.XSS**

## **1.1 Description**

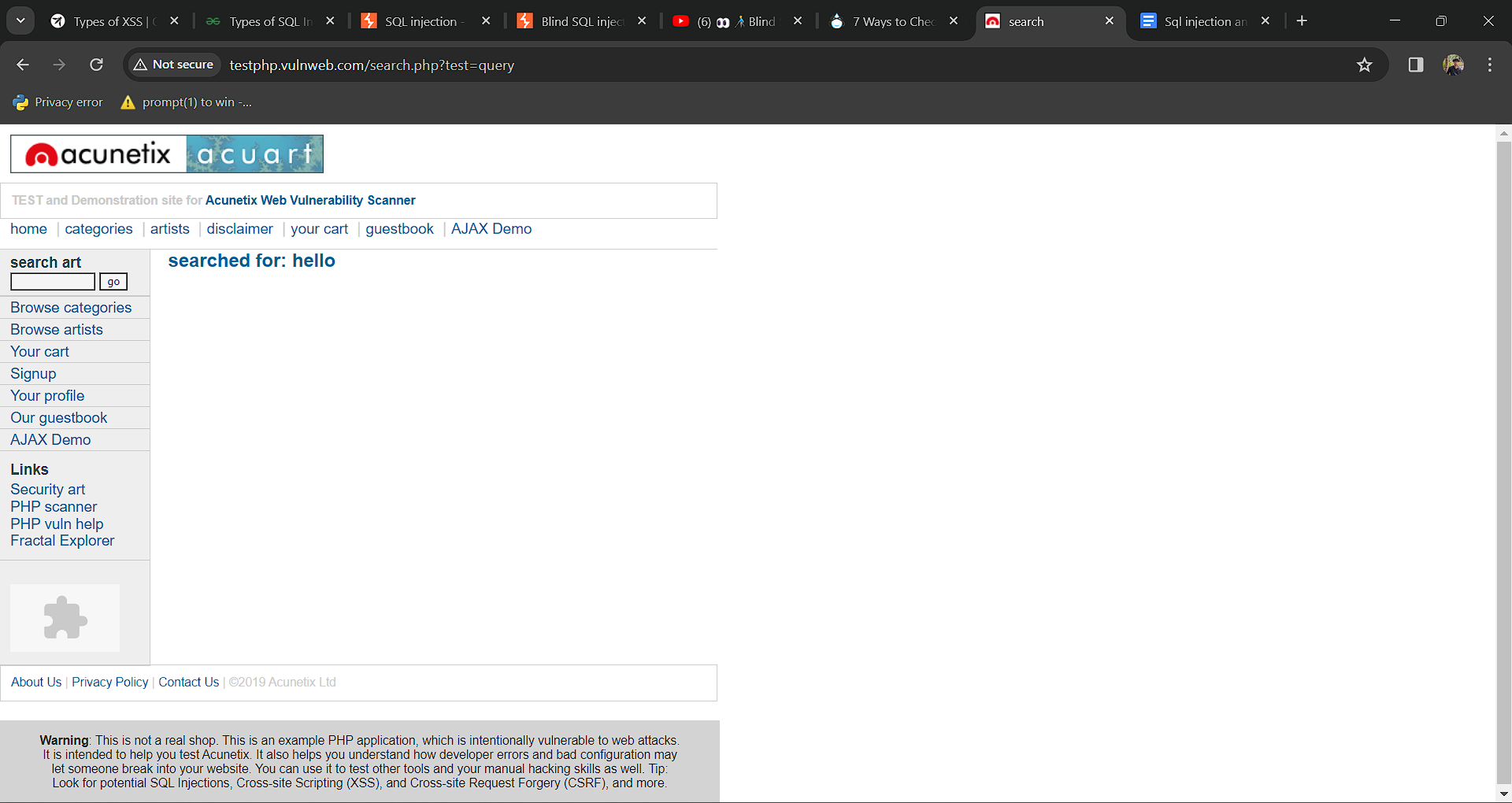
In this web application, there exists an input search box where any entries added therein are promptly displayed on the page. We could potentially exploit this vulnerability by injecting a payload into said input box.

## **1.2 Vulnerable instance**

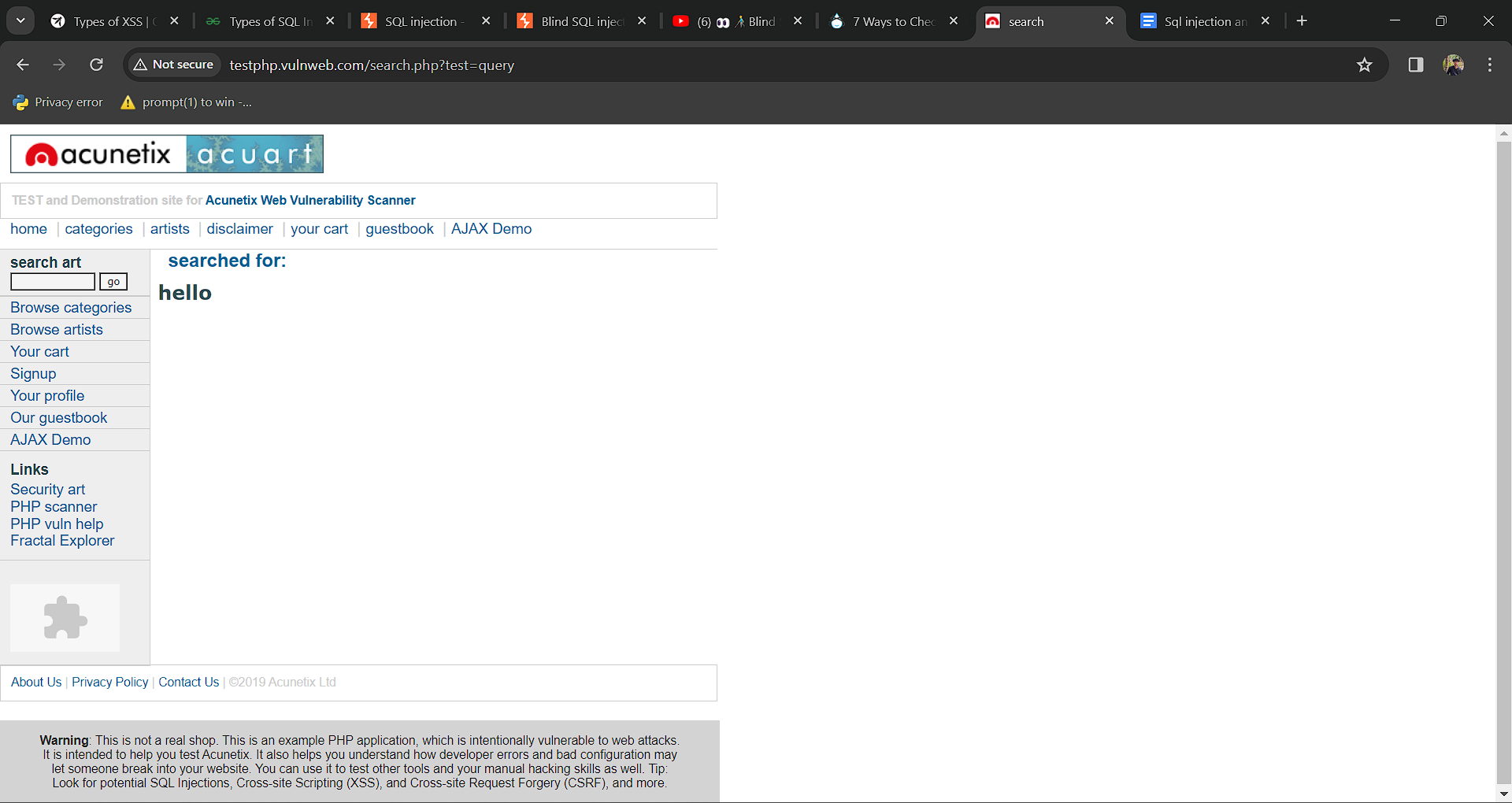
http://testphp.vulnweb.com/index.php

## **1.3 Proof of concept.**

When we input anything into the box, we can observe that all items are shown on this page.

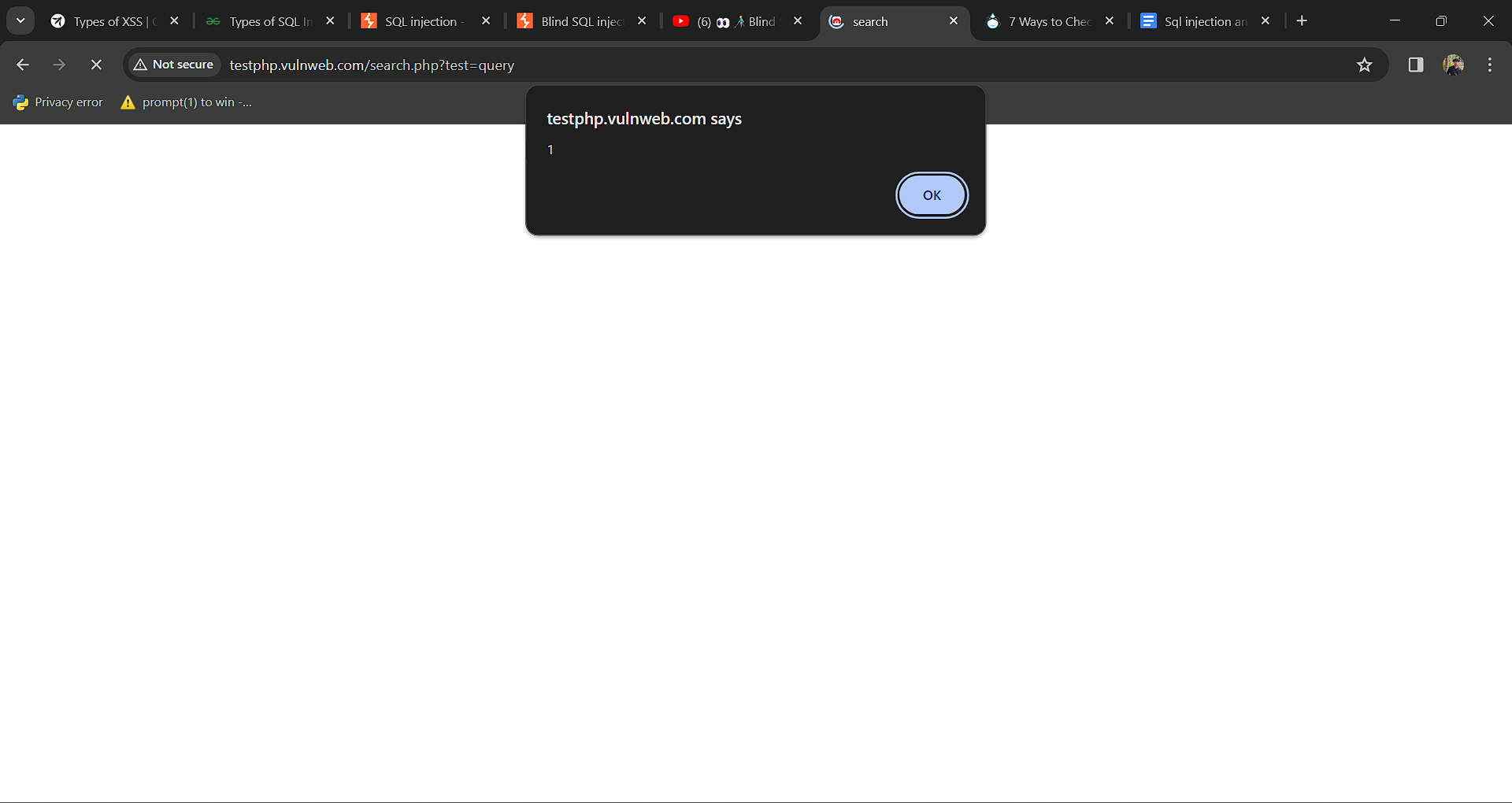


Before determining if the input box is vulnerable, our initial step is to assess its susceptibility by testing with a basic HTML <h1> tag containing "hello".



We can see that the html tag works here.

The HTML tag functions properly, allowing us to insert any payload into the input box and verify its functionality, such as: <script>alert(1)</script>.



## **1.5 Mitigation.**

**Input Validation and Sanitization:** Implement strict input validation and sanitization mechanisms to ensure that user-supplied input is free from malicious content. This includes validating input format, length, and type, as well as filtering out potentially dangerous characters or commands.

**Least Privilege Principle:** Follow the principle of least privilege by granting only the minimum necessary permissions to database users and restricting access to sensitive database operations and resources.

# **2.Command Injection**

## **2.1 Description**

The web app features a session for choosing an artist, within which a vulnerability exists. Through this vulnerability, it's possible to access the database by injecting specific payloads.

So an attacker can access the database by exploiting this vulnerability, making it a critical issue.

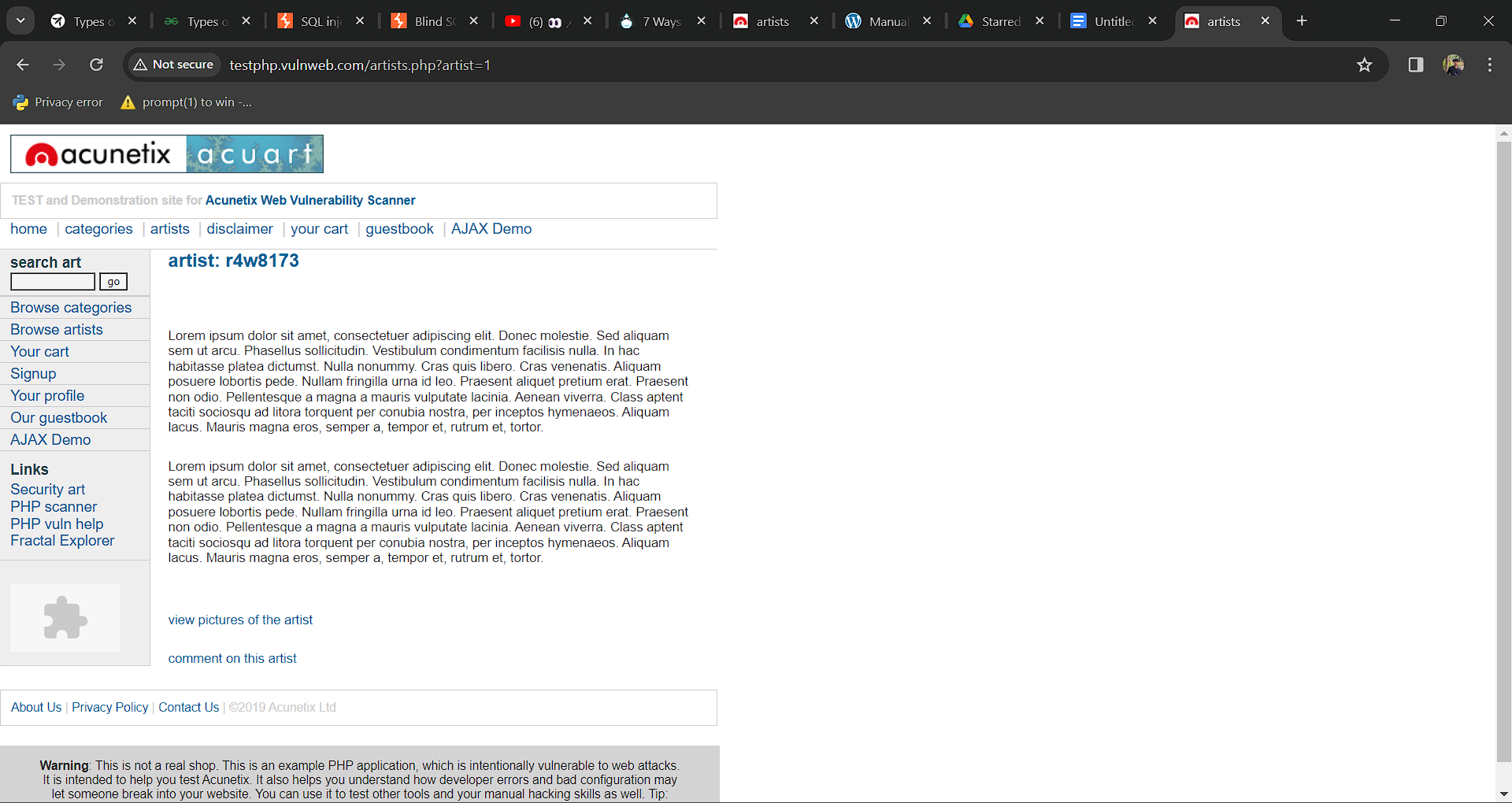
## **2.2 Vulnerable instance**

<http://testphp.vulnweb.com/artists.php?artist=1>

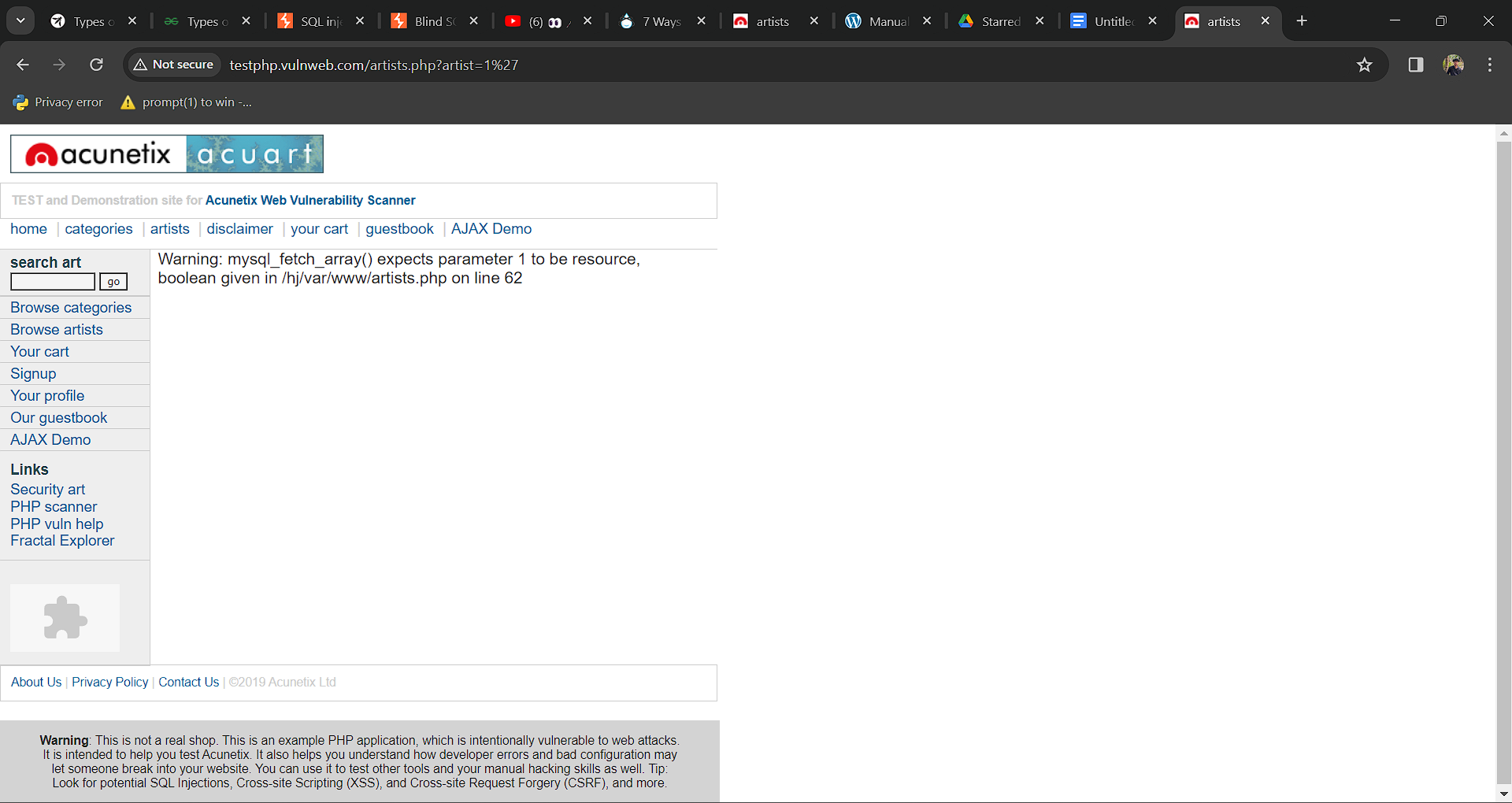
## **2.3 Proof of concept.**

First we want to access a category in the website : [testphp.vulnweb.com/artists.php?artist=1](http://testphp.vulnweb.com/artists.php?artist=1)”

we are going test SQL injection for “**id=1″**



By adding an apostrophe (**‘**) symbol at the end of input which will try to break the query. That time we can find the website is sql vulnerable, this is an error based vulnerability.



Using ORDER BY payload to find how many columns are there in the table.

http://testphp.vulnweb.com/artists.php?artist=1 order by 1

Then

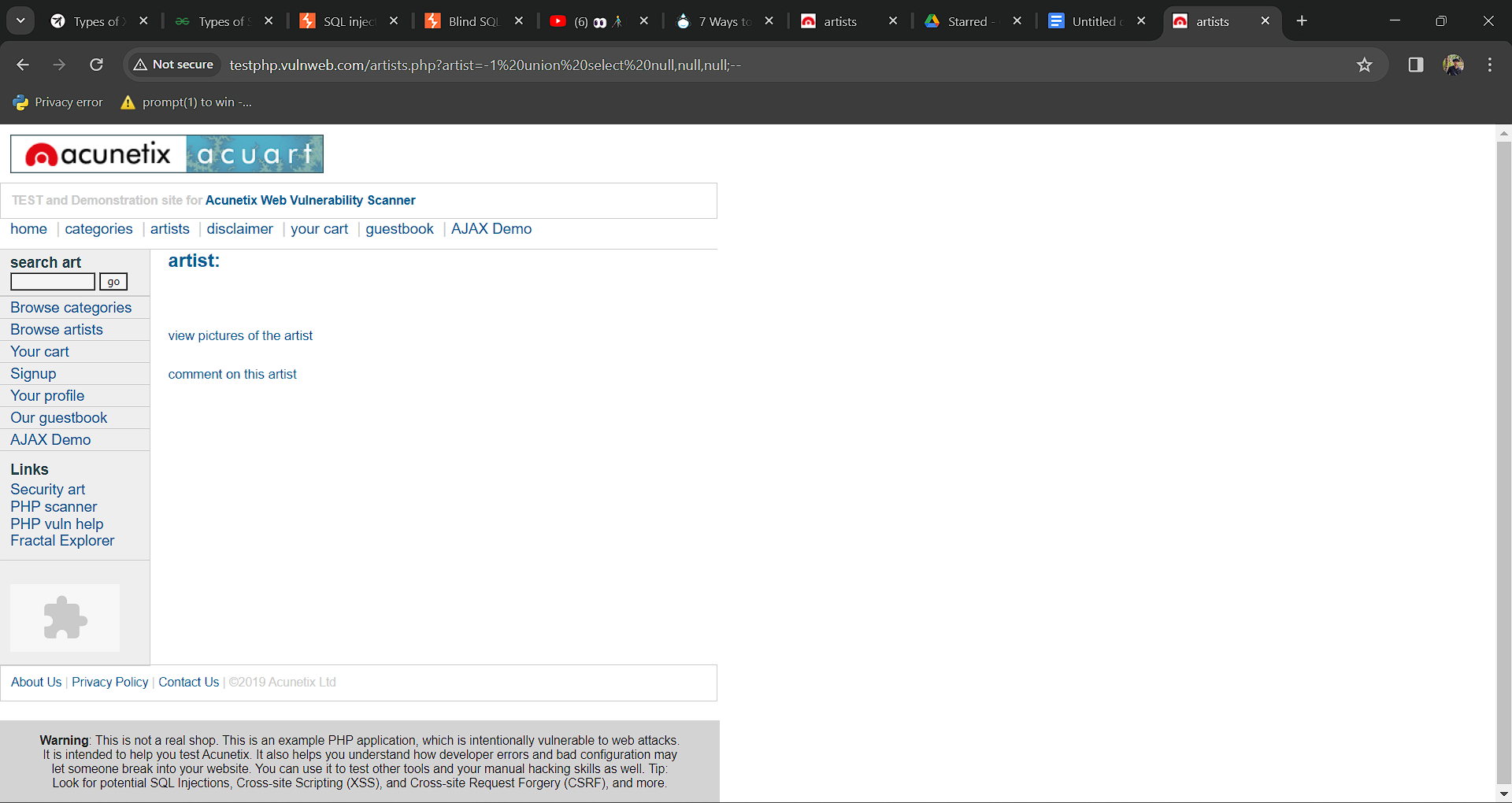
http://testphp.vulnweb.com/artists.php?artist=1 order by 4

At this time there is an error form so we find that there are 3 columns.

Then

Use union to find the data form the table:

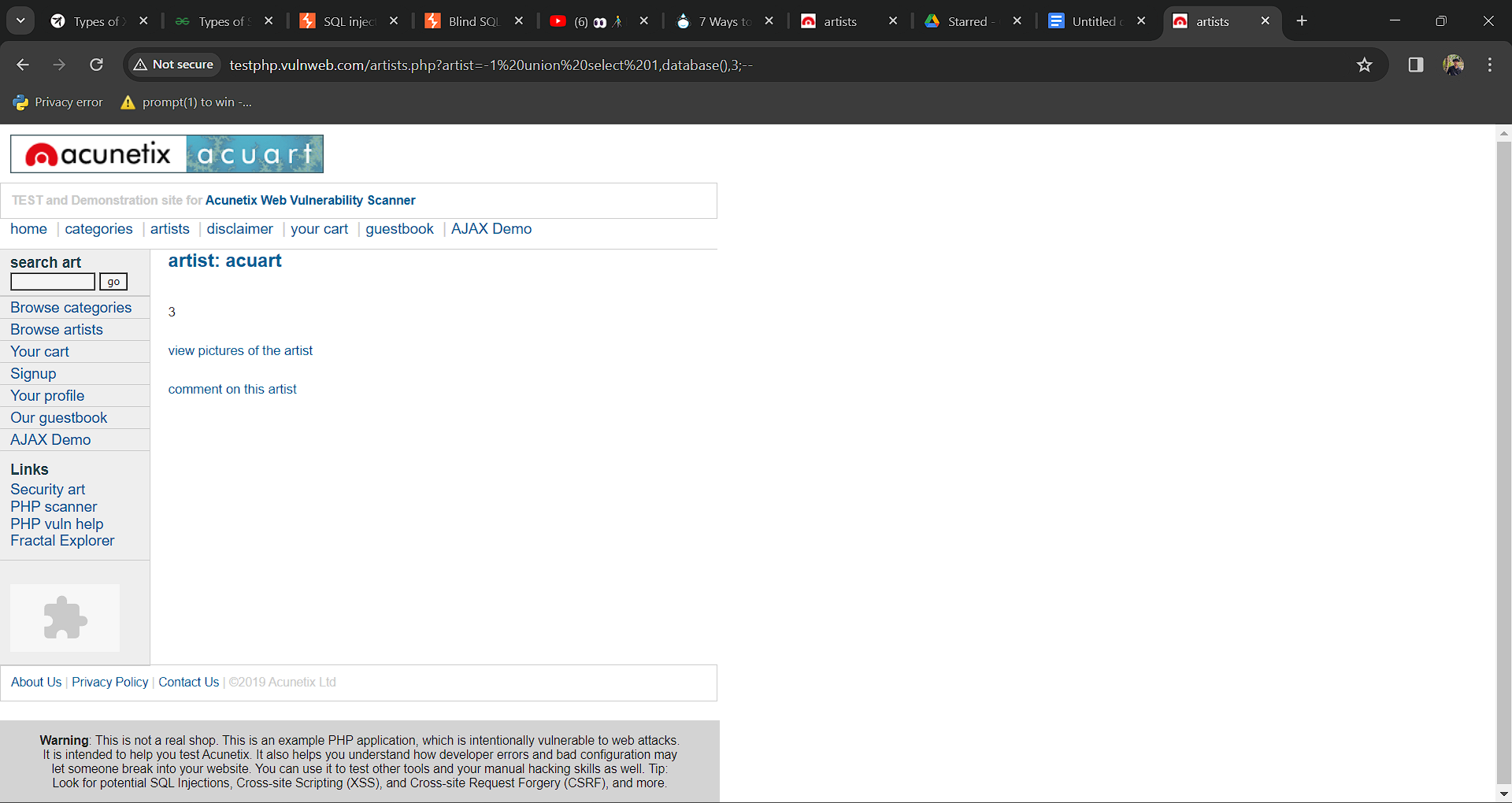
http://testphp.vulnweb.com/artists.php?artist=-1 union select null,null,null;--



Using this union query to find the name of database

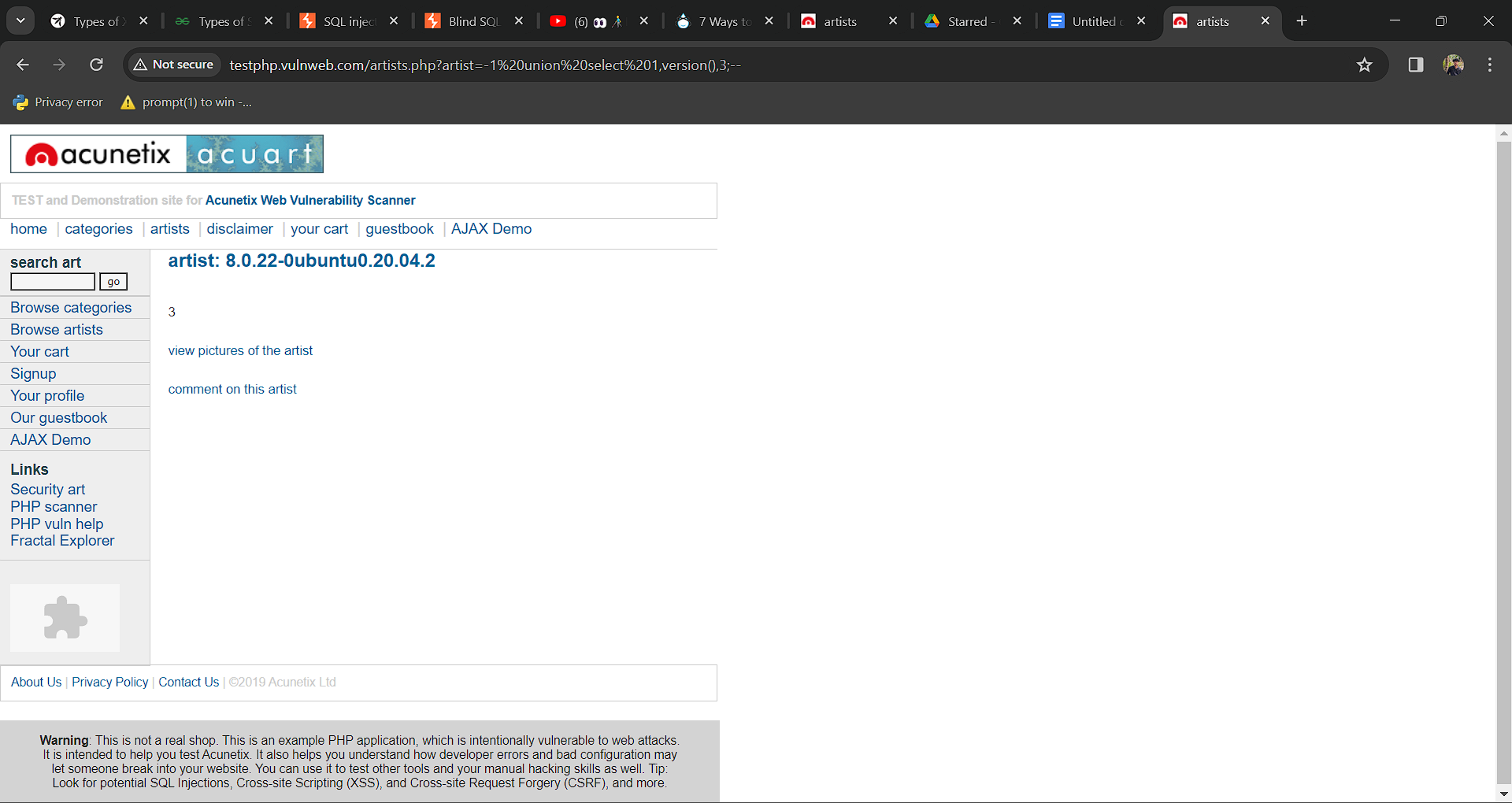
http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,database(),3;--

We get the database name is **acuart.**

****

Using this union query to find the name of version

http://testphp.vulnweb.com/artists.php?artist=-1 union select 1,version(),3;--

The version is **5.1.73 0ubuntu0 10.04.1.**

## **2.5 Mitigation**

**Patch or Update:** Ensure that all software and systems involved, including the database management system (DBMS), are up to date with the latest security patches and updates.

**Access Control:** Implement strict access control measures to limit who can access the database. Use strong authentication methods such as multi-factor authentication (MFA) and ensure that users have the minimum necessary privileges required to perform their tasks. Regularly review and revoke unnecessary privileges.

# **3.file path manipulation**

## **3.1 Description**

On this website, files are revealed upon request, making it easy for an attacker to manipulate the file name and gain access to alternative files.

## **3.2 Vulnerable instance**

[http://testphp.vulnweb.com/showimage.php?file=](http://testphp.vulnweb.com/showimage.php?file=/../../../../../../../../../../)

## **3.4 Mitigation**

**File Validation:** Implement validation to ensure that file requests are legitimate and authorized. This includes validating file names, extensions, and paths to prevent attackers from manipulating them to access unauthorized files.

**Access Controls:** Implement access controls to restrict access to sensitive files based on user roles and permissions.

## 

# **4.http parameter pollution**

# **4.1 Description**

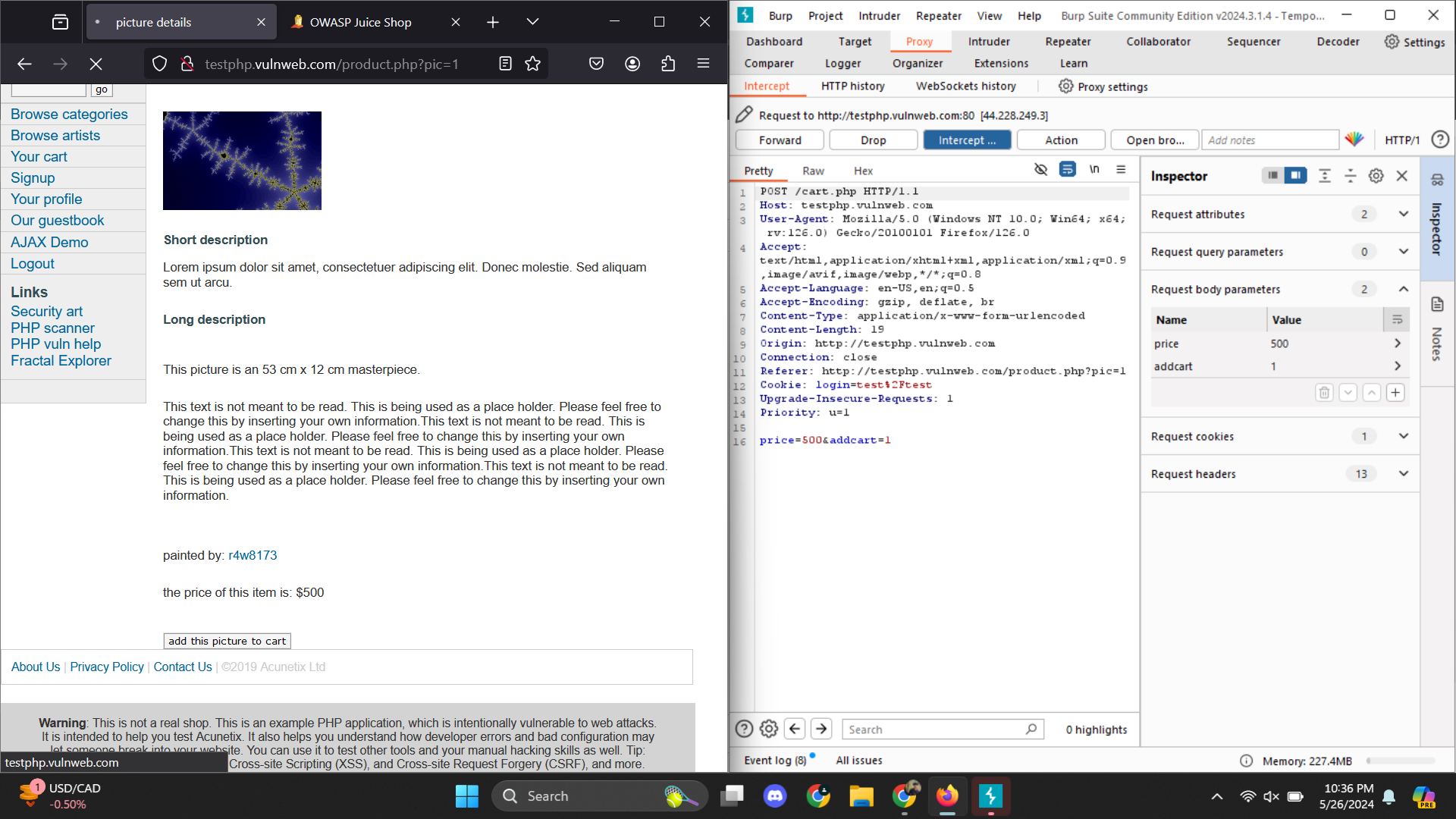
In the web application there is an option to add products to cart .In this option there is a vulnerability. When we add items in the cart , in the request the price and id are passed in plain text .So we can edit the price to 0 .

## **4.2 Vulnerable instance**

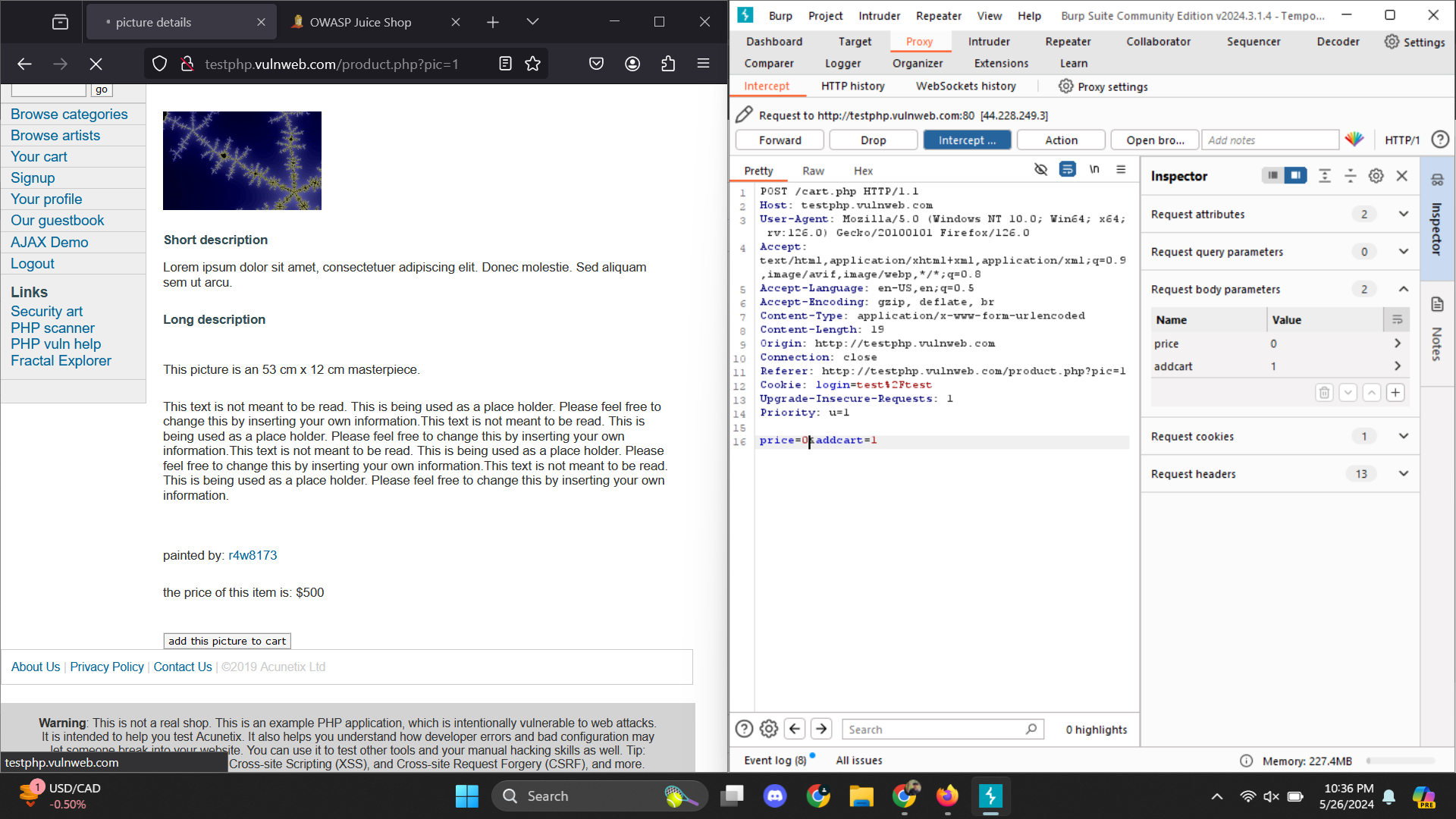
http://testphp.vulnweb.com/cart.php

## **4.3 Proof of concept.**

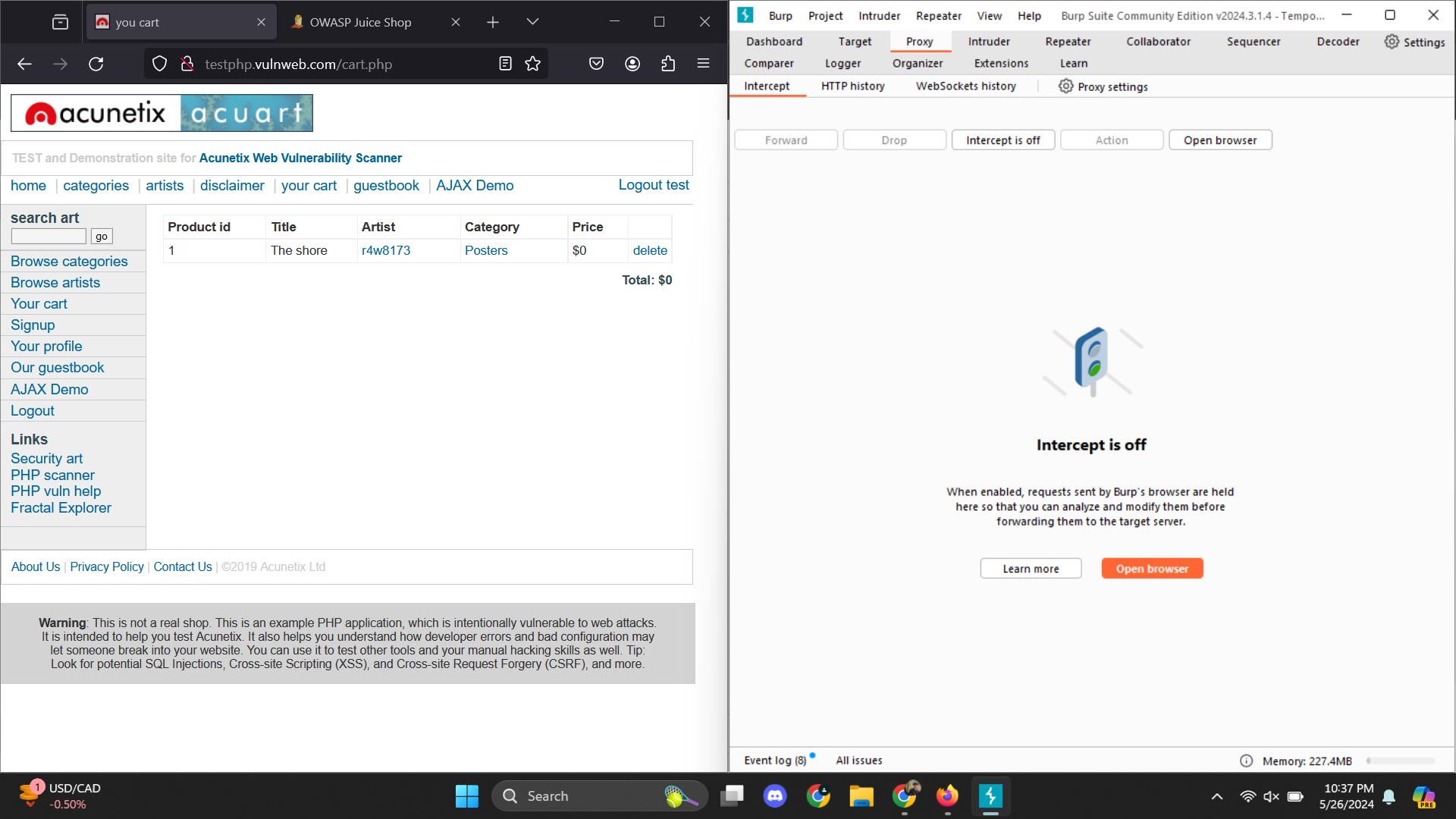
We can see that the id and price are in plain text.



We can edit the price to 0



In the cart we can see the price is zero in the cart.



## **4.4 Mitigation**

**Tokenization:** Generate unique tokens for each session/cart encapsulating price and product ID.Decode tokens on the server to retrieve original values.

**Input Validation and Sanitization:** Strictly validate and sanitize all incoming data to adhere to expected formats and values.