Web Application and API penetrationtest – Shopmore webapp

Penetration Test Report

Baisangur Dudayev 2CC01
THOMAS MORE

Contents

1 OVERVIEW

1.1 Time scope

Initial assessment (05-10-2023 - 17-12-2023):

- Executed by Baisangur Dudayev

1.2 Material scope

- Shopmore.com
  - 10.0.2.100

1.3 Document version

Version	Date	Author	Comments
v1.0	16-dec-2023	Baisangur Dudayev	First release to client

2 MANAGEMENT SUMMARY

The risk classifications assigned for all discovered vulnerabilities are as follows

CRITICAL	HIGH	MEDIUM	LOW	INFO
1 finding	1 finding	No findings	No findings	No findings

2.1 Context

Thomas More has enlisted the services of Baisangur to conduct a comprehensive web application penetration test during the initial assessment period from 5-Oct-2023 to 17-Dec-2023. The primary goal of this examination was to uncover and address any security vulnerabilities present in Thomas More's "Shop More," a web application.

This report provides Thomas More with a comprehensive overview of all identified vulnerabilities, accompanied by recommended measures to enhance and optimize the security of the application components.

2.2 Observations

2.2.1 Positive Observations

It is crucial to highlight the absence of SQL injection attacks within the application. My thorough examination revealed no indications of vulnerabilities related to SQL injection, reflecting a robust defense against this specific threat.

2.2.2 Negative Observations

In the course of my security evaluation, several concerning observations have come to light regarding the site's susceptibility to common attack methods. One notable issue is the general lack of common sanitization methods. The absence of stringent input validation and output sanitization mechanisms raises concerns about the site's resilience to various forms of injection attacks.

Despite the positive aspects highlighted earlier, my security assessment has brought to light two critical vulnerabilities. The most severe issue identified is a vulnerability within the application that exposes local system files on the server using OS command injection, posing a significant risk.

Additionally, I uncovered a Cross-Site Scripting (XSS) vulnerability, posing potential threats to end-users of the application. This vulnerability creates a scenario where a simple click on a hyperlink in an email could compromise their accounts. The deceptive nature of the URL, which seemingly points to the application server (ShopMore.com), increases the likelihood of users falling victim to the attack due to the perceived trustworthiness of the domain.

2.3 Conclusion

The application and its associated API exhibit a security maturity level that needs improvement in various critical aspects. Despite this, the vulnerabilities highlighted in this report pose significant risks that demand immediate attention, especially considering the system's current production phase with active client usage. I strongly recommend that Thomas More thoroughly investigate server logs to ascertain whether the vulnerabilities outlined in this report have been exploited previously.

To ensure ongoing security, I advocate for regular security assessments in the future to uphold the system's integrity and fortify its security measures. The implementation of a robust and secure development life cycle is paramount to proactively prevent the recurrence of such vulnerabilities in subsequent development iterations.

In summary, my evaluation leads me to the conclusion that, at present, Thomas More's "ShopMore" lacks the essential security controls needed to provide confidence in resilience against cyber-attacks.

3 LIST OF FINDINGS

Title	Status	Severity	Page
Export page vulnerable to OS command injection	Unresolved	Critical	7
Contact form vulnerable to Cross Site Scripting	Unresolved	High	10

3.1 OS command injection

CRITICAL

Description

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

Risk

An OS Command Injection vulnerability poses a significant threat by allowing an attacker to execute arbitrary commands on the underlying operating system. This could potentially lead to unauthorized access, data exfiltration, and the compromise of sensitive information. For instance, an attacker exploiting this vulnerability might gain the ability to execute commands that reveal or manipulate data, bypass security measures, or escalate privileges.

Exploitation of an OS Command Injection vulnerability could result in the compromise of the entire system, allowing the attacker to execute commands with the same level of privileges as the compromised process or service. This, in turn, could lead to unauthorized access to sensitive data, disruption of services, or even complete control over the affected system.

Attackers may leverage OS Command Injection to perform various malicious activities, such as running commands to retrieve sensitive files, execute arbitrary code, or manipulate the system's configuration. This could have severe consequences, including unauthorized access to confidential information, disruption of critical services, or the installation of malware.

Furthermore, the impact of an OS Command Injection vulnerability extends beyond the compromised system. An attacker could potentially use this vulnerability to pivot to other systems within the network, escalating the scope of the attack and compromising additional resources.

Addressing OS Command Injection vulnerabilities is crucial for maintaining the security and integrity of the system. Implementing proper input validation, using secure coding practices, and restricting the execution of arbitrary commands are essential measures to mitigate the risks associated with this type of vulnerability.

Recommendation

To mitigate the risk associated with an OS Command Injection vulnerability, it is imperative to implement robust measures for input validation and sanitization, particularly when handling user input destined for backend requests. This is especially crucial when the input may include resource identifiers like URIs or filenames. Effective strategies for mitigating this risk include:

1. **Input Validation:** Enforce stringent input validation mechanisms by implementing whitelists of allowed URLs or filenames. This ensures that only permissible and expected input is accepted, reducing the likelihood of unauthorized commands being executed.
2. **Regular Expressions:** Utilize regular expressions to validate input formats, ensuring that user-supplied data adheres to predefined patterns. This helps

prevent the injection of malicious commands by validating the input against expected structures.

3. **Network Segmentation:** Employ network segmentation practices to restrict external access to internal systems and resources. This measure is particularly beneficial when the connection is not deemed necessary from a business perspective. By segmenting the network, potential attackers are hindered from reaching critical internal components, limiting the scope and impact of the vulnerability.

These mitigation strategies collectively contribute to a more robust security posture, reducing the risk of OS Command Injection vulnerabilities and fortifying the resilience of the system against malicious exploits. Regular security assessments and monitoring are essential components of a proactive security approach to identify and address potential vulnerabilities promptly.

Proof

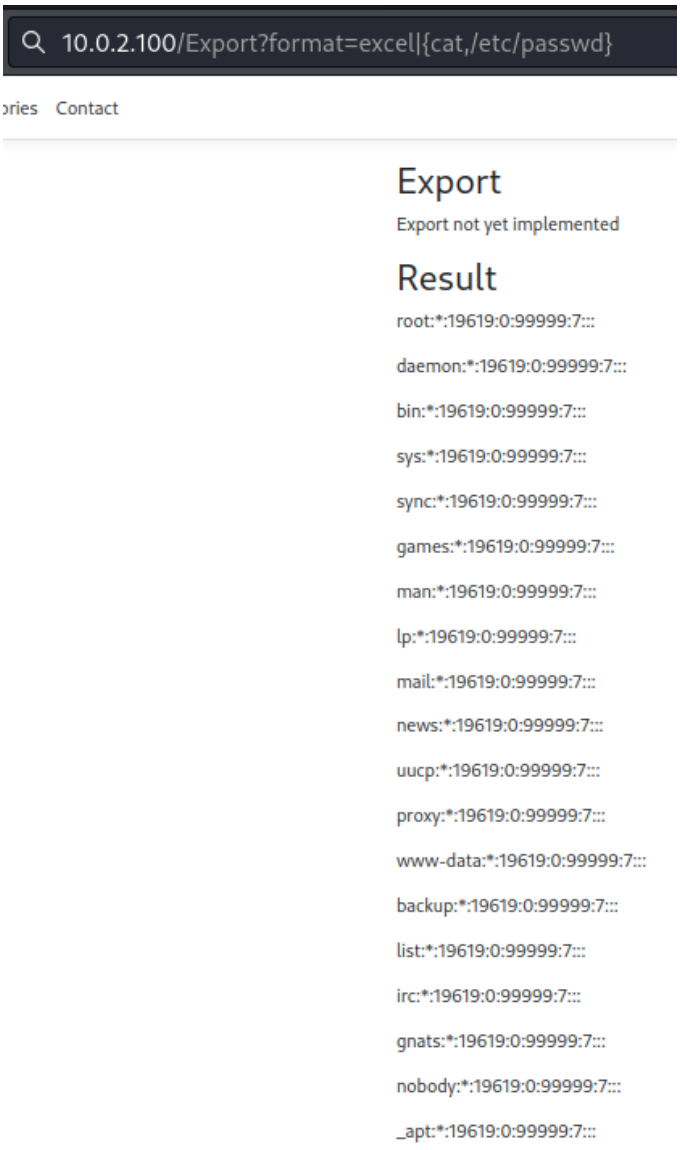
In this passage, I showcase how to confirm this finding. Keep in mind, however, that this is not designed to be a complete overview of every instance of this finding. As the owner of the application, it is your duty to delve into and pinpoint structural measures for all instances of this concern throughout the application.

The Export page of the website allows any user of the platform to inject bash command in the url giving them read & write access to all the files within the webserver. The screenshot below shows me gaining access to the/etc/passwd file which only root or users with sudo rights should have access to:

By editing the URL to include:
`http://10.0.2.100/Export?format=excel||{cat,/etc/passwd}`

I demonstrate that I have access to the file system of the web application as user root.

The screenshot provided illustrates the payload's successful execution.


10.0.2.100/Export?format=excel|{cat,/etc/passwd}
ories Contact
Export
Export not yet implemented
Result
root:*:19619:0:99999:7:::
daemon:*:19619:0:99999:7:::
bin:*:19619:0:99999:7:::
sys:*:19619:0:99999:7:::
sync:*:19619:0:99999:7:::
games:*:19619:0:99999:7:::
man:*:19619:0:99999:7:::
lp:*:19619:0:99999:7:::
mail:*:19619:0:99999:7:::
news:*:19619:0:99999:7:::
uucp:*:19619:0:99999:7:::
proxy:*:19619:0:99999:7:::
www-data:*:19619:0:99999:7:::
backup:*:19619:0:99999:7:::
list:*:19619:0:99999:7:::
irc:*:19619:0:99999:7:::
gnats:*:19619:0:99999:7:::
nobody:*:19619:0:99999:7:::
_apt:*:19619:0:99999:7:::

3.2 Stored Cross Site Scripting

HIGH

Description

CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:H

Risk

A Cross-Site Scripting (XSS) vulnerability poses a considerable risk as it allows attackers to inject malicious scripts into web pages viewed by other users. By exploiting this vulnerability, an attacker could potentially compromise the security of users' accounts, steal sensitive information, and manipulate the content displayed to victims.

In the context of a successful XSS attack, the attacker may gain unauthorized access to user sessions, leading to the compromise of confidential data. This could include personal information, login credentials, or any other sensitive details entered by users on the affected web application.

The exploitation of an XSS vulnerability enables attackers to execute scripts within the context of a user's browser, allowing them to perform actions on behalf of the user without their consent. This could range from redirecting users to malicious websites to capturing keystrokes and other sensitive information.

The impact of an XSS vulnerability extends beyond individual users, affecting the overall trustworthiness of the web application. Successful exploitation could lead to reputational damage, loss of user trust, and potential legal consequences if sensitive data is compromised.

Recommendation

Input Validation and Sanitization: Implement robust input validation mechanisms to ensure that user inputs are thoroughly validated and sanitized, preventing the injection of malicious scripts.

Proof

In this paragraph, we demonstrate how to validate this finding. Do bear in mind, however, that this is not meant to be an exhaustive overview of all instances of this finding. As the application owner, you have the responsibility of investigating and identifying structural mitigation of all occurrences of this issue across the application (and other, similar ones in your portfolio).

Both user inputs available on the Contact are not properly sanitiized causing them to be vulnerable to stored cross site scripting

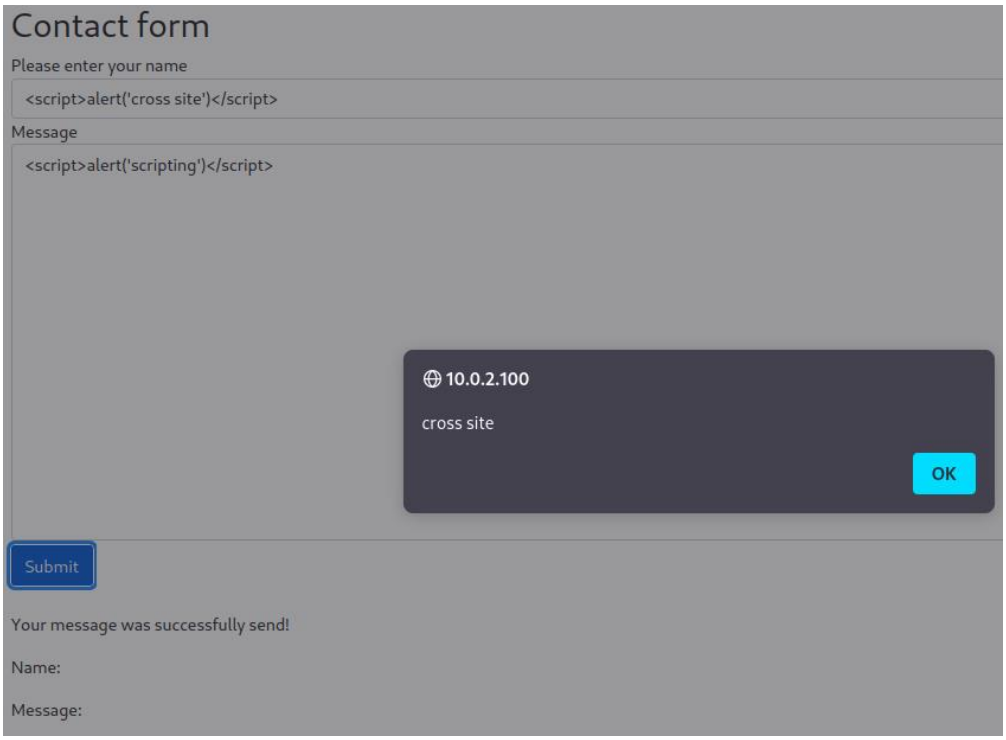
By injection the commands:

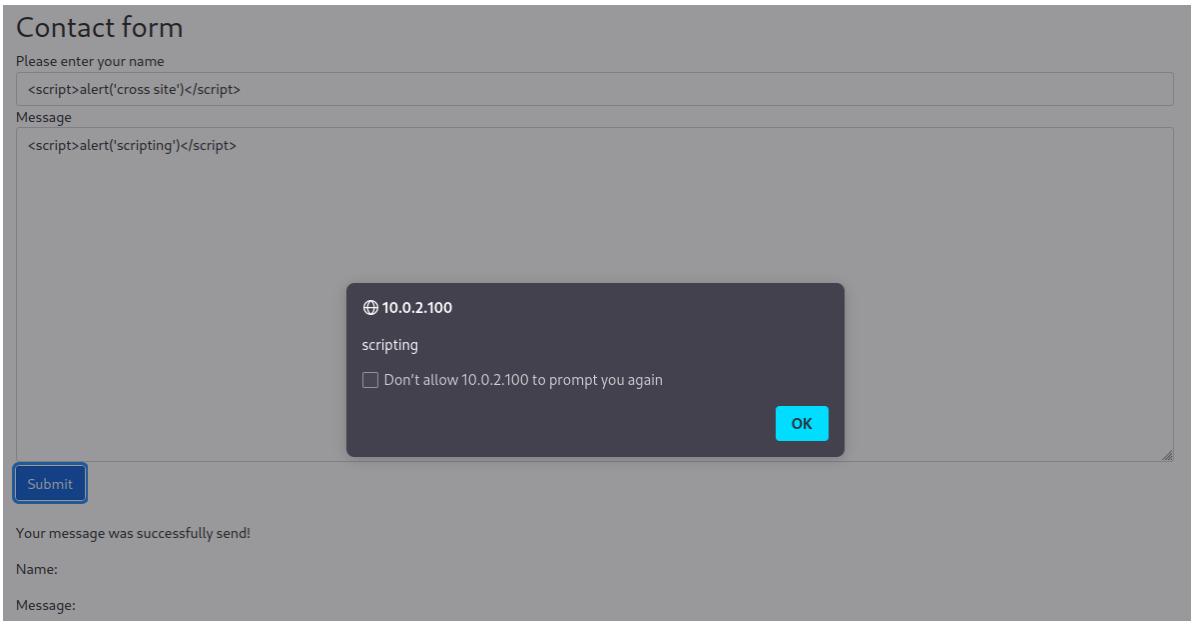
<script>alert('cross site')</script>

<script>alert('scripting')</script>

I demonstrate that both input fields on the page are vulnerable to cross site scripting.

The screenshot provided illustrates the payload's successful execution.





4 APPENDIX A: TESTCASES

The test cases outlined below underwent thorough evaluation during this penetration test. The Status column indicates the responsible party for execution in cases where the test was applicable within the current scope. Similarly, when a test was deemed not applicable, the Status column accurately reflects this determination.

Title	Description	Status
Testing for SQL Injection (WSTG-INPV-05)	- Identify potential SQL injection points within the system. Assess both the severity of the injection and the attainable level of access through exploiting this vulnerability.	Tested by Baisangur Dudayev
Testing for Command Injection (WSTG-INPV-12)	- Identify and assess the command injection points.	Tested by Baisangur Dudayev
Testing Directory Traversal File Include (WSTG-ATHZ-01)	- Identify injection points that pertain to path traversal. – Assess bypassing techniques and identify the extent of path traversal.	Tested by Baisangur Dudayev
Testing for Stored Cross Site Scripting (WSTG-INPV-02)	Identify stored input that is reflected on the client-side. - Assess the input they accept and the encoding that gets applied on return (if any).	Tested by Baisangur Dudayev
Testing for Reflected Cross Site Scripting (WSTG-INPV-01)	Identify reflected input on the client-side. Evaluate the accepted input and examine the applied encoding upon return (if any).	Tested by Baisangur Dudayev
Testing for Buffer Overflow	Identify points of access that could be user for buffer overflow attacks	Tested by Baisangur Dudayev
Testing for Stored Cross Site Scripting (WSTG-INPV-02)	Identify stored input that is reflected on the client-side. - Assess the input they accept and the encoding that gets applied on return (if any).	Tested by Baisangur Dudayev
Testing for Server-Side Request Forgery (WSTG-INPV-19)	Identify potential Server-Side Request Forgery (SSRF) injection points within the system	Tested by Baisangur Dudayev
Testing for File Upload	Identify potential file	Tested by Baisangur

Vulnerability	upload points within the application, assessing the security of the file upload functionality.	Dudayev

5 APPENDIX B: SEVERITY CLASSIFICATION

I employed version 3.1 of the Common Vulnerability Scoring System (CVSS) as a robust scoring framework to evaluate and categorize the severity of the findings in my penetration testing endeavors. This strategic utilization of CVSS not only enables a systematic assessment of vulnerabilities but also serves as a standardized metric. By employing such a well-established scoring system, I aim to provide my clients with a clear and consistent method for gauging the severity of identified vulnerabilities. This, in turn, aids them in prioritizing their remediation efforts effectively, ensuring a focused and efficient approach to enhancing their overall security posture.

5.1 Critical

Characteristics

- The vulnerability can be exploited with minimal expertise, posing a high risk.
- Tools and scripts for exploitation are publicly available.
- The vulnerability can be triggered without user interaction.
- The vulnerability poses a significant threat to data confidentiality.
- Exploiting the vulnerability could lead to a complete system compromise.

5.2 High

Characteristics

- The vulnerability can be exploited with minimal expertise, posing a high risk.
- Tools and scripts for exploitation are publicly available.
- The vulnerability can be triggered without or with minimal user interaction.

5.3 Medium

Characteristics

- Requires specific conditions for exploitation.
- Less likely to occur due to the need for specialized skills.
- Limited impact on business operations.
- Cannot be automated.

5.4. Low

Characteristics

- The vulnerability poses challenges for exploitation, requiring substantial expertise.
- Specialized skills are needed, making the occurrence less likely.
- Involves a complex chain of events for successful compromise.
- Limited to no significant impact on business operations.
- Dependent on specific conditions or conjunction with other vulnerabilities.
- Manual intervention is required, as the exploit cannot be easily automated.

5.5. Informational Severity

Characteristics

Findings in this category are Theoretical vulnerabilities which may have no realistic impact on confidentiality, integrity, or availability. Re-evaluation is recommended for findings lacking sufficient information during testing.

6 APPENDIX C: METHODOLOGY

My approach to conducting penetration tests involves working within the constraints of a defined timeframe, a common limitation for security assessments like penetration tests. The primary objective during this period is to uncover and address as many vulnerabilities as possible. The intricate nature of systems, varying programming languages, and diverse frameworks make it increasingly challenging to accurately estimate the time and effort required to identify the most significant risks. It remains imperative to proactively address these issues to prevent potential harm and malicious exploitation within the organization.

The assessment's quality hinges on the precision and relevance of the findings, directly influencing the credibility of the subsequent report. Achieving the discovery of all potential weaknesses is practically unattainable, and it's not always necessary. Some vulnerabilities are less critical, either due to their non-exploitable nature or their negligible impact if exploited. Conversely, others pose substantial risks and are highly relevant to the overall security landscape.

To carry out ethical hacking effectively, I've formulated an approach that facilitates the identification and prioritization of weaknesses specifically tailored to your business needs. This involves providing actionable and feasible recommendations that align with your unique environment. Utilizing a method that involves asking pertinent questions during a kick-off meeting and considering business-related factors during vulnerability assessments, I can precisely prioritize findings and assess the associated risks.

6.1 Initiation

Started As a key part of my Hacking-as-a-Service model, applications and other services that need testing go through an onboarding process. In this step, I check the business risks related to the target(s) under evaluation, making it easier for me to understand the findings during testing. I consider things like:

- The types and importance of the data being handled
- The common adversaries the customer faces
- How much the application and its users are exposed
- How crucial the solution is within the set limits

- And more. Based on the chosen protection level, this early assessment could be a quick check or a detailed analysis of the system under investigation. In this phase, I also set up the right testing methods and document them as a crucial part of the testing plan.

Getting Ready In the Project Kickoff meeting, I go through both the technical and non-technical details of the assessment. It's important for everyone involved, both technically and from a business perspective, to be there, making sure we all understand the goals. We talk about things like:

- When and where the testing will happen
- Who to contact at the customer's end and at my place
- Fine-tuning the scope of the assessment
- What needs to be done before we start testing
- How to report serious issues and signs of problems
- Deciding how much information I should have for testing (white/grey/black box)
- For every engagement, I work with the customer to look at the risks of testing on the systems we're focusing on. To make sure it doesn't cause too many problems, I suggest doing tests in environments that aren't used for real work. If needed, I also help with minimizing the impact on how available the systems are, like suggesting regular backups.

6.2 Testing

During testing, I stick to the plan we agreed upon. While I use tools to quickly spot easy problems, I believe it's essential to manually check everything. So, I double-check all findings. I've also put together detailed guides to help my ethical hacking go smoothly and thoroughly.

6.3 Reporting

Once the testing is done, I look at all the findings and put together suggestions in a final report. The report starts with a summary for people who aren't tech-savvy. To make sure the report is top-notch, I have quality checks in place. I check that all the steps were done properly, the findings are right and in the right order, and the suggestions make sense and can be acted upon.

6.4 Review

My reports aim to be clear and practical. To make sure my assessments have the most impact, I offer a meeting with the customer after the assignment. This meeting is a chance to talk about everything I found

and suggested, clearing up any questions the customer might have about boosting their organization's security.

7. CONFIDENTIALITY & LIABILITY

This document contains private information meant for you, myself, and others officially involved in the projects it talks about. It's important not to share this info with anyone else unless I say it's okay in writing. I'm giving you this document with trust, and you should only use it for your internal business needs. Selling, copying, or sharing this report with anyone else in any way is not allowed unless I give written permission.

I've done my best to make sure the info in this report is right. But some data might come from others or their software, and I am not responsible for any issues if there are mistakes in this document. Also, because a penetration test has a set time, my goal is to find as much as I can during that time. This means the test might not cover every possible issue in the IT system being tested.