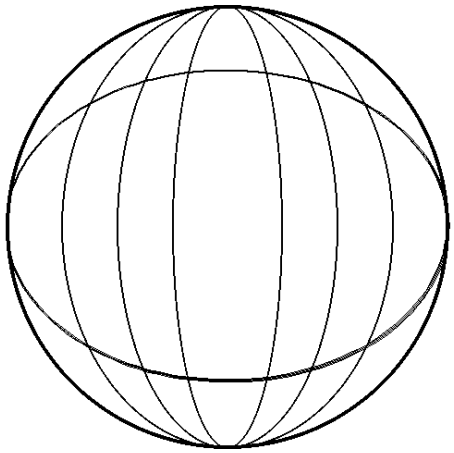
Security Audit and Testing Report for Example
5/26/2017

---CONFIDENTIAL---



www.securityandpentesting.org
auditor@securityandpentesting.org
VyperLabs™ ALL RIGHTS RESERVED

Table of Contents

Summary of Performed Operations

Project request date: 05/21/2017
Project start date: 05/22/2017
Project completion date: 05/26/2017

Project order #: 170 596
Rev: 0
Assigned pen-tester: Dexter Morgan

Hosts tested: https://www.example.com/
Testing scope: *
Exempt from testing: /admin/, /confidential/, /documents/
Testing insight level: black-box

Tests performed:

- Injection attacks (XSS and SQL injections)
- Searches for sensitive data exposure/leaks
- Brute-forcing login credentials and authentication security
- Security misconfigurations
- Access control flaws

Tests omitted (outside of customer-defined testing scope):

- Social engineering
- Zero-day exploitation
- PHP and OS command injections
- Header injections

Vulnerabilities Found By Severity and Classification

Severity:

Severity	Description	Number of found vulnerabilities	Patched
HIGH	The security flaw can allow account hijacking, backdoor access, e.t.c.	1	0/1
MEDIUM	The security flaw is difficult to take advantage, has a low likelihood of occurrence, or only has mild consequences.	1	0/1
LOW	The security flaw is mostly harmless, but should be remedied for good practice.	1	0/1

Classification:

Type	QTY Found
Code injection	1 (High)
Insecure authentication	1 (Medium)
Sensitive data exposure	1 (Low)

Technical Details of Performed Operations
(Page 1 of 6)

Simulated attack type: cross-site scripting (XSS)

URLs tested:

- https://www.example.com/login.php
- https://www.example.com/register.php
- https://www.example.com/reset.php
- https://www.example.com/account/change_password.php
- https://www.example.com/account/change_email.php
- https://www.example.com/account/send_message.php
- https://www.example.com/account/check_inbox.php

Testing process:

The simulated XSS attacks performed involved iterating through every HTTP request header, GET parameter value, POST parameter value, and HTTP request trailer and replacing them with malicious values designed to probe for an XSS vulnerability. Every such parameter value on each of the aforementioned URLs was thoroughly tested.

The parameters are modified one at a time, with only 1 modified parameter per request, and all non-edited parameters remaining as what they would normally be, designed to ensure that the server-side scripts would be unlikely to "break" before the vulnerability would be revealed.

Technical Details of Performed Operations (Page 2 of 6)

Simulated attack type: SQL injections

URLs tested:

- https://www.example.com/login.php
- https://www.example.com/register.php
- https://www.example.com/reset.php
- https://www.example.com/account/change_password.php
- https://www.example.com/account/change_email.php
- https://www.example.com/account/send_message.php
- https://www.example.com/account/check_inbox.php

Testing process:

The SQL injection tests were done with a combination of semi-automated and automated tools. Every GET and POST parameter was tested with SQL injection attack strings with an automated tool. A semi-automated tool was also used to attempt to perform SQL injection attacks on web pages that appear to perform SQL queries or commands.

Technical Details of Performed Operations (Page 3 of 6)

Simulated attack type: searches for sensitive data exposure/leaks

URLs tested:

- https://www.example.com/login.php
- https://www.example.com/register.php
- https://www.example.com/reset.php
- https://www.example.com/account/change_password.php
- https://www.example.com/account/change_email.php
- https://www.example.com/account/send_message.php
- https://www.example.com/account/check_inbox.php

Testing process:

Searches for sensitive data were done in a number of ways. The web application was brute-forced for hidden (and probably sensitive URLs), and in addition, the HTML pages were scraped via regular expressions to find sensitive information such as passwords.

Technical Details of Performed Operations
(Page 4 of 6)

Simulated attack type: Weak Authentication

URLs tested:

- https://www.example.com/login.php
- https://www.example.com/register.php
- https://www.example.com/reset.php
- https://www.example.com/account/change_password.php
- https://www.example.com/account/change_email.php

Testing process:

The login form was brute-forced using a "dictionary" of common passwords, with usernames that were scraped from the website. Password requirement security was manually evaluated. The "change password" and "change email" forms were also brute-forced to see if one could reset someone else's password or email.

Technical Details of Performed Operations (Page 5 of 6)

Simulated attack type: Security misconfigurations

URLs tested:

- https://www.example.com/login.php
- https://www.example.com/register.php
- https://www.example.com/reset.php
- https://www.example.com/account/change_password.php
- https://www.example.com/account/change_email.php

Testing process:

Cookies were examined to ensure they were secured properly (via the “HttpOnly” and “Secure” flags). Sensitive forms were also manually examined to see if they could be forged in cross-site requests (“cross-site request forgery”) or manipulated with “clickjacking”. Various other such miscellaneous tests were performed.

Technical Details of Performed Operations (Page 6 of 6)

Simulated attack type: Access control flaws

URLs tested:

- https://www.example.com/login.php
- https://www.example.com/register.php
- https://www.example.com/reset.php
- https://www.example.com/account/change_password.php
- https://www.example.com/account/change_email.php

Testing process:

The authentication processes were manually reviewed for miscellaneous design flaws that could allow someone to gain unauthorized access to other accounts, or account functionalities, e.g a web-server script relying on client-side validation before allowing it to perform a critical function, employing weakly-implemented cryptography, or predictable session-related cookies.

The account-creation process was also reviewed, as well as the security of the CAPTCHA mechanism for defending against the creation of automated bot accounts.

Technical Details of Discovered Vulnerabilities and Remediation
(Page 1 of 3)

Vulnerability type: XSS injection
Vulnerability severity: HIGH
Vulnerable page: /account/send_message.php

Potential consequences of exploitation:

A registered account is able to "inject" HTML or Javascript code into a message, which then renders or executes when the recipient reads it, in their inbox. This may allow the attacker to do things such as hijack session cookies or make requests to other sensitive functions on the web-server, such as to account/change_email.php or account/change_password.php.

Recommended remediation:

Edit the PHP code in /account/send_message.php. The "message_content" parameter should be "sanitized" before the message_content is written into the database (or however it's stored on the web-server). This means that the "<" and ">" symbols should be replaced with "<" and ">" symbols, respectively. To be safe, I recommend also encoding the following characters: <, >, &, ", ', \, and =

Technical Details of Discovered Vulnerabilities and Remediation
(Page 2 of 3)

Vulnerability type: Insecure authentication security
Vulnerability severity: Medium
Vulnerable page: /account/register.php

Potential consequences of exploitation:

The password requirement security isn't as good as it could be. Although the person who designed it was wise to require registered users to use a password that involves 1 punctuation mark and 1 number, this is still far too predictable. Almost everyone who selects a password under such requirements, will choose a password that fits the following format:

[word or name] + [punctuation] + [number]

Moreover, the "word or name" is usually something predictable or related to the purpose of the website itself. The punctuation is usually a ! or ? or a . And the number usually just starts at 1, and maybe it'll go higher if the registered user is required to update his password on a regular basis.

And I know people do this, because I used to work at a company where everyone knew each-other's passwords. And 9 out of 10 people used a password that followed that same format. This makes the passwords easy to guess if an automated tool is used, that generates such passwords programatically.

Recommended remediation:

Make it so the usernames are hidden, so someone brute-forcing the logins will need to know the usernames, as well. This will make it exponentially more difficult to guess. Or just make the passwords require that at least 1 punctuation or digit is in the middle of the string, somewhere, rather than just predictably at the end.

Technical Details of Discovered Vulnerabilities and Remediation
(Page 3 of 3)

Vulnerability type: Sensitive data exposure leak
Vulnerability severity: Low
Vulnerable page: /login.php

Potential consequences of exploitation:

Someone may discover the email address of the web-developer who worked on the login page. It could potentially be then used for social engineering.

Recommended remediation:

The email address walterwhite@yahoo.com was found in the comments of the login page, in a context that implies this email address belongs to a web developer. Just remove it from the comments.

Glossary

Vulnerability ranking:

Low: A security weakness which, by itself, is generally hard to exploit or relatively harmless. However, such flaws should not be taken lightly or ignored, lest a skilled adversary finds a way to use it in conjunction with other flaws to pull off something more malicious. Examples include: emails or internal IP address leaks, CAPTCHAs which are easy to programatically solve (allowing for the creation of bots, if an adversary is dedicated enough to doing so), or the HTTP TRACE header being enabled on the server.

Medium: A security flaw that can be exploited, either resulting in medium-severity consequences, or it could just be that the flaw itself is relatively hard to take advantage of. A vulnerability that has a low likelihood of occurring or has a bearable impact.

High: Any vulnerability that can be exploited to gain access to a functionality they shouldn't have access to, unless the functionality itself is trivial or harmless. If the worst a person can do with the vulnerability is pull off a harmless prank, it will not be rated as "high" severity. This rating is reserved for things that have a reasonable potential to be exploited and also high impact.

Vulnerability classifications:

Clickjacking:
https://en.wikipedia.org/wiki/Clickjacking

Command injection:
https://owasp.org/www-community/attacks/Command_Injection

CSRF (cross-site request forgery):
https://en.wikipedia.org/wiki/Cross-site_request_forgery

SQL injections:
https://en.wikipedia.org/wiki/SQL_injection

XSS (Cross-site scripting):
https://en.wikipedia.org/wiki/Cross-site_scripting