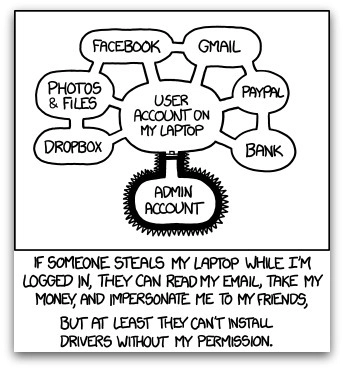
## horizontal line



IamAVulnerable Corporation Penetration Test

04.05.2017 Performed by Awesome People Dat Pen Test Company

**─**

Alexander Cannell

Awesome People Dat Pen Test Company

1337 West 3701 East

Cedar City, UT 84720/E: [alexandercannell@suu.edu](mailto:alexandercannell@suu.edu)/P: 867.5309

Table of Contents

[Document Revision History 1](#_Toc479459117)

[Executive Summary 1](#_Toc479459118)

[Summary Recommendations 1](#_Toc479459119)

[Narrative 2](#_Toc479459120)

[Findings 2](#_Toc479459121)

A0001……………………………………………………………………………………………………………………….3

A0002……………………………………………………………………………………………………………………….4

A0003……………………………………………………………………………………………………………………….6

A0004……………………………………………………………………………………………………………………….7

# Document Revision History

|  |  |  |
| --- | --- | --- |
| **Author** | **Version** | **Date** |
| Alexander Cannell | 1.0 | 03/27/2017 |
| Alexander Cannell | 2.0 | 04/05/2017 |

# 

# Executive Summary

On March 27th 2017 the Awesome People Dat Pen Test Company performed a penetration test on the IamAVulnerable Corporation’s domain with the company's permission. We performed the penetration test to determine whether or not the IamAVulnerable Corporation was vulnerable to attack. The goal was to exfiltrate as much data as possible.

Initially the IamAVulnerable domain presents several protections on the exterior of the domain. There was several vulnerabilities that we found on the IamAVulnerable domain. Listed below. SQL Injection, File Injection, XSS, and Command Injection are some of the vulnerabilities which we found. All of which are listed below. If you have any questions please don’t hesitate to contact me via email or phone.

# Summary Recommendations

Based on the finding during the Penetration Test. The following recommendations will help improve your security.

* The biggest recommendation I could give is to use more complex passwords. I was able to gain access to the DVWA site with the Username admin and Password = password. Here is a link to a guidelines page for [strong passwords.](https://www.sans.org/security-resources/policies/general/pdf/password-construction-guidelines)
* Web Application Firewalls will help against injection attacks.
* Use appropriate privileges.
* Encrypt and has appropriate data
* Keep everything Updated and Patched
* Perform proper input validation, Cleanup code, Implement Least Privilege, and Use safe API
* Neutralize meta characters and Blacklist specialized characters
* Implement XSS Filters

# Narrative

Testing began with vulnerability scans on the IamAVulnerable Domain, including Nmap port scan. Vulnerability scans turned up a brute force vulnerability. By using bruteforcing techniques we were able to get into the DVWA environment with the Username admin and the Password = password. Once in the DVWA environment we were able to perform vulnerability scans to find several vulnerabilities. Including SQL Injection, File Injection, XSS, and Command Injection. Further details are listed below of our findings. We were able to go through and test each of these vulnerabilities to exfiltrate sensitive data, and execute unwanted code.

# Findings

The following are the findings discovered during this engagement. The criteria for rating these vulnerabilities follows:

* **Critical**: Remote exploitation is possible without authentication or exploitation is trivial and grants significant privileges
* **High**: Remote exploitation is possible without authentication but additional steps are needed to have a full compromise
* **Medium**: Exploitation is possible, but relies on additional attack vectors to be successful
* **Low**: Exploitation is extremely difficult to perform and may not be reliable.
* **Info**: Information that doesn’t directly lead to exploitation but may give away details to an attacker that will prove more useful in finding vulnerabilities or attacking systems

**Title:** SQL Injection

**Vulnerability ID:** A0001

**Severity:** Critical

**Vulnerable Host:** http://10.0.2.4/dvwa/vulnerabilities/sqli/?id=1&Submit=Submit

**Explanation:**

SQL Injection is a malicious attack where SQL statements are inserted into an entry field for execution. A successful SQL Attack like the one I just performed on your SQL server allowed me to gain access to passwords and hashes of those passwords; Also I was able to gain access to the usernames associated with those passwords. I found this vulnerability under your SQL injection tab. I have included some suggestions on how to fix this vulnerability along with some documentation about SQL Injections.

**Evidence:**

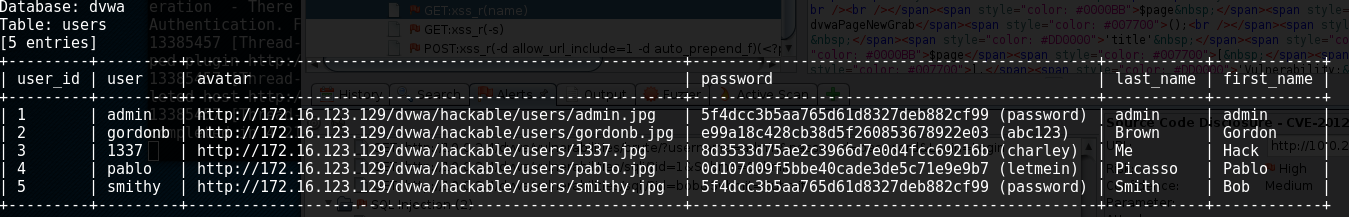


Figure 1: Passwords and password hash’s accessed via SQL Injection

**Remediation:**

1. Implement a WAF(Web App Firewall) Firewall
2. Get rid of any data database functionality that is not needed
3. Use appropriate privileges
4. Assume that your application is not secure and encrypting or hashing passwords/other sensitive data.
5. Update and patch
6. Prepare Statements with parameterized queries
7. Whitelist Input Validation
8. Escaping All User Supplied Input

**References:**

<https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet>

<http://www.enterprisenetworkingplanet.com/netsecur/article.php/3866756/10-Ways-to-Prevent-or-Mitigate-SQL-Injection-Attacks.htm>

**Title:** Command Injection

**Vulnerability ID:** A0002

**Severity:** Critical

**Vulnerable Host:**  10.0.2.4/dvwa/vulnerabilites/exec/#

**Explanation:**

Command Injection is a common malicious attack where the goal is to execute malicious commands on the host operating system (OS) via a vulnerable application. In this case your Ping for Free web app, under the Command Execution tab. I was able to extract password data from your host box which could be used to cause some real damage to your system. I have included some ways to prevent Command Injection on your site and some references also.

**Evidence:**

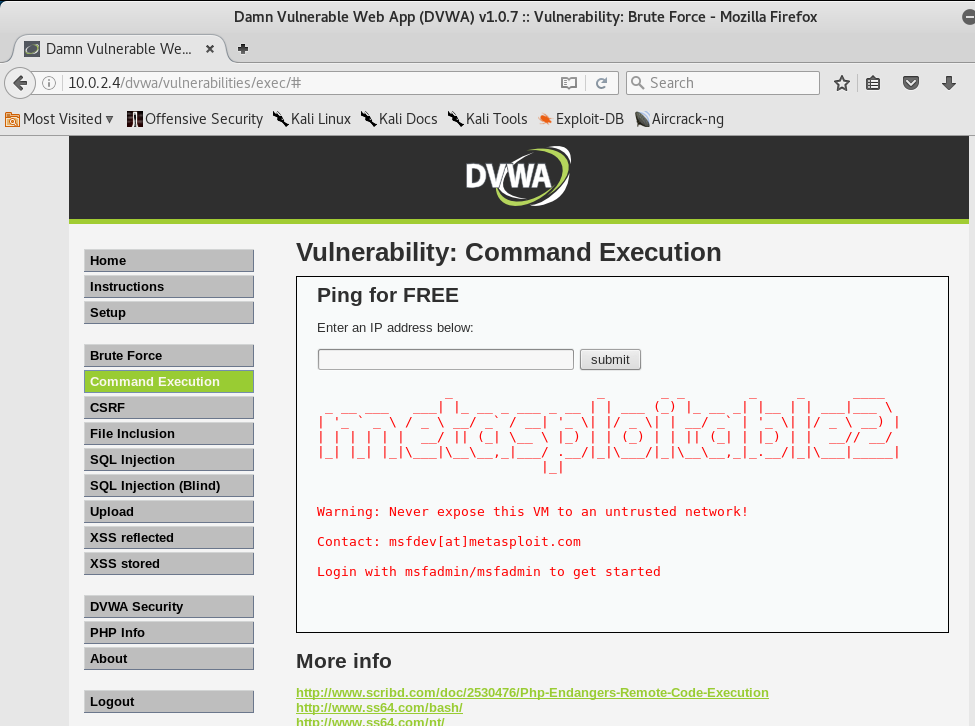


Figure 2: Using command injection; 1 | cat /etc/issue

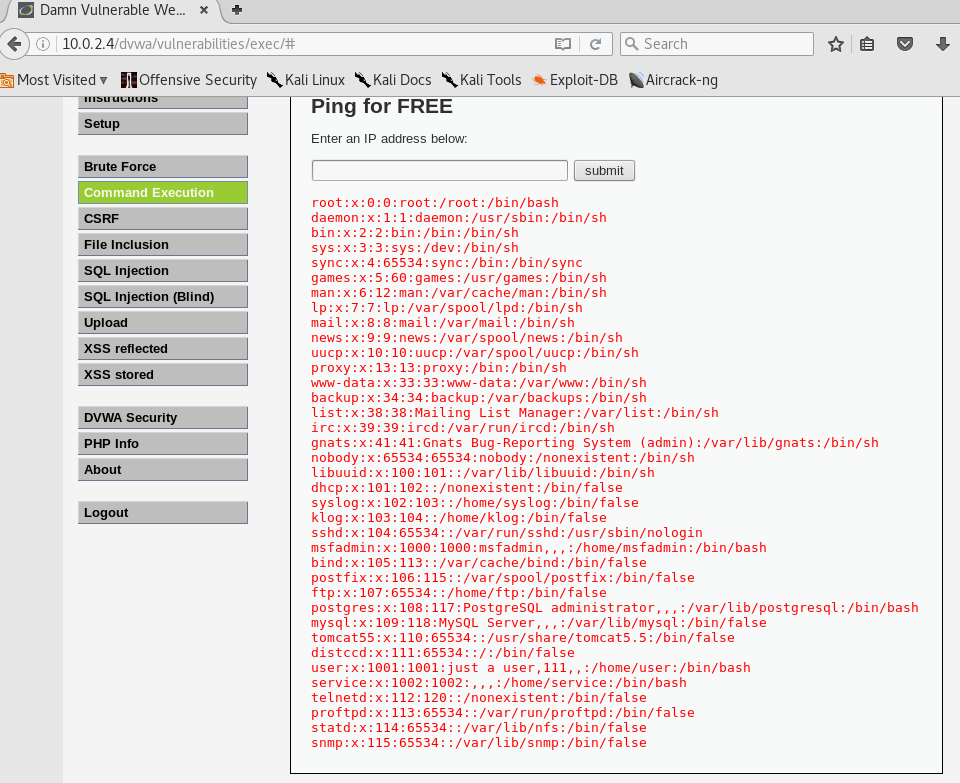


Figure 3: Extracting data using command injection; 1 | cat /etc/passwd

**Remediation:**

1. Perform proper input validation
2. Use a safe API
3. Contextually escape user data
4. Neutralize Meta Characters
5. Implement Least Privilege

**References:**

<https://www.owasp.org/index.php/Injection_Prevention_Cheat_Sheet>

<http://affinity-it-security.com/how-to-prevent-command-injection/>

**Title:** XSS - Cross Site Scripting

**Vulnerability ID:** A0003

**Severity:** High

**Vulnerable Host:** 10.0.2.4/dvwa/vulnerablilities/xss/#

**Explanation:**

Cross Site Scripting or XSS is one of the most common forms of attack. A XSS attack is a type of attack on a web app such as your what is your name Application, that allows a malicious user to execute/inject scripts and actions on the site. This flaw can be found anywhere in an application where user’s input has been taken but not properly encoded. Malicious action can be performed on the client side. Like stealing cookies and session tokens of valid users.

I found the XSS vulnerability on your “what is your name?” web app under XSS Reflected tab. I have included some recommendations on how to fix your vulnerability down below and some references.

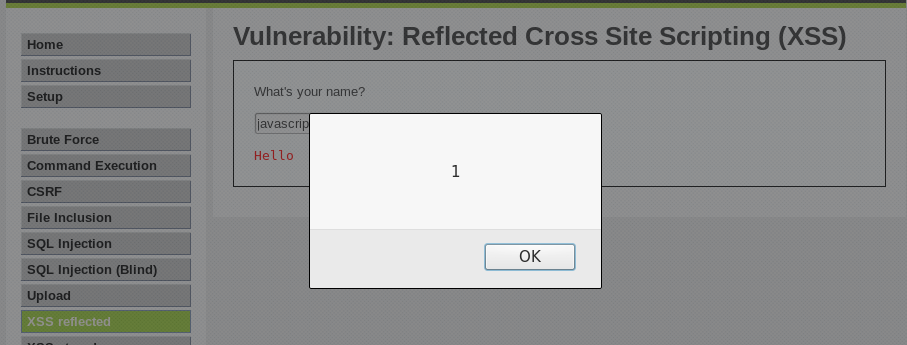
**Evidence:**

Figure 4: JavaScript Cross Site scripting using <script language="javascript">alert("1")</script>

**Remediation:**

* Create a XSS Filter
* Cleanup code
* Dont use dynaically generated JavaScript code whenever possible
* Never Insert untrusted data

**References:**

<https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet>

<http://resources.infosecinstitute.com/how-to-prevent-cross-site-scripting-attacks/#gref>

**Title:** File Injection

**Vulnerability ID:** A0005

**Severity:** Critical

**Vulnerable Host:** 10.0.2.4/dvwa/vulnerablilities/fi/?page=../../../../../../etc/passwd

**Explanation:**

File Injection is a vulnerability that allows a malicious attacker to include a file into a system. This can lead to code execution, DoS, XSS, or Sensitive information disclosure. The last one I was able to do on your system. I was able to grab the passwords file from off the host machine. I have included some ways of patching up these vulnerabilities, and some corresponding documentation.

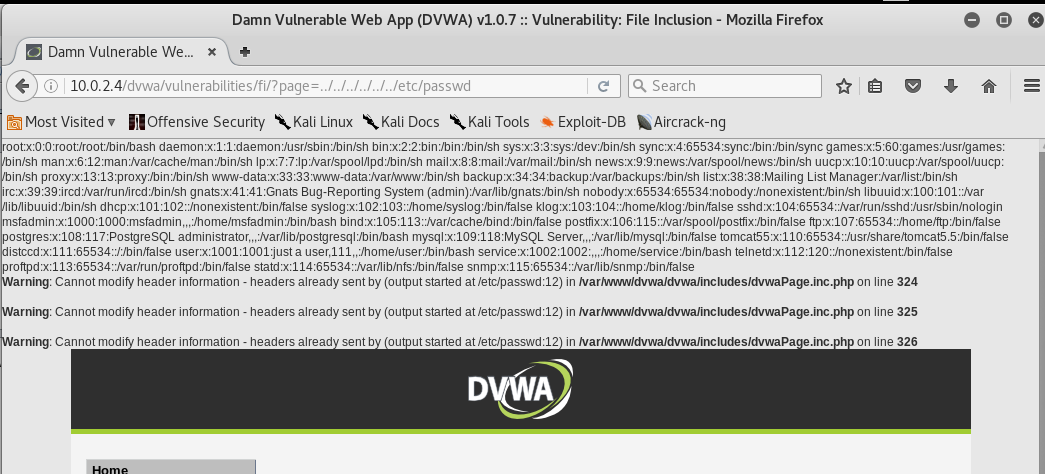
**Evidence:**

Figure 4: File Injection

**Remediation:**

* Blacklist all special characters
* Limit the API to allow only files from allowed directories.
* Avoid passing user-submitted input to any framework API.
* Maintain a whitelist of allowed files
* Reject all invalid identifier contained in any request to destroy any attack surface for malicious users

**References:**

<https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion>

<http://resources.infosecinstitute.com/file-inclusion-attacks/#gref>