By: Ryan Dewhurst

Introduction:

Damn Vulnerable Web App (DVWA) started back in late 2008 when I wanted to learn more about the art of web application security. At the time I found it difficult to find an application that would fulfil my needs. I wanted an application developed in PHP/MySQL that was vulnerable to the most common types of web application vulnerabilities, that I could practise manual exploitation and automated web application scanning tools on and of course in a legal environment.

At first I used the application to teach myself web application vulnerability exploitation and countermeasures. I thought that it may be useful to other security professionals/students, so I released the first version as 'DVWA BETA'. DVWA BETA was released on the 17th of December 2008.

At the time of writing DVWA is at version 1.0.5 with version 1.0.6 under development. Since the first release, DVWA has come a long way. We have implemented many more vulnerabilities, more features and made it look/feel like a real web application where ever possible. Currently we are averaging over 1000 downloads per month and growing.

DVWA would not be what it is today without the help of the DVWA open source community. We have had contributions and feedback from all over the world, from people with many different backgrounds and perspectives. I believe this is what makes DVWA so great.

Vulnerabilities:

As the name suggests DVWA has many web application vulnerabilities which affect it. Every vulnerability has three different security levels, low, medium and high. The security levels give a challenge to the 'attacker' and also shows how each vulnerability can be counter measured by secure coding. Below is an explanation of each security level:

High: This vulnerability level gives the user an example of how to secure the vulnerability via secure coding methods. It lets the user understand how the vulnerability can be counter measured. This level of security should be un-hackable however as we all know this is not always the case. So if you manage to bypass it, let us know.

Medium: This security level's purpose is to give the 'attacker' a challenge in exploitation and also serve as an example of bad coding/security practises.

Low: This security level is meant to simulate a website with no security at all implemented into their coding. It gives the 'attacker' the chance to refine their exploitation skills.

Below is a list of some of the vulnerabilities which affect DVWA and some brief information about each of them:

- **Brute Force:** HTTP Form Brute Force login page; used to test password brute force tools and show the insecurity of weak passwords.
- **Command Execution:** Executes commands on the underlying operating system.
- **Cross Site Request Forgery:** Enables an 'attacker' to change the applications admin password.
- **File Inclusion:** Allows an 'attacker' to include remote/local files into the web application.
- **SQL Injection:** Enables an 'attacker' to inject SQL statements into an HTTP form input box.
- **Insecure File Upload:** Allows an 'attacker' to upload malicious files on to the web server.
- **Reflected Cross Site Scripting:** An 'attacker' can inject their own scripts into the web application.
- **Stored Cross Site Scripting:** Allows an 'attacker' to inject malicious scripts into the database.
- **Easter eggs:** Full path Disclosure and Authentication bypass. (find them!)

Exploitation:

In this section of the article I will give some simple examples on exploiting some of the vulnerabilities found in DVWA.

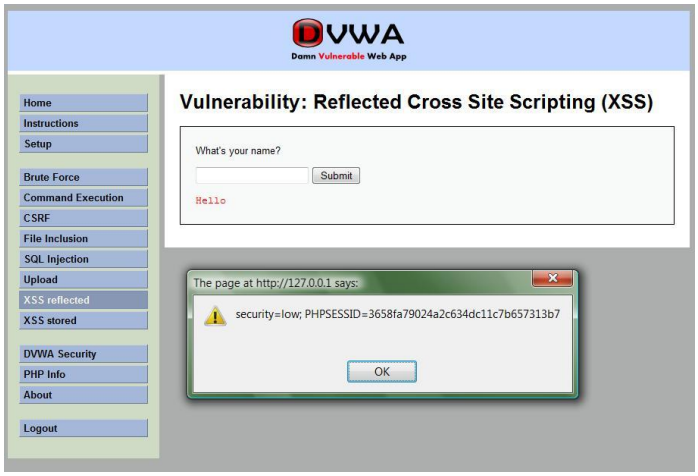
Reflected XSS:

"Reflected attacks are those where the injected code is reflected off the web server, such as in an error message, search result, or any other response that includes some or all of the input sent to the server as part of the request. Reflected attacks are delivered to victims via another route, such as in an e-mail message, or on some other web server. When a user is tricked into clicking on a malicious link or submitting a specially crafted form, the injected code travels to the vulnerable web server,

which reflects the attack back to the user's browser. The browser then executes the code because it came from a "trusted" server." *(OWASP, Cross-site Scripting (XSS))*

In this example *(Screenshot 1)* we have entered a payload into the vulnerable name variable which shows a JavaScript alert box with the web applications cookies. The code is vulnerable because it does not sanitise user supplied input before executing them. The payload used was: "*<script>alert(document.cookie);</script>".* This is just a simple proof of concept to prove that the vulnerability actually exists. Reflected XSS attacks can be used to steal cookies, deface pages, install malware, execute exploits and much more. The exploitation is only limited to your imagination.

Screenshot 1:



Command Execution:

"The purpose of the command injection attack is to inject and execute commands specified by the attacker in the vulnerable application. In situation like this, the application, which executes unwanted system commands, is like a pseudo system shell, and the attacker may use it as any authorized system user. However, commands are executed with the same privileges and environment as the application has. Command injection attacks are possible in most cases because of lack of correct input data validation, which can be manipulated by the attacker (forms, cookies, HTTP headers etc.)." *(OWASP, Command Injection)*

In this example *(Screenshot 2)* we have entered the payload into the vulnerable ping variable input box which not only executes the ping command like the application is supposed to, but also executes the 'whoami' command which prints the current logged in username *(nt authority\system)*. The payload used was: *"127.0.0.1 && whoami".* Command execution vulnerabilities are much more

powerful than just printing the current logged in user; they can be used to download files from the internet, connect back to the ‘attacker’, format the hard-drive and much more. Again the exploit is only limited to the imagination of the ‘attacker’.

Screenshot 2:

Countermeasures:

The countermeasure of each vulnerability can be seen via the ‘View Source’ button on the high security level. DVWA also has an IDS system implemented *(PHPIDS)* which can be used to view the affect of an IDS on a web application. In the example below we will look at the source code for the command execution high security level vulnerability.

Screenshot 3:

Command Execution Source

```
<?php

if( isset( $_POST[ 'submit' ] ) ) {

    $target = $_REQUEST["ip"];

    $target = stripslashes( $target );


    // Split the IP into 4 octects
    $octet = explode(".", $target);

    // Check IF each octet is an integer
    if ((is_numeric($octet[0])) && (is_numeric($octet[1])) && (is_numeric($octet[2])) && (is_numeric($octet[3]))) {

    // If all 4 octets are int's put the IP back together.
    $target = $octet[0].'.'.$octet[1].'.'.$octet[2].'.'.$octet[3];


        // Determine OS and execute the ping command.
        if (stristr(php_uname('s'), 'Windows NT')) {

            $cmd = shell_exec( 'ping  ' . $target );
            echo '<pre>'.$cmd.'</pre>';

        } else {

            $cmd = shell_exec( 'ping  -c 3 ' . $target );
            echo '<pre>'.$cmd.'</pre>';

        }

    }

    else {
        echo '<pre>ERROR: You have entered an invalid IP</pre>';
    }


}

?>
```

Command execution (high security level code in DVWA 1.0.6):

```
<?php

if( isset( $_POST[ 'submit' ] ) ) {

    $target = $_REQUEST["ip"];

    $target = stripslashes( $target );


    // Split the IP into 4 octects
    $octet = explode(".", $target);

    // Check IF each octet is an integer
    if ((is_numeric($octet[0])) && (is_numeric($octet[1])) && (is_numeric($
octet[2])) && (is_numeric($octet[3]))) {

    // If all 4 octets are int's put the IP back together.
    $target = $octet[0].'.'.$octet[1].'.'.$octet[2].'.'.$octet[3];


        // Determine OS and execute the ping command.
        if (stristr(php_uname('s'), 'Windows NT')) {

            $cmd = shell_exec( 'ping  ' . $target );
            echo '<pre>'.$cmd.'</pre>';

        } else {

            $cmd = shell_exec( 'ping  -c 3 ' . $target );
            echo '<pre>'.$cmd.'</pre>';
```

```
        }

    }

    else {
        echo '<pre>ERROR: You have entered an invalid IP</pre>';
    }


}

?>
```

The code above works on a white list approach. It first splits the IP address into 4 separate octets, 127,0,0 and 1. It then checks that each octet is an integer by using the is_numeric() function; if each octet is an integer it rebuilds the IP address and executes the command. If any of the octets contain any non integer characters the script returns an error message *(ERROR: You have entered an invalid IP).*

Conclusion:

Damn Vulnerable Web App (DVWA) has taught me much about the art of web application security and I hope that it can be used teach web developers and other ethical hacking students/professionals the same.

If you are wanting to learn web application security I advise not only using DVWA however also reading books such as “The Web Application Hackers Handbook” and using many other online resources which are freely available.

If you would like to contribute to DVWA you can visit http://www.dvwa.co.uk to find out how. We welcome contributions of any kind.

References:

OWASP. (2009) *Cross-site Scripting (XSS).* Available at: http://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29 (Accessed: 3rd October 2009).

OWASP. (2009) *Command Injection.* Available at: http://www.owasp.org/index.php/Command_Injection (Accessed: 3rd October 2009).