Directory Traversal Attack On Linux Based Web-Servers

Sai Govindarajan

956299

Contents

Introduction

Security vulnerabilities (Kumar, 2016) are weaknesses, flaws or errors found within a security system that have the potential to be taken advantage of by an adversary in order to compromise a network. When this is the case the system being targeted is under threat. A threat (Kumar, 2016) is a hypothetical event where an adversary uses such vulnerabilities to attack a system. With the current technological advances there have been many ways to circumvent such attacks to classify a security system as safe. However, as there is a technological advancement this goes both ways. Adversaries are able to overcome such security defences which keep systems constantly under threat. One such form of security vulnerability is web-based and realistically could affect any one of us that use a form of input/output file upload over the internet. This attack is known as File Path/Directory Traversal attack.

A dictionary traversal attack (Ikram Lali, Ahsan and A.F.M, 2010) is a web security vulnerability that is the product of insufficient filtering or validation of browser input from user. They can be found within web server software/ files, or in application code that is to be executed on the server. Directory traversal vulnerabilities can exist in a variety of programming languages which includes Python, PHP, Pearl and more.

On a webserver, web applications are executed relative to the web document root. The exact path depends on the operating system and the webserver, but common web document root directories include **/var/www** for Apache on Linux and **C:\Inetpub\wwwroot** for ISS on Windows. A bug in the webserver software may allow the web server process to access files outside the web document root. If a web application also uses file names taken from user without proper input validation, this could open a directory traversal vulnerability. Instead of valid file names, an adversary can then enter absolute or relative file paths to access arbitrary files, including application source code, system files and other files containing sensitive information. If combined with a file upload vulnerability, directory traversal can lead to remote code execution.

Technical Overview with example illustration

In order to understand this security vulnerability extensively, we must first understand the file system hierarchy and how each file or directory is accessed by the system. Upon this, we must understand, how an adversary uses the system efficiently (Barracuda Campus, 2019) to effectively exploit the vulnerability upon which we use examples to analyse situations where this vulnerability would/could be exploited. The below section follows this format with an aim to provide understanding of the topic of this literature.

File System Structure

A string that shows the location of a directory or a file on a system's file system is known as a path string. When we look at **Fig 1** below, we see a basic example of a hierarchal system of a Linux based system with "root" directory denoted by "/". We also have "home" and "www" directories based around the root directory. Using this figure, if one were to traverse to the root directory, the path string would be defined by "/". Similarly, in order to traverse to the home directory, the path string can be represented by "/home" and then from there to the user "bob" with the path string "/home/bob".

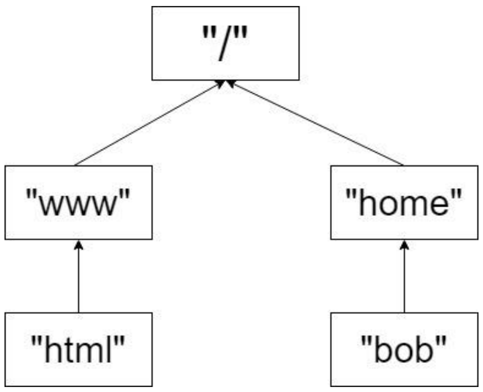


Fig 1: Example illustration of a basic file system hierarchy for a Linux based system

Additionally, there are "." and ".." contained inside each directory which are not represented in **Fig 1**. These are special characters that are included in certain files systems that aid the user in traversal of the file system hierarchy with ease. They relate to the directory relative to the directory that they are currently present within. For example, "." would be directed to the current directory the user is traversing and ".." would be directed to the parent directory of the current directory. When we apply this to our example, where we are inside "/home/bob", ".." would take us to "/home/" and "." would keep us at "/home/bob". Adversaries have the ability to use these special characters to traverse file system hierarchies.

The Security Vulnerability

Now as we analyse at **Fig 1**, we realise that when Bob decides to upload an image into a web application, it takes in a user-supplied path string to retrieve the file from inside Bob's home folder. This is then taken in and shown with a path string of "../../www/html/". When this is the case, adversaries can use the aforementioned special directory identifiers to traverse file hierarchies to reach restricted directories. By using these special characters, an adversary could potentially access the root directory "/etc/passwd" where the system's password file is stored. This exploit is commonly known as a directory traversal attack.

Directory traversal attack (Flanders, 2019) is a web security vulnerability that allows an adversary to read arbitrary files on the server that is running an application. This might include application code and data, sensitive operating system files and credentials for back-end systems. In some cases, an adversary might be able to write to arbitrary files on the server, allowing them to modify application data or behaviour and ultimately take full control of the server.

Many applications that place user input into the file paths implement some form of defences against directory traversal attacks, but these can often be evaded. If an application strips or blocks directory traversal sequences from the user-supplied filename, then it might be possible to bypass the defence using a variety of techniques.

The adversary might be able to use an absolute path from the file system root, such as "filename=/etc/passwd", to directly reference a file without using any traversal sequences. An adversary might also be able to circumvent the defence measures by using nested traversal sequences, such as "…//" or "…\/", which will revert to simple traversal sequence when the inner sequence is stripped.

Additionally, an adversary could also use various non-standard encodings, such as "..%252f" or "..%c0%af" to bypass the input filter. In order to gain understanding of such input filters, one must simply google such filters to use them. This puts the web applications that use such basic defence mechanisms under an enormous threat.

This security vulnerability requires the adversary to learn the encoding syntax for each type of server that they are trying to exploit. This includes either a UNIX based server such as an Apache server or a Windows based server such as an ISS server. Each have different encoding syntax, but they do not deviate from each other greatly.

Example Illustration – The Security Vulnerability

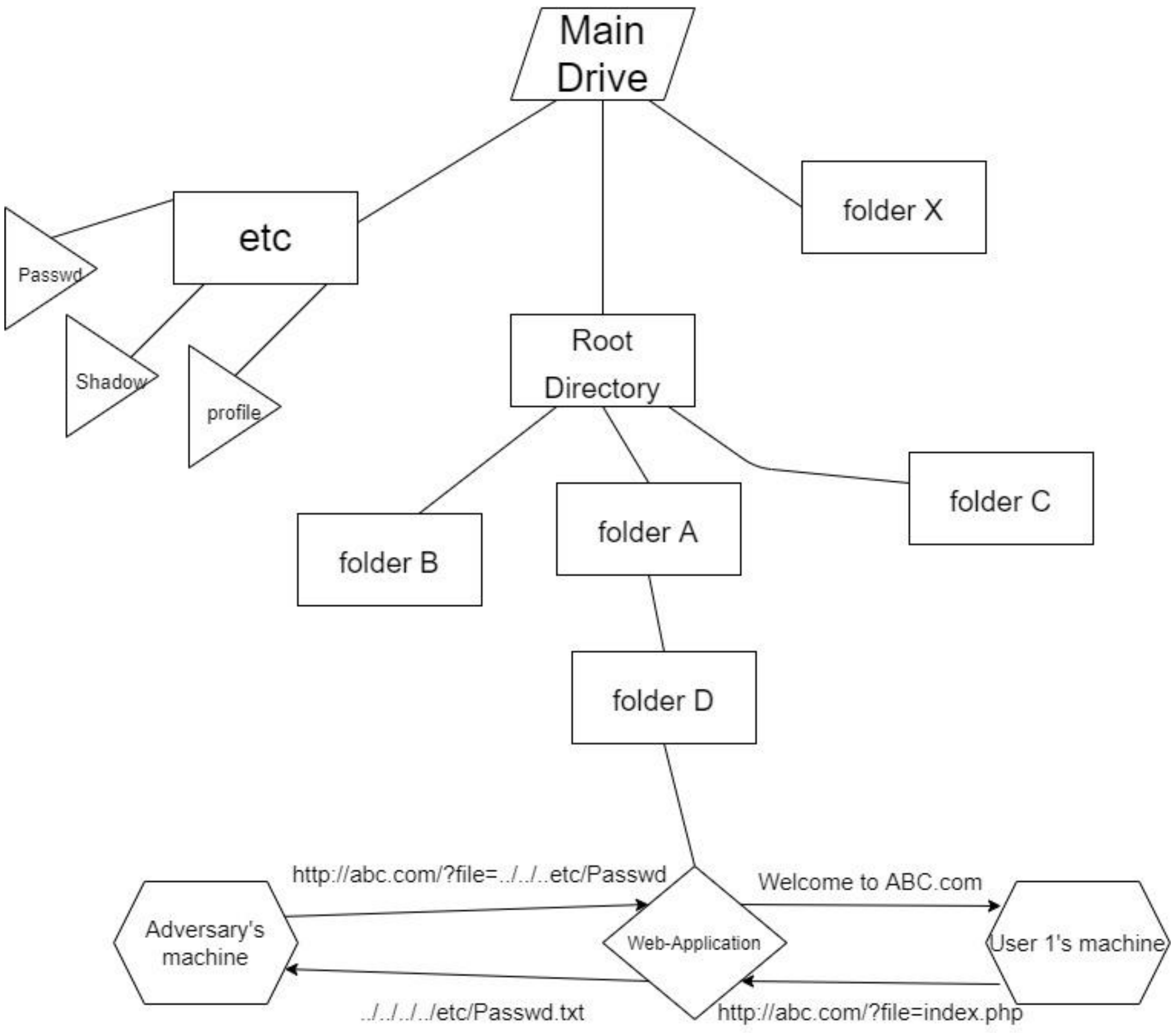


Fig 2: Example illustration of a Linux based web-application server under directory traversal attack

When we analyse at **Fig 2**, we see that it illustrates a Linux based web-application server. This sever is stored on the system's main drive. This means along with the files for this web application server, there sits the files responsible for running and maintaining the core operating system that the whole application is run on. This server is responsible for returning "abc.com" when a user requests it through an HTTP protocol.

Now as we follow the illustration **Fig 2**, we see that file access from the web-application goes to folder D then to folder A then to the root directory and then eventually to the main system drive from where all the systems folders are present. When User 1 requests abc.com from their machine, the web application using the web server API accesses the appropriate files within their respective folders to return “abc.com”. Note here, that the web application link has the suffix “?file=index.php” which is responsible for returning the homepage of “abc.com”.

Now to explain Directory Traversal attack better, we must look at the adversary’s machine. There sits and adversary, who realises that changing the value for “?file=” can gain them access to the root of system. Thus, they replace this section with as many “../” as required as, these special characters can be used traverse through file systems with ease. By doing this, the adversary is able to eventually access the system folders, such as the “etc” folder which contains information on system passwords, shadow files and the profile information. When this takes place, we can confirm that a directory traversal vulnerability has been exploited by an adversary. They can now add, edit, or even delete critical system files which could not only cause the web-app “abc.com” but also the system it sits on to crash.

Possible Fixes and Patches

We live in a world where innovation takes place regularly. As a result of this, there are many ways to outplay the adversaries and prevent this security vulnerability from being exploited further (Flanders, 2019). Such measures can be taken by simply keeping the webservers up to date to the latest software versions available. Other measures involve, avoiding using user-supplied input to the filesystem API all together. Instead, one should have the user input be an index onto the list of one of the known safe files. For example, ‘1’ could map to somethingSafe.html and ‘2’ could map to somethingElseSafe.html.

In order to ensure further security for the webserver, it must be run from a separate disk from the system disk that contains critical operating system files and if possible, avoid storing any sensitive files on the disk where the webserver sits, unlike the example in **Fig 2**.

If it is considered unavoidable to pass user-supplied path string to file system API, then the following two layers of defences should be used together to prevent the attacks from taking place:

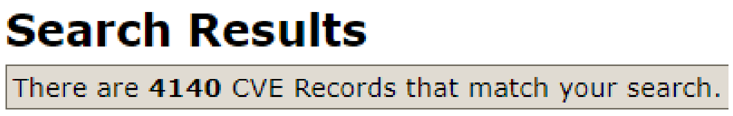
- The program must validate the user input path before processing It, preferably, the validation should compare against whitelist of permitted values. If that is not possible for the required functionality, then the validation should verify that the input contains only permitted content, such as purely alphanumeric characters.
- Upon validation of the path string, the program must append the input to the base directory and use a filesystem API to canonicalize the path. Canonicalization is the process of converting that involves more than one representation into a standard approved format. Upon performing canonicalization, the program must ensure that the canonicalized path starts at the expected root directory.

Using such defence strategies, will make it increasingly harder for adversaries to try and exploit this security vulnerability. When the current technology holds security of a system just as important as its functionality, it will eventually eradicate this threat. However, this would create newer threats but, when we place the importance of security of a system at a high esteem, such threats would eventually be a story of the past.

Systems affected by the security vulnerability in the real world

For this section of the literature, we look at the systems that have been affected by directory traversal vulnerability. Additionally, we also analyse such systems in the real-world using CVE program (Mitre.org, 2019). Developed with the U.S Department of Homeland Security, CVE aids us to identify and define publicly disclosed cybersecurity vulnerabilities.

When researching this security vulnerability, we cannot follow the conventional methods of searching news articles about this issue as, we realise that there are absolutely no news articles covering this topic. This might be due to the fact that this security vulnerability is set to have been "patched". Currently we have ample security measures that we can take to prevent it from being exploited. That being said, when we research this security vulnerability using CVE program, we realise the opposite of what was mentioned above. When searching the term "Directory Traversal" on CVE, we find that, even in today's day in age, it is a very real threat. This can be seen from **Fig 3** below.



Search Results

There are **4140** CVE Records that match your search.

Fig 3: Showing the number of cases of directory traversal attack that has been reported on CVE program (cve.mitre.org, n.d.)

Fig 3 shows the number of cases where directory traversal vulnerability has been exploited successfully by an adversary. This is a surprisingly large number of results for a security vulnerability proving for it to be a very real threat even today.

This being the case, when we choose one of the articles on this vulnerability, namely (CVE-2021-27328) (cve.mitre.org, 2021) we understand that the product Neogate Tg400 (Baltic Networks, n.d.) that has undergone the exploit is a compact 4 channel VoIP GSM gateway that connects GSM networks with VoIP. This product was exploited using directory traversal attack by allowing an authenticated user to decrypt the firmware and read sensitive information such as passwords or decryption. The reason this exploit was possible because the device was not up to date with software updates, therefore was susceptible to security vulnerabilities. Considering this product costs $540.00, having such vulnerability is not acceptable for both the company and its customers.

As we continue, we look at another case where this security vulnerability has a very high potential of taking place. For this we look at the article with the CVE ID (CVE-2020-3284) (cve.mitre.org, 2019). This article conveys to us of a web-based management of Cisco Vision Dynamic Signage Director and how it could allow an unauthenticated, remote adversary to view potentially sensitive information on the devices that are affected by this exploit. The reason for this vulnerability on this system is due to incorrect permission setting in the Linux based Apache configuration. In order for an adversary to exploit this vulnerability on this system, they must simply send a crafted HTTP request to the web-based management interface which would then allow them to view potentially sensitive information such as passwd and shadow files of the affected devices.

To Conclude

We have analysed and understood Directory Traversal vulnerability using example illustrations and real-life examples. We have also understood how to traverse Linux file systems using special characters and how an adversary might use this to their advantage when they want to run this exploit. While doing this, we have also understood the gravity of this attack and answered the question why security protection for such system is key for its functionality. To understand this security vulnerability further, we now must attempt carrying out this attack on a system under white-box conditions and with an intent to learn from it the ways of dealing with such attacks, patching the exploit and by doing so preventing reoccurrences of such vulnerabilities.

Bibliography

1. Kumar, S. (2016). Information Security Threats, Vulnerabilities and Assessment. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 5(). Available at: http://ijarcet.org/wp-content/uploads/IJARCET-VOL-5-ISSUE-5-1358-1360.pdf.

2. Ikram Lali, M., Ahsan, F. and A.F.M, I. (2010). *Path Traversal Penalty in File Systems*. Available at: https://www.researchgate.net/publication/41393197_Path_Traversal_Penalty_in_File_Systems/link/00b7d534f5c719d952000000/download.

3. Barracuda Campus. (2019). [online] Available at: https://campus.barracuda.com/product/webapplicationfirewall/doc/42049342/directory-traversal-vulnerability/.

4. Flanders, M. (2019). *A Simple and Intuitive Algorithm for Preventing Directory Traversal Attacks*. [Undergraduate Dissertation] Available at: https://arxiv.org/pdf/1908.04502.pdf.

5. Mitre.org. (2019). *CVE - Common Vulnerabilities and Exposures (CVE)*. [online] Available at: https://cve.mitre.org/.

6. cve.mitre.org. (n.d.). *CVE - Search Results*. [online] Available at: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=Directory+traversal.

7. cve.mitre.org. (2021). *CVE - CVE-2021-27328*. [online] Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-27328.

8. Baltic Networks. (n.d.). *Yeastar NeoGate GSM TG400 1-Port Gateway (4 Channels)*. [online] Available at: https://www.balticnetworks.com/yeastar-neogate-gsm-tg400-1-port-gateway-4-channels.

9. cve.mitre.org. (2019). *CVE - CVE-2020-3484*. [online] Available at: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-3484.