Path Traversal

Rajas Bipinchandra Patil

2023-04-02

This paper analyzes and suggests techniques to mitigate the Path Traversal attack, also known as 'Improper Limitation of a Pathname to a Restricted Directory' (CWE-22). In this paper, we try to understand how path traversal works and how attackers usually execute this attack. We majorly focus on prevention strategies in various phases of software development. In the end, we conclude by summarizing the techniques we have seen so far.

1. Introduction

Path Traversal, also known as Directory Traversal attack, focuses on gaining unsolicited access to files and directories outside the web root folder on the server where a web application is deployed. It comes under the umbrella of 'Broken Access Control' vulnerabilities that ranks in the first position of the OWASP Top 10:2021 vulnerabilities.

This vulnerability is typically exploited by the attacker by manipulating the URL. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal[zaproxy]. The Path traversal attack, on an elementary level, can be executed by simply adding the special character sequence '../' to the URL. Due to this, this attack is also known as a dot-dot-slash attack. By adding this to the URL of a vulnerable application, the attacker can change directories on the application server. This is dangerous as the attacker can get access to sensitive information that is present on the server.

Path Traversal vulnerabilities can have serious implications, such as data loss, privacy infringement, and system compromise. Attackers can use path traversal vulnerabilities to access sensitive files or directories that contain confidential information, such as passwords, financial data, or personal information. In some cases, path traversal vulnerabilities can also allow attackers to execute arbitrary code on the system, which can lead to system compromise or even total control of the system. This paper primarily focuses on explaining the vulnerability and providing ways in which we can try to mitigate the vulnerability as much as possible.

Due to the nature of this vulnerability that involves the attacker navigating through the directories and going back to the root folder, it is also known by names like directory climbing or backtracking. [owasp]

2. What are Path Traversal attacks and why do they work?

To understand the Path Traversal Attack, it is important to understand how a web application handles user input. Web applications process user input in the form of parameters, such as form data, query string parameters, and cookies. These parameters are then used to create a URL that will be used to request resources from the server. But what happens when these parameters carry unsanitized input that can contain a payload sent by an attacker? The system gets compromised.

In a Path Traversal Attack, the attacker uses a specially crafted input to manipulate the URL and traverse directories on the server. The URL can either be manipulated directly from the web browser or if the URL is getting built dynamically in the code, then the attacker can send in malicious input in the input field which is used to build the URL. File paths follow different conventions depending on the operating system. For example, Unix-based operating systems use slash '/', and Windows uses backslash '\' as a path separator. Also, absolute path names in Unix-based operating systems begin with a '/, but Windows either can begin with a '\\' or refer to the drive letter where the file is stored like 'C:\'. However, barring these few differences, the structure of the directories remains the same and consequently, the attack patterns remain the same in principle. For the sake of simplicity, we will follow the syntax of a Unix-based system throughout the scope of this paper.

Now to see why these attacks work, we will begin by understanding the meaning of the sequence that is entered in the payload by the attacker. The sequence of characters such as '../', in a Unix-based operating system is a command to go a level up in the hierarchy of directories. When this sequence is received in an input and is added to any URL, the system perceives this as a command to change directories. Now adding a payload that looks something like '../../../etc/passwd' can make the system go three levels up and if that is the root directory, then the password directory is compromised as the attacker now has access to the passwords of all the users on that system. Naturally, it is not that easy to access the get-through to user passwords as those directories should have another layer of security, however, we will discuss that in this paper in detail in the prevention section.[acunetix]

3. Common Techniques Used in Path Traversal Attacks

As observed in the previous section, the most elementary form of path traversal attack involves a specific sequence of characters strategically placed to access files and directories outside the web root folder. However, in contemporary real-time applications, this unsophisticated technique may not be effective due to the implementation of security mechanisms in web browsers that restrict the acceptance of untrusted input through the URL. Despite this, attackers have historically employed several methods to exploit this vulnerability, and a few commonly utilized techniques will be explored below.

3.1. File path traversal by stripping sequences non-recursively

This method is commonly employed by attackers when a rudimentary defense mechanism against path traversal attacks has been implemented. Nevertheless, such a defense mechanism may prove insufficient. For instance, if the implemented security measure involves removing the conventional "../"

sequence upon identification, an attacker may adopt a modified payload that utilizes nested traversal sequences, like "....//". In this case, the attacker can no use a slightly modified payload that might be able to use nested traversal sequences, such as '....//', which will revert to simple traversal sequences when the inner sequence is stripped [portswigger]. Due to the attackers' capacity for multiple attempts to infiltrate a system, simplistic mitigation techniques like this are often inadequate in preventing path-traversal attacks.

3.2. Null Byte Injection

The null byte injection technique constitutes a means to bypass security measures deployed against path traversal attacks. In programming languages, a null byte is a special character that signals the termination of a string. By incorporating a null byte into the path, an attacker can fulfill the necessary conditions for the request to be accepted by the server. However, when the server processes the request, the null byte functions as a terminator that enables only the malicious code to pass through. An instance of a null-byte injection attack can be illustrated through a request that requires a filename as input. The attacker may then input a payload that appears as follows: 'filename=../../etc/passwd%00.pdf'. The request proceeds since 'pdf' is a valid file format, but the null byte in the input terminates the string, and when the URL is executed, the path traversal payload becomes operational.

3.3. URL Encoding and Double URL Encoding

URL Encoding and Double URL Encoding work on a similar idea. They form the payload in such a manner that converts the malicious input into its hexadecimal code. The attacker can do this by taking into consideration the defense mechanisms that might restrict his attempt to filter the traditional '../' or its variations. Double encoding takes it a notch above by encoding the already encoded URL making it difficult for the system to recognize the pattern. For instance, if a web application filters out the "../" string to prevent path traversal attacks, an attacker can encode the string as "%2E%2E%2F" to bypass the filter. Double encoding takes in the same string and encodes it further such that the '%' is encoded as its hexadecimal code i.e., '%25'. Thus, the payload sequence is now '%252E%252E%252F'.

4. Previous instances of Path traversal

Path traversal attacks have been around for years. Here are a few instances where the system was exploited because of this vulnerability being present.

4.1. kubectl Path Traversal Vulnerability

The first path traversal vulnerability was identified in 2018 when security researcher Michael Hanselmann discovered a vulnerability in the kubectl cp and oc cp commands, which let you copy files from a pod to your client machine. In his research, Michael found that if an attacker could replace this tar binary inside the container (by infecting a base image, or in some other malicious way), he could output a specially crafted tar file that would include files with relative path names, thus performing a Path Traversal on the client machine. This vulnerability was assigned CVE-2018-1002100 with a high-risk

severity rating. It was fixed and new versions were then released. However, it led to a new vulnerability which was discovered about a year later after this was fixed. This vulnerability also exposed the same Path Traversal risk. The new vulnerability was assigned CVE-2019-1002101. This vulnerability reappeared again because the original fix for the previous exploit was incomplete. This was issued a new CVE-2019-11246. Finally, the fix for this vulnerability was a comprehensive one and it took care of the exploit for good. [Sagi2019]

4.2. Apache Struts Vulnerability

A path traversal vulnerability was identified in the Apache Struts core package in 2016. It was assigned a CVSS score of 9.8 and was a critical vulnerability. It was assigned a CVE-2016-6795.

5. Prevention Strategies for Path Traversal Vulnerabilities

The simplicity of path traversal attack is what makes it dangerous. As seen in the previous sections, these types of attacks have a very wide scope since URLs are an integral part of any application. The most effective way to prevent Path traversal vulnerabilities is to not allow any user-defined input while building URLs or supply them to an API filesystem altogether. However, this kind of approach might not seem practical. Thus, to mitigate this vulnerability to the maximum extent we need to implement security measures in three major areas of software development i.e., Architecture and Design, Implementation and Operation.

5.1. Prevention Strategies in Architecture and Design Phase

The Architecture and Design phase of software development can be considered the most important phase from a security perspective. In this phase, we can implement security measures in the application and plan the flow accordingly. However, in this phase, the programmer is very much blind to the attacks that can happen to the application in the future. Thus, the programmer needs to use preemptive measures in this phase to secure the application. Below are a few strategies used in the architecture and design phase for the prevention of Path Traversal.

5.1.1. Environment Hardening

Environment Hardening involves running the code with the lowest possible privileges i.e., taking away all the unnecessary privileges so that the system will work just fine while not leaving any openings. File permissions should be strictly given only to users who require access to read, write, or execute.

5.1.2. Input Validation

Input validation is a key strategy to be implemented against Path Traversal attacks. We will see more about it in the Implementation phase as well. In the Architecture and Design phase, developers should assess the potential inputs that the application may receive and define the appropriate validation rules to prevent attackers from exploiting vulnerabilities in the system. Integrating input validation into the architecture and design phase of software development can help identify potential vulnerabilities early in the development process, allowing for prompt remediation. This proactive approach can save time, effort, and costs associated with fixing vulnerabilities at later stages of development or even post-

release. While integrating multiple modules it is imperative to treat all input as untrusted and systems should be designed in such a manner that can separate the security layer of each module wherever possible so that it is easier to modify the code if any security-related bugs are found in the future. This leads us to the next strategy in the architecture in the design phase which is 'Sandboxing'.

5.1.3. Sandboxing

Sandboxing is a technique used to isolate and execute untrusted code in a controlled environment. This technique is more of a palliative measure rather than a preventive measure i.e. it is used to minimize the damage after it has occurred. Sandboxing divides the application into multiple modules so that even if the attack on one module is successful, the other modules remain unharmed. One of the techniques is containerization technology like Docker, where the web application is deployed in a container that is isolated from other containers and the host system. This approach ensures that if a path traversal attack is successful, the attacker's access is limited to the container only and not the host system or other containers. The use of virtualization technology like virtual machines is another approach to sandboxing. The web application is deployed within a virtual machine that is isolated from the host system, and any successful path traversal attempt will be confined within the virtual machine without access to the host system. These approaches to sandboxing provide a layer of security that limits the impact of path traversal attacks and protects the web application and its environment from unauthorized access.

5.2. Prevention Strategies in Implementation Phase

During the Implementation phase of development, we need to apply techniques in the code such as whitelisting, input validation, etc. In this phase since we are not just idealizing but developing, we have more exposure to the possible issues that can arise in the code. We should also handle file validations in this phase, as that is one of the attack surfaces which can be compromised. Let us look at the strategies in detail.

5.2.1. Input validation and whitelisting

This involves pruning traversal characters and creating allowlists for only accepting valid inputs. The prevention of directory traversal attacks often involves removing traversal characters such as "../" from the path string. However, this approach may lead to semantic errors and miss potential vulnerabilities. For instance, algorithms that adopt this approach, such as mini httpd and thttpd, may contain holes in test coverage, resulting in missed vulnerabilities. Additionally, the complexity of these algorithms makes it difficult to test and verify all possible malicious path strings, leading to the possibility of path explosion problems. As a result, these approaches may fail to prevent directory traversal vulnerabilities, as seen in CVE-201818778[NIST2018]. Thus, while removing traversal characters is a valid prevention strategy, developers should be aware of its limitations and consider additional measures to mitigate the risk of directory traversal attacks.

Whitelisting is frequently regarded as a very successful strategy for thwarting directory traversal attacks. Nonetheless, depending only on whitelisting has its drawbacks. If the length of the path string is unbounded, one such issue is the presence of an endless number of ways to refer to the same file on a file system. While whitelisting has its uses, it may also be unduly restrictive if it doesn't canonicalize and consider the infinite number of different path strings that might lead to a file. Moreover, because

whitelisted files are frequently specified during development, it may be difficult for these mitigations to safely respond to situations when the whitelist must be altered on the fly. Further, if whitelisted directories contain symbolic links pointing to non-whitelisted directories or files, attackers can abuse these symbolic links to bypass defenses and traverse the file system. [Flanders2019].

5.2.2. Canonicalization of paths

Canonicalization of path strings is a technique used to prevent directory traversal attacks. This technique resolves a user-supplied path string to a unique, absolute path string by expanding symbolic links and resolving directory references such as '/./', '/../', and extra '/' characters. The majority of modern programming languages have inbuilt functions for canonicalization. Since they involve system calls or access to the filesystem, these routines are less portable and frequently more sophisticated than other avoidance strategies. For example, rebuilding the 'realpath' function when transferring PHP code to another programming language or web application framework [Flanders2019]. Despite these restrictions, route string canonicalization remains an effective method of preventing directory traversal attacks. Consider the following code that prevents directory traversal attacks [Kohnfelder2022].

```
String filename = request.getParameter("file");
final String base = "/safedir";
File prefix = new File(base);
File path = new File (prefix, filename); // Resultant path must start with the base
path prefix
if (!path.getCanonicalPath().startsWith(base)) {
 throw new PathTraversalException(filename + " is invalid");
}
// Resultant path must be longer by more than 1 character than the base path prefix
if (!(path.getCanonicalPath().length() > prefix.getCanonicalPath().length() + 1)) {
 throw new PathTraversalException(filename + " is invalid");
}
path.delete();
```

The code initially creates a safe base directory with the name "/safedir."

Next, by concatenating the base directory with the user-supplied filename, it constructs a File object representing the base directory and another File object representing the entire path.

The code then checks to see if the resultant path begins with the base path prefix and is more than one character longer than the base path prefix. If one of these requirements is not fulfilled, the path is invalid and a PathTraversalException is thrown.

Finally, if the path is valid, the function deletes the File object's representation of the file.

5.2.3. Tokenization Algorithm

This algorithm is kind of an extension of the previous approach 'canonicalization of paths'. It canonicalizes the URL in question and generates an output string. Next, the algorithm starts by tokenizing the string with '/' as a separator. Once the tokens are formed they are pushed into a stack in the order they were tokenized. The value inside the stack is now checked. If the value in any element of the stack is a '.' it is ignored. However, if the value in any of the elements of the stack is a '..', it is

immediately discarded? This way we ensure that none of the values inside the stack hold the capacity to change the directory even if the URL were to be executed. [Flanders2019]

Apart from using the standard techniques for implementing defense mechanisms against path traversal attacks, we can write algorithms such as these catering to our specific needs. It can be an amalgamation of these techniques as well.

5.2.4. DotDotPwn - The Directory Traversal Fuzzer

DotDotPwn is a well-known tool used to discover path traversal vulnerabilities. It may be used to test web applications for path traversal vulnerabilities and automates the process of exploiting path traversal vulnerabilities. By appending "../" characters to the path in the request, the program attempts to access files outside the web root directory. If the web application is subject to path traversal attacks, the tool can access files outside the web root directory, possibly gaining access to sensitive information or executing malicious code. DotDotPwn is a handy tool for testing web applications for path traversal vulnerabilities, but it should only be used in a controlled environment with suitable permissions.

Once the tool finds a vulnerable target, it can then proceed to exploit the vulnerability by attempting to read or write files on the server, or by executing arbitrary commands. The tool can also be customized to target specific directories or file types and to use various techniques such as null-byte injection and URL encoding to bypass security measures. [sectester]

5.3. Prevention Strategies in Operations Phase

In the Operations phase, it is crucial to ensure that the deployed web application remains protected against path traversal attacks. This is to make sure that the implemented checks are working and if there are any openings in the system, they need to be handled and closed. For this, the following techniques are usually used.

5.3.1. Implementing a Web Application Firewall (WAF)

Path traversal attacks are a type of attack that web application firewalls (WAFs) are intended to defend against. Path traversal payloads can be detected and blocked in requests by a WAF, preventing them from getting to the web application. Further lowering the likelihood of a successful attack, WAFs can be set to block requests that include suspicious characters or patterns. Furthermore, a lot of WAFs have capabilities like rate restriction and IP blocking that can stop attacks involving repetitive or automated requests.

5.3.2. Regularly Update and Patch Systems

Regular patching activities are one of the common ways to fend off the majority of the attacks. This includes operating systems, web servers, and web applications. Patches often include bug fixes for known vulnerabilities, including those that could be exploited by any kind of attack. By keeping systems up to date, organizations can reduce the risk of a successful attack.

5.3.3. Logging and Monitoring

A comprehensive logging and monitoring mechanism should be in place for the online application. The logs should be inspected regularly to detect any abnormal activity or attempted assaults. All discovered

issues should be examined and resolved as soon as possible. Attackers attempted attacks might be audited into the logs and the monitoring team in the application can review those and track the attacker or identify his intentions.

6. Conclusion

Path traversal attacks are a common vulnerability in today's web applications and can have devastating consequences if left unchecked. In this paper, we have discussed the characteristics of path traversal attacks, as well as different techniques used by attackers to exploit these vulnerabilities. We have also discussed prevention strategies that can be implemented to protect applications from path traversal attacks. By following best practices and implementing these strategies, organizations can minimize the risk of a successful attack on their web applications. Upon analysis, we can conclude that path traversal attacks can be avoided and mitigated if proper checks are implemented in all phases of software development. Primarily, if proper input validation checks are in place, there can be a significant decrease in successful attacks. The best approach to input validation is to use an allowed list of acceptable inputs that strictly conform to specifications. Reject any input that does not conform to the specifications or transform it into something that does. Consider all relevant properties when validating input, such as length, type, acceptable values, syntax, consistency, and conformance to business rules. Use stringent allow lists for filenames and file extensions to avoid weaknesses. Beware of sanitizing mechanisms that may remove characters required for some exploits. Decode and canonicalize inputs before validation and use the lowest privileges possible to accomplish tasks. Use isolated accounts with limited privileges, create mappings between fixed input values and acceptable objects, and run code in a sandbox environment to restrict access to files and commands.

References

[zaproxy] OWASP ZAP – Path Traversal. https://www.zaproxy.org/docs/alerts/6-1/

[NIST2018] CVE-2018-18778 Detail. https://nvd.nist.gov/vuln/detail/CVE-2018-18778

[owasp] Path Traversal | OWASP Foundation. https://owasp.org/www-community/attacks/Path_Traversal

[sectester] ** DotDotPwn - The Directory Traversal Fuzzer **. http://dotdotpwn.sectester.net/

[snyk] Directory Traversal in org.apache.struts:struts2-core | CVE-2016-6795 | Snyk. https://security.snyk.io/vuln/SNYK-JAVA-ORGAPACHESTRUTS-30778

[portswigger] What is directory traversal, and how to prevent it? | Web Security Academy. https://portswigger.net/web-security/file-path-traversal#:~:text=Directory%20traversal%20(also%20known%20as,and%20sensitive%20operating%20system%20files.

[Sagi2019] Sagi, Daniel. CVE-2019-11246: Another kubectl Path Traversal Vulnerability Disclosed. https://blog.aquasec.com/kubernetes-security-kubectl-cve-2019-11246

[Kohnfelder2022] Kohnfelder, Loren. Heymann, Elisa. Miller Batron P. Introduction to Software Security. 2022-01 https://research.cs.wisc.edu/mist/SoftwareSecurityCourse/Chapters/3_3-Directory-Traversal.pdf

[Flanders2019] Flanders, Michael. A Simple and Intuitive Algorithm for Preventing Directory Traversal Attacks. 2019-08-13. https://arxiv.org/pdf/1908.04502

[acunetix] What is a Directory Traversal Attack? https://www.acunetix.com/websitesecurity/directory-traversal/