CVE-2019-11510

Arbitrary file reading vulnerability in Pulse Connect Secure

Martin Pretz, 7060026

May 12, 2021

Abstract

1 Introduction

2 Description

The CVE-2019-11510 is a vulnerability that allows attackers to get arbitrary file reading access after they have send a special URI. [1]

This vulnerability has been assigned the maximum CVSS classifier of 10.0 Critical [2] **TODO: Warum dieser CVSS Score?**

This vulnerability is part of the CWE-22 class which is associated with "Path Traversal". That means that by explicitly specifying an abnormal path that is "[...] intended to identify a file or directory [...]" [3] it is possible to gain access to that file or directory without owning the required rights. This is made possible because the software uses an externally specified path and due to the way the software proceeds with that path. The effect is that the software resolves the given paths to files or folders that lie outside the resitriced directory. [3]

3 Threat classification

TODO: CWE und CVSS einordnung

Regarding the STRIDE threat model, this vulnerability can be classified as an *information disclosure* in the first place, because primarily the attacker may read files unauthorized. But in the second place this vulnerability also leads to an *elevation of privileges* [4] because admin credentials are stored in plain-text inside an unrestricted file and because an attacker can arbitrary read files he can easyly obtain the admin credentials. [5]
Even though one could argue that *information disclosure* "only" affects confidentiality and privacy, this vulnerability justifiably was classified with the CVSS 10.0 Critical, because the *elevation of privileges* additionally affects every single protective goal. [4]

4 Analysis of the Protective Goals

4.1 Confidentiality

4.2 Integrity

4.3 Availability

4.4 Authenticity

4.5 Identify

4.6 Deniability

4.7 Privacy

5 Path traversal

Allthough path traversal was already mentioned in the previous chapter, one must take a closer look at path traversal in order to fully understand how it works. This will be done during the course of this chapter.
As previously mentioned path traversal attacks are used to access files or directories that lie outside the web root folder. This is usually achieved by using a path sequence containing the so called "dot-dot-slashes" (../) and its variations. Even though the attacker has access to the file system, he is also limited by the operating system's access control (meaning locked or in-use files). [6]

6 How can this vulnerability be exploited?

As already mentioned in 2, in order to exploit the vulnerability the attacker must send a request (e.g. via HTTP) to the target server that contains a path sequence used for path traversal (see chapter 5) and a special URI for the file that the attacker want to gain access to. [5]
"When a user logs into the admin interface of the VPN [...]" [5], the password is stored as plain-text within a MDB file (Microsoft Access Database). The corresponding file can be found at */data/runtime/mtmp/lmdb/dataa/data.mdb*. Since

the attacker already has arbitrary access to all files of the system he can easyly obtain the admin password. With this information the attacker could perform further attacks, e.g. exploiting the CVE-2019-11508, which allows an attacker to upload harmful files while using the credentials he obtained beforehand, since the credentials actually belong to an authanticated user. [5]
In other cases Kevin Beaumont reported that this vulnerability has lead to

7 Example usage

TODO: hier beispiel wie der exploit durchgeführt wird (mit programm)

8 Affected

All versions from between 8.2 to 8.2R12.1, 8.3 to 8.3R7.1 and 9.0 to 9.0R3.4 are affected. [1]

TODO: Länderliste

9 Prevention

Regarding the prevention of a such a path traversal attack, multiplie measures can be effectiv to different extents. Obviously the best way to prevent this type of attack is to operate the system/program without expecting the user to input a path. But since there are applications that inevitably require the user's input, this challange has to be adressed in a different way. [6]
In those cases it is advised to make sure, that the user can not specify all parts of the path, instead embeding the user input self defined paths. Another way could be to normalize the path before proceeding eliminating all irregularities. [6]
Last but not least, the usage of chroot jails or code access policies is advised in order to resitrict where files can be obtained or saved to. [6]

10 Mitigation

During the course of this chapter it will be discussed how this vulnerability can be mitigated. Thereby, the analysis is divided in three different aspects which are the implementation, the architecture and design, as well as the operation.

10.1 Implementation

One of the most important methods of prevention is the input validation. It is recommended to define acceptable input based on the "accept known good" validation strategy. So all input either must directly comply with the specifications or must be converted in a way that does comply. It is of crucial importance that the validation is performed as accurate and comprehensive as possible, otherwise the validation misses its point. More precisely, looking "[...] exclusively for malicious or malformed [paths]" [3] is not what should be done during validation. Instead try to limit directly what characters are allowed so that the attacker cannot even enter a harmful path. Concrete it should be prohibited to enter ".." character, instead only allow a single "." character. File extensions should also be concerned, meaning that the file extensions is not allowed to be included in the path but must be selected from a predefined list. This way it can be restricted what types of files can be read. Furthermore directory separators such as the the "/" or the "\" character should be prohibited. This has to be done not only once per path but multiple times in order to minimize the error rate. For example if the validation checks for the string "../" within the path ".../...//", there would still remain the "../" string, even tough "../" has been removed twice from the original path. [3]
Furthermore warnings or error messages should only contain as little information as possible that are exclusively useful to intended recipients. This way, attackers cannot obtain detailed information about the inner workings of the system. [3]

10.2 Architecture and Design

This section will summ up the most common decisions regarding the architecture and design of a programm in order to prevent or at least mitigate the impact of the vulnerability.

If any user input is to be validated on the client side, it is advised to ensure that this validation process is duplicated on the server side to prevent that the attacker can use CWE-602 vulnerabilities he makes use of CVE-2019-11510. This would allow the attacker to bypass client sided validations and thus the attacker would still be able to send an spcial path to perform a path traversal. [3]
When running the software in production, ensure that the program the lowest privilege level that is necessary to achieve the task. This way, even if an attacker sends an harmful to the server, the software which is proceessing the input is limited due to its low privileges. Though this method cannot prevent this attack from happening, it can limit the impact.[3]
Access to files can be prevented if they are stored outside the web applications root. However if this is not possible, a different solution would be to store files in a separate directory and to use the web servers access control features to restrict the access, preventing attackers from directly requesting files. For example, defining a fixed constant in each calling programm, checking if this constant does exist in the file and if absent the file was directly requested and should be

closed. [3]
If user input is required and the number of possible inputs is limited, assign/map an ID to every possible input value accepting only the predefined IDs and rejecting requests using undefined IDs. [3] "For example, ID 1 could map to "inbox.txt" and ID 2 could map to "profile.txt"." [3] This method is called an "Acces Reference Map".

10.3 Operation

Content of this section include methods of prevention for operating the program in production environment.

The use of a dedicated application firewall can be effectiv to detect and prevent attacks trageting this vulnerability. This is escpecially useful, if the vulnerability cannot be fixed by yourself because the program belongs to a third party. [3] It has to be noted that this can only act as a temporary solution until a patch is available.
Additionally, the program can be run inside a sandbox environment in way that separates it from the operating system. This will heavily restrict the accessable files and directories. But it has to be noted that this solution only reduces the impact an attack can have on the operating system, while the application itself can "[...] still be subject to compromise." [3] Furthermore the degree of effectiveness depends on the scope and possibilities of the sandbox. Eligible sandbox systems could be Unix chroot jail, SELinux or AppArmor. [3]

11 Detection

This way, it is not just prevented that an attacker can acces files, furthermore it can be detected if an file was unauthorized requested directly, providing an detection method.

12 Solution

Since this vulnerability is caused by the unintended behaviour of Pulse Connect Secure while resolving provided paths and therefore the internal implementation of this software, the vulnerability has been removed by a software patch provided by Pulse Secure.

TODO: Alternativen?

13 Conclusion

References

[1] *Nvd - cve-2019-11510*, https://nvd.nist.gov/vuln/detail/CVE-2019-11510, (Accessed on 05/10/2021).

[2] *Pulse vpn vulnerability analysis (cve-2019-11510) — awake security*, https://awakesecurity.com/blog/pulse-vpn-vulnerability-analysis-cve-2019-11510/, (Accessed on 05/11/2021).

[3] *Cwe - cwe-22: Improper limitation of a pathname to a restricted directory ('path traversal') (4.4)*, http://cwe.mitre.org/data/definitions/22.html, (Accessed on 05/10/2021).

[4] D. J. Schneider, *It-sicherheit - dr juergen schneider - dhbw mannheim - angewandte informatik - ss 2021 - 2 - angriffe und schwachstellen*, 2021.

[5] *Cve-2019-11510: Proof of concept available for arbitrary file disclosure in pulse connect secure - blog — tenable*, https://de.tenable.com/blog/cve-2019-11510-proof-of-concept-available-for-arbitrary-file-disclosure-in-pulse-connect-secure, (Accessed on 05/10/2021).

[6] *Path traversal — owasp*, https://owasp.org/www-community/attacks/Path_Traversal, (Accessed on 05/10/2021).