# Summary Paper

### Intro

The author describes the current state of web-based vulnerabilities, particularly in the directory traversal region, which constitutes 22% of 2017 web vulnerabilities reported to the National Vulnerabilities Database. The paper introduces a better way of lightweight and cross-platform algorithm to detect traversal attacks across web server providing web service to clients. Also, lack of exhaustive testing in the field of traversal attacks due to a comprehensive list of directory access on HTTP web server makes the standard defensive approach very much resource intensive. Attackers use directory traversal attacks to run unsigned and arbitrary code on the backend servers. To combat most of the scenarios of user paths and directory traversal, the author recommends to parse every string, query of web URL resource against its parser to filter out certain keywords, thus preventing implicit code execution or redirection internally.

### Background

Web 2.0 is on the rise and security vulnerabilities are also on a steady rise for the past few years, emphasizing the 2016 - 2017 report by National Vulnerabilities Database (NVD) which states having 127% of vulnerabilities increase and 51% of that pie has been consumed by Web application space. As most vulnerabilities of web applications usually refer towards ‘SQL Injection’ and ‘Cross-Site Scripting’ attacks and preventions, the author mentions that very few steps have been taken for ‘Directory traversal’ vulnerabilities, even though it constitutes about approximately 1/2 of the web application vulnerabilities stated by the NVD report[2].

The author explains the premise of the Directory system in general for specific Operating system. In the example, he used the Linux Operating system directory structure with user alias as “alice”.

So explaining with Linux directory space consisting of ‘~/‘ root directory, user, usr, bin, etc, www and var, etc., He further states that if the webserver is hosted for delivering web pages over web protocol then ‘/www/‘ folder would be the best bet for accessing the file or resource for the client requests.

But instead of directly accessing ‘./www/index.html’ if the same request gets called using ‘../user/alice/‘ while making a return callback in php, improper hardening or parsing of the input will return all of those other directories which shouldn’t be attainable in the first place.

Author also mentions that many system administrators have web hardening for not letting client request filter out certain keywords and further points out that those solutions are either bloated (not lightweight) and also not cross-platform solutions as real paths, absolute paths changes according to the webserver solution which has been deployed for servicing user requests. The author proposes a lightweight algorithm to counter all these shortcomings of the current state of the directory traversal vulnerability.

Technology

The author assumes that web servers would be using the Linux base directory file system and also using ‘httpd’ service for delivering http responses of client requests. While also stating current mitigations used in ‘mini\_httpd’ and ’thttpd’, which uses ‘de\_dotdot’ algorithm to mitigate the vulnerability to a certain extent. Also, canonicalizing the user string path combined with whitelisting of specific resources defined by the webserver admins are currently the backbone of the problem solving for this vulnerability in general.

Author uses his algorithm described in ‘Algorithm 1 and Algorithm 2’ while also discarding wildcards like ‘?’ and ‘\*’ in the user-supplied string for parsing in his algorithm which is been implement in C language hosted on Github and recommends to use symbolic links functionality to the whitelisted path array or file which would check against specific paths[3]. Depending on the path’s overall access target the interpreter would either replace ‘..’ with ‘/‘ etc. thus ending the need for more verbose type of checks required and also working with cross-platform URL checking at the same time with its conversion of parsed stack to string to ID against the whitelisted domains or path list.

The Engine was running on hardware which had specifications as follows :

Processor: Intel i5-7200U Dual Core CPU

RAM: 8 GB

Engine: KLEE [4]

He describes his algorithm with proof of correctness against four different definitions and also three Lemma with proof for specific problems. Also proving his theorem with regards to canonicalization of String S1, S2 in conjunction with whitelisting, which equates to being true if and only if both path strings S1 and S2 refers to the same path resource on the file system[5].

### References

[1] A Simple and Intuitive Algorithm for Preventing Directory Traversal Attacks [Online]. Available: https://arxiv.org/abs/1908.04502

[2] Fortify Security Research Team Link: https://microfocus.com/

[3] Github Link [Github]. Source: https://github.com/flandini/sanitize

[4] KLEE Engine, PC Specifications [Online] https://dl.acm.org/citation.cfm?id=1855741.1855756

[5] Figure 4 & 5 which describes the algorithm stack comparisons and pop methods usage for discarding the unwanted url strings [Paper]