Data Leakage Protection

Ziyad Ahmed Mohammed[1], Arshpreet Singh[2],

Manan Rajiv Sahni[3], Abdul Moid Khan Mohammed[4],

Syed Suhaib Hussain[5], Mohammad Abdul Malik Fazal[6]

[1]Dept. EEE, Chaitanya Bharathi Institute of Technology (CBIT), Hyderabad

[2,3] Dept. CSE, Manipal University Jaipur

[4]Dept:- CSE, Methodist College of Engineering and Technology

[5]Dept:-IT, KJ Somaiya College Of Engineering,Vidyavihar,Mumbai

[6]Dept – CSE, University College of Engineering , Osmania University

[1]Email:- ziyadahmed910@gmail.com , [2]Email:- sarshpreet07@gmail.com

[3]Email:- manansahni100@gmail.com , [4]Email:- skhan852000@gmail.com

[5]Email:-suhaib.hussain26000@gmail.com ,[6]Email- malikfazal268@gmail.com

1.INTRODUCTION

1.1.Theoretical Background

Applications run businesses.They have become so crucial that it is almost impossible to imagine an organization running without them. Applications run on servers and if we look almost 2 decades back we would require a physical server to run them , but the issue with that was the size and speed of the server required is unpredictable buying a huge server only to utilize 5-10% of its capabilities was just a waste of capital , thus the concept of virtualization came to be it enables tones of apps to run on the same physical server. Consider virtual machine as a slice of physical hardware , so all has the portion of the cpu , memory and disk space.The came the concept of containers they are very similar to a Vm but with one major difference THE HOST OPERATING SYSTEM. Docker is a software platform for developers and sysadmins to deploy , develop and run apps with containers.Now using this helped utilize the resources more efficiently but there at times were security holes in container configurations that could lead to compromised situations.

1.2.Motivation

Dockers and containers are one of the emerging technologies on various applications. As the use of docker increases, the severity of malicious attacks on the docker containers increases.

Creating containers in the system helps utilize the resources in an efficient way , but sometimes security holes in container configuration can lead to compromised situations which could lead to huge losses. Container security plays a crucial role in safeguarding our services and resources. We plan to build a proof of concept in order to document our findings so as to offer some insights into a solution for increasing the security of containers on a wider scale. We plan to improve the security of the application within the container and control the malicious behavior,

1.3.Aim of the proposed Work

Docker and containers brought about a major change in the industry helping not only major corporations but also small firms that initially could not have enough capital to put out really good applications that bring about a lot of positive changes in daily usage , but the security plays a crucial role in safeguarding all of this so we expect to provide a hands on recommendations to tighten the security on containers.

1.4.Objective(s) of the proposed work

We work on improving the security of an application within the container and attempt to control the malicious behavior. Our application will make users aware about the vulnerabilities that could potentially exist in the file.this is to allow for a prompt action to be taken to reduce or eliminate vulnerabilities. Credential leaks risks such as attack on your databases and third party services could be prevented.

2.LITERATURE SURVEY

2.1.Survey of the Existing Models/Work

[1] The research presents a machine learning-based method for categorizing documents as confidential or non-confidential. The technique entails utilising the SVM algorithm to categorize three types of data: private, public, and non-enterprise. This data is represented by the most common binary weighted unigrams detected across the corpus. According to the findings, the model properly identified 97 percent of all data breaches.

[2] The research presents a methodology to identify the various vulnerabilities present in the docker container image. It used the Self Organizing Map approaches along with some traditional machine learning technologies that provided the most amount of vulnerability detection.

[3] The paper presents a new model which is superior in the following aspects , detection of small sections of information in non-confidential documents and it generates an easy to understand and modifiable model.

[4] Their study has demonstrated how the psychological components capacity, opportunity, and incentive combine to create employee actions that are thought to aid in the prevention of data leakage occurrences within banks. According to their results, competence is critical for avoiding data leakage in financial institutions.

[5] In this research, improved whole data fingerprinting is provided to address flaws in fingerprints created by conventional data hashing. Ordinary fingerprints are susceptible and may be circumvented even with modest changes to the original data; hence, k-skip-n grammes are employed to generate modified fingerprints. The k-skip-n grammes present a reliable way for recognising the original data even after data change (i.e., addition, subtraction, word synonyms). In this approach, both confidential and nonconfidential documents are processed to yield fingerprints, with nonconfidential materials producing nonconfidential k-skip-n-grams. Non-confidential k-skip-n-grams aid in the elimination of unneeded n-grams in confidential papers. In practically all testing conditions, the suggested technique outperformed conventional complete fingerprinting methods.

[6] Docker has become progressively famous on the grounds that it gives proficient containers that are effortlessly run by the host system. However, the paper had analyzed the docker hub and found various vulnerabilities and security risks as well as potential attacker scripts present in docker hub. The paper served as a stepping stone for further research on securing docker.

[7] In recent years, the world of technology as well as the topics of computer science has developed rapidly. To meet the growing needs of students, there shouldn't be a need for teachers to personally supervise code creation, working and its errors. Rather, it is possible to evaluate the student’s assignments using an automated bot. The paper here showcases this usecase as an example to the utilization of docker as a sandbox environment that is capable of understanding and running the codebases, and evaluating them as per some parameters.

[8] This paper signified the use of traffic morphing which essentially means either splitting or padding of packets so as to evade attacks born from traffic analysis. The logic involved a transformative matric that decided the size of the packets. It was an effective way of preventing attacks in encrypted HTTP flow networks.

2.2.Summary/Gaps identified in the Survey

[1] The model had a simple logical categorization that required a specific description on the data which outlined its type. This type of categorization had less flexibility in terms of dealing with variation in nature and content of the data and also because of the degree of discretion exercised on various levels of hierarchical positions in an organization or companies.

[2] The model can only have a vulnerability sample size of 28 which is quite less in terms of identifying all the vulnerabilities.

[3] the model gives only a half-done solution to the problem already at hand and is unable to fully deal with the problems of rephrased texts.

[4] this method can only protect you within the organization and not an outsider

[5] intensive indexing is required for all confidential and nonconfidential documents. Thus, the extra storage and processing capabilities required are a major drawback for this method.

Summarizing all the papers we came to a conclusion that either some of the data is leaked or due to some vulnerabilities there is a backdoor available for data to be leaked or tempered with. Hence we came up with the idea of blocking any data from going out no matter the backdoor or any direct attack.

3.Overview of the Proposed System

3.1.Introduction and Related Concepts

In our day to day interaction with software applications, we can realize one important aspect of software. Security of the application in our current digital age has become an all time requirement as well as a formidable challenge. So in this context, if we observe the development of a particular software at various stages, we will notice that even the basic idea of choosing the architecture is affected to a great extent by the existing security implementations on similar architectures as well as proposed extent of features for which the software is being built for.

In order to tackle this problem, we are trying to use security as a practice, rather than just an implementation. By this we mean, that just like a software development life cycle has different phases, similarly, by maintaining a set of security practices at each phase, we can make sure that whatever software we progress through at each stage has a decent amount of security in effect. For example, while we write tests, instead of just checking, a basic set of requirements, that the software might be performing, we can also choose to have tests/ checks for any set of configuration discrepancies, or maybe a suite of test cases where the software's credibility might be compromised, in case its not implemented properly. In the usual scenario, these tests can be built in various stages, starting from some security requirements, and gradually, all different bug fixes and integrations could have their separate suites of tests to counter the different problems the software faced in various stages of its lifecycle. It can be easily seen that tests are just one fundamental part of the software development lifecycle, which is commonly used as a failcheck to prevent software mishaps from happening. For all different phases, like architecture designing, requirements planning etc, a series of steps and some compulsory practices could help develop clean and bug free software with lesser vulnerabilities. Since a lot of these practices can be memonized, hence, we can automate these steps to be performed uniformly and independently, on a lot of the software we write to ensure an optimal software development life cycle. For our starting phase, we target requirements, which is the starting point of any idea. And further along, after the software is built, it might still have a lot of complex use cases that could have loopholes, easy to be looked over by the naked eye. To tackle such a situation, we decided to build an innovative application that focuses more on preventing leakage of information rather than the obstruction of the attacks received from malicious agents.

And as an evolving feature, we wanted this development to have its own environment for testing software. So that instead of messing with the original baseline code, we decided to build a small sandbox environment that could be used to extend the same as well as allow for multiple coherent access to the environment.

3.2.Framework, Architecture or Module for the Proposed System

Our approach tries to achieve the first phase by giving a user the list of vulnerabilities of a docker image. This provides the user with more information so that he can be more well informed when using the docker image and can take necessary steps to either remove or reduce the vulnerabilities.

The second phase involves building a sample setup of a simple server(leaky) and an auto server secret detection function. This is to simulate a scenario in which when pinged or given some command the leaky server leaks out the secret information that a malicious user extracts.

After simulating our problem statement, we intend to build a reverse proxy by using the traefik tool. The tool would allow us to manage this scenario as it sits between the original leaky server and the user. It ensures that when the endpoint is pinged, an empty response is given instead of the secrets. For simulation purposes we have used only 2 secrets and a simple golang server, but it could be replicated easily with any server and for many secrets.

And finally for system maintenance from a security point of view, a customized setup to collaborate and test scripts.

3.3.Proposed System Model(ER Diagram/UML Diagram/Mathematical Modeling)

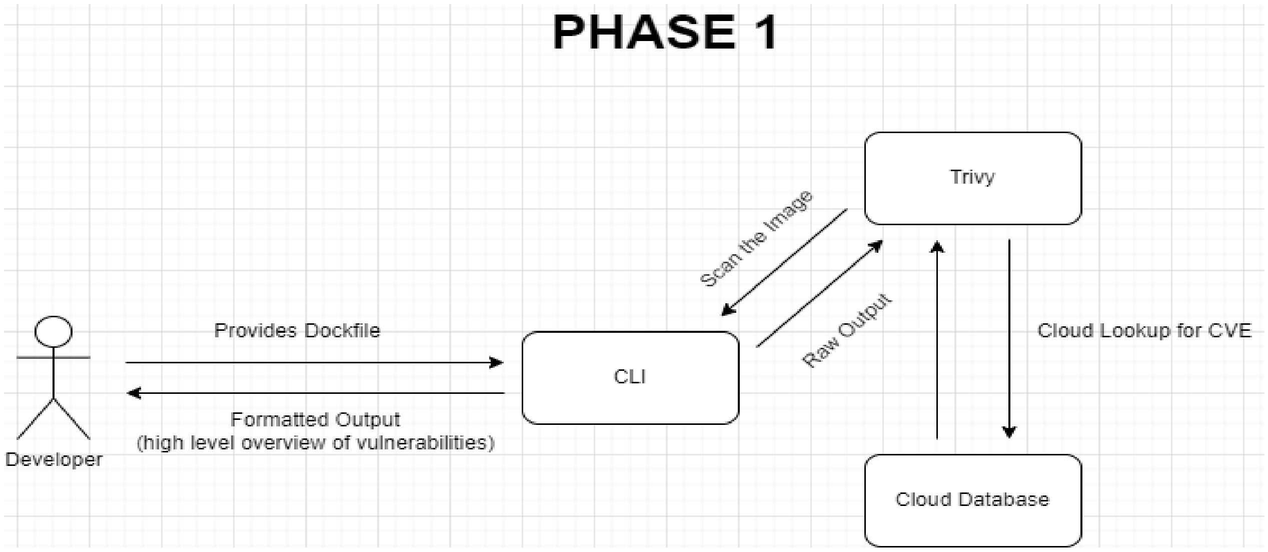


Fig1 : Phase 1

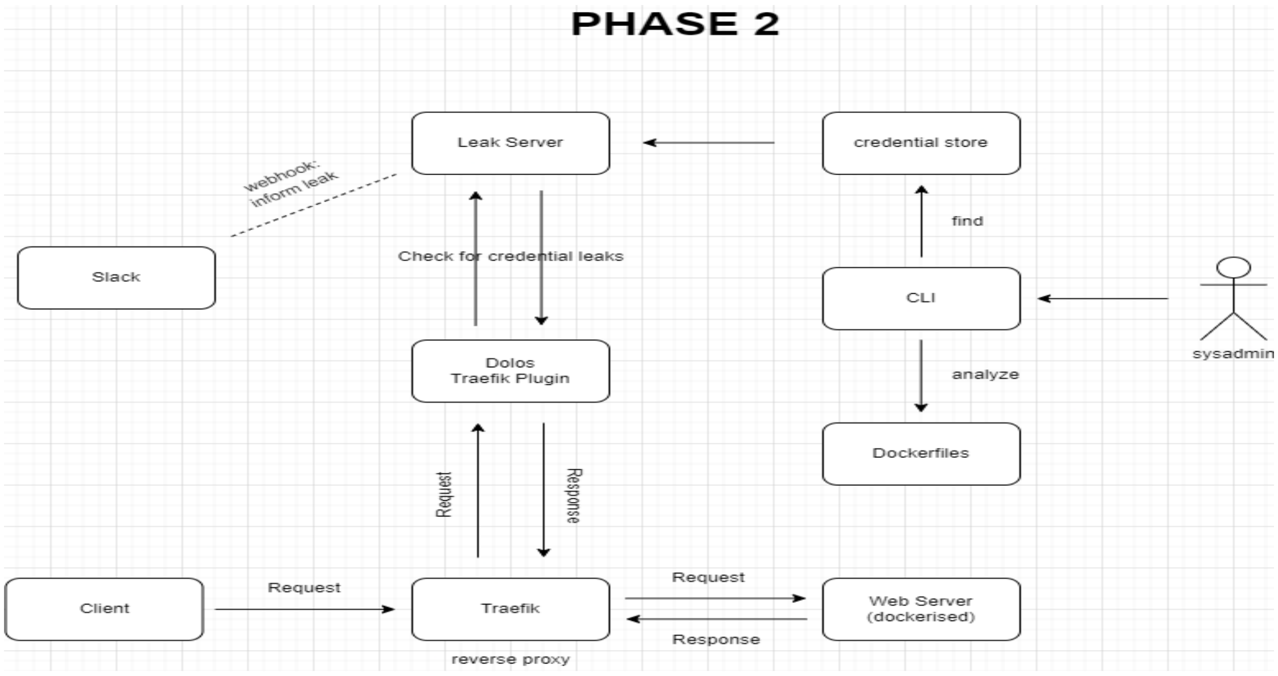


Fig2 : Phase 2

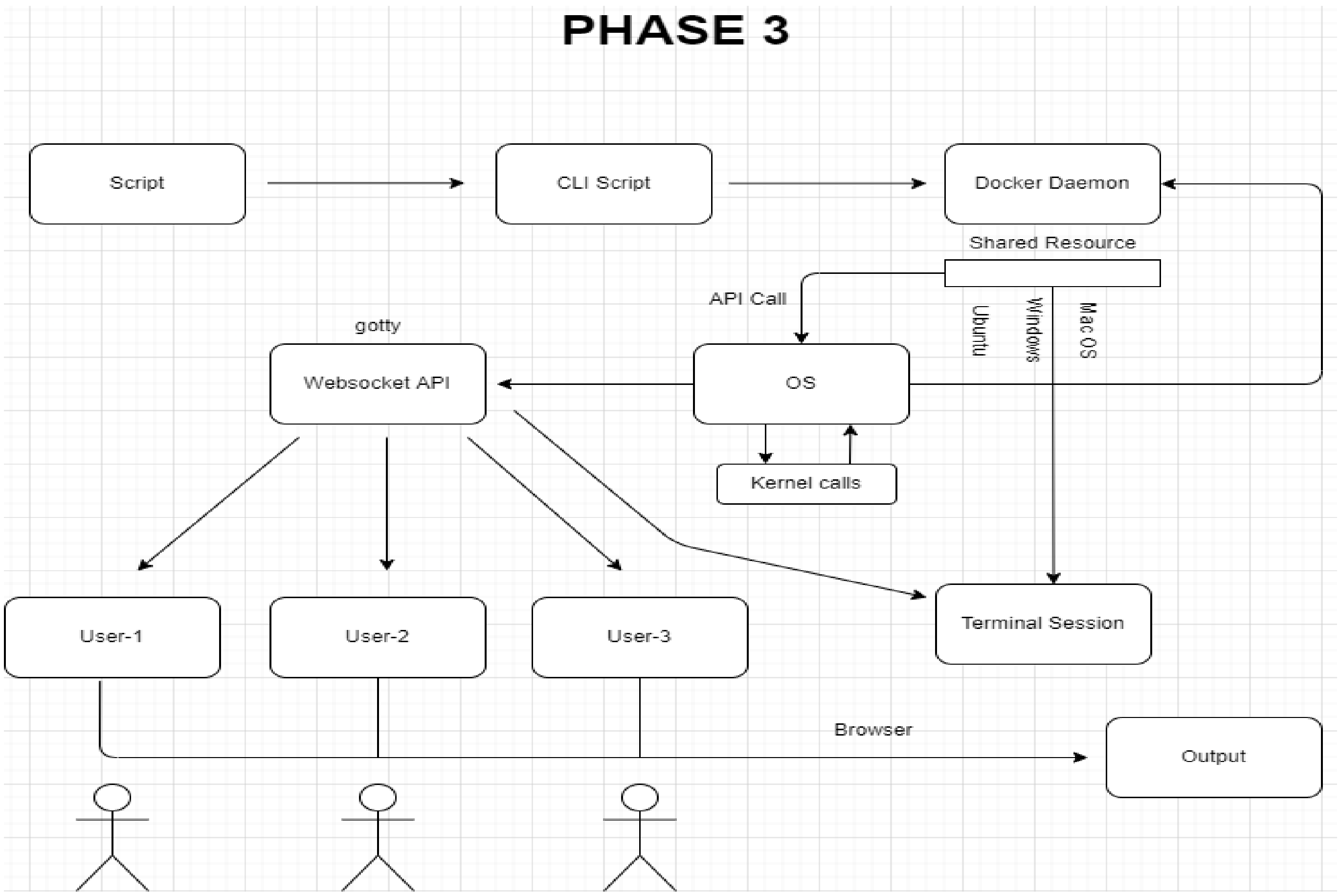


Fig3 : Phase 3

4.Proposed System Analysis and Design

4.1.Introduction

The system is designed to heighten the importance of viewing software security not just as a single stage approach but as a practice that takes place in every stage. This is to ensure that the software has some security guidelines and capabilities in place at every stable product stage. The project starts in its initial phase where it scans the requirements and the system and tries to find the vulnerabilities present in it. After recognising the various vulnerabilities that could be potentially exploited, the same could be either handled perfectly or the threat they pose could be minimized substantially.

Then, the second phase depicts an approach where the goal isn't exactly to defend the system from various known and unknown potential attacks but rather creating a stop gap that marks sensitive information unapproachable and therefore could be used in tandem with other implementations for increased security.

This provides an added layer of safety that prevents leakage of information after the malicious attackers have breached the system's inbuilt security.

Last phase includes a simple implementation of a docker based sandbox environment that allows for easy testing, increased isolation from host system and greater ease in handling scripts from unknown sources

4.2.Requirement Analysis

4.2.1.Functional Requirements

4.2.1.1.Product Perspective

The objective of the paper is to ensure that the security of the software is observed as a practice at every stage rather than as a single stage consideration. This allows for the security of the software to be enhanced greatly and provide better utility overall.

4.2.1.2.Product features

The product is capable of analyzing vulnerabilities of any system, provided that it is given the system build info. It is also capable of preventing sensitive information from exiting the system, which would otherwise be at risk due to attackers. Lastly, it also has a software sandbox environment that allows for testing of the software, being developed, without much hassle and space consumption.

4.2.1.3.User characteristics

The general user would be a software developer, quality tester or a software analyst who would be looking to either build the software or analyze the quality and safety of its live deployment. However, this does not limit the user to these professions, rather can be extended to any user that involves himself into developing or constructing any specific software.

4.2.1.4.Assumption & Dependencies

The user should be knowledgeable on simple software lifecycles and procedures as well as should have good software practices. The current implementation has been only tested out on a linux system, and hence further analysis needs to be done for other operating systems as well for software compatibility. For dependencies, the system should have gcc, a python compiler and golang compiler.

4.2.1.5.Domain Requirements

Python, Golang, Docker, Docker Compose

4.2.1.6.User Requirements

A linux operating system with golang, docker and python installed

4.2.2.Non Functional Requirements

4.2.2.1.Product Requirements

4.2.2.1.1.Efficiency (in terms of Time and Space)

For first phase, any dockerfile is scanned over from the cloud in a fixed interval of one minute

For second phase: The latency is as low as 50ms for the requests being processed by the middleware

For third phase: The sandbox takes 10-15 seconds to start up(at max) on any system with docker running(this is 50% less compared to starting a VM)

4.2.2.1.2.Reliability

All three steps are reliable enough to give consistent results

4.2.2.1.3.Portability

The use of docker allows us to build portable images, that can be shifted to remote linux systems and worked upon

4.2.2.1.4.Usability

The system is easy to use and can be used with minimum amount of technical knowledge

4.2.2.2.Organizational Requirements

4.2.2.2.1.Implementation Requirements (in terms of deployment)

N/A

4.2.2.2.2.Engineering Standard Requirements

N/A

4.2.2.3.Operational Requirements (Explain the applicability for your work w.r.to the following operational requirement(s))

- Economic : Software can be built more securely, hence reduces the impact of economic implications on development
- Environmental : Lesser utilization of resources as software is built sustainably.

- Social : N/A
- Political: N/A
- Ethical : Ensures better security practices
- Health and Safety : N/A
- Sustainability: Allows for building of sustainable and more secure softwares
- Legality: N/A
- Inspectability : Allows to inspect the code standard based on the vulnerability observed

4.2.3.System Requirements

4.2.3.1.H/W Requirements(details about Application Specific Hardware)

Linux OS which have 4GB Ram or above for optimal performance

4.2.3.2.S/W Requirements(details about Application Specific Software)

Python, Golang, Docker, Docker Compose, Trivy, Traefik

5.Results and Discussion

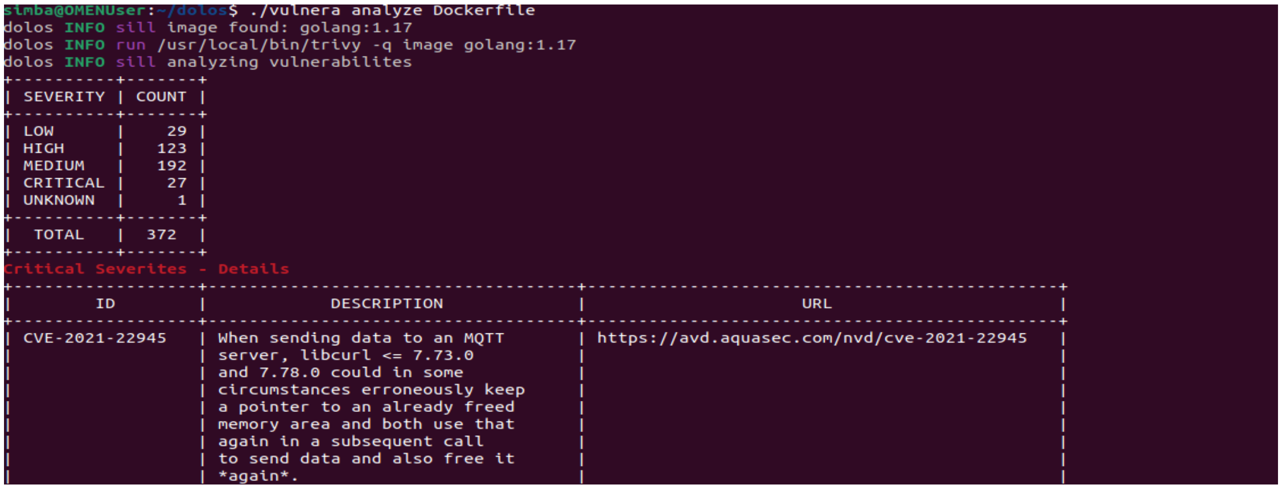
As highlighted before, the project could be seen in 3 phases. These phases being :- 1] Vulnerability Scanner, 2] Prevention of exiting of Sensitive information and 3] Sandbox Environment

The vulnerability scanner identifies the various components,requirements, architecture,etc of the system provided and therefore gives a list of all potential vulnerabilities identified and ranks them based on severity.

The second phase works in the manner of preventing information from leaking out. It works effectively in situations where the attackers have entered the system by breaking the defense mechanisms of any software. It works with the help of a tool called traefik which is a contemporary HTTP reverse proxy and load balancer that makes microservice deployment simple.

Lastly we built a dockerbased sandbox environment with a link to a database of free open source github projects that are present on github. This allows the user to not only test scripts from unknown sources but also can access scripts available online without the fear of malicious scripts. This functionality is because the environment behaves as if it's an isolated instance with little to no traceable connection to its original host computer.

Phase 1: Vulnerability Scanner

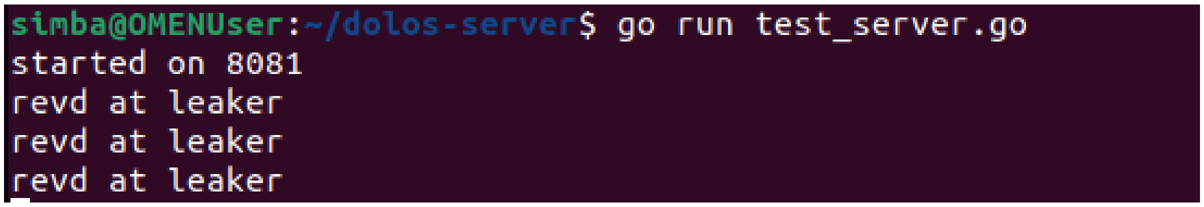


```
simba@OMENUser:~/dolos$ ./vulnera analyze Dockerfile
dolos INFO sill image found: golang:1.17
dolos INFO run /usr/local/bin/trivy -q image golang:1.17
dolos INFO sill analyzing vulnerabilites
+----------+-------+
| SEVERITY | COUNT |
+----------+-------+
| LOW      |    29 |
| HIGH     |   123 |
| MEDIUM   |   192 |
| CRITICAL |    27 |
| UNKNOWN  |     1 |
+----------+-------+
|  TOTAL   |  372  |
+----------+-------+
Critical Severites - Details
+------------------+-------------------------------------+----------------------------------------------+
|        ID        |             DESCRIPTION             |                     URL                      |
+------------------+-------------------------------------+----------------------------------------------+
| CVE-2021-22945   | When sending data to an MQTT        | https://avd.aquasec.com/nvd/cve-2021-22945   |
|                  | server, libcurl <= 7.73.0           |                                              |
|                  | and 7.78.0 could in some            |                                              |
|                  | circumstances erroneously keep      |                                              |
|                  | a pointer to an already freed       |                                              |
|                  | memory area and both use that       |                                              |
|                  | again in a subsequent call          |                                              |
|                  | to send data and also free it       |                                              |
|                  | *again*.                            |                                              |
```

Fig4 : Vulnerability Scanner Script

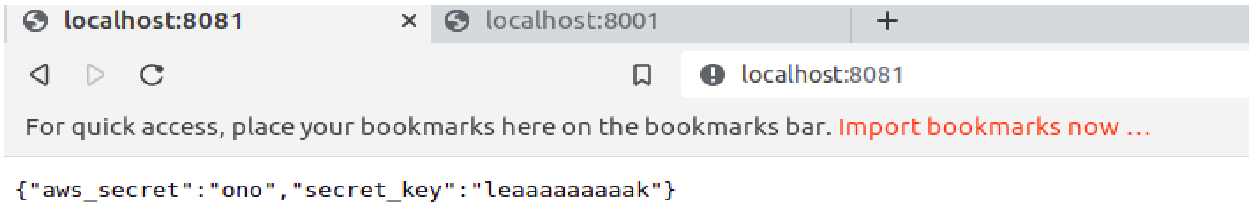
The figure depicts the vulnerability analyzer script that is analyzing the various vulnerabilities present in a previous version of golang as well as ranking them based on its severity. Lastly it provides more details about each vulnerability as well as a credible url that explains the vulnerability further.

Phase 2: Prevention of exiting of Sensitive information.



```
simba@OMENUser:~/dolos-server$ go run test_server.go
started on 8081
revd at leaker
revd at leaker
revd at leaker
```

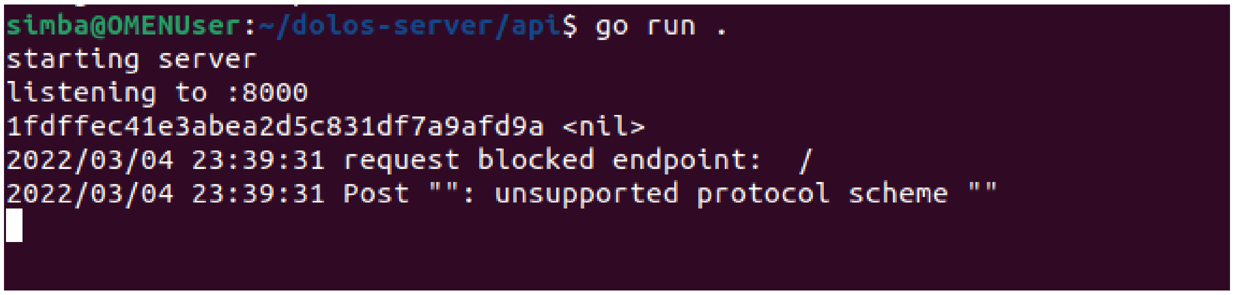
Fig5 : Script running for Leaky Server (Test Server)


localhost:8081
localhost:8001
localhost:8081
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now ...

```
{"aws_secret":"ono","secret_key":"leaaaaaaaaak"}
```

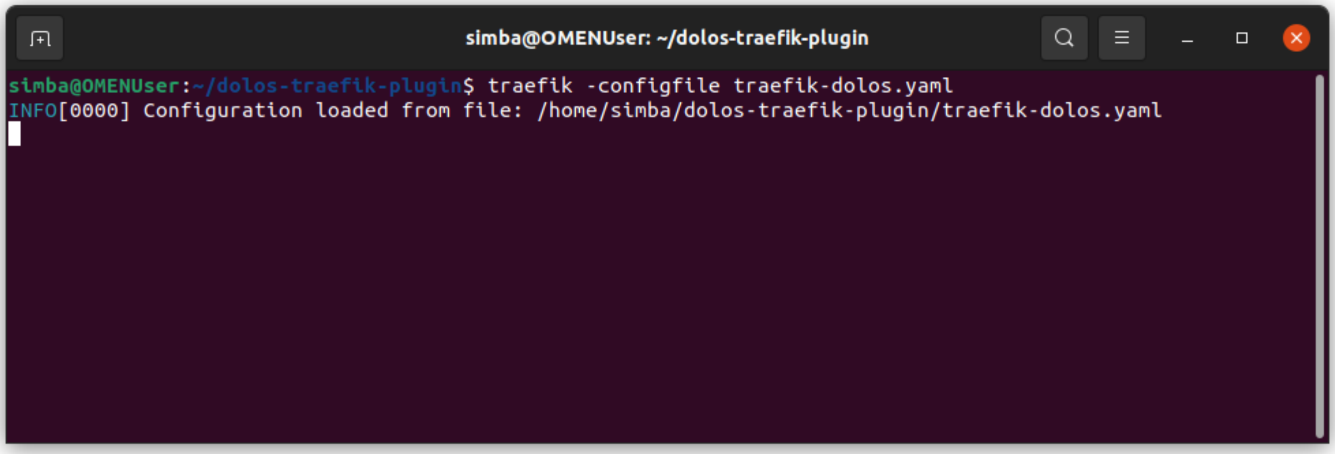
Fig6 : Leaky Server at port 8081

The figure describes the outcome when the leaky server is attacked and the secret information is easily extracted from the server, which should be prevented. For the sake of depiction we are creating only two sensitive pieces of information that are being accessed, however in a real time scenario this could be potentially disastrous as lots of very private information could be used to cause massive losses. Our solution can be seen as follows:



```
simba@OMENUser:~/dolos-server/api$ go run .
starting server
listening to :8000
1fdffec41e3abea2d5c831df7a9afd9a <nil>
2022/03/04 23:39:31 request blocked endpoint:  /
2022/03/04 23:39:31 Post "": unsupported protocol scheme ""
```

Fig7 : Running Proxy Server Implementation



```
simba@OMENUser:~/dolos-traefik-plugin$ traefik -configfile traefik-dolos.yaml
INFO[0000] Configuration loaded from file: /home/simba/dolos-traefik-plugin/traefik-dolos.yaml
```

Fig8 : Running Of Traefik Tool configuration

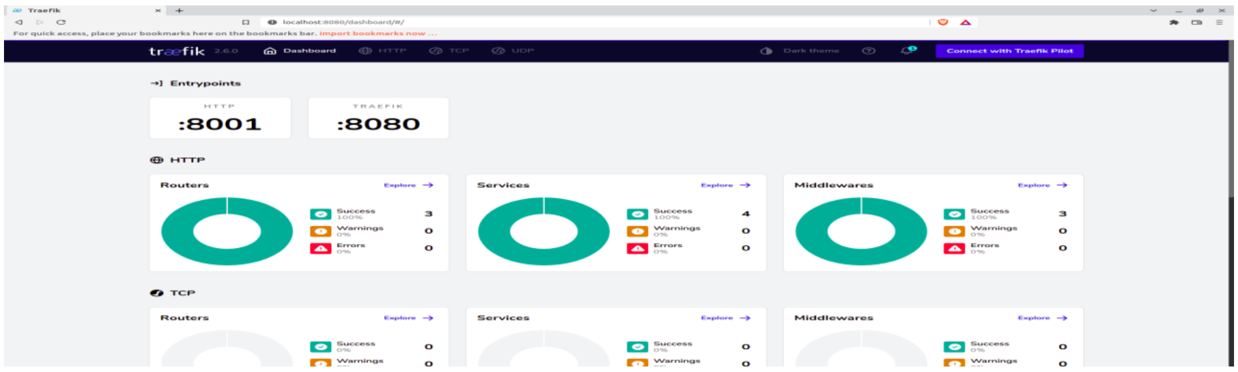
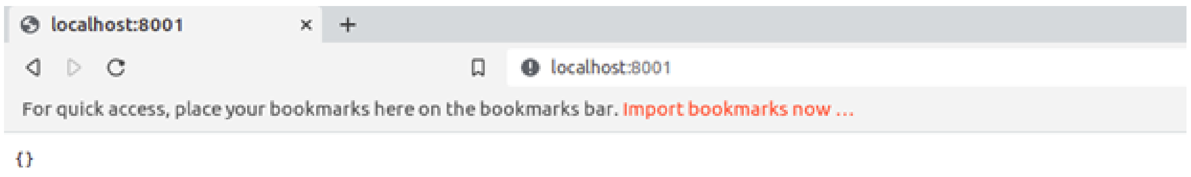


Fig9 : Observable connections from traefik tool and their management

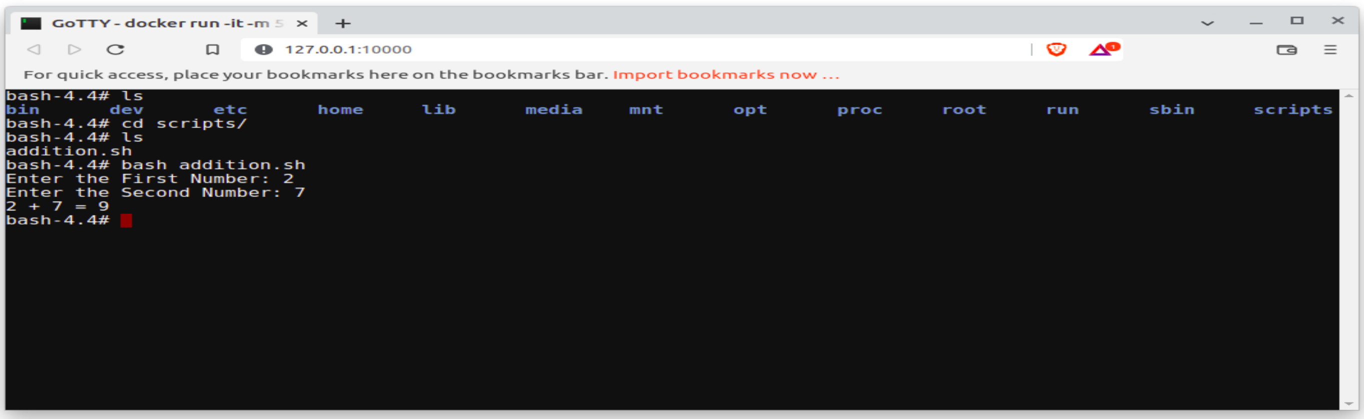

localhost:8001
localhost:8001
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now ...

```
{}
```

Fig10: Our Proxy Server at Port 8001

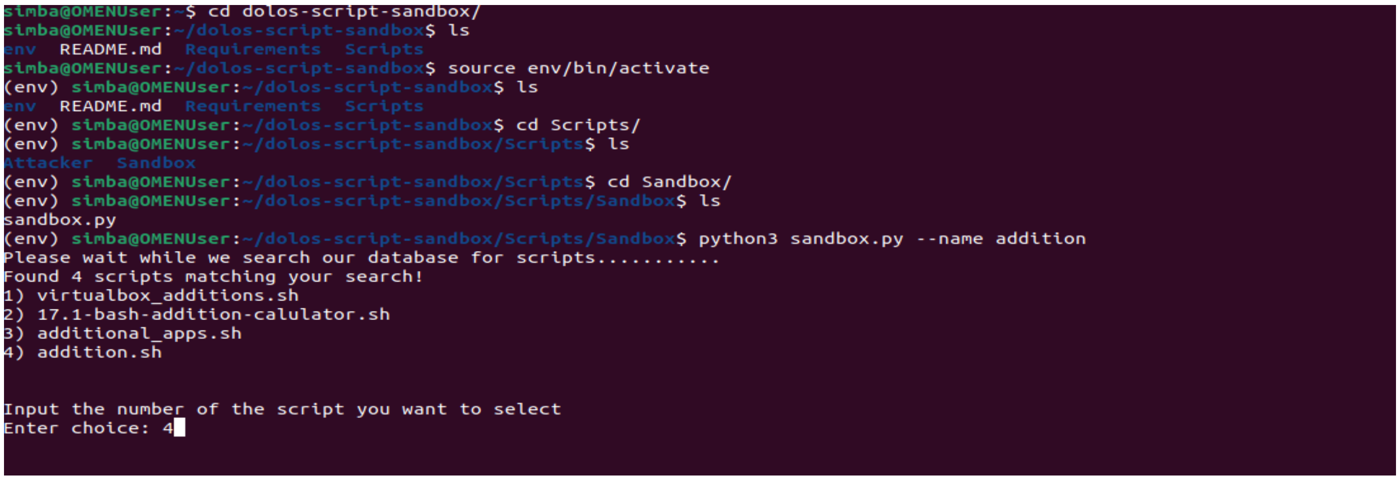
The figure depicts the successful implementation of preventing the secrets previously shared by the leaky server. This works as our implementation stands in between the user and the leaky server and ensures that blank information is given to the user in response to requests for sensitive information.

Phase 3 : Sandbox environment


GoTTY - docker run -it -m
127.0.0.1:10000
For quick access, place your bookmarks here on the bookmarks bar. Import bookmarks now ...

```
bash-4.4# ls
bin       dev       etc       home      lib       media     mnt       opt       proc      root      run       sbin      scripts
bash-4.4# cd scripts/
bash-4.4# ls
addition.sh
bash-4.4# bash addition.sh
Enter the First Number: 2
Enter the Second Number: 7
2 + 7 = 9
bash-4.4#
```

Fig : Isolated Sandbox environment



```
simba@OMENUser:~$ cd dolos-script-sandbox/
simba@OMENUser:~/dolos-script-sandbox$ ls
env  README.md  Requirements  Scripts
simba@OMENUser:~/dolos-script-sandbox$ source env/bin/activate
(env) simba@OMENUser:~/dolos-script-sandbox$ ls
env  README.md  Requirements  Scripts
(env) simba@OMENUser:~/dolos-script-sandbox$ cd Scripts/
(env) simba@OMENUser:~/dolos-script-sandbox/Scripts$ ls
Attacker  Sandbox
(env) simba@OMENUser:~/dolos-script-sandbox/Scripts$ cd Sandbox/
(env) simba@OMENUser:~/dolos-script-sandbox/Scripts/Sandbox$ ls
sandbox.py
(env) simba@OMENUser:~/dolos-script-sandbox/Scripts/Sandbox$ python3 sandbox.py --name addition
Please wait while we search our database for scripts..........
Found 4 scripts matching your search!
1) virtualbox_additions.sh
2) 17.1-bash-addition-calulator.sh
3) additional_apps.sh
4) addition.sh


Input the number of the script you want to select
Enter choice: 4
```

Fig : Sandbox script running on Host

As can be seen above the sandbox is running a separate instance of a terminal, different and isolated from the host system where it is able to run the script as requested by the user.

References

[1] M. Hart, P. Manadhata, and R. Johnson, “Text classification for data loss prevention,” in Privacy Enhancing Technologies, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 18–37

[2] O. Tunde-Onadele, J. He, T. Dai, and X. Gu, “A study on container vulnerability exploit detection,” in 2019 IEEE International Conference on Cloud Engineering (IC2E), 2019, pp. 121–127.

[3] G. Katz, Y. Elovici, and B. Shapira, “CoBAn: A context based model for data leakage prevention,” Inf. Sci. (Ny), vol. 262, pp. 137–158, 2014

[4] R. van der Kleij, R. Wijn, and T. Hof, “An application and empirical test of the Capability Opportunity Motivation-Behaviour model to data leakage prevention in financial organizations,” Comput. Secur., vol. 97, no. 101970, p. 101970, 2020.

[5] Y. Shapira, B. Shapira, and A. Shabtai, “Content-based data leakage detection using extended fingerprinting,” arXiv [cs.CR], 2013.

[6] P. Liu et al., “Understanding the security risks of docker hub,” in Computer Security – ESORICS 2020, Cham: Springer International Publishing, 2020, pp. 257–276.

[7] F. Špaček, R. Sohlich, and T. Dulík, “Docker as platform for assignments evaluation,” Procedia Eng., vol. 100, pp. 1665–1671, 2015.

[8] X. Luo, P. Zhou, E. W. W. Chan, W. Lee, R. K. C. Chang, and R. Perdisci, “HTTPOS: Sealing information leaks with browser-side obfuscation of encrypted flows,” Cloudfront.net.