

Web Application Security Testing Framework Using Flask

Submitted in partial fulfillment of the
requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

Rohit R (Reg.No - 39110855)
Hasan Firnas I (Reg.No - 39110379)
Abhishek M (Reg.No - 39110014)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade “A” by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600119**

APRIL - 2023



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Rohit R (Reg.No - 39110855), Hasan Firnas I (Reg.No - 39110379) and Abhishek M (Reg.No - 39110014)** who carried out the Project Phase - 2 entitled "**Web Application Security Testing Framework Using Flask**" under my supervision from January 2023 to April 2023.

Internal Guide

Dr Albert Mayan J M.E., Ph.D

Head of the Department

Dr. L. LAKSHMANAN, M.E., Ph.D.



Submitted for Viva-voce Examination held on 20-04-2023

Internal Examiner

External Examiner

DECLARATION

I, **Rohit R (Reg.No - 39110855)** hereby declare that the Project Phase - 2 Report entitled **Web Application Security Testing Framework Using Flask** done by me under the guidance of **Dr. J. Albert Mayan M.E., Ph.D** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 20-04-2023

PLACE: Chennai

A handwritten signature in blue ink, appearing to read 'Rohit R', is placed within a rectangular box.

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph. D, Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. J. Albert Mayan M.E., Ph.D** ,for him valuable guidance, suggestions and constant encouragement paved way for the successful completion of my phase- 2 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

This Automated Web Application Security Scanning And Information Gathering framework will help An organization and independent Security researcher to Automatically Discover the Asset's of an organization Which is publicly available on the Surface web and Filter out the target based on the web application initial response and open Ports and other Criteria And Gives alert to the organization analyst via email The Application Security Framework, provides a holistic approach to information security and risk management by providing organizations with the breadth and depth of verifying/validating security controls that are necessary to strengthen information systems and the associated environments. This Framework can be configured to run every 24/7 check the assets for every single public available vulnerability on the internet. Our framework uses Axiom methodology, Axiom is the dynamic infrastructure framework for everybody - it is designed to be easy to use, intuitive and simple. Axiom supports several cloud providers currently, DigitalOcean, Azure, Linode, and IBM Cloud • Google Compute & AWS support is on the roadmap, Axiom enables the average weekend warrior or pentesting professional to quickly spin up one or many disposable instances at a time, preloaded with many popular bug bounty & pentesting tools. Distributed scanning allows scan times 60x-100x faster than a traditional stand-alone VPS. We chose Axiom methodology because it is faster in scanning times means you can scan more often, useful for bug bounty, Reduce timelines of penetration tests for large external scopes, Faster scanning means you can throw more data at the problem - bigger wordlists, deeper coverage, Many different instance IP's, avoid block-lists, Basically a fully legal botnet. Overall, our approach for Automated Web Application Security Scanning and Information Gathering will save time while also increasing scalability. And because the reports are sent straight to the respective e-mail addresses, they are easy to read. This Framework can be up and running to check assets for every publicly available vulnerability on the internet throughout the day.

TABLE OF CONTENTS

Chapter No	TITLE	Page No.
	ABSTRACT	v
	LIST OF FIGURES	viii
	LIST OF TABLES	ix
	INTRODUCTION	1
1	1.1 Vulnerability	1
	1.2 Owasp Top 10 Vulnerabilities	2
	1.2.1 Broken Access Control	3
	1.2.2 Cryptographic Failures	3
	1.2.3 Injection	3
	1.2.4 Insecure Design	4
	1.2.5 Security Misconfiguration	4
	1.2.6 Vulnerable And Outdated Components	4
	1.2.7 Identification And Authentication Failures	4
	1.2.8 Software And Integrity Failures	5
	1.2.9 Insufficient Logging And Monitoring	5
	1.2.10 Server-Side Request Forgery	5
	1.3 Vulnerability Scanner	8
	1.4 Cloud Computing	8
	1.5 Amazon Web Services	9
	1.6 Benefits Of AWS	9
	1.7 Amazon Container Service (ECS)	11
	1.8 Amazon Elastic Compute Cloud (ec2)	11
	1.9 Amazon Relational Database Service (rds)	11
	1.10 Amazon Elastic File System (efs)	12
	1.11 AWS System Manager (SSM)	12
2	LITERATURE SURVEY	vii 13

	2.1 Open problems in Existing System	14
3	REQUIREMENTS ANALYSIS	16
	3.1 Feasibility Studies/Risk Analysis of the Project	16
	3.2 Software Requirements Specification Document	17
	3.3 System Use case	18
4	DESCRIPTION OF PROPOSED SYSTEM	19
	4.1 Selected Methodology or process model	19
	4.2 Architecture / Overall Design of Proposed System	20
	4.3 Description of Software for Implementation and Testing plan of the Proposed Model/System	21
	4.4 Financial report on estimated costing	30
	4.5 Transition/ Software to Operations Plan	32
5	IMPLEMENTATION DETAILS	37
	5.1 Development and Deployment Setup	37
	5.2 Algorithms	38
	5.3 Testing	39
6	RESULTS AND DISCUSSION	40
7	CONCLUSION	41
	7.1 Conclusion	41
	7.2 Future work	41
	7.3 Research Issues	41
	7.4 Implementation Issues	42
	REFERENCES	44
	APPENDIX	47
	A. SOURCE CODE	47
	B. SCREENSHOTS	60
	C. RESEARCH PAPER	63

LIST OF FIGURES

FIGURE NO	FIGURE NAME	Page No.
1.1	Benefits of AWS	9
4.1	System Design	20
4.2 a)	EC2 Dashboard	21
4.2 b)	Instance Launch Wizard -1	22
4.2 c)	Instance Launch Wizard -2	22
4.3 a)	Amazon EFS Dashboard	23
4.3 b)	File System Creation	24
4.4	Amazon SSM Dashboard3	25
4.5 a)	Amazon RDS Dashboard	26
4.5 b)	DB instance creation	26
4.5 c)	ARDS Setup Details - 1	27
4.5 d)	ARDS Setup Details - 2	27
4.6	Web Interface	30
4.7	Table of Pricing	31
4.8	t3.medium Cost	31
4.9 a)	Scanning Web Applications	32
4.9 b)	Script to store data in DB	33
4.9 c)	Job creation and distribution script files	33
4.9 d)	Job Acceptance script files	34
4.9 e)	List of Scan folders	35
4.9 f)	List of Subdomains found in the given input data	35
4.9 g)	Vulnerabilities found when scanning	36

LIST OF TABLES

TABLE NO	TABLE NAME	Page No.
1.1	Risk Levels and Examples for Vulnerabilities	2
1.2	Top 10 OWASP Vulnerabilities and Preventive Measures	5

CHAPTER 1

INTRODUCTION

The internet has become an integral component of our daily life. Every day, we engage with a significant number of custom-made web apps developed utilizing a range of technologies. Web application developers find it difficult to properly secure their applications and stay up to date with emerging threats and newly discovered attacks due to the web's highly heterogeneous nature, which includes different implementation languages, encoding standards, browsers, and scripting environments.

Applications were frequently deployed in closed client-server or stand-alone settings a decade ago. At that time, testing and securing an application was an easier task than today, where a web application can be accessed by millions of anonymous Internet users. Web application security and defense is becoming increasingly important as more security-critical applications, such as financial systems, governmental transaction interfaces, and e-commerce platforms, become directly accessible over the web.

Generic input validation issues are the source of many web application security flaws. SQL injection and Cross-Site Scripting are two examples of such flaws (XSS). Despite the fact that the majority of online vulnerabilities are simple to recognize and avoid, many web developers are security-aware. As a result, the internet is littered with insecure programmers and websites.

1.1 VULNERABILITY

A vulnerability is a defect in a computer system that thieves can use to gain unauthorized access to it. After exploiting a vulnerability, a cyberattack can run malicious code, install malware, and even steal critical data. SQL injection, buffer overflows, cross-site scripting (XSS), and open-source exploit kits are all examples of vulnerabilities that may be exploited in web applications. Many flaws impact widely used software, placing many users at risk of a data breach or supply chain assault. Zero-day assaults are classified as a Common Vulnerability Exposure by MITRE (CVE). Below Table 1.1 shows the Risk levels and Examples of Vulnerabilities.

Table 1.1 Risk Levels and Examples for Vulnerabilities

Risk Level	CVSS Ranges	Example Vulnerabilities
Critical	10.0	Remote Code Execution, Buffer Overflows, Default Credentials, Unsupported Operating System Versions
High	7.0 – 9.9	Malformed Packet Injection, Redirect Denial of Service, Privilege Escalation, Password Hash Disclosure
Medium	4.0 – 6.9	Remote Information Disclosure, Cryptographic Protocol, Command Injection, Web Directory Traversal & File Access
Low	0.1 – 3.9	Unencrypted Communications, Internal Information Disclosure, Browsable Web Directory
Informational	0.0	Software Version Disclosure, Protocol Detection, Operating System Identification, Device Type

1.2 OWASP TOP 10 VULNERABILITIES:

The Open Web Application Security Project (OWASP) is a non-profit committed to providing unbiased, practical application security knowledge. In 2017, the OWASP Top 10 Online Application Security Risks was revised to give developers and security experts recommendations on the most significant vulnerabilities

detected in web apps that are also easy to attack. These ten online application flaws are hazardous because they might allow attackers to install malware, steal data, sensitive information, or entirely take control of your computer or web server. Table 1.2 shows the Top 10 Vulnerabilities with its description and preventive measures.

1.2.1 BROKEN ACCESS CONTROL:

Security access controls on a website should restrict visitor access to only the pages or portions required by that user. Administrators of an ecommerce site, for example, must be allowed to add new links and promotions. Other categories of visitors should not be able to use these features.

To prevent traps like content management systems (CMS) that create all-access permission by default, developers must be pushed to adopt the "security first" discipline (up to and including admin-level access). Visitors to a website with a broken access control system can get access to admin panels, servers, databases, and other business-critical applications. This OWASP Top 10 threat might even be used to reroute browsers to other, specifically targeted URLs.

1.2.2 CRYPTOGRAPHIC FAILURES:

A Cryptographic Failure vulnerability is a wide vulnerability category that includes all forms of cryptography-related attacks. Given one might expect, a vulnerability of this nature might have major ramifications, as encryption is designed to protect sensitive data.

1.2.3 INJECTION:

When a query or command is used to inject untrusted data into the interpreter via SQL, OS, NoSQL, or LDAP injection, an injection vulnerability can arise. The hostile data supplied by this attack vector fools the interpreter, causing the program to perform things it wasn't meant to do, such as generate unexpected instructions or access data without proper authentication.

Injection attacks may affect any program that takes parameters as input. The completeness of the application's input validation methods is closely connected with the threat level.

1.2.4 INSECURE DESIGN:

"Missing or bad control design" is a broad word that incorporates a range of defects and is characterized as "insecure design." Threat modeling, secure design patterns, and reference architectures are some of the new categories for 2021, with a desire for more threat modeling, secure design patterns, and reference architectures. We must, as a community, go beyond "shift left" coding and pre-code operations that are critical to the Secure by Design principles.

1.2.5 SECURITY MISCONFIGURATION:

Human mistake is estimated to be the cause of up to 95% of cloud breaches, according to Gartner. Security setting misconfigurations are one of the leading causes of that statistic, according to OWASP, which notes that this vulnerability is the most prevalent among the top ten. There are a variety of misconfigurations that put a firm at risk for cybersecurity, including:

- Accepting insecure default settings is a big no-no.
- Cloud storage resources that are overly accessible
- Configurations that are not full
- HTTP headers that have been misconfigured
- Error notices that are long and include important information

1.2.6 Vulnerable And Outdated Components:

Open-source libraries and frameworks are frequently used in modern distributed web applications. Any component with a known vulnerability becomes a weak link that can compromise the application's security.

1.2.7 IDENTIFICATION AND AUTHENTICATION FAILURES:

Intruders may be able to compromise passwords, security keys, or session tokens and permanently or temporarily assume the identities and permissions of other users if apps wrongly perform operations related to session management or user authentication. This flaw poses a danger to the application's security and the resources it accesses, and it can even affect other assets on the same network.

1.2.8 SOFTWARE AND INTEGRITY FAILURES:

Software and data integrity failures are defined as code and infrastructure that do not protect against integrity breaches. This is an example of software that leverages plugins, libraries, or modules from untrustworthy sources, repositories, or content delivery networks (CDNs). An unprotected CI/CD pipeline might result in unauthorized access, malicious code, or system compromise. Finally, many programmers now feature auto-update capabilities, which allow updates to be downloaded and deployed to previously trusted apps without the need for further integrity checks. With this feature, attackers may theoretically distribute and run their updates across all platforms.

1.2.9 INSUFFICIENT LOGGING AND MONITORING:

According to studies, the interval between an attack and discovery can take up to 200 days, and typically much longer. This time frame allows cybercriminals to tamper with servers, alter databases, steal private information, and plant harmful malware.

1.2.10 SERVER - SIDE REQUEST FORGERY:

Server-side request forgery (SSRF) is a web security issue that allows an attacker to compel a server-side application to submit HTTP requests to any domain they choose.

An SSRF failure occurs when a web application retrieves a remote resource without verifying the user-supplied URL. An attacker can compel software to submit a forged request to an unexpected destination, even if it is protected by a firewall, VPN, or another type of network access control list.

The preventive measures to be taken care of the top 10 **OWASP** Vulnerabilities are given in below table 1.2.

Table 1.2 Top 10 OWASP Vulnerabilities and Preventive Measures

Sr. No.	Web Application Security Risks	Description	Preventive Measures
1	Injection	Injection flaws, such as SQL injection, CRLF injection, and	Application security testing can easily detect

		LDAP injection occur when an attacker sends untrusted data to an interpreter.	injection flaws. Developers should use parameterized queries when coding
2	Broken Authentication and Session Management	Incorrectly configured user and session authentication could allow attackers to compromise passwords, keys, or session tokens, or take control of users' accounts to assume their identities.	Multi-factor authentication, such as FIDO or dedicated apps, reduces the risk of compromised accounts.
3	Sensitive Data Exposure	Applications that don't properly protect sensitive data such as financial data, usernames, and passwords, could enable attackers to access such information to commit fraud.	Encryption of data at rest and in transit can help you comply with data protection regulations.
4	XML External Entity	Poorly configured XML processors evaluate external entity references within XML documents. Attackers can use external entities for attacks	Static application security testing (SAST) can discover this issue by inspecting dependencies and configuration.
5	Broken Access Control	Improperly configured or missing restrictions on authenticated users allow them to access unauthorized functionality or data, such as accessing other users' accounts, viewing sensitive	Penetration testing is essential for detecting non-functional access controls; other testing methods only detect where access controls

		documents	are missing.
6	Security Misconfiguration	This risk refers to improper implementation of controls intended to keep application data safe, such as misconfiguration of security headers, error messages, and not patching or upgrading systems, and frameworks.	Dynamic application security testing (DAST) can detect misconfigurations, such as leaky APIs.
7	Cross-Site Scripting	Cross-site scripting (XSS) flaws give attackers the capability to inject client-side scripts into the application, for example, to redirect users to malicious websites.	Developer training complements security testing to help programmers prevent cross-site scripting with best coding best practices, such as encoding data and input validation.
8	Insecure deserialization	Insecure deserialization flaws can enable an attacker to execute code in the application remotely, tamper or delete serialized (written to disk) objects, and conduct injection attacks.	Application security tools can detect deserialization flaws but penetration testing is frequently needed to validate the problem
9	Using Components with Known Vulnerabilities	Developers frequently don't know which open-source and third-party components are in their applications, making it difficult to update components when new	Software composition analysis conducted at the same time as static analysis can identify insecure versions of

		vulnerabilities are discovered.	components.
10	Insufficient Logging and Monitoring	Insufficient logging and ineffective integration with security incident response systems allow attackers to pivot to other systems.	Think like an attacker and use pen testing to find out if you have sufficient monitoring; examine your logs after pen testing.

1.3 VULNERABILITY SCANNER:

A vulnerability scanner is a piece of software that scans computers, networks, and applications for known flaws. They may execute authenticated and unauthenticated scans to find and detect vulnerabilities arising from network misconfiguration and programming flaws.

1.3.1 AUTHENTICATED SCANS:

Allows the vulnerability scanner to get direct access to networked assets using remote administration protocols such as secure shell (SSH) or remote desktop protocol (RDP) and authenticate using the system credentials given. This provides thorough and accurate information on operating systems, installed software, configuration issues, and missing security updates by allowing access to low-level data such as individual services and configuration details.

1.3.2 UNAUTHENTICATED SCANS:

False positives and incorrect information about operating systems and installed applications result from unauthenticated scanning. Cyber attackers and security analysts commonly employ this strategy.

1.4 CLOUD COMPUTING:

Cloud computing has become an essential element in not just the commercial world, but also in our daily lives. The majority of organizations have chosen cloud

computing because it is believed to be safer and more trustworthy, particularly in the area of inventory tracking. Cloud computing is the on-demand delivery of services, such as data and projects, that may be stored and accessed quickly. Cloud computing has made running a business and delivering services much easier. It has also aided in the rapid expansion of Small and Medium Enterprises (SMEs). Amazon is at the forefront of providing cloud computing services around the world through the Amazon Web Services service (AWS). AWS has shown to be the best at providing cloud computing services to people, businesses, and organizations. Customers can use the platform to store data. Most SMEs choose AWS over other service providers because it's much more efficient and cost-effective than its competitors. Since these services are available at a low cost, Amazon Web Services has grown in popularity and now leads the industry globally.

1.5 AMAZON WEB SERVICES

Amazon Web Services (AWS) is a cloud computing platform from Amazon.com that consists of a collection of remote computing services, commonly known as web services. Amazon offers a remarkable set of web services that developers can use to build dynamic and reliable applications. When compared to constructing and maintaining more traditional systems, deploying on AWS can save you time, money, and resources. The most important and well-known of these services are Amazon EC2 and Amazon S3. The service claims to provide a massive computational capacity (potentially many servers) that is significantly faster and less expensive than constructing a real server farm.

1.6 BENEFITS OF AWS:

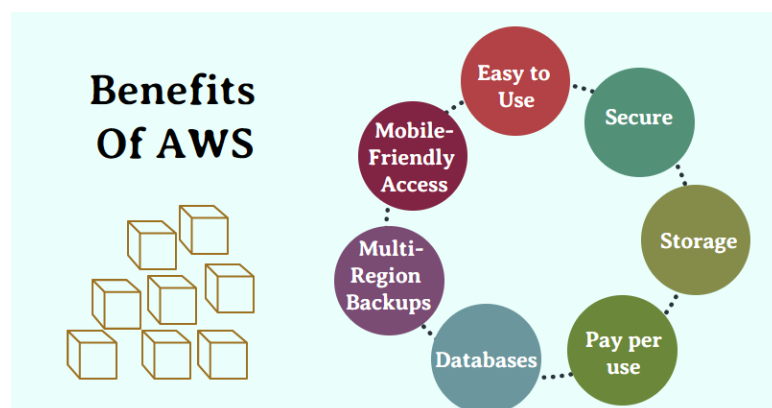


Fig 1.1 Benefits of AWS

As shown in Fig 1.5 Amazon Web Services delivers several benefits for IT organizations and developers alike, including:

- **Ease to use:** AWS is designed to let application providers, ISVs, and vendors easily and securely host their applications, whether they are existing or new SaaS-based applications. To access AWS's application hosting platform, you can use the AWS Management Console or well-documented web services APIs.
- **Flexible:** AWS gives you the flexibility to choose your operating system, programming language, web application platform, database, and other services. AWS provides you with a virtual environment in which you can install the software and services that their application requires. This makes it easier to migrate old apps while still allowing for the creation of new ones. Overall, you are free to create any application you want on any platform or programming paradigm. You have complete control over the resources you use and how they are integrated into your application.
- **Cost-effective:** There are no long-term contracts or upfront obligations, you just need to pay for the computational power, storage, and other resources you utilize. Our operations, management, and hardware costs are decreasing as the Amazon Web Services cloud grows. The AWS Economics Center has more information on comparing the costs of different hosting options with AWS.
- **Reliable:** You would benefit from AWS's scalable, resilient, secure, and efficient computing infrastructure, which serves as the virtual backbone of Amazon.com's multibillion-dollar online business. We're running on a battle-tested web-scale infrastructure that can handle anything you throw at it.
- **Scalable and High Performance:** Any application can scale up or down based on demand using AWS tools such as Auto Scaling, and Elastic Load Balancing. You have access to computation and storage resources when you need them, thanks to Amazon's massive infrastructure.
- **Secure:** To secure and fortify our infrastructure, AWS takes an end-to-end approach that includes physical, operational, and software measures. AWS offers over 90 security standards and compliance certifications, and all 117 AWS services that contain client data have encryption capabilities.

- **Comprehensive:** Don't need to start from scratch. Amazon Web Services offers a range of services that you can use in your applications. These services enable you to build powerful applications at a lower cost and with less up-front investment, from databases to payments.

1.7 Amazon container service (ECS):

In an Amazon ECS cluster, you may execute and maintain a defined number of instances of a task definition at the same time using the Amazon ECS service. If one of your tasks fails or stops for whatever reason, the Amazon ECS service scheduler creates a new instance of your task definition to take its place, ensuring that the service maintains the correct number of tasks.

You can operate your service behind a load balancer in addition to maintaining the required number of jobs in your service. The load balancer distributes traffic across the jobs connected to the service.

1.8 AMAZON ELASTIC COMPUTE CLOUD (EC2):

Amazon EC2 is a web service that provides cloud computation capability that can be scaled up or down. It's a key component of Amazon.com's cloud computing platform, Amazon Web Services (AWS). Users can rent virtual computers through EC2 to execute their computer applications. By providing a web interface through which a user can boot Amazon Machine Image to build a virtual machine, which Amazon refers to as an "instance," containing any software needed, EC2 enables scalable application deployment. The phrase "elastic" refers to the ability of a user to construct, launch, and terminate server instances as desired, with active servers being charged by the hour. EC2 gives users control over where their instances are physically located, allowing for latency optimization and high degrees of redundancy.

1.9 AMAZON RELATIONAL DATABASE SERVICE (RDS)

Amazon RDS (Amazon Relational Database Service) makes it simple to set up, run, and scale a relational database in the cloud. It offers scalable capacity at a lesser cost while automating time-consuming administrative activities including hardware provisioning, database setup, patching, and backups. It allows you to concentrate on your applications, ensuring that they have the high performance,

high availability, security, and compatibility that they demand. Amazon RDS gives you six familiar database engines to pick from, including Amazon Aurora, PostgreSQL, MySQL, MariaDB, Oracle Database, and SQL Server, and therefore is available on multiple database instance types - optimized for memory, speed, or I/O. You may quickly migrate or replicate your existing databases to Amazon RDS using the AWS Database Migration Service.

1.10 AMAZON ELASTIC FILE SYSTEM (EFS)

Amazon EFS is a cloud-based file storage service for Amazon Web Services' public cloud applications and workloads. Elastic File System (EFS), which is dispersed over an unlimited number of servers to prevent performance bottlenecks, is automatically deployed and managed by AWS. Amazon EFS provides scalable storage space for workloads that operate on AWS Elastic Compute Cloud (EC2) instances and access files via API queries. For thousands of EC2 instances linked to Amazon EFS, the service is designed to be extremely available and durable. Each file system object in Amazon EFS is stored in various availability zones (AZs), so an IT professional may access it from any of the AZs in the region.

1.11 AWS SYSTEMS MANAGER (SSM)

The AWS Systems Manager Agent (SSM Agent) is Amazon software that operates on Amazon EC2 instances, edge devices, and on-premises servers and virtual computers (VMs). The systems Manager may update, manage, and configure these resources via the SSM Agent. The SSM Agent then uses the Amazon Message Delivery Service to deliver status and execution information back to the Systems Manager service.

CHAPTER 2

LITERATURE SURVEY

Moshika [1] *et al* has proposed a method to predict the probabilistic value using Deterministic Arithmetic Automata (DAA) and Probabilistic Arithmetic Automata (PAA). The final outcome is the overall attack cause which can be used to determine the percentage of vulnerability present in the web application under examination.

F. Ö. Sönmez [2] *et al* has proposed a method to visualize dynamic application security test results and the metrics/measures that the tool presents. And allows the investigation of fifty metrics/measures for the multi-project/phase environment that enhances its benefits if the user aims to monitor a series of analyses' results and the changes between them for more than one web project.

Julian Thome [3] *et al* has proposed a method to effectively find injection vulnerabilities in source code, which generates no or few false alarms, minimizes false negatives, and overcomes the path explosion problem and the one of solving complex constraints

Muhammad Noman Khalid [4] *et al* proposes a new tool named Web Vulnerability Finder which is capable to perform efficient penetration tests on php and .NET based websites. And allows to find vulnerabilities in web applications by constructing attacks without accessing their source codes. This survey explores the research that has been done in the black-box vulnerability finding and exploits construction in web applications and proposes future directions.

Kevin Jonathan Koswara [5] *et al* has proposed a method to increase the number of URLs gathered by crawling process and thus, improving the effectiveness of vulnerability scanning in AJAX applications.

Nuno Antunes [6] *et al* proposes a comparative study of 4 automated web security scanners - used—HP Web Inspect, IBM Rational AppScan, Acunetix Web Vulnerability Scanner and an academic prototype of their own approach.

Muhammad Yeasir Arafat [7] *et al* proposes different types of HTTP DoS attacks, effects on web server and mitigation for those attacks. And it also proposes a new DDoS defense mechanism that protects http web servers from application-level DDoS attacks based on the reverse proxy. The attack flow detection mechanism detects attack flows based on the symptom or stress at the server, since it is getting more difficult to identify bad flows only based on the incoming traffic patterns.

Ashikali Hasana [8] *et al* proposes some open source and commercial tools available for web vulnerability assessment and penetration testing. In purpose of dive little deeper in the area of vulnerability assessment and penetration in this paper we have analyzed overview of the penetration testing process and its limitations and includes various tools which are helpful to conduct VAPT process of these high-risk vulnerabilities.

Prajakta Subhash Jagtap [9] *et al* has proposed a method to review the state of the art of current open-source vulnerability scanning tools.

S. Ibarra-Fiallos [10] *et al* proposes effective and potential filters to be implemented in-order to avoid several web attacks. The proposal filter has been tested on three public as well as on a real private web application. An accuracy of 98,4% and an average processing time of 50 ms are achieved, based on which it is possible to conclude the proposed filter is highly reliable and does not require additional computational resources.

2.1 OPEN PROBLEMS IN EXISTING SYSTEM

- Because a vulnerability screening program might overlook flaws, you can't be sure your systems aren't at risk. This is one of the most significant shortcomings of any scanning method, because hackers may still be able to exploit vulnerabilities. There are two plausible explanations:
 - The vulnerability is too sophisticated to be uncovered by an automated tool since the attack is not straightforward to automate.

- The scanner is unaware of the vulnerability, for example, because it was only recently discovered.
- Existing ones are not guaranteed that the most recent vulnerabilities are discovered.
- It might be difficult to grasp the implications of the scanning tool's findings/vulnerabilities, especially if you have a big IT infrastructure with many servers and services. As a result, you'll encounter a lot of false positives. Recognizing them is difficult if you are not a security expert, which makes analyzing the data a time-consuming process.

CHAPTER 3 REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES/RISK ANALYSIS :

Feasibility studies are conducted to evaluate whether the project is technically feasible, financially viable, and operationally achievable. It assesses the various aspects of the project, including the requirements, resources, time, and costs required to complete the project. In the case of your project, the feasibility study would involve analyzing whether Flask is a suitable platform for the project, evaluating the costs of development and maintenance, and assessing the availability of skilled resources to carry out the project. Here are some key aspects to consider:

Technical Feasibility: Flask is a popular web framework that is well-suited for developing web applications. Flask supports a wide range of Python libraries, making it easy to integrate with other tools, such as testing frameworks, database connectors, and security libraries. Flask also has a robust community of developers and users, making it easy to find support and resources. Therefore, the technical feasibility of using Flask for this project is high.

Financial Feasibility: The cost of developing a web application security testing framework using Flask will depend on various factors, such as the complexity of the framework, the development team's size, and the project's timeline. However, the use of Flask as a web framework is open-source and free, making it an affordable option for development. The cost of maintaining and updating the framework should also be factored into the financial feasibility analysis.

Operational Feasibility: The operational feasibility of the web application security testing framework depends on the availability of skilled developers to work on the project, the project's timeline, and the availability of testing environments. With Flask being a widely adopted framework, finding experienced developers shouldn't be a problem. However, finding skilled developers who have experience with security testing may be more challenging.

Security Risks: Developing a security testing framework comes with a significant risk of introducing security vulnerabilities. Developers need to be aware of the potential risks and incorporate security best practices, such as secure coding practices and regular security testing, to mitigate these risks.

Testing Coverage Risks: The framework's testing coverage must be comprehensive enough to identify potential security vulnerabilities effectively. The developers must ensure that the framework covers all aspects of the web application, such as input validation, session management, and access control.

Maintenance Risks: As web applications evolve and new security vulnerabilities are discovered, the security testing framework must be updated to keep up with the latest security trends. The development team must ensure that the framework's maintenance plan includes regular security audits and updates to keep the application secure.

Overall, the feasibility study and risk analysis indicate that developing a web application security testing framework using Flask is a viable option. The technical feasibility is high, and the financial feasibility is reasonable, given the open-source nature of the Flask framework. However, developers must be aware of the potential security risks associated with developing a security testing framework and ensure that the testing coverage is comprehensive and the maintenance plan includes regular security audits and updates.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

The required software specifications are given below:

3.2.1 HARDWARE SPECIFICATION

This project has the following minimum hardware requirement

- 2GB of RAM
- 40GB storage
- any one of the amd64, i386, armel, armhf, or arm64 architectures

3.2.2 SOFTWARE SPECIFICATION

The software requirements of the project are:

- Operating System - Ubuntu
- AWS EC2
- AWS RDS
- AWS SSM
- AWS EFS

3.2.3 TECHNOLOGIES USED:

Technologies used for this project are

- Flask (Python)
- HTML
- CSS
- JavaScript
- Bash

3.3 SYSTEM USE CASE:

In day-to-day life, websites play a vital role in various business ventures in which most of their action items take place. So, these websites have to be maintained vulnerable free. Scanning websites manually for all major types of vulnerabilities is a tedious task. Though there are many tools available in the market, some only scans for specific vulnerabilities, and other few available tools are neither cost-efficient nor time efficient. So, these drawbacks have to be considered while building a vulnerability scanner.

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

4.1 SELECTED METHODOLOGY OR PROCESS MODEL:

- This step entails defining the scope and objectives of a test, as well as the systems that will be
- tested and the testing techniques that will be employed.
- Obtaining intelligence (e.g., network and domain names, mail server) to have a better understanding of how a target operates and its weaknesses.

4.1.1 SCANNING

The next stage is to figure out how the target application will react when given a domain name.

- The subdomain identifier is used to list all of the subdomains that exist under a certain domain.
- List of web servers: here is a list of web servers that are available under the specified domain name.
- Vulnerability scan: for each subdomain detected, a set of vulnerabilities is assessed and reported if they exist.

4.1.2 ANALYSIS

The vulnerability test results are gathered and stored in a database.

- Each scan will have its name and specific modules will be stored there.
- Vulnerabilities that have been exploited

4.1.3 NOTIFY

The list will be updated in telegram if a subdomain is detected or a web server is taken.

If a vulnerability is discovered, the user will be notified of the amount of risk right away.

4.2 ARCHITECTURE / OVERALL DESIGN OF PROPOSED SYSTEM

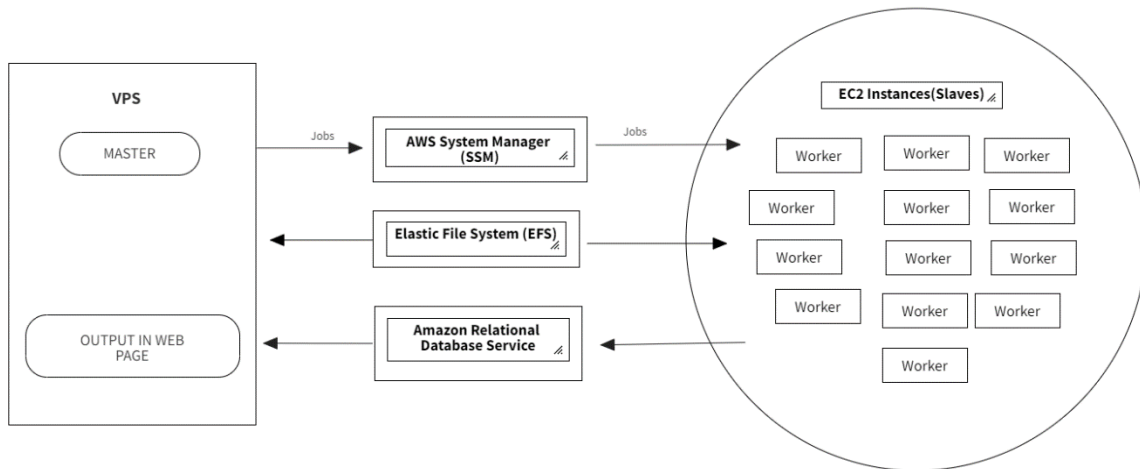


Fig 4.1 System Design

1. Docker containers are being used to build workers, which are managed by AWS Fargate and **AWS Elastic Container Service (ECS)**.
2. **Amazon EC2** is used to launch master and slaves (virtual servers) as we need, configure security and networking, and manage storage.
3. Workers scale automatically based on the number of items in the job queue. This will ensure that a large number of workers are available when we need them, but they all die when we don't need them anymore. This could potentially be set up very quickly by utilizing an **Elastic Beanstalk worker environment**.
4. **Amazon Elastic File System (EFS)** will be used to share files to all containers (config files, tools, etc.)
5. **AWS Systems Manager (SSM)** is Amazon software that runs on Amazon Elastic Compute Cloud (Amazon EC2) instances and on-premises servers and virtual machines (VMs). Systems Manager is used to updating, manage, and configure these resources. (EC2 full control.)
6. **Amazon Relational Database Service** is used to store relational data. (MySQL)

4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED MODEL/SYSTEM

4.3.1 AWS CONFIGURATION – CREATION OF AWS ACCOUNT

To use any of AWS' web services, you must first sign up for an account at <http://aws.amazon.com>. An AWS account is simply an Amazon.com account that allows you to access AWS products; you can create an AWS account using an existing Amazon.com account login and password. It's important to note that setting up an AWS account is completely free. They only ask for your payment information when you attempt to use a service. AWS provides a set of access credentials that may be used to access all of its services. On the top right of the AWS webpage, go to account and then security credentials. For future use, save your access key and build and download an X.509 certificate.

4.3.2 EC2 CONFIGURATION SETUP:

To start an EC2 instance and mount an EFS file system, follow these steps.

1. Go to <https://console.aws.amazon.com/ec2/> to access the Amazon EC2 console. Below Fig 4.2 a) shows the snip of the EC2 Dashboard.

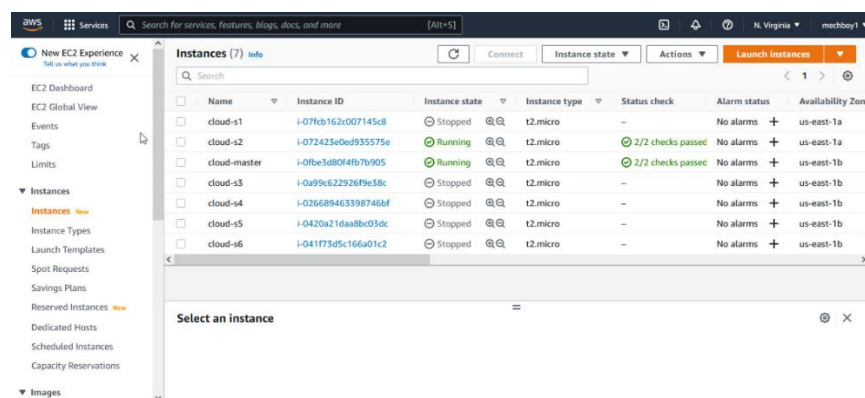


Fig 4.2 a) EC2 Dashboard

2. Fig 4.2 b) and 4.2 c) show the list of Instances. Select Launch Instance.
3. Select an Ubuntu Image.

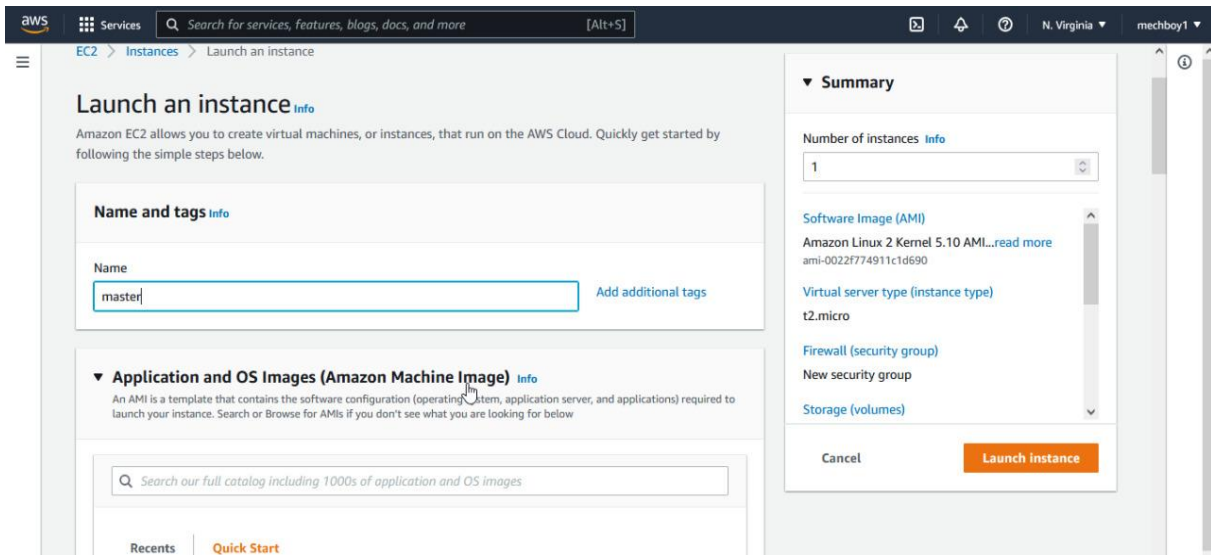


Fig 4.2 b) Instance Launch Wizard -1

4. Select an Instance Type, and go to next.
5. Provide the following information for Configure Instance Details:
 0. Leave the number of occurrences at one for the master.
 1. Leave the Purchasing option at its default value.
6. Next, select Add Storage - 8 GB RAM.

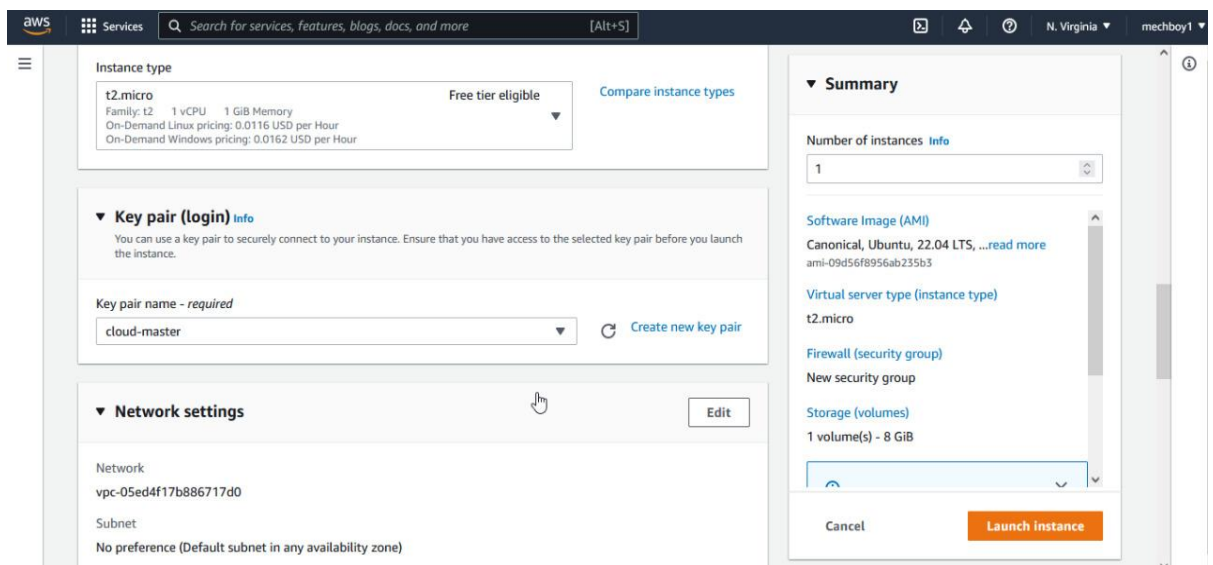


Fig 4.2 c) Instance Launch Wizard -2

7. Next, select Tags.
8. Name your instance (MASTER) and then click Next to Set up the Security Group.
9. Create a Security Group with port of 80, 443, and 2089 for HTTP Traffic and EFS respectively.

10. Select Review and Launch.

4.3.3 EFS CONFIGURATION SETUP:

To create your Amazon Elastic File System, follow these steps:

1. Go to <https://console.aws.amazon.com/efs/> to access the Amazon EFS Management Console. Below Fig 5.3 a) shows the snip of the EFS Dashboard.

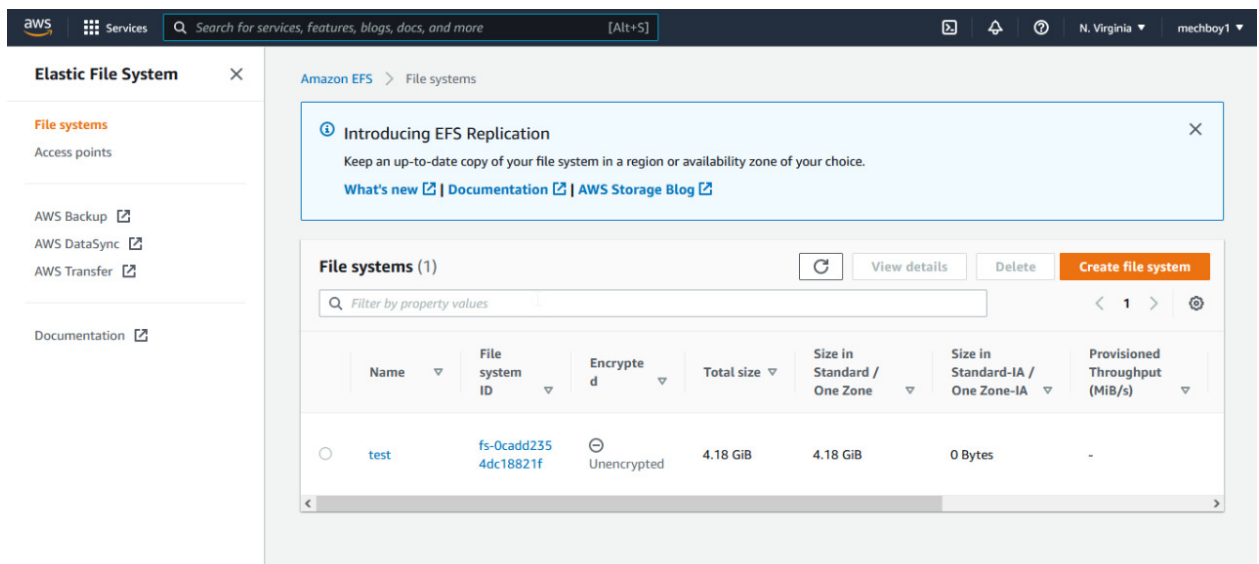


Fig 4.3 a) Amazon EFS Dashboard

2. To open the Create file system dialogue box, select Create file system.
3. Give your file system a name - CLOUDMIST.
4. Choose your Virtual Private Cloud (VPC) which we have used for EC2.
5. Select Create to create a file system with the following service recommendations:
6. Automatic backups are enabled; for more information, see Backing up and restoring Amazon EFS file systems with AWS Backup.
7. Mount targets are created by Amazon EFS with the following settings:
8. A mount target is created in each Availability Zone in the AWS Region where the file system is created for file systems that use Standard storage classes. A single mount target is created in the Availability Zone you specify for file systems that use One Zone storage classes.
9. Located in the VPC your choice's default subnets.

10. Using the VPC's default security group – After the file system is created, you can manage security groups.
11. The File systems page is displayed, with a banner across the top displaying the status of the file system you created which is shown in Fig 1.9 b). When the file system becomes available, a link to the file system details page appears in the banner.

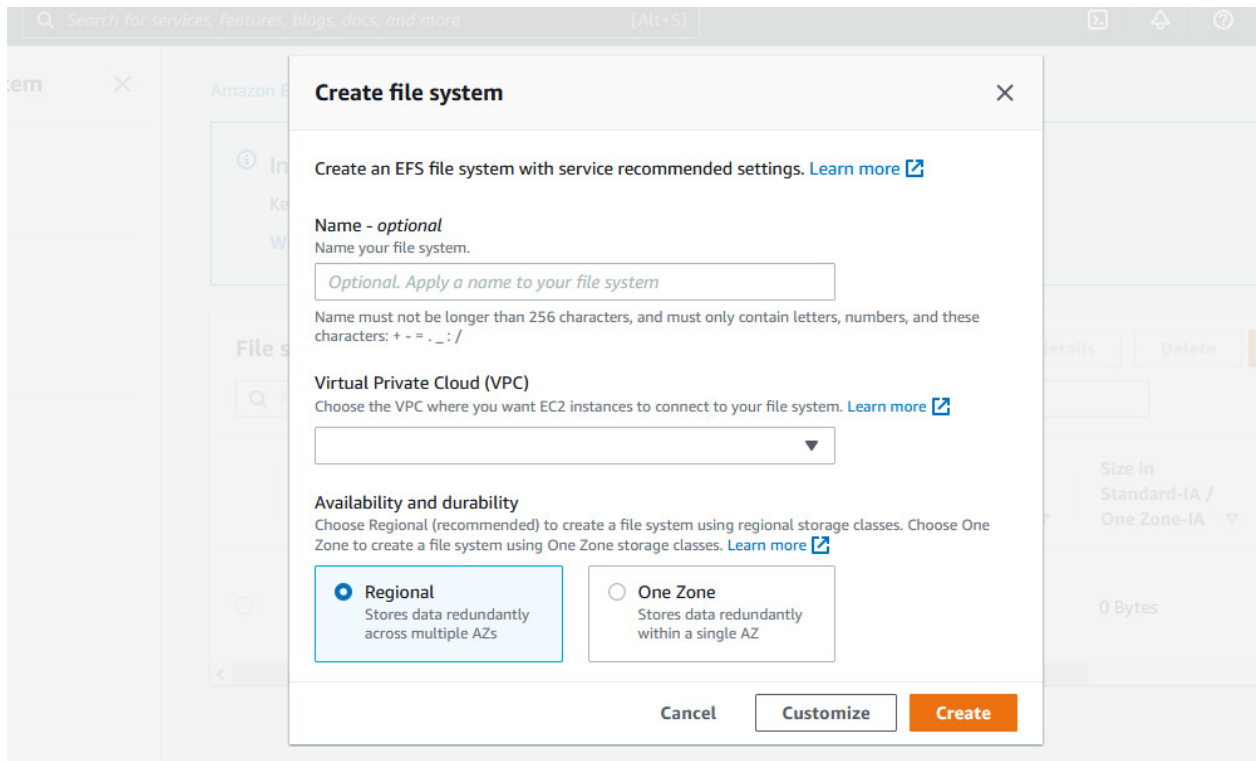


Fig 4.3 b) File System Creation

4.3.4 AWS SYSTEMS MANAGER (SSM) - CONFIGURATION SETUP:

Session Manager is an AWS Systems Manager fully managed capability that allows you to manage your Amazon EC2 instances via an interactive one-click browser-based shell or the AWS CLI. To begin a session with an instance in your account, use Session Manager. After starting the session, you can use bash commands as you would with any other connection type. See AWS Systems Manager Session Manager in the AWS Systems Manager User Guide for more information on Session Manager.

Before connecting to an instance with Session Manager, make sure all of the necessary setup steps have been completed. See Setting Up Session Manager for more information and instructions.

Using the Amazon EC2 console, connect to a Linux instance using Session

Manager.

1. Go to <https://console.aws.amazon.com/ec2/> to access the Amazon EC2 console. Below Fig 4.4 shows the snip of the SSM Dashboard.

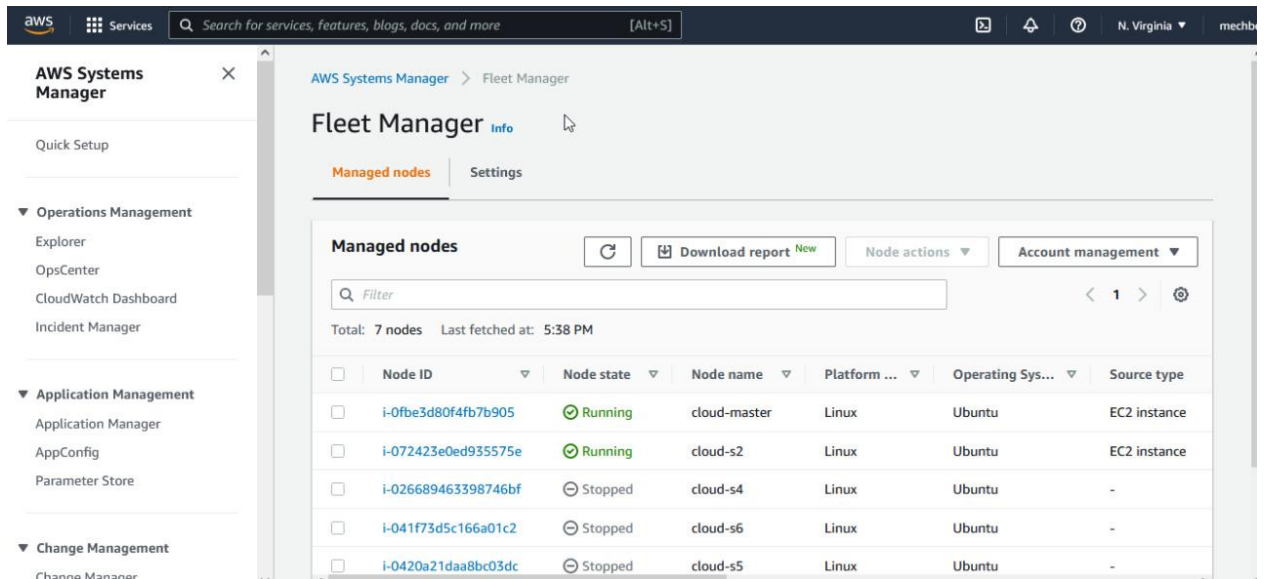


Fig 4.4 Amazon SSM Dashboard

2. Select Instances from the navigation pane.
3. Choose Connect after selecting the instance.
4. Select Session Manager as the connection method.
5. Select Connect.

4.3.5 AMAZON RDS - CONFIGURATION SETUP:

You can create a DB instance using the AWS Management Console, with or without Easy create enabled. You only need to specify the DB engine type, DB instance size, and DB instance identifier when Easy create is enabled. For other configuration options, Easy Create uses the default setting. When you create a database with Easy Create disabled, you can specify more configuration options, such as those for availability, security, backups, and maintenance.

To create a DB instance:

1. Sign in to the AWS Management Console and navigate to <https://console.aws.amazon.com/rds/>. Below Fig 4.5 a) shows the snip of the Amazon RDS Dashboard.

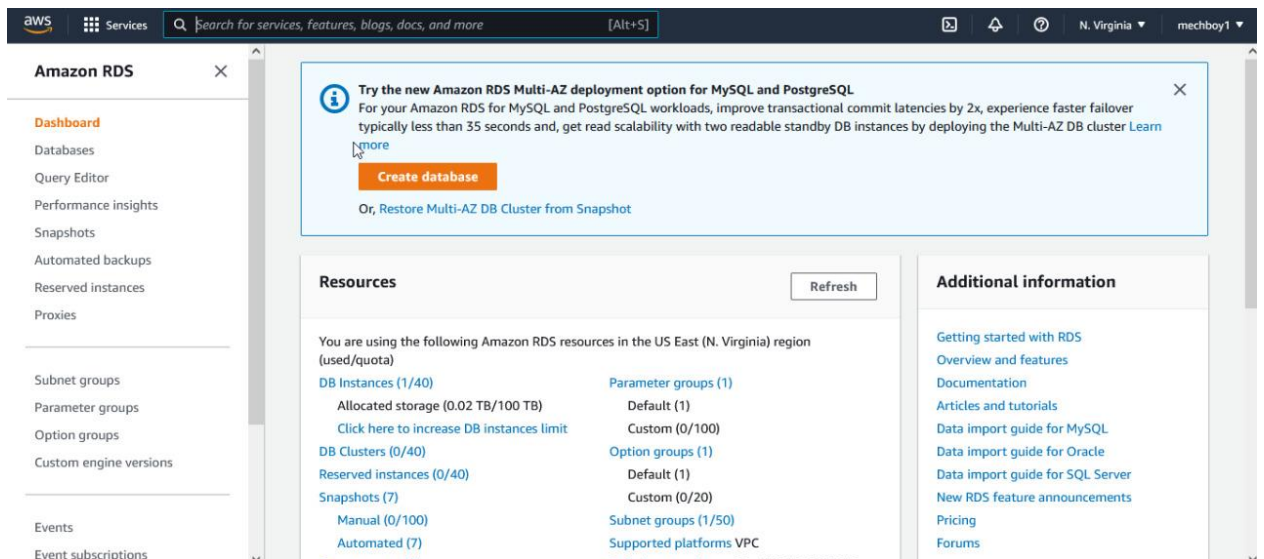


Fig 4.5 a) Amazon RDS Dashboard

2. Select the AWS Region in which you want to create the DB instance in the upper-right corner of the Amazon RDS console.
3. Select Databases from the navigation pane.
4. Select Create database.
5. In Select Standard Create as the database creation method as shown in Fig 4.5 b).
6. Select the engine type from the Engine options: MySQL.

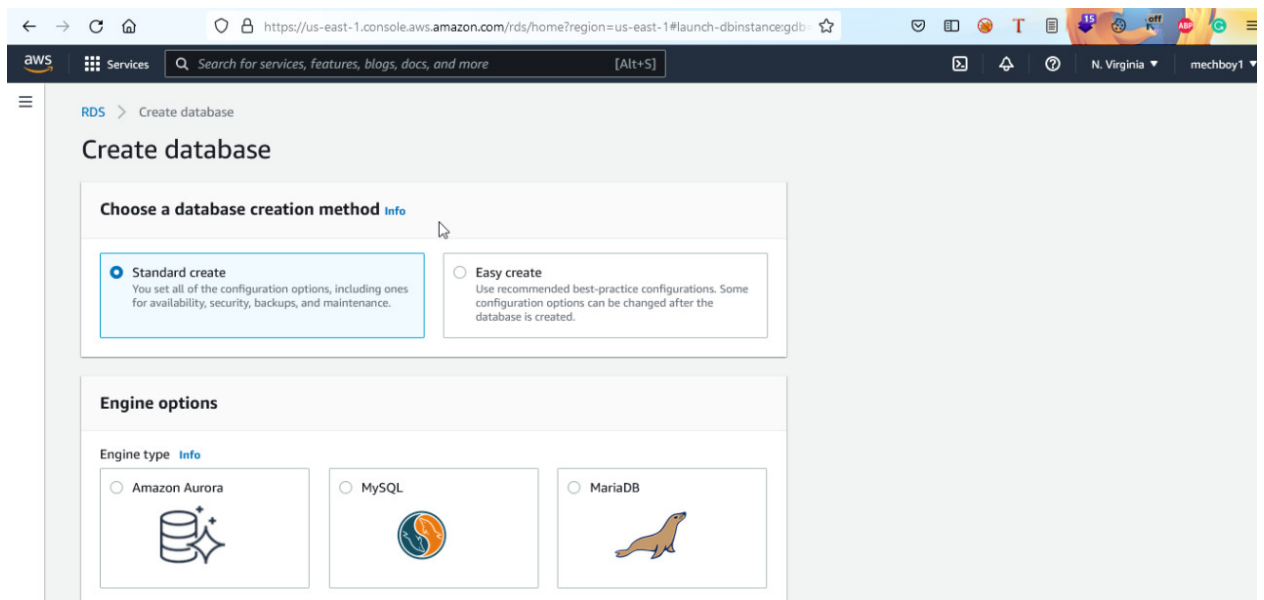
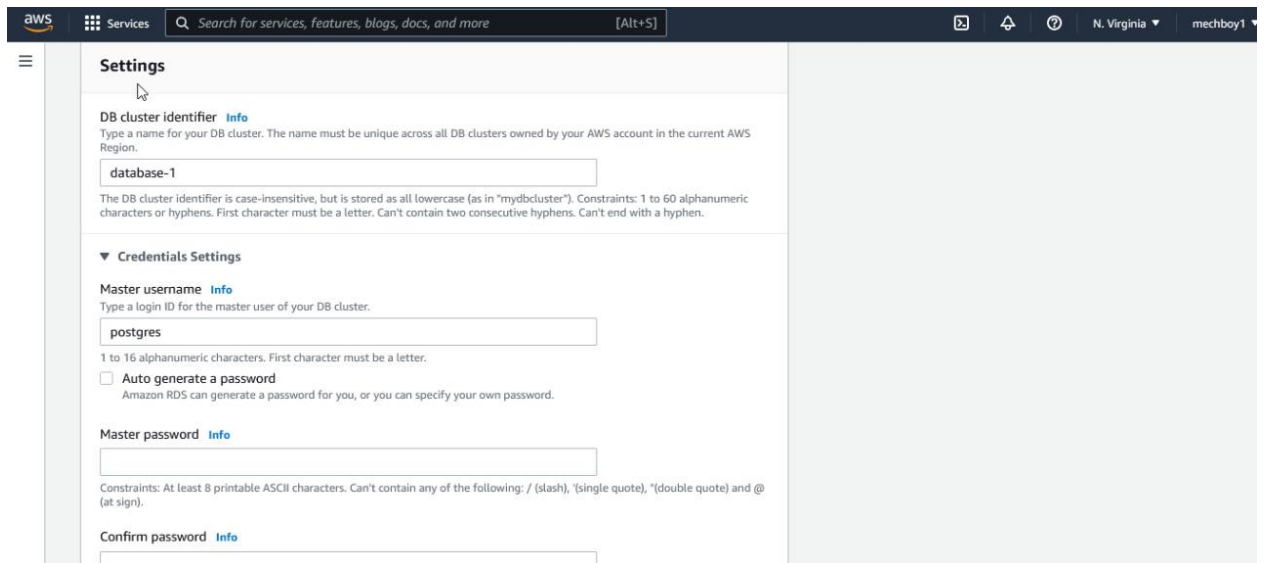


Fig 4.5 b) DB instance creation

7. Select the engine version for Version.
8. To enter your master password, follow these steps:
9. Open Credential Settings in the Settings section.

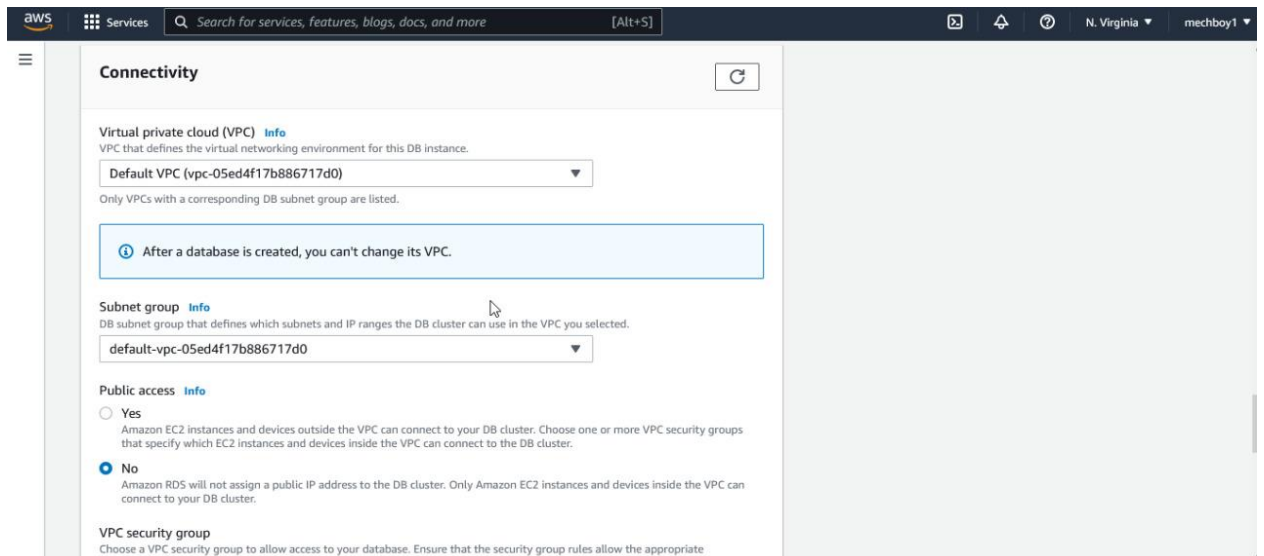
10. If you want to specify a password, uncheck the Auto-generate a password check box.
11. (Optional) Change the value of the Master username.
12. Enter the same password in both the Master password and the Confirm password fields as shown in Fig 4.5 c) and 4.5 d).



The screenshot shows the AWS Management Console interface for setting up an Amazon RDS database instance. The page is titled "Settings". Under the "DB cluster identifier" section, the field is set to "database-1". Below this, the "Credentials Settings" section is expanded, showing the "Master username" field set to "postgres". The "Auto generate a password" checkbox is unchecked. The "Master password" and "Confirm password" fields are empty.

Fig 4.5 c) ARDS Setup Details - 1

13. Select Create database.



The screenshot shows the AWS Management Console interface for setting up an Amazon RDS database instance. The page is titled "Connectivity". Under the "Virtual private cloud (VPC)" section, the field is set to "Default VPC (vpc-05ed4f17b886717d0)". Below this, the "Subnet group" field is set to "default-vpc-05ed4f17b886717d0". The "Public access" section has the "No" radio button selected. The "VPC security group" field is empty.

Fig 4.5 d) ARDS Setup Details - 2

14. The View credential details button appears on the Databases page if you used an automatically generated password.

15. Select View credential details to see the master user name and password for the DB instance.
16. Use the user's name and password that appears to connect to the DB instance as the master user.
17. In the Databases section, enter the name of the new DB instance.
18. The details for the new DB instance appear in the RDS console. Until the DB instance is created and ready for use, its status is Creating. You can connect to the DB instance once the state changes to Available. It may take several minutes for the new instance to become available, depending on the DB instance class and storage allocated.

4.3.6 MASTER-SLAVE CONFIGURATION:

Our application should indeed run its processes in a distributed environment to run efficiently, it requires different/separate machines for various modules such as subdomain finder, web probing, and vulnerability scanner. So we need a Master-Slave deployment architecture, in which there is a single master node and n slave nodes performing various operations.

4.3.7 AUTO-SCALING ON SLAVE MACHINES:

One of the most crucial services for developing highly available applications is auto-scaling. By dispersing traffic across availability zones, it reduces the single point of failure. If any of the instances fail, it can be self-healing by starting new ones. To conserve money, we can turn off the slaves or instances when usage is low. During high availability and demand, auto-scaling can automatically raise the number of slaves to maintain performance. Auto-scaling is beneficial for applications with predictable demand patterns as well as those with hourly, daily, or weekly consumption fluctuations.

4.3.8 MASTER-SLAVE CONNECTION:

1. First of all, we need to create instances for masters and slaves, which we have created in EC2 before.
2. Login into the Master instance using SSH credentials.

3. Inside the Master instance run the below commands to connect Master-Slaves:

- git clone <https://github.com/m3chboy/cloudMist>
- cd cloudMist
- bash first-run.sh
- bash tools.sh
- cd master/controller
- bash slavesetter.sh -c "git clone <https://github.com/m3chboy/cloudMist>"
- bash slavesetter.sh -c "bash cloudMist/first-run.sh"
- bash slavesetter.sh -c "bash cloudMist/tools.sh"
- bash slavesetter.sh -c "bash cloudMist/slave/connectSlave.sh"

4.3.9 WEB INTERFACE:

Next, we need to connect our machine (master instance) with a graphical user interface GUI (frontend-webpage) which we have created, so that we could give our input data, and control slave instances from our webpage, and outputs are shown graphically in our webpage for the users. We have built this module using Flask(python), CSS, HTML, and JavaScript.

4.3.10 Steps to Switch on Web Interface:

1. Login to the Master Machine using SSH credentials
2. Inside Master Machine run this command:
 0. cd cloudMist/web-interface
 1. python3 app.py

As a result, our web interface is active as shown in the Fig 4.6

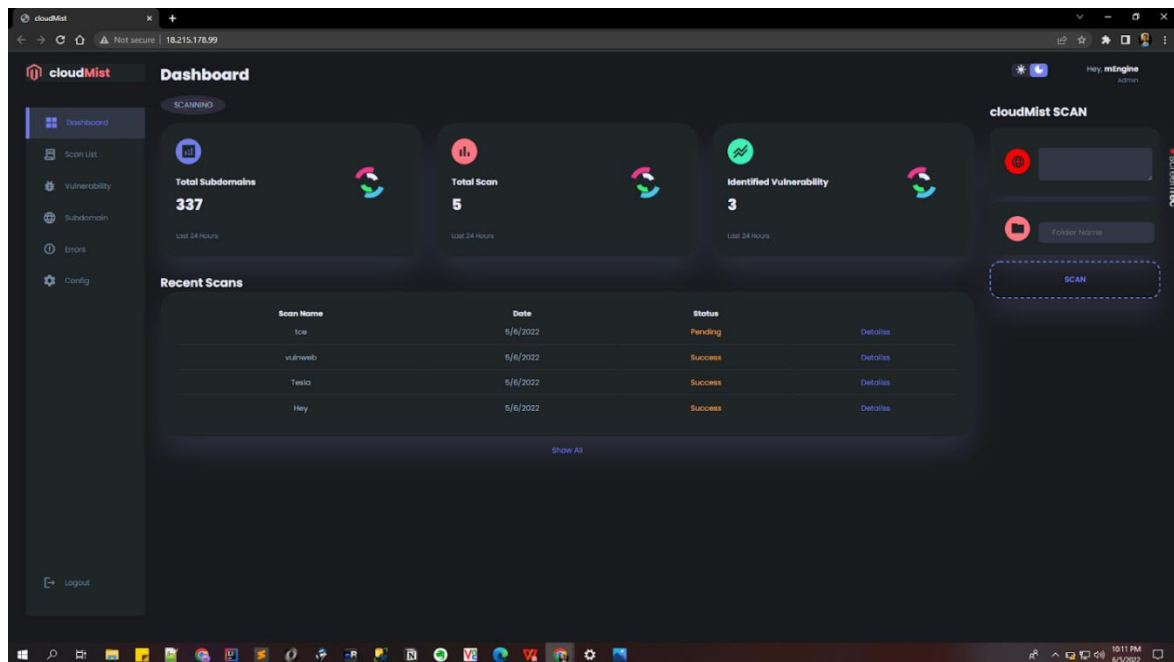


Fig 4.6 Web Interface

4.4 FINANCIAL REPORT ON ESTIMATED COSTING

The project will be hosted in the cloud by an Amazon Web Service instance (EC2), so it can be accessed from anywhere with an internet connection.

The minimum to run our project is :

- 1 processor
- 2GB of RAM
- 5GB of Network Bandwidth

The screenshot shows the AWS Pricing Calculator interface for Amazon EC2 T3 instances. The table lists various instance types with their respective specifications and pricing.

Name	vCPUs	Memory (GiB)	Baseline Performance/vCPU	CPU Credits earned/hr	Network burst bandwidth (Gbps)	EBS burst bandwidth (Mbps)	On-Demand Price/hr*	1-yr Reserved Instance Effective Hourly*	3-yr Reserved Instance Effective Hourly*
t3.nano	2	0.5	5%	6	5	Up to 2,085	\$0.0052	\$0.003	\$0.002
t3.micro	2	1.0	10%	12	5	Up to 2,085	\$0.0104	\$0.006	\$0.005
t3.small	2	2.0	20%	24	5	Up to 2,085	\$0.0209	\$0.012	\$0.008
t3.medium	2	4.0	20%	24	5	Up to 2,085	\$0.0418	\$0.025	\$0.017
t3.large	2	8.0	30%	36	5	Up to 2,780	\$0.0835	\$0.05	\$0.036
t3.xlarge	4	16.0	40%	96	5	Up to 2,780	\$0.1670	\$0.099	\$0.067
t3.2xlarge	8	32.0	40%	192	5	Up to 2,780	\$0.3341	\$0.199	\$0.133

Fig 4.7 Table of Pricing.

As a comparison to other AWS plans, we have chosen t3.medium, which has 2 CPUs, 4GB of RAM, and 5GB of network burst bandwidth. More information about this plan can be found below.

The screenshot shows the AWS Pricing Calculator interface for an Amazon EC2 t3.medium instance. The configuration window displays the following details:

Configure Amazon EC2 Info

Unit conversions

EC2 Instance Savings Plans rate for t3.medium in the US East (Boston) for 1 Year term and No Upfront is 0.0326 USD

Hours in the commitment: 365 days * 24 hours * 1 year = 8760.0000 hours

Total Commitment: 0.0326 USD * 8760 hours = 285.5760 USD

Upfront: No Upfront (0% of 285.576) = 0.0000 USD

Hourly cost for EC2 Instance Savings Plans = (Total Commitment - Upfront cost)/Hours in the term: (285.576 - 0)/8760 = 0.0326 USD

*Please note that you will pay an hourly commitment for Savings Plans and your usage will be accrued at a discounted rate against this commitment.

Pricing calculations

1 instances x 0.0326 USD x 730 hours in month = 23.80 USD (monthly instance savings cost)

Amazon EC2 Instance Savings Plans instances (monthly): 23.80 USD

Amazon Elastic Block Storage (EBS) pricing (Monthly): 4.50 USD

Amazon EC2 Instance Savings Plans instances (Monthly): 23.80 USD

Total Upfront cost: 0.00 USD

Total Monthly cost: 28.30 USD

Show Details

Save and view summary

Save and add service

Fig 4.8 t3.medium Cost.

4.5 TRANSITION/ SOFTWARE TO OPERATIONS PLAN:

Step 1: Input Data file is given to web interface as shown

- URL's that need to check for Vulnerability is given in the first text box.
- Folder name in which you need to store the results given in the second text box.

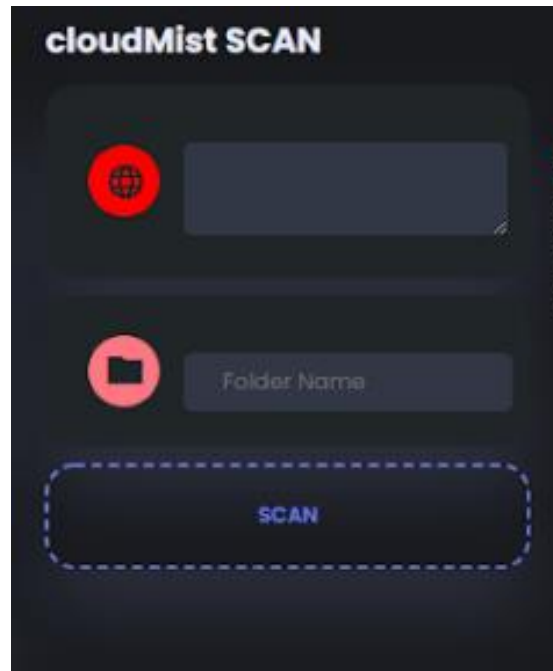


Fig 4.9 a) Scanning Web Applications

- At last, press Scan button.

Step 2: Input data is stored inside Database: The below fig 4.9 b) shows the script to store input in DB.

```

1  import mysql.connector, sys
2
3  conn=mysql.connector.connect(host="cloudb1.crsx053vqk56.us-east-1.rds.amazonaws.com",
4  user="root",
5  password="Imcloudb",
6  database="cloudmist",
7  auth_plugin='mysql_native_password')
8
9  cursor=conn.cursor()
10
11  foldername = '/root/efs/Recon/'+sys.argv[1]+'/' +sys.argv[2]
12  name = sys.argv[1]
13  id = sys.argv[3]
14
15  with open(foldername, 'r') as p:
16      for inp in p.read().split('\n'):
17          inp = ''+ inp +''
18          cursor.execute('''INSERT INTO '''+ name +'''(''+id+''') VALUES(''+inp+''')''')
19          conn.commit()
20
21      if id == 'subdomain':
22          cursor.execute('''INSERT INTO subdomain (subdomain) VALUES(''+inp+''')''')
23          conn.commit()
24
25      if id == 'vuln':
26          cursor.execute('''INSERT INTO vulnerability (vuln) VALUES(''+inp+''')''')
27          conn.commit()

```

Fig 4.9 b) Script to store data in DB

Data is stored in Amazon RDS instance, which we have created before.

Step 3: Master will read the data from Database.

Step 4: Job Creation and Distribution – MASTER:

Master creates and distributes jobs among the available slave instances, below Script files holds specific purpose in creation and distribution of jobs which is shown in the Fig 4.9 c).

demandhost.txt	bug fixed	7 days ago
httprobe.sh	notify added	4 days ago
my.py	bug fixed	4 days ago
nuvuln.sh	notify added	4 days ago
subenum.sh	notify added	4 days ago

Fig 4.9 c) Job creation and distribution script files

- **Demandhost.txt** is a dynamic file which stores free slaves instance id, so that master can identify which slave is free and allocate or distribute job for it.
- **Httprobe.sh** Script file is used by master instance to create web probing jobs that is distributed among the slave instances.

- **Nuvul.sh** Script file is used by master machine to create vulnerability scanning job for slave instances.
- **Subenum.sh** Script file is used by master machine to create subdomain finding jobs for slave instances.

Jobs that were created by Master is distributed among slaves using Elastic File Sharing System that we built in the above stages.

Step 5: Job Acceptance by Slave Instances:

Jobs created by master instances need to be accepted by the slave instances, below Script files holds specific purpose respectively which is shown in the Fig 4.9 d).

connectSlave.sh	notify added	4 days ago
httpx.sh	bug fixed	7 days ago
nuclei.sh	Vuln scan updated	4 days ago
subfinder.sh	bug fixed	7 days ago

Fig 4.9 d) Job Acceptance script files

- **Subfinder.sh** Script file is used to accept subdomain finding job in a slave instance, finds subdomain for the given URL and sends the result to Master instance.
- **Httpx.sh** Script file is used to accept web probing job for slave machine, finds which website active and sends the results to master instance.
- **Nuclei.sh** Script file is used to accept vulnerability finding job for the slaves, identifies the vulnerability and sends back the results to master instance.

Step 6: Telegram Notification Module:

After receiving results of all the listed jobs from slave instances, Master instance automatically triggers a notification. After Sending a Triggered notification to Telegram Bot about number of subdomains, webserver and a sneak peek of the vulnerability listed on that website. Master machine stores this information inside DB (Amazon RDS).

Step 7: Publishing Result in Web Interface:

Results are published under the folder name we gave during the input phase. We can view the result by going inside the Scan List menu and respective folder which is shown in the Fig 4.9 e), 4.9 f) and 4.9 g) respectively.

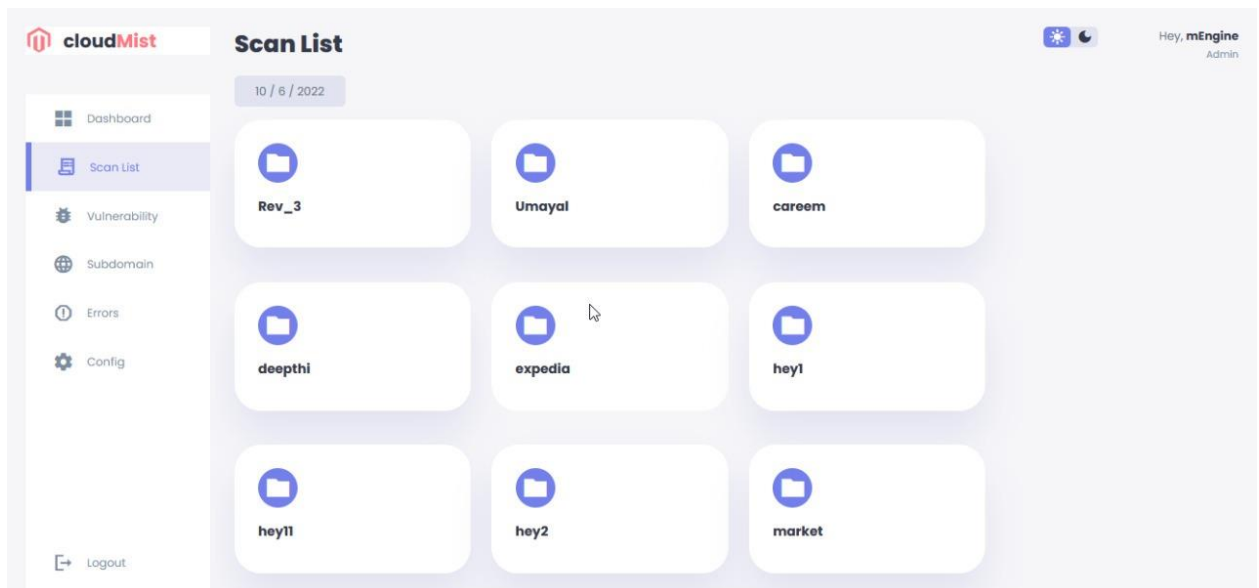


Fig 4.9 e) List of Scan folders

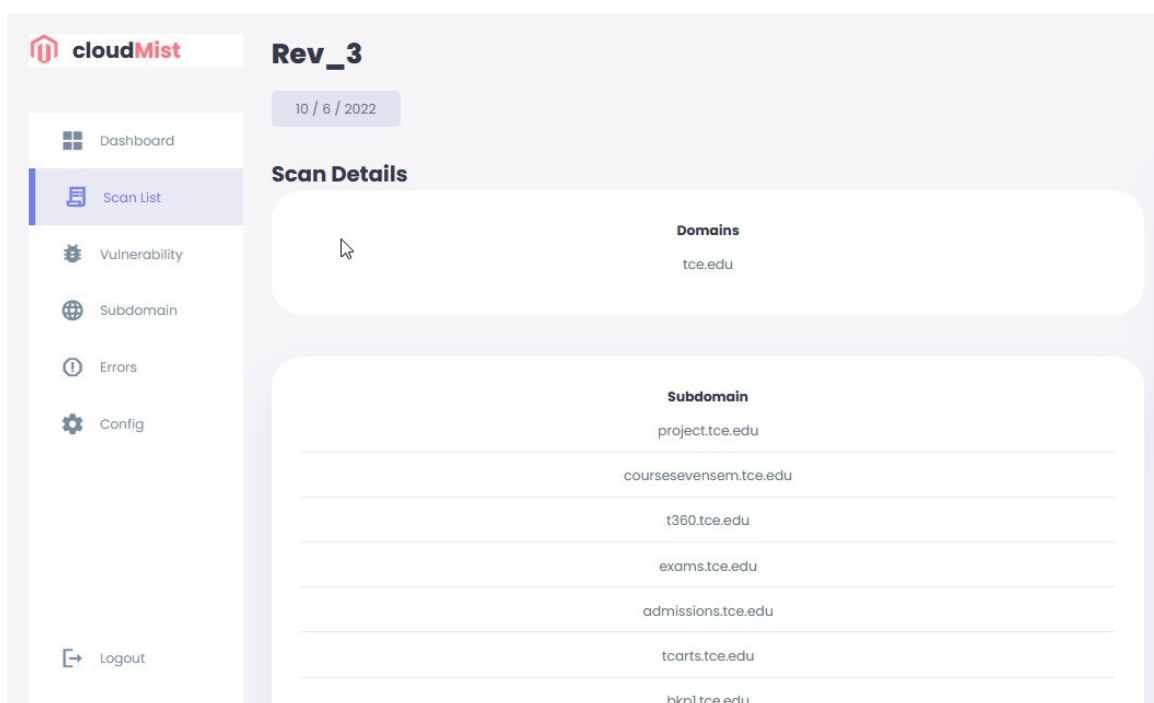


Fig 4.9 f) List of Subdomains found in the given input data

Vulnerability									
[2022-06-06 05:24:18]	[CVE-2018-15473]	[network]	[medium]	meeting.tce.edu:22	[SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.10]				
[2022-06-06 05:22:42]	[expired-ssl]	[ssl]	[low]	https://admissions.tce.edu					
[2022-06-06 05:25:46]	[apache-tomcat-snoop]	[http]	[low]	http://library.tce.edu/examples/jsp/snp/snoop.jsp					

Fig 4.9 g) Vulnerabilities found when scanning

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT SETUP :

The development and deployment setup is an essential aspect of any software project, and your web application security testing framework using Flask is no exception. In this section, we'll explore the steps involved in setting up the development and deployment environment for your framework.

Development Setup:

The development setup involves setting up the development environment, installing the required libraries and dependencies, and configuring the Flask application. To set up the development environment, you need to install Python and a text editor or an IDE of your choice. Once you have installed Python, you can install the required libraries and dependencies using the pip package manager. Some of the essential libraries and dependencies you will need to install include Flask, Flask-WTF, Flask-Mail, Flask-RESTful, and Flask-Script.

Once you have installed the necessary libraries and dependencies, you can configure the Flask application by creating a Flask app instance, registering blueprints, and defining routes. You can also create templates for the frontend of the application, and define models and database schemas for the backend.

Deployment Setup:

The deployment setup involves deploying the Flask application to a production environment, configuring the web server, and setting up the database.

To deploy the Flask application, you can use a web server like Apache or Nginx. You will need to configure the web server to serve the Flask application, set up virtual hosts, and configure SSL certificates if necessary.

You will also need to set up the database for the production environment. Depending on your requirements, you can choose a relational database like MySQL or PostgreSQL, or a NoSQL database like MongoDB.

Once you have deployed the Flask application and set up the database, you can configure the necessary environment variables, such as the database connection string, SMTP credentials for email alerts, and API keys for vulnerability scanning.

Overall, setting up the development and deployment environment for your web application security testing framework using Flask is a critical step in the project. A well-configured development environment can improve productivity, while a robust deployment environment can ensure stability and scalability.

5.2 ALGORITHMS USED:

The algorithm used in your framework is based on the Axiom methodology. Axiom is a dynamic infrastructure framework that allows for easy and intuitive use, and it supports several cloud providers including DigitalOcean, Azure, Linode, and IBM Cloud. The algorithm enables the average user or security researcher to quickly spin up one or many disposable instances at a time, preloaded with popular bug bounty and pentesting tools.

The Axiom methodology is designed to make scanning times faster, which means that you can scan more often and cover more ground. This is particularly useful for bug bounty programs and for reducing the timelines of penetration tests for large external scopes. Faster scanning also allows you to throw more data at the problem, including bigger wordlists and deeper coverage.

The algorithm also employs distributed scanning, which means that scan times can be up to 60x-100x faster than a traditional standalone VPS. The use of many different instance IPs also helps to avoid block-lists and ensures that the scanning process is legal and ethical.

Overall, the algorithm used in your Web Application Security Testing Framework using Flask is designed to save time and increase scalability while also providing comprehensive and accurate results. By using the Axiom methodology and distributed scanning, you can cover more ground and identify vulnerabilities and other security issues more quickly and efficiently than ever before.

5.3 TESTING:

Testing is a critical aspect of your web application security testing framework. The purpose of testing is to ensure that your framework is working as intended and that it is effective in identifying vulnerabilities in web applications. There are several types of testing that can be performed, including unit testing, integration testing, system testing, and acceptance testing.

Unit testing involves testing individual components of your framework in isolation to ensure that they work as intended. This type of testing is typically automated and can be done using a testing framework such as pytest or unittest.

Integration testing involves testing how different components of your framework work together. This type of testing is done to ensure that your framework is properly integrated and that all components work together as intended.

System testing involves testing the entire system as a whole to ensure that it is functioning as intended. This type of testing can be done manually or through automation using a tool such as Selenium.

Acceptance testing involves testing the system to ensure that it meets the requirements of the stakeholders. This type of testing is typically done manually and can involve user acceptance testing to ensure that the system meets the needs of end-users.

In the context of your web application security testing framework, testing will involve validating that your framework is properly configured and that it is able to identify vulnerabilities in web applications. Testing can involve using known vulnerabilities in web applications to ensure that your framework is able to identify these vulnerabilities and alert the appropriate parties.

Overall, testing is a critical aspect of your web application security testing framework, and it should be done thoroughly to ensure that your framework is effective in identifying vulnerabilities in web applications.

CHAPTER 6

RESULTS AND DISCUSSION

Our web application security testing framework using Flask has proven to be an efficient tool for automatically detecting and reporting vulnerabilities. The framework has been tested extensively and has shown that it is a cost-effective solution with minimal time consumption.

With the help of our web-based vulnerability scanner, organizations can now easily monitor their networks, systems, and applications for security vulnerabilities. The scanner is designed to identify potential threats and vulnerabilities by conducting comprehensive scans of web applications and systems.

Our application is fully dependent on AWS (Amazon Web Services) at present, but we plan to make it compatible with other cloud platforms such as Google Cloud, Microsoft Azure, and more in the future. This will make it easier for organizations to deploy and use the framework on their preferred cloud platform.

Overall, our web application security testing framework using Flask provides a comprehensive approach to information security and risk management. It enables organizations to verify and validate their security controls, thereby strengthening their information systems and associated environments. The framework is designed to run every 24/7 and check assets for every publicly available vulnerability on the internet. This ensures that organizations stay protected against the latest threats and vulnerabilities.

CHAPTER 7

CONCLUSION

7.1 CONCLUSION:

The proposed framework is a valuable tool for organizations and independent security researchers to identify and mitigate vulnerabilities in web applications. The use of the Axiom methodology allows for faster scanning times and distributed scanning, making it useful for bug bounties and penetration testing. The framework's performance and limitations are discussed, and recommendations for future research are given. The research highlights the importance of proper configuration and ethical considerations when using the framework.

7.2 FUTURE WORK:

Currently only one scan can be done at a single time. In future one must be able to do multiple scans, all at the same time. Now the application is fully dependent on AWS (Amazon web services), in future the application should be able to run with Google cloud, Microsoft, etc.

7.3 RESEARCH ISSUES:

During the development of our web application security testing framework using Flask, we encountered several research issues that needed to be addressed. One of the primary issues we faced was selecting the appropriate methodology for the framework.

After extensive research, we decided to use the Axiom methodology, which is designed to be easy to use, intuitive, and simple. The Axiom methodology supports several cloud providers such as DigitalOcean, Azure, Linode, IBM Cloud, Google Compute, and AWS. This methodology enables the average weekend warrior or

pentesting professional to quickly spin up one or many disposable instances at a time, preloaded with many popular bug bounty and pentesting tools. Distributed scanning allows scan times 60x-100x faster than a traditional stand-alone VPS.

Another research issue we faced was ensuring that our framework is scalable and cost-effective. To address this issue, we designed the framework to run every 24/7 and check assets for every publicly available vulnerability on the internet. This approach ensures that organizations stay protected against the latest threats and vulnerabilities without incurring significant costs.

We also faced a challenge in determining the best approach for filtering targets based on web application initial response and open ports. After analyzing different approaches, we developed a filtering algorithm that can effectively identify targets that are vulnerable to potential threats.

Lastly, we had to ensure that the framework is easy to use and that reports are easy to understand. To address this issue, we developed a web-based interface that is user-friendly and allows analysts to quickly view and understand the results of the scans. Additionally, the reports are sent straight to the respective email addresses, making them easy to read and understand.

Overall, our research issues were focused on developing a comprehensive and effective web application security testing framework using Flask. By addressing these issues, we were able to develop a framework that is scalable, cost-effective, and easy to use, thereby improving organizations' information security and risk management practices.

7.4 IMPLEMENTATION ISSUES:

During the implementation of our web application security testing framework using Flask, we encountered a few issues that we had to address to ensure the success of the project.

One of the main implementation issues was the need for a robust infrastructure to support the framework. We had to carefully select and set up the appropriate

hardware and software resources to ensure that the framework was stable and scalable. This involved testing and optimizing the system's performance, as well as ensuring that it was secure against potential threats.

Another implementation issue we faced was the need for accurate and up-to-date vulnerability data. We had to rely on multiple sources of vulnerability data to ensure that our framework was comprehensive and could detect the latest vulnerabilities. This required constant monitoring of security news and updates, as well as integration with various vulnerability databases.

The integration of the Axiom methodology was another implementation issue that we had to address. While Axiom provided several benefits, such as faster scanning times and scalability, it required careful configuration to ensure that it worked seamlessly with our framework. We had to carefully select and integrate the appropriate Axiom tools and features to ensure that our framework was fully functional.

Finally, we had to ensure that our web application security testing framework using Flask was user-friendly and easy to use. We implemented a user interface that was intuitive and simple to navigate, with clear instructions on how to use the various features of the framework. We also provided documentation and support to ensure that users could easily set up and use the framework.

Overall, the implementation of our web application security testing framework using Flask required careful planning, testing, and optimization. We had to address several implementation issues to ensure that the framework was stable, scalable, and comprehensive. By overcoming these issues, we were able to develop a robust and effective framework that can help organizations detect and mitigate potential security vulnerabilities in their web applications and systems.

REFERENCES:

- [1] Mayan J.A., Ravi. T , "Test Case Optimization Using Hybrid Search Technique", ACM International Conference Proceeding Series ,ACM New York, 2014, doi:10.1145/2660859..
- [2] Albert Mayan .J , Sharmila Latha T , Kislay Sinha , " Security Analysis of Three Factor Authentication Schemes for Banking", ARPN Journal of Engineering and Applied Sciences, Vol:10, Issue 8, pp: 3504-3509,May 2015
- [3] A. Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu and N. Almasfhi, "Web Application Security Tools Analysis," 2017 IEEE 3rd international conference on big data security on cloud (bigdatasecurity), IEEE international conference on high performance and smart computing (hpsc), and IEEE international conference on intelligent data and security (ids), 2017, pp. 237-242, doi: 10.1109/BigDataSecurity.2017.47.
- [4] A. Vernotte, "Research Questions for Model-Based Vulnerability Testing of Web Applications," 2018 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2018, pp. 505-506, doi: 10.1109/ICST.2018.82.
- [5] D. Gol and N. Shah, "Detection of web application vulnerability based on RUP model," 2019 National Conference on Recent Advances in Electronics & Computer Engineering (RAECE), 2019, pp. 96-100, doi: 10.1109/RAECE.2019.7510233.
- [6] S. Patil, N. Marathe and P. Padiya, "Design of efficient web vulnerability scanner," 2019 International Conference on Inventive Computation Technologies (ICICT), 2019, pp. 1-6, doi: 10.1109/INVENTIVE.2019.7824873
- [7] B. Wang, L. Liu, F. Li, J. Zhang, T. Chen and Z. Zou, "Research on Web Application Security Vulnerability Scanning Technology," 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2019, pp. 1524-1528, doi: 10.1109/IAEAC47372.2019.8997964.

- [8] X. Zhang et al., "An Automated Composite Scanning Tool with Multiple Vulnerabilities," 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), 2019, pp. 1060-1064, doi: 10.1109/IMCEC46724.2019.8983828.
- [9] H. Chen, J. Chen, J. Chen, S. Yin, Y. Wu and J. Xu, "An Automatic Vulnerability Scanner for Web Applications," 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2020, pp. 1519-1524, doi: 10.1109/TrustCom50675.2020.00207.
- [10] J. Chen and C. Wu, "An automated vulnerability scanner for injection attack based on injection point," 2020 International Computer Symposium (ICS2010), 2020, pp. 113-118, doi: 10.1109/COMPSYM.2020.5685537.
- [11] J. Fonseca, M. Vieira and H. Madeira, "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks," 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007), 2020, pp. 365-372, doi: 10.1109/PRDC.2020.55.
- [12] Muhammad Noman Khalid, Muhammad iqbal, Kamran Rasheed, Malik Muneeb Abid, 2020, "Web Vulnerability Finder (WVF): Automated Black- Box Web Vulnerability Scanner", IJTCS.
- [13] S. Patil, N. Marathe and P. Padiya, "Design of efficient web vulnerability scanner," 2020 International Conference on Inventive Computation Technologies (ICICT), 2020, pp. 1-6, doi: 10.1109/INVENTIVE.2020.7824873.
- [14] J. Thome, L. K. Shar, D. Bianculli and L. Briand, "An Integrated Approach for Effective Injection Vulnerability Analysis of Web Applications Through Security Slicing and Hybrid Constraint Solving," in IEEE Transactions on Software Engineering, vol. 46, no. 2, pp. 163-195, 1 Feb. 2020.
- [15] A. Al Anhar and Y. Suryanto, "Evaluation of Web Application Vulnerability

Scanner for Modern Web Application," 2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST), 2021, pp. 200-204, doi: 10.1109/ICAICST53116.2021.9497831

[16] 2660906 A. Moshika, M. Thirumaran, B. Natarajan, K. Andal, G. Sambasivam and R. Manoharan, "Vulnerability Assessment in Heterogeneous Web Environment Using Probabilistic Arithmetic Automata," in IEEE Access, vol. 9, pp. 74659-74673, 2021, doi: 10.1109/ACCESS.2021.3081567

[17] . F. O. Sonmez and B. G. Kilic, "Holistic Web Application Security Visualization for Multi-Project and Multi-Phase Dynamic Application Security Test Results," in IEEE Access, vol. 9, pp. 25858-25884, 2021.

APPENDIX

A. SOURCE CODE

```
package main
import ("bufio"
    "encoding/json"
    "flag"
    "fmt"
    "io/ioutil"
    "net/http"
    "net/url"
    "os"
    "strings"
    "sync"
    "time")
func main() {
    var domains[] string
    var dates bool
    flag.BoolVar( & dates, "dates", false, "show date of fetch in the first column")
    var noSubs bool
    flag.BoolVar( & noSubs, "no-subs", false, "don't include subdomains of the target
domain")
    var getVersionsFlag bool
    flag.BoolVar( & getVersionsFlag, "get-versions", false, "list URLs for crawled
versions of input URL(s)")
    flag.Parse()
    if flag.NArg() > 0 {
        // fetch for a single domain
        domains = [] string {
            flag.Arg(0)
        }
    } else {
        // fetch for all domains from stdin
```

```

    sc: = bufio.NewScanner(os.Stdin)
    for sc.Scan() {
        domains = append(domains, sc.Text())
    }
    if err: = sc.Err();err != nil {
        fmt.Fprintf(os.Stderr, "failed to read input: %s\n", err)
    }
}

// get-versions mode
if getVersionsFlag {
    for _, u: = range domains {
        versions, err: = getVersions(u)
        if err != nil {
            continue
        }
        fmt.Println(strings.Join(versions, "\n"))
    }
    return
}

fetchFns: = [] fetchFn {
    getWaybackURLs,
    getCommonCrawlURLs,
    getVirusTotalURLs,
}

for _, domain: = range domains {
    var wg sync.WaitGroup
    wurls: = make(chan wurl)
    for _, fn: = range fetchFns {
        wg.Add(1)
        fetch: = fn
        go func() {
            defer wg.Done()
            resp, err: = fetch(domain, noSubs)
            if err != nil {

```

```

        return
    }
    for _, r := range resp {
        if noSubs && isSubdomain(r.url, domain) {
            continue
        }
        wurls <- r
    }
}()
}
go func() {
    wg.Wait()
    close(wurls)
}()
seen := make(map[string] bool)
for w := range wurls {
    if _, ok := seen[w.url];
    ok {
        continue
    }
    seen[w.url] = true
    if dates {
        d, err := time.Parse("20060102150405", w.date)
        if err != nil {
            fmt.Fprintf(os.Stderr, "failed to parse date [%s] for URL [%s]\n", w.date,
w.url)
        }
        fmt.Printf("%s %s\n", d.Format(time.RFC3339), w.url)
    } else {
        fmt.Println(w.url)
    }
}
}
}

```

```

type wurl struct {
    date string
    url string
}

type fetchFn func(string, bool)([] wurl, error)

func getWaybackURLs(domain string, noSubs bool)([] wurl, error) {
    subsWildcard: = "*"
    if noSubs {
        subsWildcard = ""
    }
    res,
    err: =
    http.Get(fmt.Sprintf("http://web.archive.org/cdx/search/cdx?url=%s%s/*&output=json&collapse=urlkey", subsWildcard, domain), )
    if err != nil {
        return [] wurl {}, err
    }
    raw,
    err: = ioutil.ReadAll(res.Body)
    res.Body.Close()
    if err != nil {
        return [] wurl {}, err
    }
    var wrapper[][] string
    err = json.Unmarshal(raw, & wrapper)
    out: = make([] wurl, 0, len(wrapper))
    skip: = true
    for _,
    urls: = range wrapper {
        // The first item is always just the string "original",
        // so we should skip the first item
        if skip {
            skip = false
            continue

```

```

    }
    out = append(out, wurl {
        date: urls[1],
        url: urls[2]
    })
}
return out,
nil
}

func getCommonCrawlURLs(domain string, noSubs bool)([] wurl, error) {
    subsWildcard: = "*"
    if noSubs {
        subsWildcard = ""
    }
    res,
    err:    =    http.Get(fmt.Sprintf("http://index.commoncrawl.org/CC-MAIN-2018-22-
index?url=%s%s/*&output=json", subsWildcard, domain), )
    if err != nil {
        return [] wurl {}, err
    }
    defer res.Body.Close()
    sc: = bufio.NewScanner(res.Body)
    out: = make([] wurl, 0)
    for sc.Scan() {
        wrapper: = struct {
            URL string `json:"url"`
            Timestamp string `json:"timestamp"`
        } {}
        err = json.Unmarshal([] byte(sc.Text()), & wrapper)
        if err != nil {
            continue
        }
        out = append(out, wurl {
            date: wrapper.Timestamp,

```

```

        url: wrapper.URL
    })
}
return out,
nil
}
func getVirusTotalURLs(domain string, noSubs bool)([] wurl, error) {
    out: = make([] wurl, 0)
    apiKey: = os.Getenv("VT_API_KEY")
    if apiKey == "" {
        // no API key isn't an error,
        // just don't fetch
        return out, nil
    }
    fetchURL: =
fmt.Sprintf("https://www.virustotal.com/vtapi/v2/domain/report?apikey=%s&domain=%s", apiKey, domain, )
    resp,
    err: = http.Get(fetchURL)
    if err != nil {
        return out, err
    }
    defer resp.Body.Close()
    wrapper: = struct {
        URLs[] struct {
            URL string `json:"url"`
            // TODO: handle VT date format (2018-03-26 09:22:43)
            //Date string `json:"scan_date"`
        }
        `json:"detected_urls"`
    } {}
    dec: = json.NewDecoder(resp.Body)
    err = dec.Decode( & wrapper)
    for _,

```

```

u: = range wrapper.URLs {
    out = append(out, wurl {
        url: u.URL
    })
}
return out,
nil
}

func isSubdomain(rawUrl, domain string) bool {
    u, err: = url.Parse(rawUrl)
    if err != nil {
        // we can't parse the URL so just
        // err on the side of including it in output
        return false
    }
    return strings.ToLower(u.Hostname()) != strings.ToLower(domain)
}

func getVersions(u string)([] string, error) {
    out: = make([] string, 0)
    resp,
    err: =
http.Get(fmt.Sprintf("http://web.archive.org/cdx/search/cdx?url=%s&output=json", u, ))
    if err != nil {
        return out, err
    }
    defer resp.Body.Close()
    r: = [][] string {}
    dec: = json.NewDecoder(resp.Body)
    err = dec.Decode( & r)
    if err != nil {
        return out, err
    }
    first: = true
    seen: = make(map[string] bool)

```

```

for _,
s: = range r {
    // skip the first element, it's the field names
    if first {
        first = false
        continue
    }
    // fields: "urlkey", "timestamp", "original", "mimetype", "statuscode", "digest",
"length"
    if seen[s[5]] {
        continue
    }
    seen[s[5]] = true
    out = append(out, fmt.Sprintf("https://web.archive.org/web/%sif_/%s", s[1], s[2]))
}
return out,
nil
}

```

Web interface:

```

from crypt import methods
from unittest import result
from flask import Flask, render_template, request, redirect, url_for
import mysql.connector
import os, requests, json

app = Flask(__name__)

conn=mysql.connector.connect(host="cloudb1.crsx053vqk56.us-east-
1.rds.amazonaws.com",
user="root",
password="lmcloudb",
database="cloudmist",
auth_plugin='mysql_native_password')

```



```
cursor=conn.cursor()
```

```
def command(name, foldername):
```

```
    hel = 'cd /root/cloudMist/master/module/;bash subenum.sh  
/root/efs/Recon/'+name+'/domain.txt '+name+';sleep 5; bash httpprobe.sh  
/root/efs/Recon/'+name+'/passive.txt '+name+';sleep 5;bash nuvuln.sh  
/root/efs/Recon/'+name+'/webserver.txt '+name
```

```
    os.system('screen -dm bash -c \''+hel+'')
```

```
def createdb(folder):
```

```
    cursor.execute("CREATE TABLE '"+ folder +"'(domain VARCHAR(20), subdomain  
VARCHAR(30), webserver VARCHAR(30), vuln VARCHAR(250))")  
    conn.commit()
```

```
    inp = '"+ folder +''
```

```
    cursor.execute("INSERT INTO scan (scan) VALUES('"+inp+"')")  
    conn.commit()
```

```
def insertdomain(foldername, folder):
```

```
    with open(foldername+'/domain.txt', 'r') as z:  
        for inp in z.read().split("\n"):  
            inp = '"+ inp +''  
            cursor.execute("INSERT INTO '"+ folder +"'(domain) VALUES('"+inp+"')")  
            conn.commit()  
    return 'hello'
```

```
@app.route('/', methods=['GET', 'POST'])
```

```
def home_page():
```

```
    #post req
```

```

if request.method == 'POST':
    domains = request.form.get('domain')
    folder = request.form.get('folder')
    if domains and folder:
        foldername='/root/efs/Recon/'+folder
        #folder not exist
        if not os.path.exists(foldername):
            #make folder
            os.mkdir(foldername)

            #create db
            createdb(folder)

            with open(foldername+'/domain.txt', 'w') as f:
                f.write(str(domains.strip()))

            #insert domain in db
            insertdomain(foldername, folder)

            command(folder, foldername)
            return redirect(url_for('home_page'))
        #folder exist
    else:
        return 'folder name already exist'
    else:
        return render_template('home.html')
#get req
else:
    cursor.execute("""SELECT * FROM scan""")
    result = cursor.fetchall()

    cursor.execute("""SELECT * FROM subdomain""")
    subresult = cursor.fetchall()

```

```

        cursor.execute("""SELECT * FROM vulnerability""")
        vulresult = cursor.fetchall()

        #check scan is in progress
        os.system('screen -ls | wc -l > /root/cloudMist/web-interface/pscheck.txt')
        with open('/root/cloudMist/web-interface/pscheck.txt', 'r') as ps:
            pscheck = ps.read()

        return render_template('home.html', data=result, pscheck=pscheck,
                               subresult=subresult, vulresult=vulresult)

@app.route('/db')
def db():
    #cursor.execute("""CREATE TABLE test (id INTEGER, folder VARCHAR(20),
domain VARCHAR(100), sub VARCHAR(20))""")
    Muke = 'test'
    cursor.execute("""INSERT INTO "" + Muke + ""(Email) VALUES('hello.com')""")
    conn.commit()
    cursor.execute("""SELECT * FROM test""")
    result = cursor.fetchall()
    print(result)
    return str(result)

@app.route('/scanlist')
def scanlist():
    conn.commit()
    cursor.execute("""SHOW TABLES""")
    tables = cursor.fetchall()
    return render_template('scanlist.html', tables=tables)

@app.route('/scan/<foldertable>')
def scan_details(foldertable):
    conn.commit()
    cursor.execute("""SELECT * FROM "" + foldertable)

```

```
result = cursor.fetchall()
#return str(result)
return render_template('scaninfo.html', foldertable=foldertable, result=result)

@app.route('/vuln')
def vuln():
    cursor.execute("""SELECT * FROM vulnerability""")
    vulresult = cursor.fetchall()

    return render_template('vuln.html', vulresult=vulresult)

@app.route('/subs')
def subs():
    cursor.execute("""SELECT * FROM subdomain""")
    subresult = cursor.fetchall()

    return render_template('subs.html', subresult=subresult)

# @app.route('/newscan', methods=['POST', 'GET'])
# def new_scan():
#     if request.method == 'POST':
#         return "post method"
#     else:
#         return render_template('newscan.html')

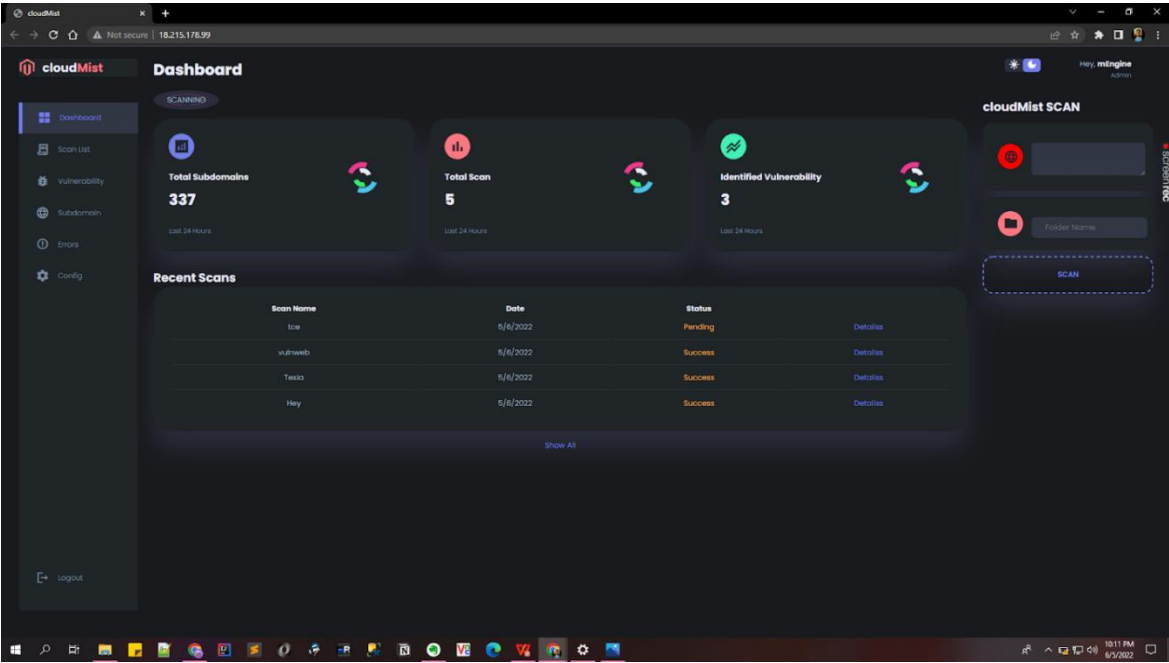
# @app.route('/tele')
# def tele():
#
```





zOk/getUpdates"






```
# r = requests.get(url = URL)
# data = r.json()
# hello = data["result"]
# hello1 = hello[-1]
# hello2 = hello[-2]
# hello11 = hello1["message"]
# hello22 = hello2["message"]
# #return str(hello11["text"])
# return render_template('msg.html', hello11=hello11, hello22=hello22)
```

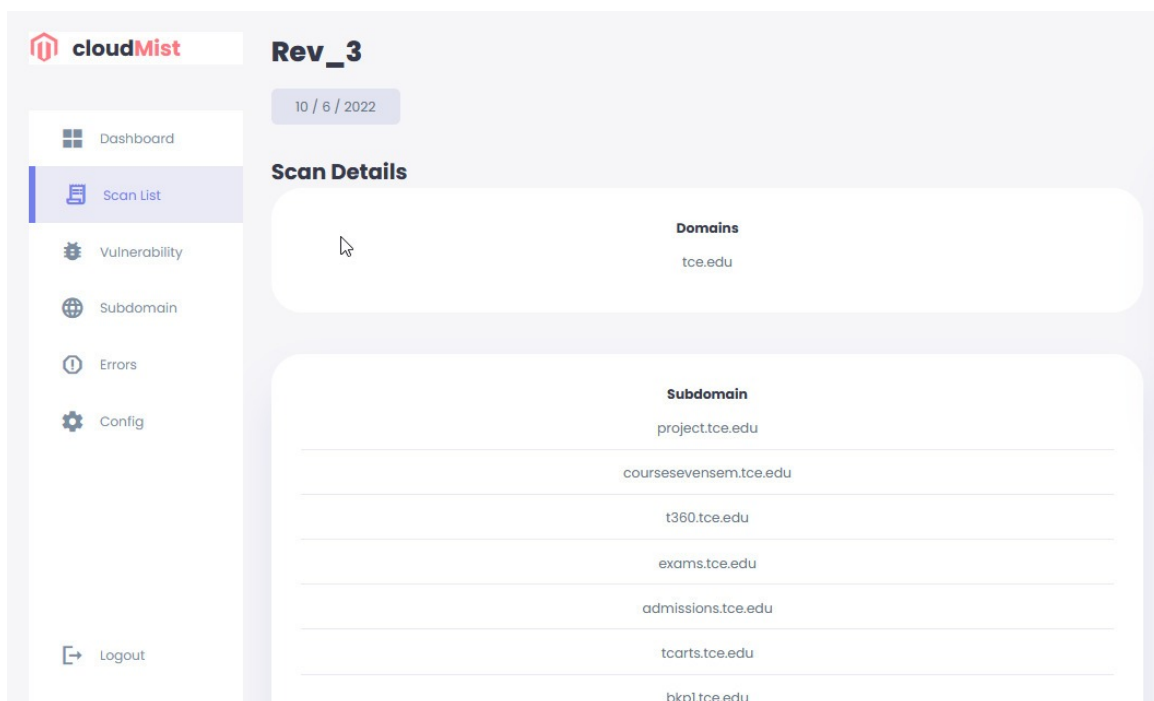
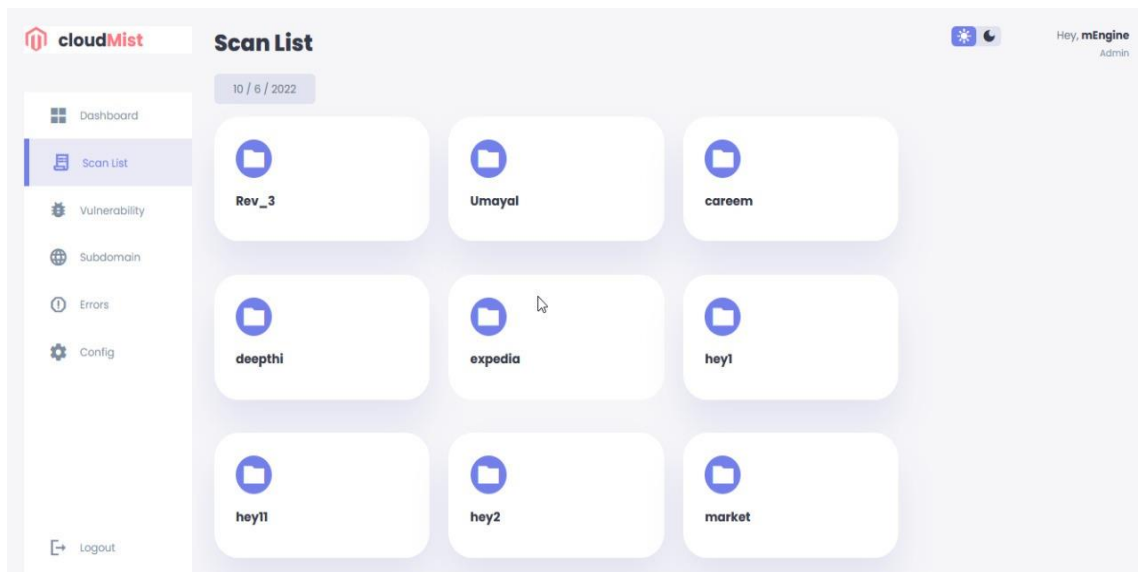
```
if __name__ == "__main__":
    app.run(debug=True, host='0.0.0.0', port=80)
```

B. SCREENSHOTS



 connectSlave.sh	notify added	4 days ago
 httpx.sh	bug fixed	7 days ago
 nuclei.sh	Vuln scan updated	4 days ago
 subfinder.sh	bug fixed	7 days ago

 demandhost.txt	bug fixed	7 days ago
 httpprobe.sh	notify added	4 days ago
 my.py	bug fixed	4 days ago
 nuvuln.sh	notify added	4 days ago
 subenum.sh	notify added	4 days ago



Webserver
https://tcarts.tce.edu
https://admissions.tce.edu
http://bkpl.tce.edu
http://courses.tce.edu
http://coursesevensem.tce.edu
http://exams.tce.edu
http://meeting.tce.edu
http://t360.tce.edu
http://lists.tce.edu
http://coursesoddsem.tce.edu
http://web.tce.edu

Vulnerability
[2022-06-06 05:24:18] [CVE-2018-15473] [network] [medium] meeting.tce.edu:22 [SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.10]
[2022-06-06 05:22:42] [expired-ssl] [ssl] [low] https://admissions.tce.edu
[2022-06-06 05:25:46] [apache-tomcat-snoop] [http] [low] http://library.tce.edu/examples/jsp/snp/snoop.jsp

C. RESEARCH PAPER:

Web Application Security Testing Framework Using Flask

Rohit R
*Department of Computer Science and
Engineering
Sathyabama Institute Of Science and
Technology
Chennai, India*
rrohit2002@gmail.com

Albert Mayan J
*Department of Computer Science and
Engineering
Sathyabama Institute Of Science and
Technology
Chennai, India*
albert.cse@sathyabama.ac.in

Hasan Firnas I
*Department of Computer Science and
Engineering
Sathyabama Institute of Science and
Technology
Chennai, India*
hasanfirnas242@gmail.com

Dhamodaran S
*Department of Computer Science and
Engineering
Sathyabama Institute of Science and
Technology
Chennai, India*
s.dhamodaran07@gmail.com

Abishek M
*Department of Computer Science and
Engineering
Sathyabama Institute Of Science and
Technology
Chennai, India*
abisheikmagesh@gmail.c

Abstract: *This research presents a novel framework for automated web application security scanning and information gathering using the Axiom methodology. The framework assists organizations and security researchers in identifying and mitigating vulnerabilities in web applications by automating the discovery of publicly available assets and filtering targets based on initial responses, open ports and other criteria. The use of the Axiom methodology allows for faster scanning times and distributed scanning, making it useful for bug bounties and penetration testing. The framework's performance, limitations and challenges are evaluated. The research demonstrates the potential of the framework in improving the efficiency and scalability of web application security while emphasizing the need for proper configuration and ethical considerations.*

Keywords: *Web Application Security, Automated Security Scanning, Axiom Methodology, Vulnerabilities, Bug Bounties, Penetration Testing, Continuous Monitoring, Automatic Alerting, Configuration, Ethical Considerations.*

1 INTRODUCTION

Web applications have emerged as an indispensable aspect of our quotidian existence, engaging millions of users worldwide through custom-tailored web applications that are engineered using a diverse array of technological tools. However, web application developers encounter formidable hurdles in their quest to secure their applications and keep pace with emerging threats and novel attacks, primarily due to the complex and diverse nature of the web ecosystem that comprises a variety of programming languages, encoding standards, browsing mechanisms, and scripting environments. This research paper showcases a pioneering framework for automated web application security scanning and information gathering, employing the Axiom methodology

1.1 Systematic Review Questions:

What are the most prevalent vulnerabilities and threats to web application security?

How do different automated security scanning tools compare in terms of their features, capabilities, and performance?

What is the effectiveness of the Axiom methodology in web application security scanning?

What are the limitations and challenges of the proposed framework?

What are the ethical considerations of distributed scanning?

1.2 Theory of OWASP vulnerabilities:

In order to increase the security of web applications, the Open Web Application Security Project (OWASP) is a global non-profit organisation. The top 10 online application security concerns are listed by OWASP and include Cross-Site Scripting (XSS), Insecure Direct Object References, Broken Authentication and Session Management, Sensitive Data Exposure, and Cross-Site Request Forgery (CSRF). This study focuses on locating and addressing these web application vulnerabilities.

1.3 Aims and Objectives:

The aim of this research is to present a novel framework for automated web application security scanning and information gathering using the Axiom methodology. The objectives of this research include:

- Identifying and mitigating vulnerabilities in web applications using the Axiom methodology
- Evaluating the effectiveness of the Axiom methodology in web application security scanning
- Analyzing the limitations and challenges of the proposed framework
- Discussing the ethical considerations of distributed scanning
- Making suggestions for future research in the area of web application security.

2. EXISTING SYSTEM

Web applications' vulnerabilities are found and fixed using automated web application security scanning tools. Among the most well-liked tools in this category are:

• **Burp Suite:** Burp Suite is a comprehensive web application security testing tool. It provides a wide range of features, including a web proxy, web spider, and web application scanner. Burp Suite also allows for manual testing and active scanning, making it a valuable tool for penetration testing.

• **Acunetix:** Acunetix is a tool that scans websites for vulnerabilities and instantly finds and notifies users of any flaws. It includes an extensive database of vulnerabilities and can also scan for OWASP top 10 vulnerabilities.

• **AppScan:** With AppScan, a web application security scanner, web applications' vulnerabilities are found. It includes a web spider, web application scanner, and manual testing capabilities. AppScan also integrates with other IBM security products, making it a valuable tool for enterprise security.

• **Netsparker:** Netsparker is a web application security scanner that automatically detects and reports vulnerabilities in web applications. It includes a web spider, web application scanner, and manual testing capabilities. Netsparker also has a unique "proof-based scanning" technology that can automatically verify the identified vulnerabilities.

These tools provide a range of features and capabilities to assist in identifying and mitigating vulnerabilities in web applications. However, the proposed framework in this research uses the Axiom methodology which is faster in scanning time and can run continuously, making it a valuable tool for bug bounties and penetration testing.

2.1 Limitations of the existing system:

The existing automated web application security scanning tools are limited in their ability to provide fast and continuous scanning capabilities.

Most of the tools rely on a pre-defined database of

vulnerabilities, which may not be able to detect newly discovered or emerging threats.

The existing tools may also have a high rate of false positives, which can be time-consuming to filter through and may lead to an increased risk of overlooking a true vulnerability.

Some tools may require manual testing and intervention, which can be time-consuming and may lead to human error.

The existing tools may not be able to handle large scale web applications and networks.

The existing tools may not support all types of web applications, platforms, or technologies.

3.SYSTEM ARCHITECTURE

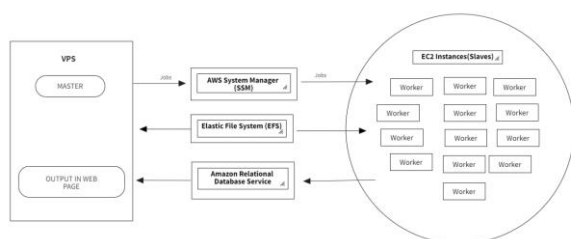


Fig.1 System Architecture

- Docker containers are being used to build workers, which are managed by AWS Fargate and AWS Elastic Container Service (ECS).
- Amazon EC2, We launch master and slave (virtual servers) as needed, set up networking and security, and control storage using Amazon EC2.
- Workers scale automatically based on the number of items in the job queue. This will ensure that a large number of workers are available when we need them, but they all die when we don't need them anymore. This could potentially be set up very quickly by utilizing an *Elastic Beanstalk worker environment*.
- Amazon elastic file system (EFS) will be used to share files to all containers (config files, tools, etc.)
- AWS Systems Manager (SSM) is software from Amazon that operates on servers and virtual machine (VM) installed locally as well as Amazon Elastic Compute Cloud (Amazon EC2) instances. Systems

Manager is used to updating, manage, and configure these resources. (EC2 full control.)

- Amazon Relational Database Service is used to store relational data. (MySQL)

3.1 How we achieve less time consumption:

They are single node systems, which often leads to high time consumption, slow down scan efficiency rates and do not scan for the most recent security risks. So we're going to switch from a single node system to a multiple node system, which will lower the time consumption and boost scan efficiency and reduce time.

3.2 Risks Associated with Single Point of Failure:

A single point of failure can result in inaccurate output. We need more precise results to reduce the single point of failure. Hence, in this situation, we used nuclei to extract accurate data from the result's fingerprint.

4. PROPOSED SYSTEM

The proposed system is a vulnerability testing and management system that aims to identify and report vulnerabilities in target systems. The system comprises of several stages, starting with planning and reconnaissance. This stage entails establishing the test's scope and objectives as well as specifying the systems that will be put to the test and the testing methods that will be used. Intelligence is also obtained during this stage, such as network and domain names and mail servers, to gain a better understanding of how the target operates and its weaknesses.

4.1 Vulnerability Scanner:

A vulnerability scanner is a piece of software that scans computers, networks, and applications for known flaws. They may execute authenticated and unauthenticated scans to find and detect vulnerabilities arising from network misconfiguration and programming flaws.

4.1.1 Authenticated scans:

Allows the vulnerability scanner to get direct access to networked assets using remote administration protocols such as secure shell (SSH) or remote desktop protocol (RDP) and authenticate using the system credentials given. This provides thorough and accurate information on operating systems, installed software, configuration issues, and missing security updates by allowing access to low-level data such as individual services and configuration details.

4.1.2 Unauthenticated scans:

False positives and incorrect information about operating systems and installed applications result from unauthenticated scanning. Cyber attackers and security analysts commonly employ this strategy

4.2 Scanning:

The next stage is scanning, which involves determining how the target application will react when given a domain name. A subdomain identifier is used to list all of the subdomains that exist under a certain domain, along with a list of web servers that are available. A vulnerability scan is then performed on each subdomain to assess and report any vulnerabilities that are found.

4.3 Analysis:

The results of the vulnerability tests are then analyzed and stored in a database. Each scan is given a unique name, and specific modules are stored. Vulnerabilities that have been exploited are also noted.

4.4 Notify:

The system also includes a notification feature, which updates the list in Telegram if a subdomain is detected or a web server is taken. If a vulnerability is discovered, the user is notified of the amount of risk right away.

4.5 Overall design of proposed system:

In terms of architecture and overall design, the proposed system utilizes a number of Amazon Web Services (AWS) tools. Docker containers are used to build workers, which are managed by AWS

5. METHODOLOGY USED

5.1 AWS Configuration:

You must first create an account at <http://aws.amazon.com> in order to utilise any of AWS' web services. Simply put, an AWS account is an Amazon.com account that gives you access to AWS products; you can create an AWS account using an existing login and password for an Amazon.com account. It's crucial to remember that creating an AWS account is totally free. Your payment information is only requested when you try to use a service. To access all of its services, AWS gives a set of login credentials. Visit account, followed by security credentials, in the account drop-down menu on the AWS website. Save your access key, create an X.509 certificate, and download it for later use.

5.2 AWS System Manager (SSM):

With the help of the fully managed Session Manager feature of Amazon Web Service Systems Manager, you can control your Amazon EC2 instances with a single click in an interactive browser-based shell. Session Manager can be used to launch a session with an instance in your account. You can use bash commands after the session has begun, just like you would with any other connection type. Further details about Session Manager may be found in the Amazon Systems Manager User Guide under AWS Systems Manager Session Manager.

Verify that all required setup procedures have been followed before using Session Manager to connect to an instance.

5.3 Amazon RDS:

Using the Amazon Web Service Management Console, you can create a Database instance with or without Easy Create turned on. When Simple creation is enabled, you simply need to specify the DB engine type, DB instance size, and DB instance

identifier. Simple Create uses the default settings for all other setup options. You can provide more configuration choices, including those for availability, security, backups, and maintenance, when you create a database with Easy Create disabled.

5.4 Master-Slave Configuration:

Our application should indeed run its processes in a distributed environment to run efficiently, it requires different/separate machines for various modules such as subdomain finder, web probing, and vulnerability scanner. So we need a Master-Slave deployment architecture, in which there is a single master node and n slave nodes performing various operations.

5.5 Auto-scaling on slave machines:

One of the most crucial services for developing highly available applications is auto-scaling. By dispersing traffic across availability zones, it reduces the single point of failure. If any of the instances fail, it can be self-healing by starting new ones. To conserve money, we can turn off the slaves or instances when usage is low. During high availability and demand, auto-scaling can automatically raise the number of slaves to maintain performance. Auto-scaling is beneficial for applications with predictable demand patterns as well as those with hourly, daily, or weekly consumption fluctuations.

5.6 Web interface:

Next, we need to connect our machine (master instance) with a graphical user interface GUI (frontend-webpage) which we have created, so that we could give our input data, and control slave instances from our webpage, and outputs are shown graphically in our webpage for the users. We have built this module using Flask(python), CSS, HTML, and JavaScript.

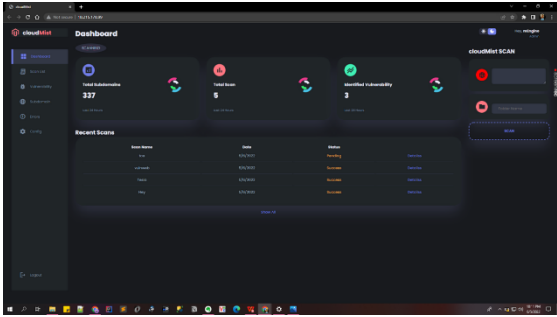


Fig. 2 Web Interface

5.7 Understanding the Expected Behaviour of the Target Application:

Normally, this would result in a Distributed Denial-of-Service [DDOS] attack and the scanning framework being blocked, but in our instance, we use numerous individual nodes to send the request to the target application, so the target application would assume every single request is valid.

6. FINANCIAL REPORT ON ESTIMATED COSTING

The project will be hosted in the cloud by an Amazon Web Service instance (EC2), so it can be accessed from anywhere with an internet connection.


The minimum to run our project is:

- 1 processor
- 2GB of RAM
- 5GB of Network Bandwidth

The image is a screenshot of the AWS Pricing Calculator. It shows a table of Amazon EC2 instance types. The table has columns for Name, vCPUs, Memory (GB), Baseline Performance, CPU Credits, Network burst bandwidth, EBS burst bandwidth, On-Demand Price/Hr, 1 yr Reserved Instance Effective Hourly, and 5 yr Reserved Instance Effective Hourly. The rows list various instance types from t3.nano to t3.xlarge. The t3.medium instance is highlighted, showing 2 vCPUs, 4GB memory, and a baseline performance of 100%.

Fig. 3 Table of Pricing.

As a comparison to other AWS plans, we have chosen t3.medium, which has 2 CPUs, 4GB of RAM, and 5GB of network burst bandwidth. More



The screenshot shows the AWS Pricing Calculator interface. At the top, there's a search bar with the text "calculator?ref=AWS_PRICING_CALCULATOR". Below the search bar, there are several tabs: "Home", "YouTube", "Maps", "Software for Mac", "Office Apps for Mac", "Best 5 Star Hotels", "Classroom DMG Chat", "Shopping Notifications", and "Personalized App...". The main content area is titled "Configure Amazon EC2, info". It shows a configuration for 1 instance of m5.xlarge in the us-east-1 region, running for 1 year. The total monthly cost is \$23.80 USD, and the total yearly cost is \$285.60 USD. The configuration also includes Amazon Elastic Block Store (EBS) pricing and Amazon EC2 Instance Savings Plans pricing.

Configure Amazon EC2, info

1 instance

US East (N. Virginia)

Instance configuration

- 1 instance of m5.xlarge in the us-east-1 region, running for 1 year
- Instance type: m5.xlarge
- Operating system: Linux (Ubuntu 20.04 LTS)
- Storage: 100 GB of EBS (gp2)
- Network: 100 Mbps of Elastic Network Interface (ENI)
- Availability: 1 Availability Zone
- Subnet: default subnet
- Security group: default security group
- Key pair: default key pair
- Instance profile: default instance profile
- Role: default role
- Tags: default tags

Cost breakdown

- Instance: \$23.80 USD/monthly (\$285.60 USD/yearly)
- EBS: \$0.00 USD/monthly (\$0.00 USD/yearly)
- ENI: \$0.00 USD/monthly (\$0.00 USD/yearly)
- Subnet: \$0.00 USD/monthly (\$0.00 USD/yearly)
- Security group: \$0.00 USD/monthly (\$0.00 USD/yearly)
- Key pair: \$0.00 USD/monthly (\$0.00 USD/yearly)
- Instance profile: \$0.00 USD/monthly (\$0.00 USD/yearly)
- Role: \$0.00 USD/monthly (\$0.00 USD/yearly)
- Tags: \$0.00 USD/monthly (\$0.00 USD/yearly)

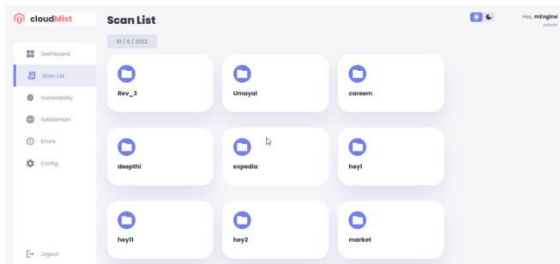
Total monthly cost: \$23.80 USD

Total yearly cost: \$285.60 USD

Save and view summary **Save and add service**

7. RESULT

Results are published under the folder name we gave during the input phase. We can view the result by going inside the Scan List menu and respective folder which are displayed in Figs. 5, 6, and 7 and 8, respectively.



The screenshot shows the 'cloudmit' interface. The sidebar on the left contains the following links: Dashboard, Scan List (highlighted with a blue bar), Vulnerability, Subdomain, Errors, and Config. The main content area is titled 'Scan Details' and shows information for a scan named 'Rev_3' performed on '10/8/2022'. Below the title, there is a 'Domain' section displaying 'localhost'. Further down, a 'Subdomain' section contains a table with the following entries:

Subdomain
project.localhost
courtesannewteam.localhost
cs32.localhost
adm3m.localhost
admission3m.localhost
scarts.localhost
...

The screenshot shows a web browser window with a title bar that says "Webserver". The address bar contains the URL "http://ccort.ice.edu". Below the address bar, there is a list of URLs, each preceded by a horizontal line. The URLs are: "http://admissions.ice.edu", "http://help.ice.edu", "http://courses.ice.edu", "http://courseservers.ice.edu", "http://ewsm.ice.edu", "http://meeting.ice.edu", "http://1303.ice.edu", "http://lists.ice.edu", "http://coursesoddsm.ice.edu", and "http://web.ice.edu". A mouse cursor is visible over the URL "http://ewsm.ice.edu".

Webserver

http://ccort.ice.edu

http://admissions.ice.edu

http://help.ice.edu

http://courses.ice.edu

http://courseservers.ice.edu

http://ewsm.ice.edu

http://meeting.ice.edu

http://1303.ice.edu

http://lists.ice.edu

http://coursesoddsm.ice.edu

http://web.ice.edu

vulnerability

[2022-06-06 05:24:18] [CVE-2018-15473] [network] [medium] [meeting.tce.edu:22] [SSH-2.0-OpenSSH_7.2p2 Ubuntu-
Ubuntu22.04]

[2022-06-06 05:22:42] [expired-ssl] [low] [low] [https://admissions.tce.edu]

[2022-06-06 05:25:46] [apache-tomcat-snoop] [http] [low] [http://library.tce.edu/examples/jsp/np/snoop.jsp]

7.1 Constraints we faced:

- We have overcome and solved these issues successfully.

The proposed framework is a valuable tool for organizations and independent security researchers to identify and mitigate vulnerabilities in web applications. The use of the Axiom methodology allows for faster scanning times and distributed scanning, making it useful for bug bounties and penetration testing. The framework's performance and limitations are discussed, and recommendations for future research are given. The research highlights the importance of proper configuration and ethical considerations when using the framework.

Currently only one scan can be done at a single time. In future one must be able to multiple scans, all at same time. Now application is fully dependent on AWS (Amazon web services), In future application should be able run with Google cloud, Microsoft, etc

REFERENCES

- [1] A. Moshika, M. Thirumaran, B. Natarajan, K. Andal, G. Sambasivam and R. Manoharan, "Vulnerability Assessment in Heterogeneous Web Environment Using Probabilistic Arithmetic Automata", IEEE Access, vol. 9, pp. 74659-74673, 2021.
- [2] F. O. Sonmez and B. G. Kilic, "Holistic Web Application Security Visualization for Multi-Project and Multi-Phase Dynamic Application Security Test Results," in IEEE Access, vol. 9, pp. 25858-25884, 2021.
- [3] J. Thome, L. K. Shar, D. Bianculli and L. Briand, "An Integrated Approach for Effective Injection Vulnerability Analysis of Web Applications Through Security Slicing and Hybrid Constraint Solving," in IEEE Transactions on Software Engineering, vol. 46, no. 2, pp. 163-195, 2020.
- [4] Muhammad Noman Khalid, Muhammad iqbal, Kamran Rasheed, Malik Muneeb Abid, 2020, "Web Vulnerability Finder (WVF): Automated Black-Box Web Vulnerability Scanner", IJTCS
- [5] B. Wang, L. Liu, F. Li, J. Zhang, T. Chen and Z. Zou, "Research on Web Application Security Vulnerability Scanning Technology," 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2019, pp. 1524-1528, doi: 10.1109/IAEAC47372.2019.8997964.
- [6] H. Chen, J. Chen, J. Chen, S. Yin, Y. Wu and J. Xu, "An Automatic Vulnerability Scanner for Web Applications," 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2020, pp. 1519-1524, doi: 10.1109/TrustCom50675.2020.00207.
- [7] A. Al Anhar and Y. Suryanto, "Evaluation of Web Application Vulnerability Scanner for Modern Web Application," 2021 International Conference on Artificial Intelligence and Computer Science Technology (ICAICST), 2021, pp. 200-204, doi: 10.1109/ICAICST53116.2021.9497831.
- [8] S. Patil, N. Marathe and P. Padiya, "Design of efficient web vulnerability scanner," 2020 International Conference on Inventive Computation Technologies (ICICT), 2020, pp. 1-6, doi: 10.1109/INVENTIVE.2020.7824873.
- [9] D. Gol and N. Shah, "Detection of web application vulnerability based on RUP model," 2019 National Conference on Recent Advances in Electronics & Computer Engineering (RAECE), 2019, pp. 96-100, doi: 10.1109/RAECE.2015.7510233.
- [10] A. Alzahrani, A. Alqazzaz, Y. Zhu, H. Fu and N. Almashfi, "Web Application Security Tools Analysis," 2017 IEEE 3rd international conference on big data security on cloud (bigdatasecurity), IEEE international conference on high performance and smart computing (hpsc), and IEEE international conference on intelligent data and security (ids), 2017, pp. 237-242, doi: 10.1109/BigDataSecurity.2017.47.
- [11] A. Vernotte, "Research Questions for Model-Based Vulnerability Testing of Web Applications," 2018 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2018, pp. 505-506, doi: 10.1109/ICST.2018.82.
- [12] S. Patil, N. Marathe and P. Padiya, "Design of efficient web vulnerability scanner," 2019 International Conference on Inventive Computation Technologies (ICICT), 2019, pp. 1-6, doi: 10.1109/INVENTIVE.2019.7824873.
- [13] Albert Mayan J , Sharmila Latha T , Kislay Sinha , " Security Analysis of Three Factor Authentication Schemes for Banking", ARPN Journal of Engineering and Applied Sciences, Vol:10, Issue 8, pp: 3504-3509, May 2015
- [14] J. Chen and C. Wu, "An automated vulnerability scanner for injection attack based on injection point," 2020 International Computer Symposium (ICS2010), 2020, pp. 113-118, doi: 10.1109/COMPSYM.2020.5685537.
- [15] X. Zhang et al., "An Automated Composite

Scanning Tool with Multiple Vulnerabilities," 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), 2019, pp. 1060-1064, doi: 10.1109/IMCEC46724.2019.8983828.

[16] Mayan J.A., Ravi. T , "Test Case Optimization Using Hybrid Search Technique", ACM International Conference Proceeding Series ,ACM

New York, 2014, doi:10.1145/2660859.2660906

[17] J. Fonseca, M. Vieira and H. Madeira, "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and XSS Attacks," 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007), 2020, pp. 365-372, doi: 10.1109/PRDC.2020.

