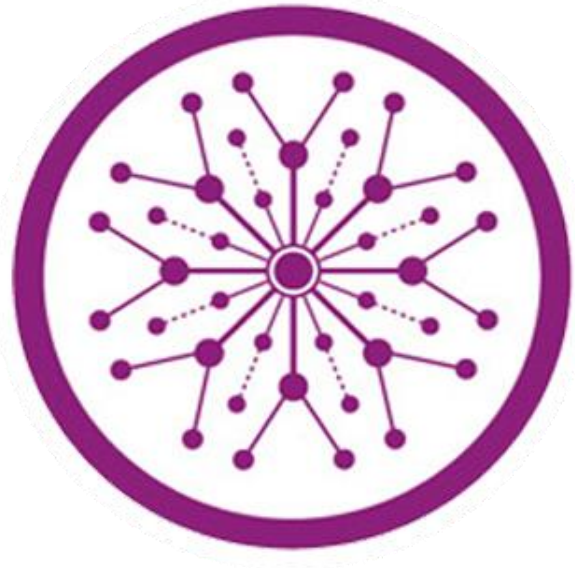
Attack Surface Management

Final Year Project

Session 2020-2023

A project submitted in partial fulfillment of the degree of

BS in Computer Science



Department of Computer Science

Faculty of Computer Science & Information Technology

The Superior University, Lahore

Fall 2023

Type (Nature of project)	[✔] **D**evelopment [ ] **R**esearch [ ] **R&D**
Area of specialization	Cyber Security, Software development
FYP ID	FYP-BCSM-S23-002

Project Group Members

Sr.#	Reg. #	Student Name	Email ID	*Signature
(i)	Bcsm-f19-383	Muhammad Husnain	bcsm-f19-383@superior.edu.pk	
(ii)	Bcsm-f19-384	Hafiz Abdul Rehman	bcsm-f19-384@superior.edu.pk	
(iii)	Bcsm-s20-010	Saad Ali	bcsm-s20-010@superior.edu.pk	

*The candidates confirm that the work submitted is their own and appropriate credit has been given where reference has been made to work of others

Plagiarism Free Certificate

This is to certify that, I Muhammad Husain S/D of Muhammad Sarwar, group leader of FYP under registration no Bcsm-f19-383 at Computer Science Department, The Superior University, Lahore. I declare that my FYP report is checked by my supervisor.

Date: Name of Group Leader: **Muhammad Husnain** Signature: _____

Name of Supervisor: Mr. Ameer Hamza

Designation: Lecturer

Signature: _____

HoD: Dr. Irfan Ud Din

Signature:_____

Project Report

Attack Surface Management

Change Record

Author(s)	Version	Date	Notes	Supervisor's Signature
Muhammad Husnain Abdul Rehman Saad Ali	1.0	12-02-2023	Initial draft of the FYP report.	
Muhammad Husnain Abdul Rehman Saad Ali	1.1	03-03-2023	Incorporated feedback from the supervisor.	
Muhammad Husnain Abdul Rehman Saad Ali	1.2	29-03-2023	Addressed comments from faculty members.	
Muhammad Husnain Abdul Rehman Saad Ali	1.3	28-04-2023	Added a detailed project plan with milestones.	
Muhammad Husnain Abdul Rehman Saad Ali	1.4	23-05-2023	Integrated additional feedback from the supervisor.	
Muhammad Husnain Abdul Rehman Saad Ali	2.0	06-06-2023	Finalized the entire report for submission.	
Muhammad Husnain Abdul Rehman Saad Ali	2.1	27-08-2023	Updated literature review based on recent findings.	
Muhammad Husnain Abdul Rehman Saad Ali	2.2	29-09-2023	Revised methodology to improve clarity.	
Muhammad Husnain Abdul Rehman Saad Ali	2.3	20-10-2023	Incorporated feedback on the results section.	
Muhammad Husnain Abdul Rehman Saad Ali	2.4	05-11-2023	Added additional charts to the results for better visualization.	
Muhammad Husnain Abdul Rehman Saad Ali	3.0	27-11-2023	Final version with all revisions completed.	

APPROVAL

`

PROJECT SUPERVISOR

Comments: _____

_____

Name:_____

Date:_____ Signature:_____

`

PROJECT MANAGER

Comments: _____

_____

Date:_____ Signature:_____

HEAD OF THE DEPARTMENT

Comments: _____

_____

Date:_____ Signature:_____

Dedication

To Dr. Abdul Qadeer Khan and Dr. Aafia Siddiqui, whose contributions and sacrifices have left an indelible mark on our nation's history. To my parents and teachers, who have been the guiding stars in my educational journey, providing unwavering support and wisdom. And to Ameer Hamza, my FYP supervisor, for his invaluable guidance and mentorship throughout this project

Acknowledgements

I extend my heartfelt gratitude to everyone who played a pivotal role in the realization of this project. Special thanks to my project supervisor, Ameer Hamza, for his expert guidance, patience, and encouragement. I am also grateful to my family for their endless love and support, and to my teachers and peers at the university for their valuable insights and contributions. This project would not have been possible without the collaborative efforts and shared wisdom of each individual involved

Executive Summary

The executive summary of the Surfaceer project provides a concise overview of its key aspects. Surfaceer is a state-of-the-art cybersecurity platform specializing in attack surface management. It integrates various technologies and methodologies, such as deep scanning and real-time threat analysis, to offer comprehensive security solutions. The project's development journey, from initial conception to final implementation, involves rigorous design, testing, and optimization stages. The summary highlights the project's main achievements, including innovative feature development, integration of cutting-edge technologies, and successful deployment. It also touches on the lessons learned and potential future enhancements, emphasizing the project's commitment to evolving cybersecurity needs.

Table of Contents

List of Figures

Chapter 1

Introduction

Chapter 1: Introduction

In this chapter, we introduce the escalating need for robust attack surface management within the realm of cybersecurity, a concern intensified by the rapid digital transformation and increasing cyber threats. Recent studies indicate that cybercrimes are expected to cost the world $10.5 trillion annually by 2025, reflecting a dire need for advanced security measures. We examine the capabilities and shortcomings of current cybersecurity platforms like Shodan, Nmap, VulnDB, and OpenCVE. Acknowledging these challenges, the chapter further introduces our project's cornerstone, Surfaceer, an innovative tool envisaged to enhance cybersecurity practices. This introduction sets the stage for a comprehensive exploration of the current state of cybersecurity, the pressing need for improved attack surface management, and the journey towards developing Surfaceer.

1.1. Background

The cybersecurity landscape has drastically changed in recent years, notably with a 600% increase in cybercrime since the onset of the COVID-19 pandemic. By 2025, cybercrime is projected to cost a staggering $10.5 trillion annually, reflecting about 1% of the global GDP. Enterprises are facing an average of 130 security breaches annually, and over 71.1 million individuals fall victim to cybercrime each year, with average losses of $4,476 per person. The FBI's Internet Crime Complaint Center (IC3) 2022 Report highlighted the immense financial impact of these crimes, totaling $10.3 billion in losses with over 2,175 daily complaints. Predominant cybercrimes include phishing, government impersonation, data breaches, investment scams, business email compromise (BEC), and tech support scams, with ransomware being the most prevalent and damaging. In fact, ransomware damages are projected to reach $265 billion by 2031.

Since 2020, phishing attacks have seen an 81% increase. The first half of 2022 marked a surge in zero-day vulnerabilities and email cyberattacks, leading to substantial financial losses. Supply chain attacks, constituting 40% of all cyberattacks, are predicted to cause about €242 billion in losses over 2022 and 2023.

A growing area of concern is IoT security, as incidents like the MiCODUS MV720 GPS tracker breach demonstrate the vulnerabilities in these devices. Despite the increasing threats, there's a concerning trend of budget reductions in cybersecurity, as noted by security executives in 2021. This challenging backdrop underscores the necessity for effective and advanced cybersecurity solutions in the face of evolving threats.

1.2. Motivations and Challenges

The primary goal of the project is to bolster cybersecurity measures to counter advanced threats. The core challenges involve precise identification and assessment of vulnerabilities, proactively adapting to the constantly changing threat landscape, and effectively integrating a range of tools for all-encompassing security coverage. This approach aims to address the intricate and dynamic nature of modern cyber threats.

1.3. Goals and Objectives

The project's goal centers around developing an enhanced attack surface management solution, aiming to surpass the capabilities of current tools. Key objectives include:

- **Comprehensive Asset Discovery:** Identifying and cataloging all digital assets across various networks for thorough coverage.
- **Real-Time Threat Analysis:** Implementing advanced algorithms to detect and analyze threats as they occur, ensuring timely response.
- **Streamlined Vulnerability Management:** Simplifying the process of vulnerability assessment and remediation, integrating automated workflows for efficiency.
- **Scalable Architecture:** Ensuring the solution is adaptable to growing organizational needs and emerging cybersecurity challenges.
- **User-Friendly Interface:** Designing an intuitive interface to facilitate ease of use for various user roles.

1.4. Literature Review/Existing Solutions

Operational Technology (OT) Security Challenges: The rise of Industry 4.0 has brought significant risks to OT environments. With the expansion of the attack surface due to cloud adoption, Industrial IoT, and increased mobility, securing OT networks has become more complex. Traditional security measures like perimeter firewalls are inadequate against sophisticated threats, including high-stakes ransomware and internal breaches. The literature emphasizes the need for comprehensive Attack Surface Management (ASM) strategies tailored for OT environments.[1]

Enhancing Nonautomated Security Testing: Existing research focuses on improving automated security processes like DAST and SAST. However, nonautomated methods like Penetration Testing and Red Teaming face scalability challenges. A novel approach using agglomerative hierarchical clustering has shown promise in reducing attack surface complexity, as demonstrated by a significant reduction in internet-facing hosts. This technique also aids in mapping network services and understanding vulnerabilities like Log4Shell.[2]

Cyber-Deception in Content Distribution Networks: Security strategies in Content Distribution Networks (CDNs) have evolved to include cyber-deception. By minimizing client delay variance, these strategies create a homogeneous attack surface, hindering adversaries from exploiting latency differentials. This innovative approach unifies proactive obfuscation and reactive randomization, enhancing the network's security against attacks like link flooding.[3]

Security Posture of EV Charging Ecosystems: With the growing adoption of electric vehicles (EVs), the security of EV charging mobile applications has become crucial. Recent studies leveraging static and dynamic analysis reveal vulnerabilities that could be exploited remotely. These vulnerabilities highlight the need for enhanced security measures in the rapidly evolving EV charging infrastructure.[4]

These reviews underscore the diverse and evolving nature of cybersecurity challenges and the importance of developing adaptive and comprehensive strategies to address them.

1.5. Gap Analysis

Despite the strengths of existing tools, gaps in real-time threat detection, holistic asset management, and user-friendly interfaces are identified. This analysis highlights the need for a more integrated and automated solution.

The gap analysis in this study reveals significant differences in the detection capabilities of various Attack Surface Management (ASM) tools used for identifying vulnerabilities in Operational Technology (OT) networks. The study highlights the inconsistency in exposure identification among different ASM tools, despite targeting identical Industrial Control Systems (ICS) equipment. However, it also notes that each tool offers unique contributions to the overall understanding of the attack surface. Therefore, the study suggests using a combination of these tools to achieve a more comprehensive and accurate assessment of the attack surface in OT networks.[4]

The research aims to delineate the attack surface in IoT networks, highlighting the independent operational nature of IoT devices. A key gap identified is the escalating security issues as IoT devices proliferate and evolve. The study's threat model systematically assesses security solutions from the design phase, focusing on vulnerabilities within different IoT zones. It also examines the interplay between IoT devices across various domains and their associated vulnerabilities and threats. The gap analysis underscores the need for a holistic security approach, considering the diverse and evolving nature of IoT devices and their attack vectors.[5]

1.6. Proposed Solution

The proposed solution is a new platform, Surfaceer, offering comprehensive, automated attack surface management. It integrates asset discovery, vulnerability assessment, and threat intelligence in a user-friendly interface.

1.7. Project Plan

The project plan outlines the steps for developing Surfaceer, including research, design, development, testing, and deployment phases, ensuring a systematic approach to the project. Overall like this:

Initiation Phase:

- Define project scope and goals.
- Assemble Agile project team.

Research and Conceptualization:

- Conduct market analysis and feasibility studies.
- Gather requirements through stakeholder interviews and surveys.

Design Phase:

- Create initial design mock-ups and prototypes.
- Conduct iterative design reviews and refine based on feedback.

Development Phase:

- Implement Agile sprints for developing software functionalities.
- Regular stand-up meetings for progress updates and addressing impediments.

Testing Phase:

- Continuous integration and testing within sprints.
- User acceptance testing (UAT) and bug fixing.

Deployment Phase:

- Prepare and execute deployment plan.
- Monitor deployment and provide immediate support for any issues.

Post-Deployment:

- Gather user feedback for future improvements.

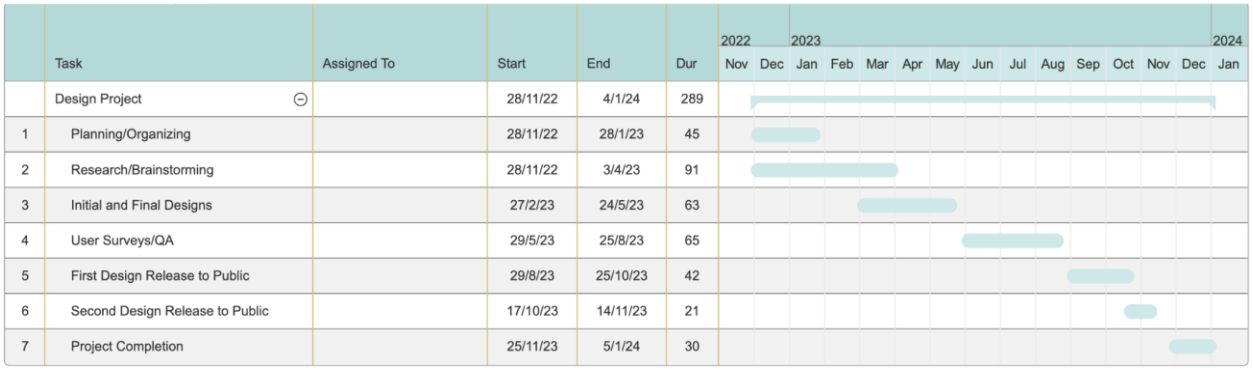
- Continuous monitoring and maintenance.

Throughout the project, Agile principles guide the process, emphasizing flexibility, customer collaboration, and responsiveness to change.

1.7.1. Work Breakdown Structure

The work breakdown structure details the project's components, assigning responsibilities, timelines, and resources for each task, ensuring efficient progress towards the project goals.

1.7.2. Gantt Chart



	Task	Assigned To	Start	End	Dur
	Design Project		28/11/22	4/1/24	289
1	Planning/Organizing		28/11/22	28/1/23	45
2	Research/Brainstorming		28/11/22	3/4/23	91
3	Initial and Final Designs		27/2/23	24/5/23	63
4	User Surveys/QA		29/5/23	25/8/23	65
5	First Design Release to Public		29/8/23	25/10/23	42
6	Second Design Release to Public		17/10/23	14/11/23	21
7	Project Completion		25/11/23	5/1/24	30

Chapter 2

Software Requirement Specifications

Chapter 2: Software Requirement Specifications

2.1 Introduction

2.1.1 Purpose

The purpose of this SRS is to present a clear and detailed set of requirements for Surfaceer. This document covers the full scope of the software's functionalities, including automated asset discovery, threat analysis, and reporting capabilities, as outlined in the product roadmap detailed in the Gantt chart.

2.1.2 Document Conventions

This SRS follows the standard typographical conventions: 'New Courier' font for system commands, bold for user interface elements, and italics for emphasis. Requirements are prioritized at the feature level, with the understanding that detailed requirements inherit the priority of the overarching feature they support.

2.1.3 Intended Audience and Reading Suggestions

This document is intended for stakeholders involved in the development and implementation of Surfaceer, including developers, project managers, marketing staff, users, testers, and technical writers. It is organized to first present an overview, followed by detailed requirements. Readers should start with the overview to understand the software's context and then proceed to sections relevant to their role.

2.1.4 Product Scope

Surfaceer is a comprehensive attack surface management software designed to automate the identification, analysis, and mitigation of cyber threats. Its goal is to enhance an organization's cybersecurity posture through advanced detection algorithms and real-time threat monitoring. The software aligns with corporate goals of reducing cyber risk and adhering to industry standards for digital security.

2.1.5 References

"Cybersecurity Framework," National Institute of Standards and Technology (NIST), Version 1.1, April 2018.

"ISO/IEC 27001 Information Security Management," International Organization for Standardization, 2013.

2.2 Overall Description

2.2.1 Product Perspective

Surfaceer is a new, self-contained product, but it integrates with existing cybersecurity solutions to provide a more holistic defense strategy. It acts as a complementary layer to enhance current systems with advanced ASM capabilities.

2.2.2 Product Functions

- Automated asset discovery and inventory.
- Real-time threat analysis and alerting.
- Vulnerability assessment and prioritization.
- Reporting and dashboards for security insights.
- Compliance tracking with industry standards.

2.2.3 User Classes and Characteristics

- Cybersecurity analysts (daily users, advanced functions)
- IT Administrators (occasional use, system settings)
- Executive Management (reporting functions, decision-making data)

2.2.4 Operating Environment

Surfaceer is designed to operate efficiently within cloud-based environments, specifically leveraging Amazon Web Services (AWS) infrastructure. It will be primarily hosted on EC2 instances, ensuring scalability and reliability across both Windows and Linux OS platforms. The

software architecture is built using Python with FastAPI for backend services, ensuring high performance, and Next.js with GraphQL for a dynamic and robust frontend experience. Development and version control will be managed through GitHub, with Visual Studio Code (VS Code) as the preferred IDE for development.

2.2.5 Design and Implementation Constraints

The development process for Surfaceer will comply with regulatory standards such as GDPR and HIPAA, ensuring the software meets global compliance mandates. The choice of technologies like Python, FastAPI, Next.js, and GraphQL is strategic to maintain a balance between performance and developer productivity. Using AWS services, including EC2, the design will be constrained to align with the capabilities and limitations of these platforms.

2.2.6 User Documentation

Surfaceer's user documentation will be comprehensive, including detailed user manuals, responsive online help systems, and engaging tutorial videos to ensure a smooth user experience. These materials will be accessible in multiple formats, including downloadable PDFs and interactive HTML content, catering to various user preferences.

2.2.7 Assumptions and Dependencies

The development of Surfaceer assumes the availability and stability of integration APIs from third-party cybersecurity tools. It also presumes that the AWS services will continue to provide the necessary infrastructure without significant changes in functionality or compliance policies. Dependencies include the continued support for the specified programming languages and the use of AWS cloud services without major service disruptions.

2.3 External Interface Requirements

2.3.1 User Interfaces

Surfaceer will offer user interfaces for both web and mobile platforms. The web interface will be built with Next.js to provide a responsive, accessible experience, while the mobile application will be developed using Flutter for cross-platform compatibility. Both interfaces will conform to the company's established UX standards, emphasizing intuitive design and user engagement.

2.3.2 Hardware Interfaces

The software will be designed to operate on AWS infrastructure, adhering to industry-standard hardware interfaces and communication protocols to ensure seamless integration and high availability.

2.3.3 Software Interfaces

Surfaceer will interface with a variety of cybersecurity tools and databases. It will incorporate secure coding practices to maintain data integrity and system security across all software interactions.

2.3.4 Communications Interfaces

Communication functions will rely on standard internet protocols, ensuring compatibility and security. Surfaceer will utilize secure channels, such as those provided by services like Discord for team communication, ensuring encrypted data transmission and secure authentication processes.

2.4 System Features

Priority: High

Stimulus/Response: User initiates threat scan, system provides report.

Functional Requirements:

- REQ-SF1-1: System shall allow scheduled threat scans.
- REQ-SF1-2: System shall provide real-time alerts.
- REQ-SF1-3: System shall generate comprehensive reports.

This template serves as a starting point. Each section should be expanded with the specific details pertinent to the Surfaceer system as per the requirements of your project.

2.5 Other Nonfunctional Requirements

2.5.1 Performance Requirements

Surfaceer will support concurrent user sessions and ensure real-time threat response within seconds. System uptime should be at least 99.9%, and backup operations should not impact system performance.

2.5.2 Safety Requirements

Surfaceer will comply with international safety standards for software products, ensuring no feature leads to loss or harm. Safety measures will include secure data handling and emergency system shutdown procedures.

2.5.3 Security Requirements

The product will meet GDPR and other relevant regulations for data security. It will include robust encryption, multi-factor authentication, and regular security audits to meet industry certifications.

2.5.4 Software Quality Attributes

Surfaceer will prioritize flexibility, interoperability, maintainability, and usability, aiming for high reliability and robustness. Ease of use will be considered over ease of learning

2.5.5 Business Rules

Only authorized personnel will have access to sensitive data within Surfaceer, with specific roles delineated for different levels of system interaction.

2.6 Other Requirements

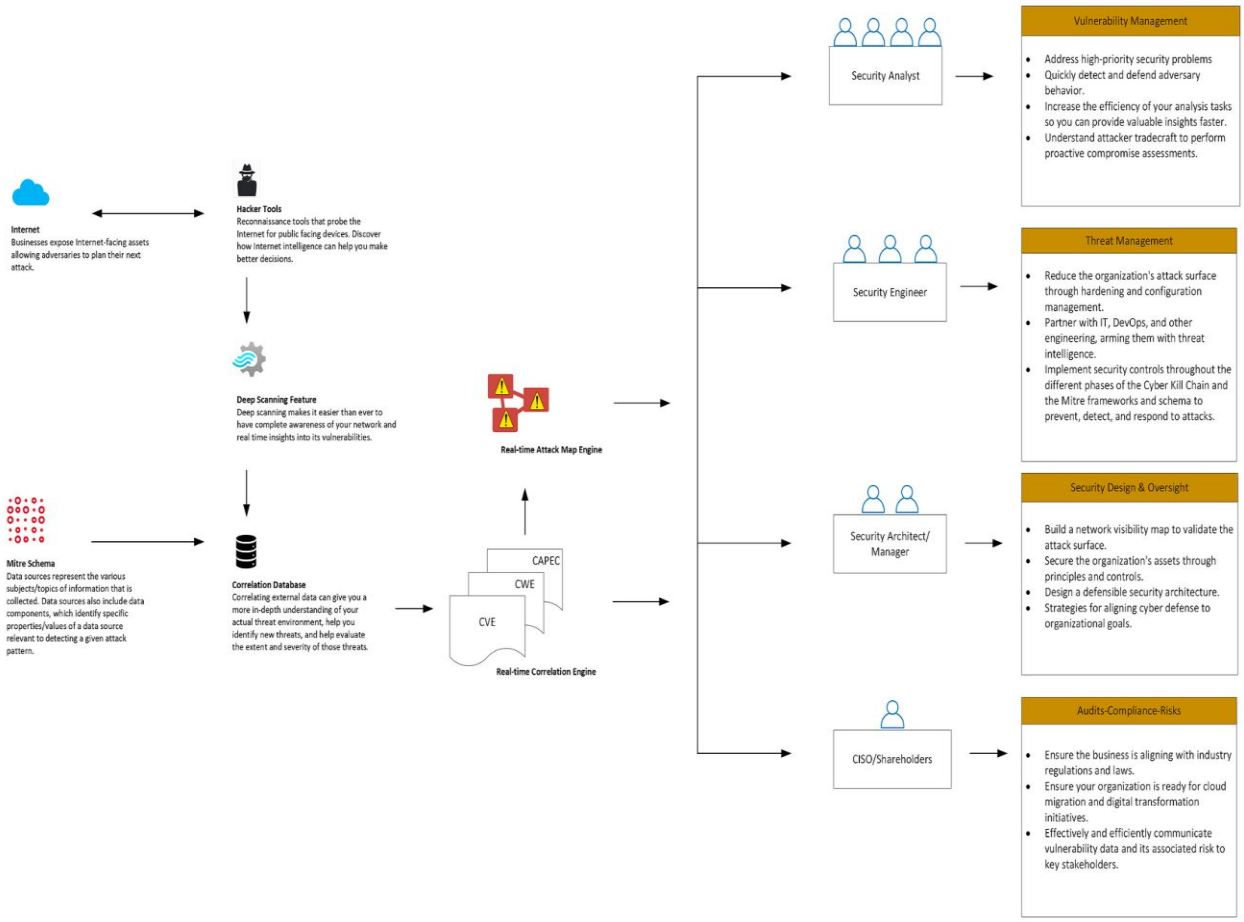
Surfaceer will be designed for internationalization, including support for multiple languages and compliance with global data protection laws. It will aim for high reusability and maintainability to facilitate future updates and integrations.

Chapter 3

Use Case Analysis

Chapter 3: System Analysis

3.1. Use Case Model



3.2. Use Case Descriptions

The diagram appears to depict a cybersecurity workflow that integrates various components and processes to enhance security management and oversight. It begins with the internet, where businesses' assets are exposed, creating potential attack surfaces. Hacker tools perform reconnaissance to identify public-facing devices, which feeds into a deep scanning feature that offers insights into network vulnerabilities. The Mitre schema, correlating various cybersecurity topics, feeds into a correlation database that helps understand the threat environment.

The workflow then branches out to different user roles, each with specific responsibilities:

- Security Analysts focus on vulnerability management, analyzing threats and attacker tradecraft to provide proactive assessments.
- Security Engineers work on threat management, reducing the attack surface through hardening and configuration management, and partnering with IT and DevOps.
- Security Architects/Managers are responsible for security design and oversight, including building visibility maps and securing assets.
- CISOs/Shareholders are involved in audits, compliance, and risk management, ensuring that the business aligns with regulations and communicates vulnerabilities effectively.

Central to this process is the Real-time Attack Map Engine and Real-time Correlation Engine, which leverage databases of common vulnerabilities and exposures (CVE), common weakness enumerations (CWE), and common attack pattern enumeration and classification (CAPEC) to inform and respond to potential cyber threats.

Vulnerability Identification and Deep Scanning

The reconnaissance data gathered is processed through a deep scanning feature, providing real-time insights into network vulnerabilities that could be exploited.

Mitre Schema Application

Incorporating the Mitre schema allows the system to correlate cybersecurity information, enhancing the understanding of the current threat landscape.

Role-Based Security Management

- Security Analysts: They play a crucial role in vulnerability management, utilizing insights from the system to assess potential threats.
- Security Engineers: Their focus is on threat management, working to reduce the attack surface by applying hardening techniques and collaborating with IT and DevOps teams.

- Security Architects/Managers: They oversee the security design, including the development of network visibility maps and the implementation of security measures.

- CISOs/Shareholders: They ensure that the organization's security measures are compliant with industry regulations and that vulnerabilities are communicated effectively.

Real-time Threat Intelligence Engines

At the heart of the workflow are the Real-time Attack Map Engine and the Real-time Correlation Engine. These engines utilize CVE, CWE, and CAPEC databases to provide a dynamic response to cyber threats.

Chapter 4

System Design

Chapter 4: System Design

This chapter outlines the structural design of the cybersecurity workflow, detailing the integration of various system components that contribute to robust security management. It describes the roles and processes from threat detection to vulnerability management and compliance, correlating with the depicted workflow involving internet exposure, hacker tools, deep scanning features, and the Mitre schema's role in a comprehensive cybersecurity strategy. The chapter further explains how this design supports different user roles, including Security Analysts, Engineers, Architects, and CISOs, in their respective functions and responsibilities for maintaining the system's security integrity.

4.1. Architecture Diagram

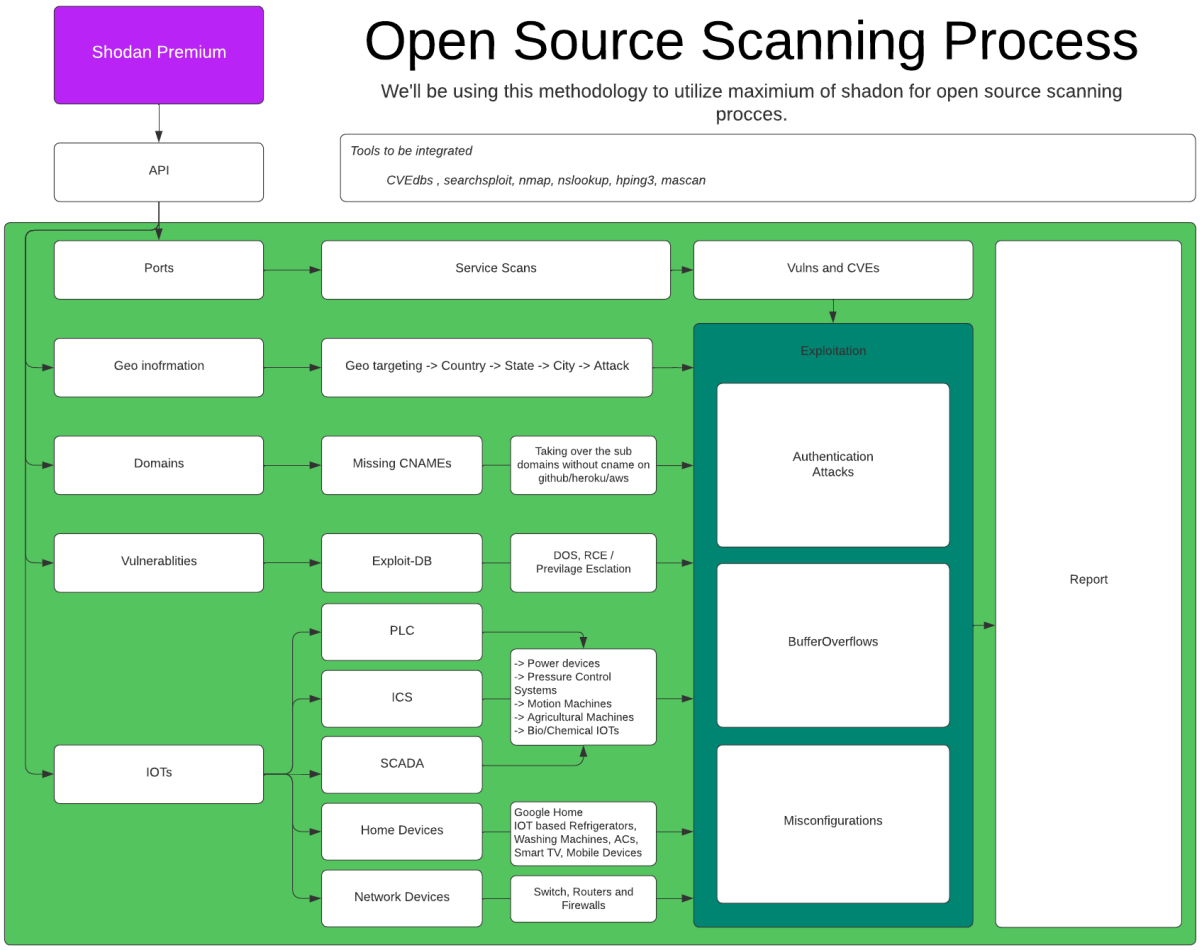


Figure 1

The diagram represents an Open Source Scanning Process that leverages Shodan Premium's API. It outlines a methodology for maximizing the use of Shodan for open-source scanning, integrating various tools such as CVE details, searchsploit, nmap, and others. The flow starts with the collection of data from Shodan, including ports, geo-information, domains, The process integrates tools like CVE details for vulnerability tracking, searchsploit for finding exploits, and Nmap for network mapping. This methodical approach allows for a thorough assessment of potential security risks, leading to a more fortified network defense strategy.

4.2. Operation contracts

Operation contracts in software development are agreements or specifications that detail the changes made to the system by an operation. They typically include:

- Operation Name: A clear identifier for the operation.
- Cross-References: Links to related use cases or requirements.
- Preconditions: The state of the system before the operation is executed.
- Postconditions: The state of the system after the operation is completed, outlining the changes.

These contracts are vital for understanding the impact of specific operations within a system, aiding in design, development, and testing phases.

4.3. Deployment Diagram

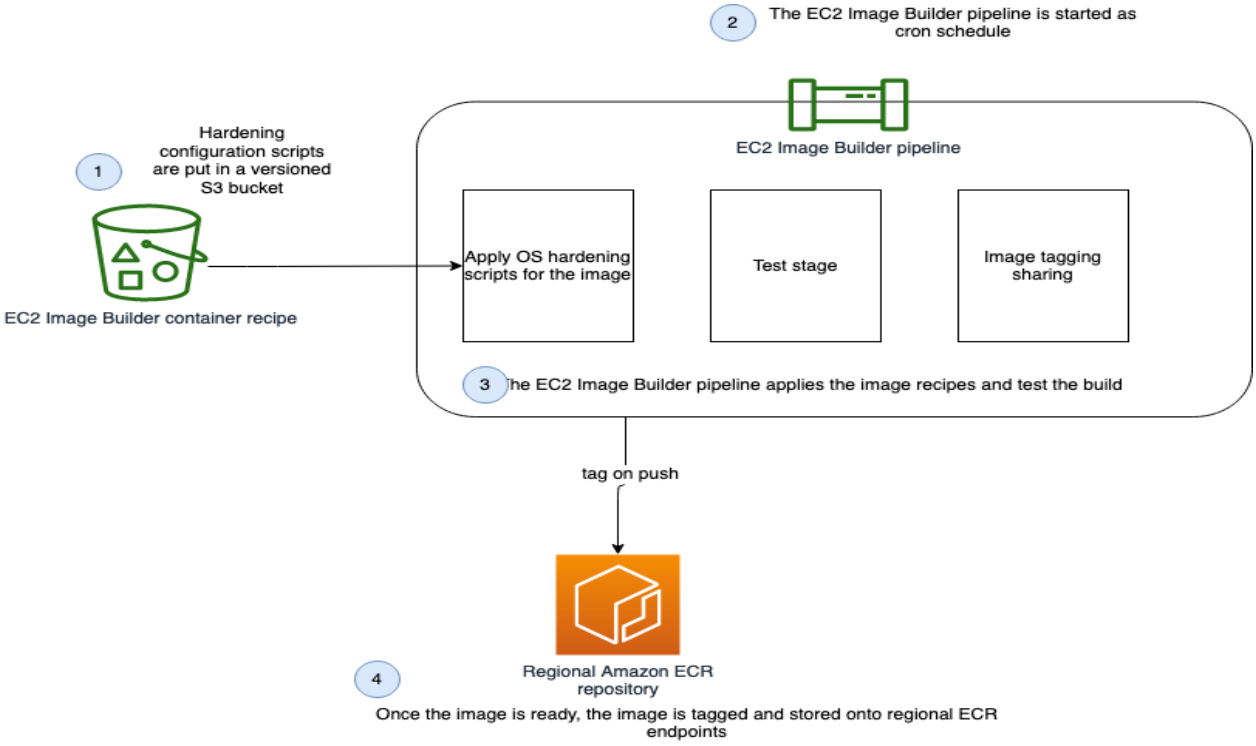


Figure 2

The diagram illustrates an AWS EC2 image building and deployment process:

Hardening Scripts: Begins with hardening configuration scripts placed in a versioned Amazon S3 bucket.

EC2 Image Builder Pipeline: Triggered on a schedule (cron), this pipeline performs several actions:

- Applies the OS hardening scripts to the image.
- Proceeds to a test stage to verify the image build.
- Concludes with image tagging for identification and sharing.

Pipeline Execution: The EC2 Image Builder pipeline applies the container recipe and conducts tests to ensure the build is successful.

ECR Repository Storage: Once the image passes the test stage, it's tagged and pushed to a regional Amazon Elastic Container Registry (ECR) repository, making the hardened image available for deployment.

4.4. Data Flow diagram

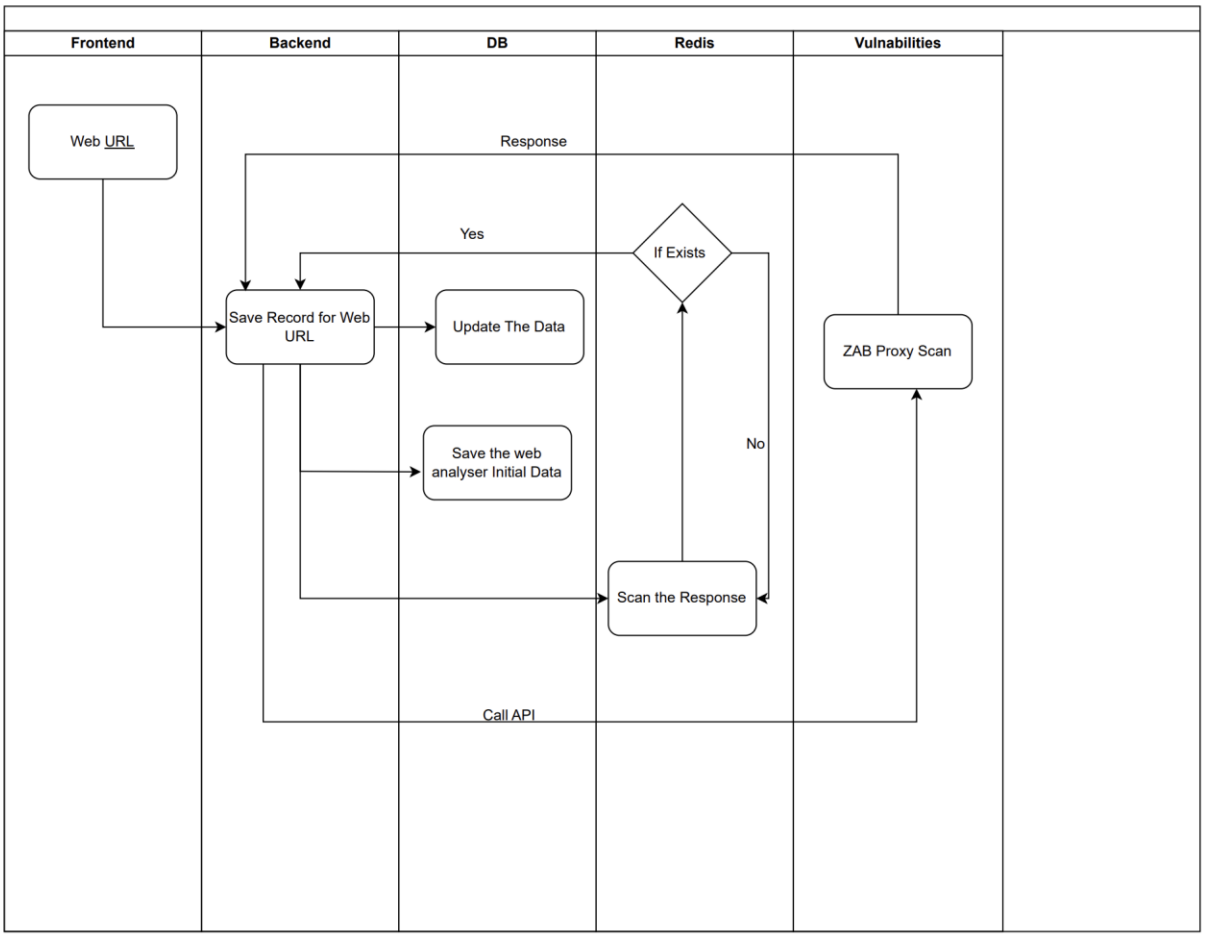


Figure 3

Description Of DFD:

Input from Frontend: A web URL is input from the frontend.

Backend Processing:

- The backend receives the web URL and proceeds to save a record for it.
- If the web URL record already exists in the database (DB), the existing data is updated.
- If the record is new, the backend saves the initial data from the web analyzer.

Redis Cache Check:

- The system checks in Redis if the data for the given URL already exists.
- If the data exists, the system proceeds to the DB update step.
- If the data does not exist, the system initiates a response scan.

Vulnerability Scanning:

After scanning the response, the system performs a ZAB Proxy Scan to identify any vulnerabilities associated with the web URL.

Data Storage:

The results of the scan are presumably stored or used to update the DB and Redis cache.

Loopback for Continuous Monitoring: The system seems to be designed for continuous monitoring, as indicated by the feedback loop to the Redis decision point, suggesting repetitive scanning for real-time vulnerability management. This DFD outlines the systematic flow of data from the initial URL input to the backend processing, through decision points using Redis, and concluding with a vulnerability scan, effectively summarizing the web vulnerability scanning process.

Chapter 5

Implementation

Chapter 5: Implementation

This chapter provides a comprehensive overview of the implementation phase of the project, detailing the deployment of code, front-end and back-end development processes, and the utilization of cloud services. It describes the practical application of the system design within the chosen technology stack, including FastAPI, Next.js, MongoDB, and the integration with AWS and Vercel for deployment. The use of GitHub as a version control system is explained, alongside the implementation of flow controls, the use of various libraries, web services, and the establishment of a robust deployment environment. Tools and techniques for development, best practices for coding standards, and version control methodologies are also outlined to ensure a structured and efficient implementation process.

5.1. Important Flow Control/Pseudo codes

```
@router.get("",response_model=ScanDataForTableResponce,tags=["Scan Data"],include_in_schema=False)
@router.get("/",response_model=ScanDataForTableResponce,tags=["Scan Data"])
async def get_scans(current_user: dict = Depends(get_current_user), page: int = 1):
  try:
    query = Scan.find(Scan.username == current_user.username)
    scan_result = (
      await query.limit(PAGE_SIZE)
      .skip((page-1)*PAGE_SIZE)
      .sort(-Scan.created_at)
      .to_list()
    )
    return {
      "items": scan_result,
      "page": page,
      "total": len(scan_result),
    }
  except:
    raise HTTPException(
        status_code=404, detail="No data found against the the given scan id"
      )
```

This code snippet defines a FastAPI route for fetching scan data:

- Endpoint: Two GET methods are defined for the same function, get_scans. The first one is hidden from the schema due to include_in_schema=False.
- Functionality: The function retrieves a paginated list of scan results for the current user. It performs a query on the Scan model, filtering scans by the user's username.
- Pagination: It implements pagination using page and a predefined PAGE_SIZE, skipping records and limiting the result set for the requested page.
- Sorting and Return: The scan results are sorted by creation date in descending order. The function returns a dictionary with the scan items, current page number, and total items count.

Error Handling: If no data is found, or an error occurs during the query execution, it raises an HTTP 404 exception with a detailed message.

```
@router.post("",response_model=ScanResponce ,tags=["Scan Post"],include_in_schema=False)
@router.post("/",response_model=ScanResponce,tags=["Scan Post"])
async def post_scan(
  body: ScanRequest,
  current_user: dict = Depends(get_current_user),
):

  global redis_caches
  Processed_obj = None
  redis_caches = await init_redis_pool()
  check_cheches = await redis_caches.get(body.ip)
  created_at = datetime.utcnow()
  Processed = ProcessedScan(
        user_id= current_user.id,
        total_scans= 1,
        scans_performed= 1 if check_cheches else 0,
        created_at= created_at,
  )
```

```
Processed_obj = await Processed.save()
if check_cheches:
    check_cheches = json.loads(check_cheches)
    logger.info('got data for this ip from from redis')
    return {
            '_id': ObjectId(await create_empty_scan(body, current_user, check_cheches)),
            'ip': check_cheches['ip'],
            'shodan': check_cheches['shodan'],
            'nmap':check_cheches['nmap'],
            'username': check_cheches['username'],
            'vuldb': check_cheches['vuldb'],
            'attack': check_cheches['attack'],
            'status': check_cheches['status'],
            'created_at':check_cheches['created_at'],
            'updated_at': check_cheches['updated_at'],
            'score': check_cheches['score'],
            'attack_level':check_cheches['attack_level'],
        }
try:
    inserted_id = await create_empty_scan(body, current_user)
    inserted_scan = await create_scan(Processed_obj.id, inserted_id, body, current_user)
    if inserted_scan is not None:
        return {
            '_id': inserted_scan.id,
            'ip': inserted_scan.ip,
            'shodan': inserted_scan.shodan,
            'nmap':inserted_scan.nmap,
            'username': inserted_scan.username,
            'vuldb': inserted_scan.vuldb,
            'attack': inserted_scan.attack,
            'status': inserted_scan.status,
            'created_at':inserted_scan.created_at,
            'updated_at': inserted_scan.updated_at,
            'score': inserted_scan.score,
            'attack_level':inserted_scan.attack_level,
        }
    raise HTTPException(status_code=404, detail="Scan Not Found")
except Exception as error:
    logger.error(f"Error in scan is : {error}")
```

```
    #                    result                    =                    await                    (ProcessedScan.id                    ==
Processed_obj.id).update(Set({ProcessedScan.scans_performed:int(ProcessedScan.scans_performed)-1}))
    # if result.modified_count > 0:
    #    raise HTTPException(
    #          status_code=404,
    #          detail=error,
    #       )
    # else:
    raise HTTPException(
       status_code=404,
       detail=f"job not updated in when we have error when we create scan {error}",
    )
```

This FastAPI route handles the creation of a new scan record:

- Endpoint Definition:
    - Defines two POST methods for the post_scan function. One is hidden from the schema.
- Functionality:
    - Receives a scan request (ScanRequest) and the current user details.
    - Initializes or gets the Redis cache.
- Redis Cache Check:
    - Checks if scan data for the given IP is already in Redis. If present, it uses this cached data.
- Processing and Saving Scan Data:
    - If no cache data is found, it creates a new ProcessedScan object with the user's ID and other details, then saves it.
    - If cache data is found, it returns an existing scan with details fetched from Redis.

- Response Generation:
    - If the scan data is new, it performs additional scan creation logic and returns the scan details.

    - If an error occurs, logs the error and raises an HTTPException.
- Error Handling:
    - Handles exceptions, logging errors, and returning appropriate HTTP responses.
    - This route is crucial for adding new scans and efficiently utilizing cached data to reduce redundant scanning operations.

5.2. Components, Libraries, Web Services and stubs

The Surfaceer project incorporates a variety of components, libraries, and web services. The backend relies on FastAPI, a modern, fast web framework for building APIs with Python, chosen for its performance and ease of use. The frontend is developed with Next.js, a React framework that enables server-side rendering and generates a polished user interface. For database operations, MongoDB is used for its flexibility and powerful querying capabilities. The 'requests' library is included to handle HTTP requests efficiently. Stubs are utilized in the development phase to simulate the behavior of actual software components, allowing for parallel development and testing.

5.3. Deployment Environment

The deployment environment for Surfaceer is configured on AWS, leveraging services such as EC2 instances for hosting and Vercel for continuous integration and deployment. This cloud-based environment ensures scalable and on-demand resource management, crucial for the expected performance and reliability of the system.

5.4. Tools and Techniques & Technologies

Development utilizes VS Code for its comprehensive coding environment and GitHub for source code management, ensuring collaborative and trackable progress. These tools support a range of plugins and integrations that streamline the development workflow.

5.5. Best Practices / Coding Standards

We adhere to industry-standard coding practices including clear naming conventions, code reviews, and comprehensive documentation. Regular refactoring and adherence to the DRY (Don't Repeat Yourself) principle ensure maintainability and scalability.

5.6. Version Control

GitHub serves as the version control system, providing a robust framework for branching, merging, and version tracking. It facilitates collaborative development and helps in maintaining a history of changes, feature branches, and release management. Following are the features for reason to use the Version Control:

- GitHub is used as the version control system.
- It enables branching, merging, and version tracking.
- GitHub facilitates collaborative development.
- It maintains a history of code changes.
- Issues and milestones are created to track tasks.
- Coding progress is tracked using GitHub's features.

Chapter 6

Testing and Evaluation

Chapter 6: Testing and Evaluation

This chapter provides a detailed account of the testing and evaluation procedures employed to ensure the software meets quality and performance standards. It covers various testing methodologies, from use case scenarios to stress testing, aimed at identifying defects and evaluating the software's capabilities under different conditions.

6.1. Use Case Testing

In use case testing, we create test scenarios based on functional specifications. Each scenario is an end-to-end flow of a particular use case, ensuring the software behaves as expected in real-world situations. This involves a mix of manual testing to mimic user interactions and automated testing to execute repetitive and regression test cases efficiently. Unit testing is integral to this process, where individual units of code are tested for correctness. Automated testing frameworks can streamline this process, running a suite of tests upon each code commit to validate recent changes.

6.2. Equivalence partitioning

Equivalence Partitioning is a testing technique that divides input data into partitions of equivalent data from which test cases can be derived. By doing so, it ensures that taking one test case from each partition is sufficient to test that group. This technique is often combined with boundary value analysis to include test cases that explore the edges of each partition. The partitions are typically defined by analyzing the requirements and identifying cases where the software's behavior should be consistent. This method is effective for both manual and automated testing, helping to identify classes of errors in a systematic manner.

6.3. Boundary value analysis

Testing the system's response at the boundary values of input data ranges, where errors are most likely to occur. Boundary value analysis plays a crucial role in testing the system's response to input data boundaries. For example, when defining the range of acceptable input values for a specific feature, we conduct tests at the lower and upper limits, as well as just beyond them. This ensures that the software can handle extreme data values effectively and avoids potential errors that may occur at these critical boundaries. By systematically testing these scenarios, we enhance the reliability and robustness of the Surfaceer application.

6.4. Data flow testing

Data flow testing is a white-box testing method that focuses on the points at which variables receive values and the points at which these values are used or referenced. It validates the software's capacity to correctly process data through each function, subroutine, or object. The testing method ensures variables are initialized properly, not used before initialization, and not lost unintentionally. It tracks the flow of data to prevent any defects that could result from improper use or handling of data, such as data leakage, corruption, or loss during transit between different modules or systems. This type of testing is crucial in environments where data integrity and consistency are paramount.

6.5. Unit testing

Unit testing for the Surfaceer app involves creating a series of automated tests that individually verify the functionality of the smallest testable parts of the application, such as functions, methods, or classes. For Surfaceer, which might involve components like network scanning, vulnerability assessment, and reporting modules, each unit test would:

- Isolate each part of the program and show that the individual parts are correct.
- Mock dependencies to ensure that the unit tests only assess the functionality of the unit and not its integration with external components.

- Assert that the unit behaves as expected under various conditions by checking the results against expected outcomes.

For example, if Surfaceer has a function scan_network that takes an IP range as input and returns a list of active devices, a unit test would provide specific IP ranges, run the function, and verify that the output matches the expected list of devices. This would be repeated with different inputs, including edge cases, to fully test the function's reliability.

6.6. Integration testing

Integration testing via Postman involves testing the interactions between different components of the Surfaceer app by making API requests and verifying responses. You'd typically proceed as follows:

- Set up Collections: Organize your API endpoints into Postman Collections representing the different aspects of the Surfaceer app, like user authentication, asset discovery, and vulnerability scanning.
- Create Environments: Define environments in Postman with variables representing different testing stages – development, staging, and production.
- Write Test Scripts: For each API request, write test scripts in the Postman Tests tab to assert conditions that must be true after the API call is made, such as HTTP response status, response time, and the structure and data of the response body.
- Chain Requests: Use Postman's ability to chain requests where the output of one API call is used as the input for another, mimicking real-world usage and data flow in the app.
- Automate: Automate the running of these tests using Postman's Collection Runner or Newman, Postman's command-line companion, to execute collection runs against your API.

- Analyze Results: Evaluate the results in Postman's Runner or through generated reports in Newman to identify any discrepancies or failures between the integrated components.

6.7. Performance testing

Performance testing assesses how the Surfaceer application behaves under normal conditions. It measures response times, throughput, scalability, and resource utilization to ensure the application performs well when accessed by the intended number of users.

6.8. Stress Testing

Stress testing pushes the Surfaceer application beyond normal operational capacity to see how it handles extreme loads. It identifies the application's breaking points and helps observe how it recovers from failure. This test ensures the application can handle high traffic and data processing demands before returning to normal conditions.

Chapter 7

Summary, Conclusion and Future Enhancements

Chapter 7: Summary, Conclusion & Future Enhancements

7.1. Project Summary

The Surfaceer project represents a significant stride in cybersecurity, focusing on attack surface management. Throughout its phases, the project has successfully integrated various open-source tools and proprietary algorithms to create a robust platform capable of real-time threat detection, assessment, and mitigation. It encapsulates rigorous design, development, and testing processes, ensuring high reliability and performance in safeguarding digital assets.

7.2. Achievements and Improvements

Key achievements of the Surfaceer project include the development of an innovative deep scanning feature, integration of real-time data flow with existing cybersecurity frameworks, and the establishment of a resilient cloud-based deployment model. Continuous improvement efforts have led to enhanced usability, better performance under load, and more precise vulnerability detection and response capabilities. Like

- Developed an advanced deep scanning feature.
- Achieved seamless integration with existing cybersecurity frameworks.
- Established a resilient, cloud-based deployment infrastructure.
- Enhanced user interface for better usability.
- Improved system performance under high load conditions.
- Refined vulnerability detection with more precise responses.

7.3. Critical Review

The Surfaceer project's critical review reveals its robustness in real-time threat detection, though it also highlights areas for improvement. The integration of various tools proved to be both an asset and a challenge, requiring complex coordination. The project's strength lies in its robust detection algorithms and integration of diverse cybersecurity tools, providing comprehensive protection. However, this integration, while beneficial, introduced complexities in system coordination and interoperability. Challenges faced in achieving seamless integration highlighted the need for a more streamlined approach in combining various cybersecurity components. Understanding these dynamics offers valuable insights into balancing the integration of diverse tools with system efficiency and user-friendliness.

7.4. Lessons Learnt

Key lessons include the importance of modular design for easier updates and the necessity for extensive testing, especially in real-world scenarios, to ensure reliability. The key lessons from the Surfaceer project extend beyond technical know-how, encompassing valuable insights into software development and cybersecurity practices:

- Modular Design: Emphasizing modular architecture for easy updates and scalability.
- Real-World Testing: Underlining the need for extensive testing in real-world scenarios to verify system reliability and performance.
- Programming Skills: Enhancing programming capabilities to handle complex cybersecurity algorithms.
- Agile Development: Learning the effectiveness of agile methodologies in managing and adapting to changes throughout the project lifecycle.
- Cybersecurity Awareness: Gaining deeper understanding in cybersecurity, emphasizing proactive threat hunting and defense strategies.
- Collaborative Learning: Recognizing the importance of teamwork and knowledge sharing in tackling multifaceted challenges.

7.5. Future Enhancements/Recommendations

Recommendations for future enhancements include implementing AI-driven analytics for predictive threat modeling, adopting more granular user access controls, and expanding the integration with emerging cloud services for wider coverage. Additionally, implementing more granular user access controls would improve security by limiting access based on user roles and requirements. Expanding integration with emerging cloud services will also ensure Surfaceer remains compatible with the latest technologies and extends its protective coverage, keeping pace with evolving cybersecurity landscapes. These enhancements would position Surfaceer at the forefront of cybersecurity solutions.

Reference and Bibliography

Reference and Bibliography

1. Veshne, J., 2023. Attack Surface Management: Principles for simplifying the complexity of OT security.
2. Sarieddine, K., Sayed, M.A., Torabi, S., Atallah, R. and Assi, C., 2023. Investigating the security of ev charging mobile applications as an attack surface. ACM Transactions on Cyber-Physical Systems, 7(4), pp.1-28.
3. Everson, D., 2023. Cyber Attack Surface Mapping For Offensive Security Testing.
4. James, F., 2023. Security Management of Smart Home Internet-Of-Things: A Framework, Finite-State Attack Modeling, and Worst Attack Vulnerability Analysis (Doctoral dissertation, University of Missouri-Kansas City).
5. Ashley, T., Gourisetti, S.N.G., Brown, N. and Bonebrake, C., 2022. Aggregate attack surface management for network discovery of operational technology. *Computers & Security*, *123*, p.102939.
6. Rizvi, S., Orr, R.J., Cox, A., Ashokkumar, P. and Rizvi, M.R., 2020. Identifying the attack surface for IoT network. *Internet of Things*, *9*, p.100162.
7. Manadhata, P.K. and Wing, J.M., 2010. An attack surface metric. *IEEE Transactions on Software Engineering*, *37*(3), pp.371-386.
8. Szefer, J., Keller, E., Lee, R.B. and Rexford, J., 2011, October. Eliminating the hypervisor attack surface for a more secure cloud. In *Proceedings of the 18th ACM conference on Computer and communications security* (pp. 401-412).
9. Matherly, J., 2015. Complete guide to shodan. *Shodan, LLC (2016-02-25)*, *1*.
10. Genge, B. and Enăchescu, C., 2016. ShoVAT: Shodan-based vulnerability assessment tool for Internet-facing services. *Security and communication networks*, *9*(15), pp.2696-2714.
11. Chen, Y., Lian, X., Yu, D., Lv, S., Hao, S. and Ma, Y., 2020. Exploring Shodan from the perspective of industrial control systems. *IEEE Access*, *8*, pp.75359-75369.
12. Rae, J.S., Chowdhury, M.M. and Jochen, M., 2019, May. Internet of things device hardening using shodan. io and shovat: A survey. In *2019 IEEE international conference on electro information technology (EIT)* (pp. 379-385). IEEE.