National College of Ireland

Automated Detection of web and cloud Misconfigurations

MSc Research Project
M.Sc. Cybersecurity

Nishanth Madasamy
Student ID: x23396369

School of Computing
National College of Ireland

Supervisor: Khadija Hafeez

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Nishanth Madasamy

Student ID: X23396369

Programme: M.Sc. Cybersecurity **Year:** 2025

Module: Practicum

Supervisor: Khadija Hafeez

Submission Due Date: 11/12/2025

Project Title: Automated detection of web and cloud misconfigurations

Word Count: 5770 **Page Count** 18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

NISHANTH MADASAMY

Signature:

Date:

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

AUTOMATED DETECTION OF WEB AND CLOUD MISCONFIGURATIONS

Nishanth Madasamy
X23396369

Abstract

Modern web applications are increasingly relying on distributed APIs, cloud storage services, and federated authentication mechanism. While these technologies provide scalability and interoperability, they also introduce a large attack surface if not properly configured. Misconfigurations in policies associated with Cross-Origin Resource Sharing (CORS), Amazon S3 bucket permissions and OAuth/OpenID Connect (OIDC) identity flows continue to be among most prevalent and damaging vulnerabilities exploited in real-world attacks. Existing security tools usually target misconfigurations in only one of these domains and therefore produce fragmented security assurance assessments and inconsistent coverage during penetration testing and cloud auditing. This research proposes a Unified Web Misconfiguration Detection Suite, which is a multi-vector command-line-based scanning framework that aims to detect critical misconfigurations in CORS, S3 and OAuth/OIDC implementations. The system consists of three specialized scanners implemented in Python, each designed to emulate attacker behavior. The CORS misconfiguration scanner which tests for origin reflection, credential leakage and preflight vulnerabilities, the S3 bucket scanner which performs unauthenticated testing of public listing, read and write permissions and the OAuth/OIDC scanner which conducts passive analysis of discovery endpoints, JavaScript files and authentication flows for insecure configurations. A vulnerable laboratory environment with a Flask-based CORS test server, intentionally misconfigured OAuth endpoints and multiple S3 bucket scenarios were constructed to validate and benchmark the scanners. Experimental results show that the unified suite reliably detects many different misconfigurations causing high-severity issues. reflected CORS origins, publicly writable S3 buckets, insecure token flows, missing PKCE, exposed secrets and improper metadata in OAuth/OIDC providers. The results validate that grouping the misconfiguration detection in one framework offers superior visibility when compared to using separate tools. This project contributes a practical approach from an attacker perspective to multi-vector web and cloud misconfiguration assessment, as well as the importance of automating tooling to mitigate security risks in the modern web application ecosystem. **Keywords: CORS misconfiguration, S3 buckets, Web application security, Automated Security Testing.**

1 Introduction

Modern web applications are increasingly being built on distributed architectures that include browser-based clients, API backends, cloud storage platforms and third-party identity providers. While these components improve scalability and usability , they introduce complex security configurations that must be properly implemented across multiple layers. Misconfiguration has become one of the most widespread and destructive types of security weaknesses and is often ranked among the most damaging categories of cloud and web security incidents. Errors such as permissive CORS, public-facing Amazon S3 buckets, and insecure OAuth/OpenID Connect (OIDC) flows have led to large-scale data breaches, account compromise, and unauthorized access to sensitive cloud resources.

CORs policies control browser interactions with cross-origin requests to ensure that web applications can access resources only from trusted domains. However, misconfigurations, such as origin reflection, the combination of wildcard origins with

credential sharing, or overly permissive preflight responses, can bypass browser security controls and allow attackers to steal private information through malicious websites. Similarly, Amazon S3 buckets, while secure by default, are often made public due to improper access control lists (ACLs) or bucket policies, which allow anonymous user to list, read, or even write objects. OAuth and OIDC, which form the basis of modern authentication and single sign-on (SSO) systems, also suffer from common misconfigurations, including insecure handling of redirect URIs, lack of PKCE protection, use of obsolete implicit flows, and leakage of sensitive metadata or secrets.

Despite the frequency and severity of these issues, existing tools tend to address these misconfigurations in isolation. CORS scanners only test cross origin headers, S3 applications only investigate bucket access and OAuth analysis tools look into protocol compliance but not into deployment flaws. Security testers and cloud auditors are therefore forced to rely on multiple tools and to perform manual correlation, not to mention the inefficiencies and heightened likelihood of oversight. A fragmented approach to application assessment is no longer adequate as web applications continue to integrate cloud, identity and API services.

To fill this void, this research proposes a Unified Web Misconfiguration Detection Suite, which is a multi-vector scanning system aimed at finding misconfigurations in CORS, S3 and OAuth/OIDC environments from the perspective of an adversary. Unlike traditional scanners, this suite combines three security modules in one workflow, which allows for complete detection of cross origin, cloud storage and identity misconfigurations. The scanners are written in Python and embedded into a command line launcher which allows an accessible interface to penetration testers, bug bounty hunters, and cloud security analysts.
This research investigates whether a non-destructive, CLI-based scanning tool can reliably detect high-risk misconfigurations across web and cloud environments without requiring privileged access or specialized infrastructure. In particular, the core research question guiding this work is
“Can a non-destructive CLI Scanner detect high-risk CORS misconfigurations with low false-positive rate on authorized targets?”
To address this gap, the objective of this project is to design and implement a unified, attacker-oriented misconfiguration detection suite capable of reliably identifying:
(i)high-risk CORS weakness such as origin reflection, wildcard-credential combinations, and unsafe preflight responses, (ii)insecure cloud storage permissions including public s3 read, write and list exposure, (iii)OAuth/OIDC configuration flaws such as missing PKCE, vulnerable implicit flows, exposed secrets, and unsafe metadata. The system aims to demonstrate that such misconfigurations can be detected safely, quickly, and accurately using only standard HTTP behavior, without generating harmful traffic or requiring access to private credentials.

2 Literature Survey

Many web and cloud ecosystems require cross-origin communication, cloud-based object storage, and federated identity protocols. While these technologies provide flexibility and scalability, research has repeatedly shown that misconfiguration remains a dominant factor in security breaches. This section reviews existing academic work related to CORS, S3 buckets, and OAuth/OIDC security, which correspond to the three components of the unified scanner developed for this project.

2.1 CORS, Same-origin Policy, and web Security Headers

The Same-Origin Policy (SOP) remains the main defense against cross-site attacks but several research papers show that SOP is inconsistent and limited. Schwenk et al. [4] performed one of the most thorough analyses of SOP between the modern browsers, and found behavioral inconsistencies that the attacker can exploit. These inconsistencies formed the basis for subsequent research on bypass techniques and cross-origin resource manipulation.

Chen et al. [1] conducted a large-scale analysis of CORS deployments, and discovered that a large number of websites expose themselves to credential theft, data exfiltration, and arbitrary cross-origin interaction as a result of careless CORS deployment. Meiser et al. [2] further pointed out that there are logic flaws in cross-origin communication in which incorrect trust assumptions between origins could be exploited.

Golinelli et al. [3] built on this by analyzing the disconnect between intended and actual security outcomes, demonstrating how even developers that try to follow the best practices fail to implement them correctly. Parallel research by Weissbacher et al. [5] showed that the implementation of modern security headers - especially CSP - is still inconsistent with often incorrect or incomplete application. Collectively, these studies identify a recurring challenge with the interpretation and/or implementation of cross-origin security controls by developers, which collectively means that opportunities for exploitation exist across the board.

2.2 Cloud Storage and S3 bucket misconfiguration

Misconfigured cloud storage remains one of the most frequently exploited attack surfaces in the cloud. Continella et al. [6] show that even small errors - including public "list" permissions - can lead to serious data exposures. Their findings emphasize how many organizations are not aware of the implications of partial public access, believing as they do that buckets are private when in fact they are not.

Further evidence of this is provided by the Strato-Sphere project [7] that presented techniques for automatically discovering publicly exposed buckets on a scale. Their study found thousands of organizations leaking sensitive data, some of which were found to be logs, credentials, and internal configuration files. This highlights the need for active probing rather than relying solely on cloud configuration metadata.

Yadmani et al. [8] performed an in-depth empirical analysis of data leaks in cloud buckets and found that storage misconfigurations often expose API Keys, private tokens, and confidential documents. The study also looked at responsibility for disclosure procedures and found that it took many organizations months to remediate leaks

2.3 OAUTH and OIDC Security

OAuth 2.0 is a standard for authentication and third-party access delegation; However, studies have consistently indicated that improper application of technology presents crucial vulnerabilities. Fett et al. [10] conducted a thorough formal study of OAuth and found that the protocol has vulnerabilities that emerge when implementers deviate from the specification.

Their contribution is still considered as a foundation in the knowledge of protocol-level risks. In this regard, Rahat et al. [9] created Cerberus, which evaluates the correctness of OAuth deployment and detects configuration-level errors in the production environment. Their findings demonstrate that real-world OAuth implementations often fall short, with issues such as insecure redirect URIs, disabled PKCE support, weak token implementation, and unsafe HTTP endpoints.

RFC 9700 [11] codifies best practices for OAuth 2.0 security highlights the importance of PKCE implementation, secure transport (only HTTPS), limited whitelisting of redirect URIs with specific redirect URI, and appropriate exposure of JWKS metadata. Failure to follow these recommendations remains a common cause of OAuth-based account compromise.

This research works directly support the design of the OAuth/OIDC scanner component of the whole suite that examines metadata documents, verifies PKCE and implicit flow problems, and identifies secrets or tokens that are being disclosed with the use of client-side JavaScript.

2.4 Summary of research gaps

In all the three areas, CORS, S3, and OAuth, the research shows there is a common theme: security controls are very powerful but highly prone to misuse. However, despite the many academic studies and standards, there is no one tool that:

- Reflectively and actively tests CORS,
- Storage unidentified access, and
- Performs OAUTH/OIDC metadata and JavaScript validation

3 Research Methodology

The methodology is structured and empirical and combines software engineering practices and attacker-oriented security testing. The overall goal is to determine whether a multi-vector scanning framework can be used to identify CORS, Amazon S3 and OAuth/OIDC misconfigurations in modern web applications.

The methodology has five elements, namely research design, data collection, scanner development, test environment construction, and evaluation strategy. Ethical and legal considerations are also discussed in order to ensure responsible security testing.

3.1 Research Design

The research design is based on the Design Science Research Methodology (DSRM) proposed by Hevner et al. which is commonly used for research involving the creation of a new security tool, security framework or process. DSR is appropriate for this project because the major output is a working software artefact -- a unified misconfiguration scanning suite -- which solves a real and well-defined security problem. The DSR process involves the following stages:

3.1.1 Problem Identification and Motivation

The literature review demonstrates that misconfigurations of CORS, S3 buckets and OAuth/OIDC systems are prevalent, and they cause many real-world security breaches. Current tooling is fragmented as each type of vulnerability requires a separate scanner. This fragmentation increases complexity and leaves room for oversight in security assessments. Thus, the problem is clearly defined: There is no unified and attacker-oriented tool that scans multiple high-impact web and cloud misconfiguration classes from one interface.

3.1.2 Solution Objectives Definition

The system aims to:

- Detect misconfigurations in CORS, S3 buckets and OAuth/OIDC deployments
- Consolidate these tests into a single command-line interface
- Impersonate the behavior of an attacker
- Provide actionable and readable reports for penetration testers and security analysts

3.1.3 Design and Development of Artefact

The artefact is made up of three independent modules of the scanner and a unifying launcher framework written in Python. Each scanner uses specifically targeted detection strategies based on empirical research.

3.1.4 Demonstration

The artefact is demonstrated in a controlled vulnerable environment that consists of intentionally ill- configured services:

- A CORS Misconfigured flask server
- A set of misconfigured and S3-compatible storage buckets
- An unsafe OAuth/OIDC provider

3.1.5 Evaluation and Analysis

The performance of the scanners is tested with regard to detection accuracy, coverage, false positives/negatives and practical usability.

3.1.6 Communication of Results

Research findings are documented in this report and can be demonstrated through practical use cases for academic assessment or professional penetration testing.

3.2 Methodological Approach Rationale

This project is rather practical, applied and tooloriented in contrast to theoretical security research. The research requires:

- Building real code
- Testing against Real vulnerabilities
- Quantifying actual scanner output

3.3 Data Collection strategy

This project does not rely on user data or datasets; instead, it focuses on behavioural data collected from live system interactions. The data collection process is performed programmatically through scanner operations and includes:

3.3.1 HTTP Behavioral Data (CORS)

Captured from:

- Manipulated Origin header requests
- Preflight OPTIONS responses
- Reflected headers
- Response code patterns
- CORS header combinations

This is critical in detecting origin reflection, wildcard + credential mixes or insecure method allowances.

3.3.2 **Cloud storage access behavior (S3) Collected from attempts to:**

- List objects anonymously
- Read objects
- Upload (PUT) objects
- Retrieve storage metadata

Compared results of access are compared with the known codes of S3 errors (AccessDenied, AllAccessDisabled, NoSuchKey, etc.) to make a guess on misconfigurations.

3.3.3 OAuth/OIDC Metadata and Endpoint Behavior

Information gathered includes:

- .well-known/openid-configuration JSON
- Authorization_endpoint, token_endpoint, jwks_uri fields
- Redirect URI validation behaviour
- Hardcoded secrets detected in javascript files
- Subdomain probing results

3.4 System Development Methodology

The system has been developed in an incremental, modular and test-driven manner, so that each individual scanner can be developed, validated and refined separately of one another.

3.4.1 Technology stack justification

- Python 3.10+: Ideal for rapid prototyping and security tooling.
- Requests library: reliable for HTTP testing.
- Argparse: Simplifies CLI interface creation

3.4.2 Modular structure

The architecture consists of three components

1. **CORS Scanner Module**
   - Implements dynamic origin manipulation
   - Performs credentialed and uncredentialed tests
   - Sends altered preflight requests
   - Analyses CORS vulnerability patterns based on established patterns

2. **S3 Scanner Module**
   - Attempts anonymous read/list/write operations
   - Uses REST-Style S3 URL patterns(https://bucket.s3.amazonaws.com/object)
   - Detects weak ACLs or bucket policies.

3. **OAuth/OIDC Scanner Module**
   - Retrieves discovery metadata (/.well-known).
   - Flags missing fields(issuer,jwks_uri, etc).
   - Scans JavaScript file for: secrets, Token literals, OAuth flows and Redirect URIs
   - Tests subdomains (eg., auth.*, login. *, sso. *,)

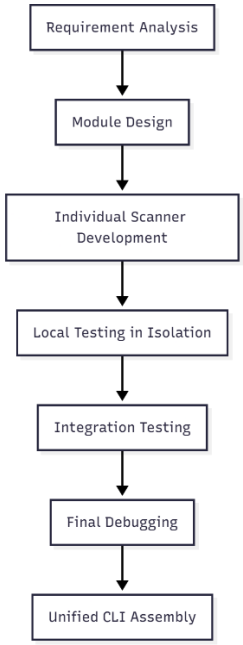


Figure 1 workflow diagram

4 Design and Implementation Specifications

This section represents the design architecture, system components, module interactions and implementation details of the Unified Web Misconfiguration Detection Suite. The system is made up of three independent scanners (CORS, S3, OAuth/OIDC) that are connected using a single unified command-line interface (CLI). The design philosophy adheres to the principles of modularity, extensibility and attacker-oriented behavior and enables the accurate detection of misconfiguration in the web and cloud environments.

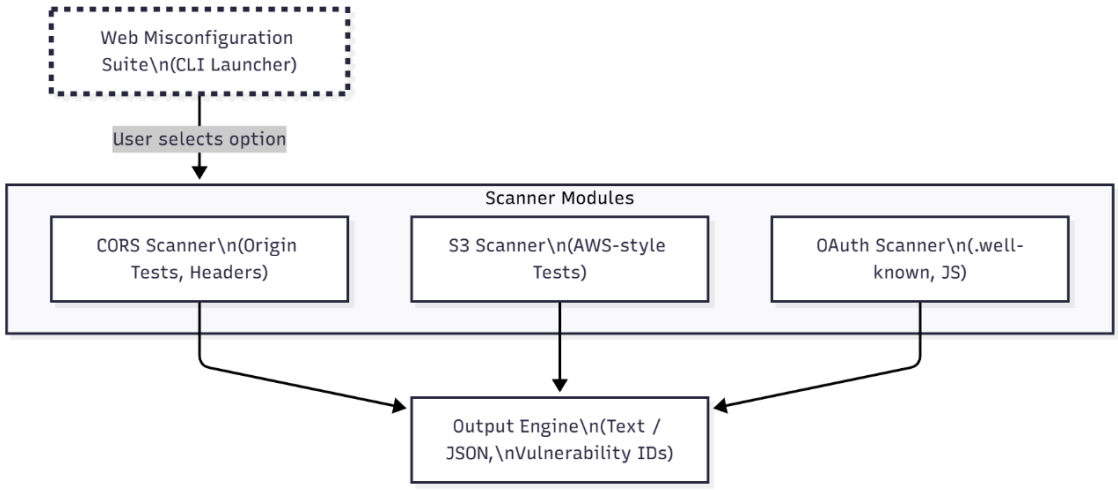


Figure 2 high level architecture of suite

Figure 2 represents the high-level architecture of the scanning suite developed and illustrates a structured and modular approach. The CLI launcher serves as the starting point to the system by providing a simple and easy-to-use interface to choose the scanner of interest. Once a scanner is selected, then execution is delegated to the corresponding scanning module. Each module, that is CORS, S3 or OAuth implements specialized routines for testing misconfigurations in their own respective domain.

4.1 Design Specification

1. Modularity

Each scanner is a separate module that has:

- Independent execution
- Separate detection logic and vulnerability heuristics
- Unique outputs and identifier

2. Attacker oriented behavior

The Scanners simulate real-world attacker actions:

- Spoofed origins
- Anonymous S3 access attempts
- OIDC metadata reconnaissance
- JavaScript enumeration

3. Performance

Concurrency is used for

- JS Scanning
- Subdomain enumeration
- S3 object testing

4.2 System Architecture Overview

The Unified Misconfiguration Detection Suite has been built on a modular, service-oriented architecture, in order to provide separation of responsibilities, ease of maintenance, and extensibility. Each of the different scanners (CORS, S3, and OAuth/OIDC) is implemented as

a different executable component with its own logic, its own input validation routines, and its own output formatting. These modules are orchestrated by a central command line interface launcher which serves as the primary interaction layer for the user. This architecture design means that improvement, bug fixes or extra scanning modules can be added without altering the basic framework of the system. Attacker-oriented behavior is also emphasized by the architecture which allows the suite to replicate realistic threat scenarios by generating crafted HTTP requests and probing storage endpoints anonymously and collects metadata from identity providers.

4.3 Implementation strategy

The implementation of the suite was done using a bottom-up approach and started with individual scanner prototypes and ultimately brought them together to form a unified framework. Each scanner used to be developed as a separate Python script so that each module could be validated separately. The CORS scanner was deployed by crafting custom made headers in the form of the http requests and by analysing the server response for misconfiguration patterns. The S3 scanner was based on raw use of the HTTP operations instead of cloud API, which allows the probing to be true to unauthenticated attacker behavior. The combination of metadata retrieval and JavaScript enumeration used by the OAuth/OIDC scanner was able to see problems that may not be visible in protocol-level documentation. Once each scanner was made stable, they were incorporated into the unified CLI interface, allowing scanning modes to be user-driven in controlled workflow

4.4 Internal module Interaction

Although each module is functioning autonomously, internal interaction between components is coordinated by the central CLI launcher. This launcher is responsible for input validation, command routing and error management as well as output presentation. When a user selects a scanner, the CLI calls the relevant module and passes it the necessary parameters such as target URLs, origins & file paths. The returned results are presented using a consistent structure for the presentation to ensure that the user gets uniform feedback from all types of scans. This interaction model also provides for the potential future integration of asynchronous execution, batch scanning or automated reporting engines without the need to change the underlying scanning logic.

4.5 Severity Classification Logic

To ensure that identified issues are reported in a meaningful and actionable manner, Unified Web Misconfiguration Detection Suite provides a specific severity rating to each finding based on security impact, exploitability and probability of leading to real-world compromise. This criticality model enables penetration testers and analysts to fast-track remediation prioritization and gain insight on the relative significance of every misconfiguration. The scanner defines four severity levels: High, Medium, Low, and Informational which are consistently applied in the CORS, S3, and OAuth/OIDC modules.

1. **CORS Severity Logic (CORS-001 to CORS-007):**
    - **CORS-001- Wildcard Origin+ Credentials Allowed(High Severity)**
      This problem is marked as High as the application of the Access-Control-Allow-Origin: + with the Access-Control-Allow-Credentials: true can be considered a violation of the browser-enforced same-origin protections.

A malicious web-hosting site will be capable of accessing sensitive authenticated data, by merely making a request to the target.

- **CORS-002-Reflected Origin (High Severity)**
  This occurs when the server reflects any Origin value sent by the client without validation. Attackers can falsely identify themselves to the server in order to steal the session data through malicious websites. It is fundamentally insecure and therefore classified as High Severity.

- **CORS-003-Credentials Permissive Origin (Medium Severity)**
  When credentials are permitted and the origin is too broad (e.g. anything that ends with .example), the risk is less than direct reflection, but nevertheless still too large to deny third-party access. Rated medium since an attacker must first gain control of a trusted subdomain or ineffective origin filtering.

- **CORS-004-Preflight Reflected Origin (High Severity)**
  The unsafe cross-origin request may be permitted during a preflight (OPTIONS) request as it indicates to the attacker's origin that the request is safe. Given that many of the high-risk operations must be pre-approved before takeoff, such a misconfiguration can be used to cross-originally modify the resources that were protected.

- **CORS-005-Preflight allows Risky Methods (Medium Severity)**
  Allowing any origin to use methods such as PUT/DELETE, which expands the attack surface. This misconfiguration promotes cross-origin alteration of server data although this type of alteration may not always be straightforward and uncertified.

- **CORS-006-Sensitive Headers Exposed (Medium Severity)**
  If the server exposes headers such as Authorization or Set-Cookie, the attackers can retrieve authentication or session cookies. It is a Medium severity which escalates to High, in relation to reflection or wildcard origins.

- **CORS-007-overly Long Max -Age(Low Severity)**
  When the value of Access-Control-Max-Age is large, browsers store CORS decisions over long periods of time. Although it cannot be used directly, it makes incorrect policies persist for longer periods and makes it difficult to recover quickly.

- **CORS-000-No CORS Headers(informational)**
  Lack of the CORS headers is not a vulnerability. The scanner puts it as informational only.

2. **S3 bucket Severity Logic:**
   he S3 scanner determines risk depending on the presence of the read, write, or list permissions. These are directly correlated to attacker capabilities.
   - **Public Write (High Severity)**
     With write access, an attacker may upload malicious files, serve phishing kits or replace the existing data. This can usually result in malware hosting, ransomware staging or reputational harm. Thus, the write access of the public is never less than High.
   - **Public Read (Medium Severity)**
     In case the objects can be read publicly, the attacker can steal confidential files, logs, personal information, backups, or API keys. Severity is Medium due to the inability of the attacker to alter the content and has the ability of exfiltrate sensitive data.
   - **Public List (Medium Severity)**
     Attackers will be able to enumerate all the stored files in case the bucket allows / ?list-type=2. Enumeration has a significant impact on exposure, and even though it is not as harsh as write access, it is rated as Medium.
   - **Bucket Does exist but Denied (Low Severity)**
     This discloses details on internal bucket names to assist attackers in reconnaissance to takeover or aggressively enumerate cloud resources. Low rating based on a minor direct impact.

3. **OAUTH/OIDC Scanner Severity logic**
   The OAuth/OIDC module uses the severity of whether misconfigurations have the potential to result in account takeover, token leakage or session impersonation.
   - **HTTP_ENDPOINT - Use of HTTP instead of HTTPS(High Severity)**
     The OAuth flows should be on TLS. The use of HTTP allows attackers to intercept and manipulate tokens and credentials . According to RFC 9700, it is a High severity problem.

   - **IMPLICT_FLOW – Insecure Token Flow Enabled (High Severity)**
     This implicit flow reveals tokens in urls. When turned on, the attackers can steal the tokens through redirect interception or browser history. This earns a High rating.

   - **MISSING_PKCE – PKCE not supported (Medium Severity)**
     The modern OAuth security requires PKCE. In its absence, authorization code interception attacks can take place. Medium rating based on the fact that it depends on other conditions.

   - **HARDCODED_SECRETS in JavaScript(High Severity)**
     Client secrets exposed in front-end code are permanently compromised. Attackers have the ability to masquerade as apps or mint tokens.

- **TOKEN_LITERAL_IN_JS – Token strings Found(High Severity)**
  When an access token or ID token is embedded into the JavaScript files, this means that the leakage is severe. This is High Severity.

- **THIRD_PARTY_IDP -Reference (Low Severity)**
  Not inherently dangerous, however, mentioned as a violation of the trust limits. May indicate integration misconfigurations. Rated Low.

- **Missing revocation/introspection endpoints(Low Severity)**
  These do not affect security directly but make the implementation of OAuth weaker.

5 Evaluation

This Chapter evaluates the performance, accuracy, usability, and practical relevance of the Unified Web Misconfiguration Detection Suite. This Evaluation was conducted in a controlled laboratory environment containing intentionally misconfigured CORS endpoints, S3-style storage buckets, and OAUTH/OIDC identify services. The goal of the evaluation is to determine whether the proposed tool reliably identifies misconfigurations and how it compares to existing security tools commonly used by penetration testers and cloud auditors.

5.1 Evaluation Objective

Does the cli Scanner accurately detect known CORS, S3 and OAUTH/OIDC misconfigurations in a website?

5.2 Evaluation method

Since the project does not involve machine learning, evaluation was performed using following methods:

1. **Functional Testing**
- Each Scanner was tested against known misconfigurations in a purposely vulnerable environment.
2. **Scenario-Based Analysis**
- Foreach module (CORS, S3, OAuth) multiple misconfigured and correctly configured scenarios were tested
3. **Comparative Tool Analysis**
- The tool was compared against existing open-source and commercial tools.
4. **Performance Evaluation**
- Measured using execution time and responsiveness

5.3 Comparison with Existing tools

Feature	Our suite	CORScanner	Prowler	ScoutSuite	Trufflehog	Burp/OWASP ZAP(by manual testing)
CORS Detection	*	*	X	X	X	*
Preflight test	*	X	X	X	X	X
S3 public read /list	*	X	*	*	X	X
S3 public Write	*	X	X	X	X	X
OAuth Metadata Analysis	*	X	X	X	X	X
PKCE/Implicit Flow checks	*	X	X	X	*	*
Js Secret Detection	*	X	X	X	X	X

Table 1 comparison with existing tool

The table 1 comparison clearly shows that none of the single existing tools offer a full coverage in the domain of CORS, S3 storage and OAuth/OIDC misconfiguration detection. While CORScanner only addresses CORS, cloud auditing tools such as Prowler and ScoutSuite analyze the configuration of a cloud but cannot do active S3 read/write probing

5.4 Performance Evaluation

Module	Avg time
CORS Scanner	55 sec
S3 Scanner	45 sec
OAuth Scanner	47 Sec
Web misconfiguration suite	~53 Sec

Table 2 performance evaluation

Table 2 represents the performance is that each scanner in the suite has a reasonably efficient window of execution and completes their tasks in less than one minute. The CORS scanner took the most time because it must perform multiple origin and preflight checks, while the S3 scanner took a bit longer because it is only read/list write probing. The OAuth scanner has moderate execution time, the metadata retrieval and JavaScript analysis steps. These results show that the tool is suitable for quick security assessments and can fit into penetration testing or bug bounty processes very well without causing a lot of delays.

6 Discussion and conclusion

6.1 Discussion

The main goal of this project was to design and implement a unified web misconfiguration detection suite that can detect common vulnerabilities in three important areas of web security: CORS misconfigurations, S3 storage weaknesses and OAuth/OIDC configuration weaknesses. The results, both from development and testing, show that the proposed system achieves these goals effectively and offers capabilities not found in any individual security tool available today.

6.1.1 Multi-Vector Coverage

One of the most significant contributions of this project is the integration of three independent scanners into one tool

- CORScanner – focuses only on CORS
- Prowler/Scout site – cloud compliance and policy auditing
- Trufflehug – Secret detection
- BurpSuite -general web proxy tools with partial coverage with manual testing

However, no single tool provides:

- Full CORS header + preflight
- Active s3 read/list/write probing
- Oauth/OIDC metadata validation
- Javascript secret detection

This integration makes the proposed suite more efficient for the testers by reducing the need for multiple tools and improving the consistency of detection results.

6.2 Conclusion

This project was able to successfully develop a unified misconfiguration detection suite that incorporates multiple scanning capabilities that typically don't exist together in the same tool. The design approach focused on lightweight attacker-oriented testing techniques so that the scanner closely mirrors real world exploitation techniques whilst still being fast and easy to use.

A major success of this research is the development of a modular, Python-based framework comprising three independent detection engines (for CORS misconfigurations, S3 storage exposures and OAuth/OIDC identity configuration issues). These components were integrated into a one unified command line interface, and the security tester could conduct multi-vector misconfiguration analysis without single tool switching. To properly validate the system's performance, a dedicated vulnerable environment was built along with misconfigured CORS endpoints, s3 compatible buckets, and OAuth/OIDC discovery metadata. This ensured that the evaluation was controlled as well as realistic

The empirical evaluation proved that the scanner performed reliably under all test scenarios. It was able to identify reflected origins, permission issues with wildcards, unsafe preflight behaviours, public s3 read/list/write exposures, missing PKCE enforcement, insecure implicit flows, and hardcoded java script secrets. Compared with existing tools, the proposed suite has

wider feature coverage compared with any single free tool, making it more suitable for penetration testers, cloud security analysts and bug bounty practitioners who are in need of misconfiguration detection with less setup efforts.

Overall, this research demonstrates that you do not need to scan for misconfigurations in a very heavyweight enterprise environment or a complete cloud-credentialed audit. The system that has been developed within this project is therefore both of academic and practical interest and can address the specific issue of providing a unique contribution to the field of lightweight, focused security testing.

7 Future Works

While the unified misconfiguration detection suite has met its key goals, there are a number of opportunities to expand and improve the system for future research. One of the most promising directions is the introduction of the authenticated scanning capabilities. The current implementation has a single focus on unauthenticated, attacker style probing which is very effective for surface configurations misconfigurations. However, there are many security issues in the real world that are only relevant when authentication flows are involved, such as credentialed CORS failures, insecure OAuth redirect behaviour, token leakage and session mismanagement. Supporting authenticated testing, full OAuth authorization code flows in particular, would greatly increase the applicability of the scanner within enterprise environments.

Another useful improvement would be deep integration with clouds, especially for S3 like storage. While the current tool proactively tests for anonymous read, write, and list misconfigurations, it does not analyse IAM policies, inheritance of ACL and organisation-wide cloud configurations. Future versions could be used to directly interact with cloud provider APIs such as AWS STS, Azure Identity or Google Cloud IAM to get granular policy analysis and cross-account permission evaluation. This would be sufficient to fill the gap between lightweight misconfiguration testing and complete cloud posture management.

Expanding the scanner to identify more types of misconfigurations for the web and the cloud is also a natural step. Future work could include adding modules to analyze Content Security Policies (CSP), validate cookie security, validate JWT signatures, inspect security headers in the configuration of an application, and also to automatically detect insecure third party dependencies. Such additions would be based on the modular architecture already implemented, and would further increase the coverage of the scanner, while retaining its lightweight design philosophy.

Beyond technical capacities, future iterations of the system may include some improved usability and accessibility. Developing a graphical user interface or a web-based dashboard would make the tool more approachable for less technical users and be able to centralise viewing of scan results, exportable reports and real-time alerts.Overall, these improvements would greatly extend both the functionality and practicality of the system, enhancing the value of the tool as a security assessment tool and paving the way for more advanced academic and industry-focused research.

References

[1] J. Chen *et al.*, “We Still Don’t Have Secure Cross-Domain Requests: an Empirical Study of CORS We Still Don’t Have Secure Cross-Domain Requests: an Empirical Study of CORS,” 2018.

[2] G. Meiser, P. Laperdrix, and B. Stock, “Careful Who You Trust: Studying the Pitfalls of Cross-Origin Communication,” *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, May 2021.

[3] M. Golinelli, E. Arshad, D. Kashchuk, and B. Crispo, “Mind the CORS.” Accessed: Dec. 04, 2025.

[4] J. Schwenk, M. Niemietz, and C. Mainka, “Same-Origin Policy: Evaluation in Modern Browsers Same-Origin Policy: Evaluation in Modern Browsers,” 2017.

[5] M. Weissbacher, T. Lauinger, and W. Robertson, “Why is CSP Failing? Trends and Challenges in CSP Adoption.” Accessed: Dec. 04, 2025.

[6] A. Continella, M. Polino, M. Pogliani, and S. Zanero, “There’s a Hole in that Bucket!,” *Proceedings of the 34th Annual Computer Security Applications Conference*, Dec. 2018.

[7] F. J. Acm Reference, D. Cable, L. Gregory, Z. Izhikevich, and Durumeric, “Strato-sphere: Finding Vulnerable Cloud Storage Buckets,” vol. 13, 2021.

[8] S. Yadmani, O. Gadyatskaya, and Y. Zhauniarovich, “The File That Contained the Keys Has Been Removed: An Empirical Analysis of Secret Leaks in Cloud Buckets and Responsible Disclosure Outcomes.

[9] Rahat, Tamjid Al, Y. Feng, and Y. Tian, “Cerberus,” *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, Nov. 2022.

[10] D. Fett, R. Küsters, and G. Schmitz, “A Comprehensive Formal Security Analysis of OAuth 2.0 *,” 2016.

[11] Torsten Lodderstedt, J. Bradley, Andrey Labunets, and D. Fett, “RFC 9700: Best Current Practice for OAuth 2.0 Security,” *IETF Datatracker*, 2025.