International Conference on Advanced Computing, Security, and Intelligent Systems (ICACSIS-2025)
ISSN: 3107-9458| ICACSIS-2025 - Conference Proceeding
DOI: https://doi.org/10.64137/3107-9458/ICACSIS-105

Original Article

Securing REST APIs: OAuth2 vs JWT Weaknesses

Kavya Muppaneni
Senior Software Engineer at HCL Global Systems, USA.

Abstract: *With the ongoing digital transformation, modern software applications are increasingly using REST APIs to facilitate communications among distributed systems, mobile apps, and cloud services, turning API security into one of the most pressing issues in the digital world today. Since APIs are a common route to access sensitive data and key functionalities, improper controls of their authentication and authorization mechanisms can be exploited to cause serious damages. OAuth2 and JSON Web Tokens (JWT) are arguably two of the most popular technologies by which REST APIs can be safeguarded; consequently, they have set the standard for the industry because of their scalability and flexibility. OAuth2 sets up an authorization delegation architecture under which a client can be given access without having to expose the credentials, whereas the JWT functions as a compact, self-contained token carrier of the claims between two parties. Due to the fact that they have been adopted en masse, it might be assumed that they do not have any security risks, however, if for some reason the design, configuration or implementation are done incorrectly, they will each have a market share of such risks. The paper documents the main security failures of OAuth2 and JWT to token leakages, insufficient token validations, insecure token storages, over-privileged scopes, replay attacks, and algorithm-related vulnerabilities. An extensive exposure of OAuth2 and JWT on various aspects of security such as confidentiality, integrity, token lifecycle management, and attack surface is done through a comparative analysis methodology. Besides, a case study of the real world is incorporated to expose the impact of some common misconfiguration and design choices in security frameworks otherwise thought to be strong. It is demonstrated that the major problems are not the result of the standards themselves but rather the misuse of developers, lack of threat modeling and inadequate enforcement of best practices. This paper sheds light on the security weaknesses of OAuth2 and JWT, depicts the real risks faced by REST APIs today, and provides some guidance towards better decision making from the security point of view by architects and developers in the selection and implementation of API authentication strategies.*

Keywords**:** *REST APIs, API Security, OAuth2, JSON Web Token (JWT), Authentication, Authorization, Web Security, Access Control.*

1. INTRODUCTION

1.1. CHALLENGES IN SECURING REST APIS

A drastic change in software architecture occurred when developers began embracing RESTful APIs as the major component of their apps. Microservices-based systems, cloud-native platforms, mobile applications, and third-party integrations are examples of architecture styles that use APIs as a medium to communicate between different components or systems in a distributed environment. This transition not only brought architectures an increase in scalability, flexibility, and speed of development but at the same time security vulnerabilities have been escalated due to the increase of attack surface. Basically, every exposed API endpoint is another potential doorway for hackers, hence API security has become a major concern across organizations regardless of their size.

A REST API's statelessness is one of its main features that distinguishes it from other APIs. The difference between them and the old web apps lies in web session management: REST Web services do not maintain any client context on the server side between two subsequent requests. This architecture really has made it easier to scale and has increased the system's performance but keeping authentication context becomes a client's responsibility through tokens normally generated by the server. Clients these tokens are part of each request, normally in HTTP headers, which makes it vulnerable for attackers to intercept and thus misuse the token. The dangerous consequence of a stolen token is the attacker having almost unlimited access to the system without necessarily needing to go through server-side validation again.

At the center of the new wave of threats to REST APIs are token leakages that have become the most exposed area. The storing of tokens on the client side, the logging of sensitive header information, or the sending through an improperly secured channel are typical examples of ways this problem continues to appear. In addition to the aforementioned abuse of tokens (replay attacks) where the attacker sends the token repeatedly to the server to masquerade as a genuine user, the API has been further exposed through the misconfigurations mentioned such as the granting of excessive privileges, failure to set rate limits, and use of insufficient token validation. Moreover, the modern API ecosystems complicate matters, where different services from different trust domains interact with each other.

Furthermore,there is an overall complexity increment due to the progressive implementation of authentication and authorization.APIs need to serve different types of clients such as web browsers, mobile apps, server-to-server communication, and third-party integrations. Each client type comes with its own set of security requirements and risk models.So, a developer is very often forced to turn to well-known standards and protocols like OAuth2 and JWT for the handling of access control. However, the security architecture can become vulnerable if these methods are not properly understood.Besides choosing the most suitable security solutions, the problem is also about having the capability to correctly apply them in a complex and highly interconnected environment.

1.2. PROBLEM STATEMENT

Although there are good well-established standards for API security, numerous real-world systems are still exposed to critical vulnerabilities. The core problem is that most developers unwisely depend excessively on OAuth2 and JWT thus the default solutions for securing REST APIs, often these are done without a complete understanding of their intended use, limitations, and security implications. Such technologies have been considered "plug-and-play" solutions with the result that it is assumed that just because they have been adopted, security is guaranteed. However, in reality, this assumption is the furthest from the truth.

OAuth2 is an authorization framework that allows delegation of access on behalf of a resource owner, whereas JWT is a token format designed to carry claims between parties. Usually, developers get the two mixed up or even use them interchangeably in most cases, thus they blur the line between technicalities. Hence, resulting in architectural confusion. Besides that, developers may justifiably or otherwise use JWTs as a substitute for OAuth2 flows, or they may improperly use OAuth2 access tokens as a means of authentication. The scenario that entails mixing authentication (who you are) with authorization (what you are allowed to do) has granted attackers a way;

It is often the case that OAuth2 and JWT are poorly implemented in live systems. Some of the examples cited are: using access tokens with long life span without proper rotation; not validating the token signature and claims; handling refresh tokens in an unsafe manner; using weak or outdated cryptographic algorithms. Sometimes, APIs trust blindly tokens issued by external identity providers without any validation of scopes, audiences, or issuers. Such errors usually occur due to lack of security knowledge, poor documentation or the need to rapidly deliver features.

Moreover, not many thorough, security-focused comparative researches have been done that clearly indicate the scenarios and ways of OAuth2 and JWT integration or separation. Generally, most of the available resources dwell on features and compatibility rather than on threat models and risk assessments. Therefore, developers and architects miss the assistance to decide proper security solutions for the case and threat at hand. This issue demands a detailed examination which concentrates on the vulnerabilities, threats, and actual security impacts of OAuth2 and JWT in REST API environments.

1.3. MOTIVATION

The purpose of this investigation is to raise awareness about security vulnerabilities through APIs by growing the number of cases and their aftermath. Huge scandals have demonstrated that leaked tokens can be utilized to cause massive data leakage, gaining control over user accounts, or silently breaking into systems. While businesses extend their APIs, attackers are getting more daring to outsmart token-based authentication mechanisms for unauthorized access. These scenarios indicate that it is essential not just to accept the standards of the industry but also to understand the security premises and drawbacks of these standards.

Besides,there is still a pretty big misunderstanding among the industry professionals about how to properly use OAuth2 and JWT. That misunderstanding is also one of the main reasons for this study. A lot of developers get confused about which technology does what, as a result, their use is sometimes inconsistent and insecure. Moreover, tutorials, sample code, and third-party libraries often

break down security concepts too much, thus they promote the use of patterns that may be fine functionally but fail when an adversary is present. The level of confusion is very evident in the case of the stateless authentication, token storage, and token validation practices talk.

Another aspect is the increasing demand for useful, step-by-step instructions addressing developers and system architects specifically. Formal specs define the OAuth2 and JWT mechanism, but they don't always show the usual mistakes or attacks that happen in the real world. Besides, security guidelines are frequently spread out in different documents, which makes it hard to get a unified picture of the best security practices. That leads to a disconnect between the theoretical models of security and the actual system design.

This essay aims at doing just that: it is primarily a write-up to bridge the gap between the security theory and practice of OAuth2 and JWT. Looking through a security-oriented lens, the paper explores the security of OAuth2 and JWT both in theory and when their capabilities are pushed to the extreme in the case of REST APIs. The emphasis is on the usage of these standards in the actual world, the common mistakes, and how to handle them effectively. Ultimately, the present paper's mission is to guide developers and system architects in making sound decisions, designing security architectures for APIs capable of resisting attacks, and not being deceived by blindly trusting the security of standards without knowing the correct way to implement them.

2. LITERATURE REVIEW

2.1. EVOLUTION OF API AUTHENTICATION MECHANISMS

Initially,the ways to secure web communication relied mainly on very simple authentication mechanisms like HTTP Basic Authentication and session-based login methods. These methods were mostly applied in monolithic type web applications, where the server was holding the session state and was in control of both the client and the backend. The approaches, which were acceptable for very limited contexts, were NOT enough any longer as systems continually evolved and got turned into distributed architectures. Web services and APIs brought in new demands such as supporting automated machine-to-machine communication, being able to scale to very large user bases, and making completely independent systems interoperable.

Today, token-based authentication methods are gaining quite a lot of popularity mainly due to their stateless nature and the mass adoption of REST APIs. At first, API keys most probably were the very first broadly accepted solution with the main advantage of being very simple and easy to use. Nevertheless, subsequent research and industry analysis have unraveled that they were short of several features, such as, absence of fine-grained access control, limited capability of revocation, and susceptibility to leak. Therefore, more advanced frameworks have been developed allowing defining permissions, handling token lifecycles, and enabling the delegation of access.

The emergence of microservices and cloud-native applications has also contributed to signaling the necessity for standardized authentication and authorization solutions. APIs started to be used by various client types, including browsers, mobile apps, and third-party services, each having its own model of trust. Consequently, OAuth2 was adopted as a flexible authorization framework and JSON Web Tokens as a compact, portable token format. Both academic and industry literature draw attention to these mechanisms as the main enablers of scalable and interoperable API security, along with recognizing the new security issues that come with them.

2.2. OAUTH2 FRAMEWORK OVERVIEW FROM EXISTING RESEARCH

OAuth2 has been widely examined as a framework proposal for a unified delegated authorization system. It was initially presented as a next step to OAuth 1.0 where the main goals were to make the authorization procedures less complex and to allow for greater usability and extensibility. The conceptual model of OAuth2 has been illustrated in the literature as a role-based system that includes resource owners, clients, authorization servers, and resource servers. Also, the main focus is on granting the third-party applications access to the user protected resources without the need for the user credentials to be disclosed.

Several pieces of work have brought out the aspect of OAuth2's adaptability manifested through its various forms of grants such as a code of authorization, client credentials, and implicit flows. Such adaptability facilitates OAuth2 to cater to different types of situations regardless of whether the applications are user-facing or are backend services communicating. The truth that OAuth2 maintains a clear separation between authentication and authorization concerns has been identified as a great step forward from the previous approaches where identity has been mixed with access control, by the scholars.

Nevertheless, the papers also underline the fact that OAuth2 was not intended to be an authentication protocol. A number of research works highlight this point by stating that the mixing up of the two has caused misuse at a certain degree, especially when OAuth2 access tokens are considered as a proof of the identity. In order to close this gap, other protocols like OpenID Connect have been suggested and used widely. Studies have highlighted that OAuth2 is an excellent authorization framework but its security is largely dependent on the appropriate choice of the flow, the strict application of validation rules, and the secure management of tokens.

2.3. JWT STRUCTURE, ADOPTION, AND COMMON USE CASES

JSON Web Tokens are widely used nowadays to send claims between parties in a very concise and URL-safe manner. The JWT standard describes a token structure that is made up of three different parts: the header, the payload, and the signature. The header identifies the signing algorithm, the payload carries claims like subject, issuer, and expiration time, and the signature guarantees the token integrity. However, research papers mainly point out the self-contained aspect of JWT as a major advantage that thus allows stateless authentication and authorization without the need for a continuous communication with an identity provider.

JWTs are nowadays one of the most popular methods for conveying access, identity, and session tokens in REST APIs. Their portability and handling simplicity have caused microservices to really fall in love with them, especially in situations where several services have to validate tokens independently of a central authority. Experimental data has shown that JWTs reduce latency and increase scalability since they do not rely on a central session store. Hence, JWTs are typically combined with OAuth2 frameworks or simply used in custom authentication scenarios.

On the contrary, the documentation also reveals that an irresponsible use of JWTs can be very harmful. That JWTs contain all the information inside themselves means, among other things, that it becomes challenging to revoke them if the tokens are very long-lived. The writers have noticed that programmers are frequently using JWTs in wrong ways, for instance, by embedding confidential data in token payloads or by neglecting to set up expiration and rotation processes. Most likely, these revelations serve as a proof that JWTs are extremely potent instruments, however, their security is basically a result of the stringent implementation of techniques and lifecycle management.

2.4. DOCUMENTED OAUTH2 VULNERABILITIES

A large number of research papers report security flaws in OAuth2 implementations. Token misuse is one of the most frequently mentioned problems and it is explained as the exposure of access tokens through unsecure transport, logging mechanisms, or client-side storage. According to the studies, attackers can reuse compromised tokens without being detected, especially in systems that do not have token binding or contextual validation features.

Scope abuse is also a term frequently mentioned when talking about vulnerabilities. Moreover, the research shows that clients are frequently granted overly broad scopes thus violating the principle of least privilege. In these situations, tokens that have been compromised give attackers the ability to access much more than they deserve for a particular operation. Besides, a number of documents show that resource servers perform inadequate or no scope validation at all hence they accept tokens even if the tokens lack proper permissions.

OAuth2 research has primarily focused on redirect URI attacks. Poor redirect URI validation can be exploited by an attacker who can then gain authorization codes or tokens by re-routing users to their malicious endpoints. The studies also indicate that the problem with redirect URI handling still continues to exist and even though the documentation thoroughly specifies it, there are still many misconfigurations. All these security vulnerabilities reveal that the OAuth2 protocol security depends very heavily on the details of implementation and configuration choices.

3. PROPOSED METHODOLOGY

In order to explore OAuth2 and JWT as potential ways for securing REST APIs, this research made up its mind to take a pretty comprehensive and security-centric approach. And not just from the angle of functions or efficiencies, the very first stage of the methodology is security features, real threats, and typical implementation problems in the security field. The framework of the approach is a comparative security analysis that is combined with threat modeling techniques to systematically identify the pros, cons, and trade-offs of each mechanism.

3.1. COMPARATIVE SECURITY ANALYSIS FRAMEWORK

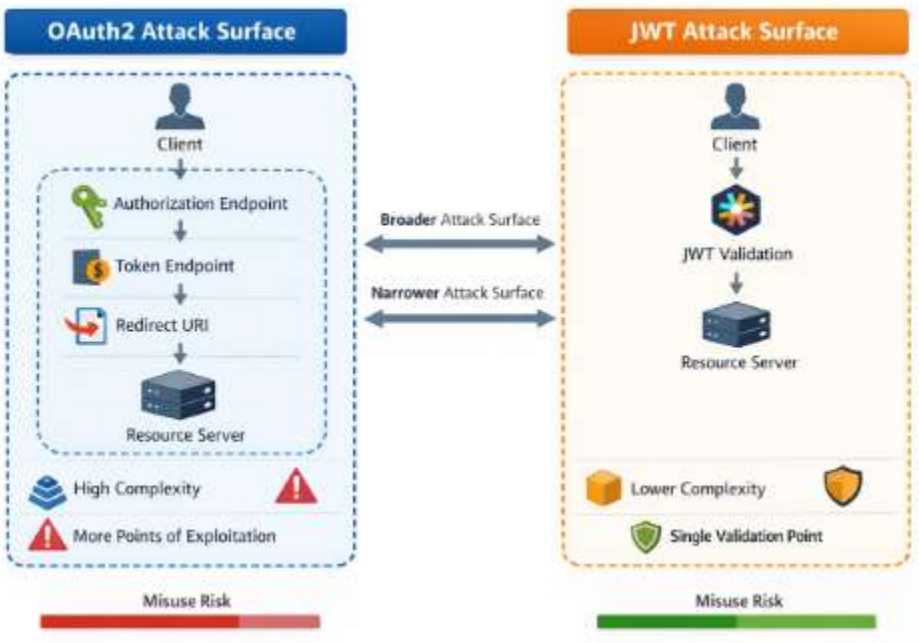

Figure 1. Oauth2 Vs JWT Attack Surface Comparison

One core element of the method that examines different security aspects of OAuth2 and JWT is the comparative analysis framework. The framework treats OAuth2 and JWT not as competitors but as two parts that can be combined or used separately for API security architectures. Besides that, the research also investigates how the two techniques deal with typical attacks and what influence design decisions have on the security of the system.

The implementation of the framework is split mainly into two significant parts. Firstly, the OAuth2 and JWT methods are studied individually regarding their security features, understanding why they are used in the respective contexts, and recognising their possible weaknesses. Then, their functionalities are directly compared to each other to identify the features where one method offers a stronger security or is more exposed to risks. Performing such a systematic comparison greatly aids in gaining a deep understanding of how security choices at the architectural level can influence the security of REST API environments.

3.2. EVALUATION CRITERIA

First of all, the comparative analysis between these two is carried out based on a limited set of evaluation criteria to maintain consistency and depth. The criteria have been chosen from the main themes of API security literature and the common attack vectors perpetrators usually exploit in real-world systems.

- Token Lifecycle Management: Token lifecycle management plays a major role in the security of APIs. After all, tokens are what basically decide who's allowed into stateless systems. This particular point looks at OAuth2 and JWT methods in handling token issuance, expiration, renewal, revocation, and rotation. The capability of OAuth2 in providing short-lived access tokens and the use of refresh tokens is discussed against the background of JWT being self-contained, which makes revocation a less straightforward task. Besides, the paper also highlights the risks of improper token lifecycle management in terms of token theft and replay attacks.
- Confidentiality and Integrity: Confidentiality and integrity aspects are measured based on the degree to which each technology safeguards tokens against unauthorized disclosure and tampering. In the case of OAuth2, this would refer to secure authorization flows, transport-layer security, and the validation of tokens by the server. For JWT, the focus is on cryptographic signing, picking the right algorithm, and verifying the claims.
  Besides that, it also examines the issue of storing sensitive information inside tokens being a potential risk and also the possible negative consequences of having weak or misconfigured cryptographic controls.

- Scalability and Performance: Security trade-offs are only one aspect of the discussion along with the scalability and performance features. Stateless methods including JWT are mostly preferred for performance reasons because they eliminate the need of a session store at a central point. Nevertheless, the resulting more scalable system might be less capable of token revocation and usage tracking. Adding authorization servers in OAuth2 increases the network overhead, but it also makes it possible to enforce security from a more-centralized point. The method evaluates the extent to which the changes in system architecture affect the performance of the system as well as its security.
- Attack Surface Exposure: Attack surface exposure essentially refers to the extent to which different mechanisms can be blamed for raising or lowering an attacker's opportunities. For example, the assessment includes checking the standard attack vectors which attackers may exploit, such as authorization endpoints, token endpoints, redirect URIs, and client-side token storage.The approach recognizes that intricate OAuth2 flows may result in configuration errors, while the straightforwardness of JWT may result in misuse. The criterion illustrates by a simple usage case that simplicity of architecture is not always more secure.

3.3. COMPARATIVE EVALUATION SUMMARY

Table 1. Comparative Evaluation of Oauth2 and Jwt against the Criteria of the Proposed Methodology

Evaluation Criterion	OAuth2	JWT
Token Lifecycle Management	Strong support via short-lived tokens and refresh flow	Limited revocation; relies heavily on expiration
Confidentiality & Integrity	Depends on secure flows and server-side validation	Strong cryptographic integrity if properly validated
Scalability & Performance	Requires authorization server interaction	Highly scalable due to stateless validation
Attack Surface Exposure	Larger surface due to flow complexity	Smaller surface but higher risk of misuse

This methodology is based on a combination of comparative analysis and threat modeling. It provides a systematic and security-oriented approach for the evaluation of OAuth2 and JWT. The approach illustrates the designed functionalities of the mechanisms as well as their real performance in real-life situations, thus preparing the ground for major results and recommendations in the subsequent sections.

4. CASE STUDY

4.1. REAL-WORLD REST API SCENARIO

This case study is about a made-up but believable fintech Software-as-a-Service (SaaS) platform which provides digital payment processing, account management, and transaction analytics through REST APIs. Such a platform serves different kinds of customers, for instance, a web dashboard for end users, mobile apps, and third-party partner services.Since financial data is very sensitive and there are also implications of the law, it is imperative that the company has strong authentication and authorization mechanisms to secure its data and also prevent unauthorized access and fraud.

The platform provides different REST API endpoints that allow you to do various things like getting account balances, looking up transaction history, making payments, and handling administrative tasks. In order to keep the system stateless, hence scalable, these endpoints have been secured using token-based security methods.This case study explores two distinct security scenarios: one is using OAuth2 authorization flows, and the other is a JWT-based authentication system without implementing a full OAuth2 framework. They both are run in the same operational context to measure, in a fair manner, the performances of the setups.

4.2. OAUTH2-BASED API IMPLEMENTATION

Implementing a platform using OAuth2, it primarily relies on a central authorization server for managing the access control. Users log in via the platform's identity provider and then authorization server issues the users access tokens which have a short validity period and it also issues refresh tokens which lasts longer. Access tokens are restricted to the minimal set of API permissions. For instance, with obtaining account data permission alone or making a transaction permission only.

While making API requests, clients include the access token in the Authorization header. The resource server checks each token to make sure the signature, issuer, audience, expiration time, and granted scopes are all correct.Refresh tokens are confidential to the client and serve to obtain new access tokens when the current one expires. A setup of this nature gives the platform the option to cut off a user's access simply by invalidating refresh tokens or altering scope policies at the authorization server.

However, adopting this method implies that the infrastructure needs to be more complex, but at the same time, it is a trade-off for having a centralized control of both access decisions and token lifecycles. Also, the app is able to make detailed authorization decisions and is in compliance with the requirements of least-privilege access rule in the finance sector.

4.3. JWT-BASED API IMPLEMENTATION

The platform opts for a lighter, stateless authentication architecture with its JWT-based implementation. After a user successfully logs in, the system returns a signed JWT with user identity, roles, and permissions info. Being self-contained, the JWT also carries issuer, subject, expiration time, and audience as the standard claims.

The clients include the JWT in the API request header every time, and the resource server validates the token locally by using the shared public key. There is no central authorization server or refresh token mechanism. Instead, tokens have} a pretty long expiry time so that the users are less inconvenienced and the authentication overhead is reduced.

This method makes deployment easier and gives a performance boost as there are no extra network calls left to be made for token validation. Yet, access control is done solely based on claims that were put inside the token at the time of issuance. To reflect changes in user permissions, tokens have to be reissued, and instant revocation is a challenge without token blacklists or some other infrastructure.

4.4. ATTACK SIMULATIONS

Security posture of both implementations was assessed by running a number of attack simulations focusing on common API threat scenarios.

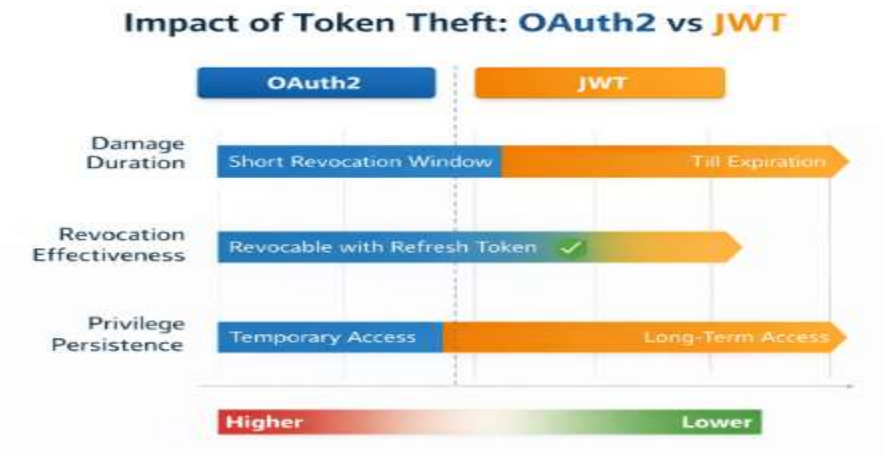

Figure 2. Impact of Token Theft: Oauth2 Vs Jwt

- Token Replay Attack: In the OAuth2 configuration, the attacker was able to replay a stolen access token. But since access tokens last only for a very short time, the window during which the token could be reused was very limited. Moreover, the scope restrictions stopped the attacker from doing high-privilege operations. In contrast, a JWT-based system would allow the attacker to replay the token as long as the token is still valid because the token can be used until the expiration time without any context binding.
- Token Theft: Simulated token theft involved forcibly taking tokens from unsuspecting local client storage. In the OAuth2 implementation, stolen access tokens had a short lifespan, therefore, the potential long-term harm decayed pretty fast. Though refresh tokens were indeed more vulnerable, they enjoyed an extra layer of validation and storage safeguards. In the JWT scenario, if tokens got stolen, the thief could simply enjoy the full level of the access rights for the entire token life, thereby making the impact of the theft much worse.
- Misconfiguration: Misconfiguration scenarios covered such aspects as token claims not being properly validated and permission settings being too permissive. Permissions that were misconfigured in OAuth2 scopes caused overly excessive access which, being only one case scenario, was easier for the central authority to identify and fix. In contrast, in a JWT-based

system, the act of putting too many privileges inside tokens led to continuous over-authorization that lasted until new tokens were created.

- Observed Behavior and Vulnerabilities: The case study showed clear differences in the behaviors of the two approaches. OAuth2 was more resistant to token abuse thanks to its well-defined lifecycle management and centralized control. On the other hand, its complexity led to an increased risk of configuration errors, especially in redirect URI handling and scope definitions. Security based on JWT gave ease and efficiency advantages but the verbatim security risk was increased when tokens were stolen, the victim suffered replay attacks and the victim's permissions fluctuated as the victim's permissions changed.

The results indicate that neither of the two methods is safe or unsafe by nature; rather, the security effects are greatly dependent on the choices made regarding implementation and the knowledge of the threats.

4.5. COMPARATIVE CASE STUDY SUMMARY

Table 2. Comparative Analysis of Oauth2-Based and Jwt-Based Authentication Approaches

Scenario	OAuth2-Based Implementation	JWT-Based Implementation
Token Replay	Limited impact due to short-lived tokens	High impact until token expiration
Token Theft	Reduced exposure; revocation possible via refresh	Full access for token lifetime
Misconfiguration Risk	Higher complexity but centralized correction	Simpler setup but persistent privilege issues

The case study illustrates that the vulnerabilities of OAuth2 as well as JWT-based REST API security can be revealed through real-world attack scenarios. These findings provide additional evidence of the fact that a combination of proper design, flawless execution, and well-informed decision on the security measures according to the system requirements and threat models are necessary.

5. RESULTS AND DISCUSSION

5.1. COMPARATIVE RESULTS SUMMARY

The comparative analysis and case study results reveal that OAuth2 and JWT present different types of security guarantees at a fundamental level when they are used for protecting REST APIs. OAuth2 is good at handling the authorization complexity through the use of flows, centralized control, and managing permissions at a fine level. In contrast, JWT mainly focuses on statelessness, portability, and performance, and it generally sacrifices the dynamic control and revocation for these.

The results show that the security effectiveness of a solution depends not only on a particular mechanism but more on how well its design fits the system's threat model and operational requirements. In all the cases analyzed, OAuth2 could contain the damage of the token abuse through its short-lived access tokens and refresh mechanisms. On the other hand, JWT-based solutions showed better performance characteristics but at the same time a bigger risk when the token is stolen, replayed, and the privileges continue. The outcomes confirm once again the significance of implementation and lifecycle management factors in API security as stressed in the literature."

5.2. SECURITY STRENGTHS AND WEAKNESSES OF OAUTH2

The main security advantage of OAuth2 is that it allows authentication and authorization to be performed separately. In addition, access control is enforced by the use of roles and scopes. Also, when access tokens only last for a short period, the exposure window is limited in case of token compromise, whereas, refresh tokens make it possible for controlled renewal without reauthentication. Besides, there are major benefits of centralized authorization servers such as the ability to immediately implement policy changes, revoke tokens, and monitor requirements, which are critical in regulated industries such as finance and healthcare.

Another advantage identified in the report is that OAuth2 enables the enforcement of the least-privilege policy. By assigning very specific scopes, APIs can limit clients to exactly the resources and operations they need. This will greatly decrease the possible damage if tokens are stolen. Moreover, OAuth2 is highly compatible with other extensions like OpenID Connect, which deal more specifically with authentication needs.

On the other hand, OAuth2 brings with it several significant weaknesses. Its complexity is such that it can very easily lead to errors in the configuration, especially if one doesn't know how to properly deal with things like redirect URI validation, scope design, and

grant type selection. The evidence suggested that incorrect OAuth2 configurations have vulnerabilities similar to those of the simplest methods. Besides that, the need for various elements authorization servers, resource servers, and clients in an OAuth2 system increases exposure to attacks as well as the complexity of managing operations. These vulnerabilities reported by the study are also seen in the literature describing OAuth2 as a very powerful technology that can be very error-prone if developers do not have a deep understanding of security principles.

5.3. SECURITY STRENGTHS AND WEAKNESSES OF JWT

Probably the biggest reason JWT is secure is that they can't be tampered with cryptographically. When done well, JWT signing and very carefully verifying is a guaranteed perfect protection against all token forgery attempts. Since they are self-contained, they allow resource servers to validate tokens locally, that is, without depending on any centralized infrastructure, thus reducing both latency and points of failure. This style of architecture fits microservices environments like a glove as each service will have to authenticate the request itself anyway.

Besides, the straightforwardness of JWT-based authentication also accounts for lesser configuration complexity. When there are fewer, various components, the opportunities for incorrect configurations in relation to authorization flows or token endpoints get reduced drastically. In conditions characterized by fixed access settings and almost no necessity for dynamic revocation, JWTs are capable of providing a viable and streamlined solution.

The paper, however, brings up some significant challenges in the use of JWTs. One of the major challenges is the revocation of tokens, which is still a topic of debate, especially in the case of durable tokens. A JWT, after it is issued, remains valid until it expires even if the user permissions have been changed or the account has been disabled. This is a major risk if the token is stolen. The results of the paper also disclose the very same vulnerabilities that are already known - for example, improper algorithm validation and insufficient claim checks - which can compromise the cryptographic security of JWT.

Besides that, the way tokens are stored also increases the dangers of using JWTs. For example, if JWTs are stored in insecure places on the client-side, they can be easily compromised via cross-site scripting (XSS). On the other hand, loading token claims with a multitude of privileges results in users always having excessive access. Such security issues also serve as the evidence for the research paper which advises against the use of JWTs as a complete security solution but only as a token format.

5.4. PERFORMANCE AND SCALABILITY TRADE-OFFS

Performance and scalability among other things were highlighted as major differences between the OAuth2 and JWT-based solutions. Implementations that were based on JWT showed better scaling performance because of their single-value authorization model. The resource servers could validate the requests without making additional network calls, which means lesser latency and more capability for horizontal scaling. This is why JWTs are considered to be the best option for very busy APIs and systems that are distributed globally.

OAuth2 on the other hand allows the authorization server interactions and token endpoint invocations to add extra load. While this can slow down the system, the research has shown that this load is generally tolerable if one gets in return higher levels of security control and security visibility. Nowadays, caching mechanisms and token introspection can be used to great effect to reduce the performance costs of the operations.

Findings from this research suggest that performance compromises should not be looked at alone. Those systems, which put the highest priority on speed and simplicity, are more likely to go for JWT-based solutions whereas the ones that need strong governance, auditing, and dynamic access control capabilities will benefit more from using OAuth2. This observation is consistent with the literature which points out that security architecture must strike a balance between performance and risk tolerance.

5.5. PRACTICAL IMPLICATIONS FOR DEVELOPERS

The results urge developers and system architects to understand the different roles of OAuth2 and JWT. It is extremely important to think of OAuth2 as an authorization framework, not a token format, and similarly, JWT should be seen more as a token carrying claims than a full-fledged security solution. Most of the insecure implementations originate from mixing up this difference.

Developer's really need to figure out how to set aside time for proper configuration, scope designing, and validation activities when dealing with the implementation of OAuth2. In particular, they might be required to devise very stringent redirect URI policies, shrink token lifetimes to absolute minimums, and validate all token claims. Equally, systems leveraging JWT can be helped with a set of best practices that primarily suggest having short expiry times, making sure the storage is secure, and very thorough signature verification. Besides that, the research highlights that the correct combination of the two mechanisms and the harmonious co-existence is a strong point. Using OAuth2 for authorization and JWT as the format for an access token is a perfectly balanced pair that can work well if done right. By the way, such a mixture should be at the mercy of threat modeling and security requirements, not to the point of convenience.

6. CONCLUSION AND FUTURE SCOPE

6.1. CONCLUSION

This research aimed to analyze the security of REST APIs by comparing OAuth2 and JSON Web Tokens. It explored how both the security architecture and the implementation practices have a direct impact on the security level. The finding reveals that OAuth2 provides more precise control over authorization through carefully designed flows, fast token expiration, central policy enforcement, and very detailed scopes. If these features are properly implemented, they significantly reduce the risks of token theft and misuse. On the flipside, JWT stands out in high performance, scalability, and user-friendliness mainly because of its stateless validation and cryptographic integrity features, thus it fits well with distributed and high-throughput systems. However, the paper also reveals that if token expiry is postponed, there is no token revocation, or secret information is stored in tokens, the use of JWT-based models may open up more security holes. One of the main conclusions is that neither OAuth2 nor JWT are inherently secure or insecure; their security largely depends on how they are properly used, managed during their lifecycle, and whether they fit the system's threat model. Hence, a set of proper REST API security practices should include a clear differentiation between authentication and authorization, token lifetimes as short as possible, thorough checking of token claims, secure client-side storage, and the regular application of the least privilege principle. When integrated in a well-considered manner using OAuth2 as the authorization framework and JWT as a token format these two tools may result in a balanced and strong API security architecture.

6.2. FUTURE SCOPE

By keeping REST API security constantly updated, we can trace and tackle new, increasingly sophisticated methods of a hacking attack. For example, new standards such as OAuth 2.1 are designed to make flows simpler and discard insecure, outdated practices, while techniques such as Demonstration of Proof of Possession (DPoP) and mutual TLS (mTLS) are solving problems of token replay and token binding. Zero Trust security models, which imply continuous authentication, making access decisions based on the context, and minimizing the amount of implicit trust in API ecosystems, are also becoming very popular. Furthermore, there are many options available for an automatic security check such as policy-as-code, continuous token validation testing, as well as the automated detection of misconfigurations. Researchers could look into how such breakthroughs might be merged with OAuth-based and token-based authentication methods to significantly ramp up REST API security in complex, distributed environments of the future.

REFERENCES

[1] ELHejazi, Mahmoud F., and Wisam HA Muragaa. "Improving the Security and Reliability of SDN Controller REST APIs Using JSON Web Token (JWT) with OpenID and auth2. 0." *2024 IEEE 4th International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*. IEEE, 2024.

[2] Talakola, Swetha, and Sai Prasad Veluru. "Managing Authentication in REST Assured OAuth, JWT and More." *International Journal of Emerging Trends in Computer Science and Information Technology* 4.4 (2023): 66-75.

[3] Gbenle, Toluwase Peter, et al. "Applying OAuth2 and JWT Protocols in Securing Distributed API Gateways: Best Practices and Case Review." (2022).

[4] Beshiri, A. R. B. Ë. R., A. N. A. S. T. A. S. Mishev, and I. V. A. N. Chorbev. "Security issues in the RESTful API (service) using OAuth 2.0 for authentication and authorization." *Proc. Int. Conf. Engineering Technologies*. 2021.

[5] Ferry, Eugene, John O Raw, and Kevin Curran. "Security evaluation of the OAuth 2.0 framework." *Information & Computer Security* 23.1 (2015): 73-101.

[6] Alghawli, Abed Saif Ahmed. "Analysis of Authentication Methods and Secure Web Application Realization With an Integrated Authentication System." *Appl. Math* 19.5 (2025): 1027-1038.

[7] Helenius, Marko, and Minna Vallius. "REST API SECURITY: TESTING AND ANALYSIS." (2022).

[8] Kumar, Ritesh. "Enhancing API Security: A Comparative Analysis of OAuth 2.0, OpenID Connect, and SAML."

[9] Jangam, Sandeep Kumar, Nagireddy Karri, and Partha Sarathi Reddy Pedda Muntala. "Advanced API Security Techniques and Service Management." *International Journal of Emerging Research in Engineering and Technology* 3.4 (2022): 63-74.

[10] Siriwardena, Prabath. *Advanced API security: OAuth 2.0 and beyond*. Apress, 2019.

[11] Mendoza Jiménez, Francisco. *Securing a REST API Server*. MS thesis. Universitat Politècnica de Catalunya, 2022.

[12] Visočnik, Vid. *Comparison of JWT and OAuth 2.0 for authorisation and authentication in rest services*. Diss. Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, 2018.

[13] Phanireddy, Sandeep. "Securing RESTful APIs in Microservices Architectures: A Comprehensive Threat Model and Mitigation Framework." *International Journal of Emerging Research in Engineering and Technology* 4.2 (2023): 64-73.

[14] Ali, AzraJabeen Mohamed. "Ensuring Secure Access: Authentication and Authorization in ASP .NET Web API." (2022).

[15] Badhwar, Raj. "Intro to API security-issues and some solutions!." *The CISO's Next Frontier: AI, Post-Quantum Cryptography and Advanced Security Paradigms*. Cham: Springer International Publishing, 2021. 239-244.