Jurnal Minfo Polgan
Volume 13, Nomor 2, Januari 2025
DOI : https://doi.org/10.33395/jmp.v13i2.14406
e-ISSN : 2797-3298
p-ISSN : 2089-9424

Terbit : 01 Januari 2025

Memperkuat Autentikasi dan Integritas Data REST-API Menggunakan Token HMAC SHA-256

Hendra[1*], Awan[2], Waisen[3], Wilianto[4], Yudi[5]
[1,2,3,4]Universitas IBBI, Medan, Indonesia
[1]hendra.soewarno@gmail.com, [2]one.wan@gmail.com, [3]whisen@gmail.com,
[4]wiliantogan@gmail.com, [5]yudifanggawa@gmail.com

ABSTRAK

Keamanan dalam transaksi REST API sering kali terancam oleh serangan man-in-the-middle (MITM), terutama ketika alat uji keamanan dengan kemampuan SSL/TLS interception disalahgunakan oleh pihak tidak bertanggung jawab. Supaya serangan ini tidak mudah terdeteksi dan merupakan serangan berbahaya karena memungkinkan penyusupan ke dalam trafik HTTP S untuk memodifikasi data yang ditularkan antara klien dan server Penelitian ini dirumuskan untuk merancang formulasi pembangkitan token untuk pengauthenticationan dan deteksi perubahan data yang ditularkan antara perjanjian antara klien dan server yang terhubung ke server. Metode yang digunakan di sini adalah hmac sha-256 untuk pembangkitan token permintaan dan token balasan. Berdasarkan hasil uji fungsional dan pengujian keamanan, diskusi formulasi ini adalah efektif dalam pengauthenticationan klien, melindungi data yang bertukar, dan mencegah serangan balasan di dalam lingkungan yang rentan, di mana penyerang dapat mengganggu data pengauthenticationan respons melalui SSL / TLS interception.

Kata Kunci: Keamanan REST API, serangan man-in-the-middle (MITM), intersepsi SSL/TLS, pembuatan token API

ABSTRACT

Security in REST API transactions is often threatened by man-in-the-middle (MITM) attacks, especially when security testing tools with SSL/TLS interception capabilities are misused by malicious parties. The attacks enable HTTPS traffic interception to covertly tamper with data being exchanged between clients and servers. In this work, we aim to construct a token generation formulation that can be used for authentication, detecting data changes between agreements, and avoiding replay attacks. HMAC SHA-256 is used to generate request tokens and response tokens. The functional and security testing results prove that the designed formulation is effective to ensure authentication, data intregity and prevention of replay attacks, even in atack scenarios where the protected communication is exposed to the attacker through a misuse of the SSL/TLS interception.

Keywords: *REST-API Security, Man-in-the-middle-attack (MITM), SSL/TLS interception, API Token Generation*

PENDAHULUAN

Zed Attack Proxy atau biasa disingkat ZAP adalah alat pengujian keamanan aplikasi web yang mengklaim mampu melakukan intersepsi SSL. Artinya, ZAP dapat dieksploitasi untuk meluncurkan serangan MITM terhadap lalu lintas HTTPS. Serangan ini diizinkan melalui memasang sertifikat CA ZAP root CA di sistem operasi pemakai atau browser sehingga semua komunikasi HTTPS melewati ZAP mengambil sertifikat yang sama. Fitur ini kemudian memungkinkan ZAP untuk mendekripsi, menganalisis, memodifikasi, dan mengenkripsi ulang lalu lintas menggunakan CA sertifikat server sasaran tanpa tersetuji oleh klien atau server. Meskipun demikian, serangan jenis ini sama-sama dengan mudah bisa disalahgunakan oleh penyusup yang meretas sistem perangkat lunak sehingga mereka bisa menyerang semua lalu lintas HTTPS.

Dalam banyak kasus, REST API diakses menggunakan HTTPS agar token dan data yang ditukar antara klien dan server tetap aman. Tetapi pendekatan ini tidak mengecualikan potensi penyalahgunaan

Jurnal Minfo Polgan
Volume 13, Nomor 2, Januari 2025 e-ISSN : 2797-3298
DOI : https://doi.org/10.33395/jmp.v13i2.14406 p-ISSN : 2089-9424

software ZAP. Seorang penyerang yang tidak bermoral dan atau terhubung ke jaring internal dapat memanfaatkan fungsi ZAP untuk melakukan serangan MITM, dan akibatnya, ini dapat mengancam kesalahan token dan konfidensial data yang diperdagangkan selama transaksi brokering dengan REST API.

Objektif dari penelitian ini adalah untuk mengimplementasikan sistem otentikasi berbasis token dan mendeteksinya, dan dalam waktu yang sama menolak token rahasia yang tersimpan di klien yang digunakan kembali untuk transaksi REST API. Aplikasi dari penelitian ini adalah meningkatkan keamanan komunikasi antara klien dan server, yang masing-masing dieliminasi dari berbagai jenis intersep bagi SSL saat menggunakan ZAP. Maka dengan sistem ini tidak akan merugikan jika transaksi REST APInya disadap. Selain itu, aplikasi dari penelitian ini terkait dengan perlindungan data yang dipertukarkan di antara klien dan server agar penjelasannya tidak tertukar, dan kemungkinan ancaman dari meta man in the middle banyak menggunakn tools dari ZAP sesuai studi kausalitas pengujian yang dilakukan selama penelitian.

KERANGKA TEORI

Application Programming Interface

Antarmuka pemrograman aplikasi (API) adalah kumpulan aturan dan protokol yang mengatur bagaimana aplikasi berinteraksi satu sama lain secara sederhana, jelas, dan terstandarisasi (Biehl, 2016). API bertindak sebagai perantara yang memungkinkan pertukaran data dan fungsionalitas antar aplikasi tanpa terikat pada teknologi, bahasa pemrograman, atau platform tertentu. Standar yang ditentukan oleh API membantu pengembang perangkat lunak sisi klien berkomunikasi dengan API yang disediakan oleh server.

Salah satu arsitektur API yang paling banyak digunakan adalah Representational State Transfer (REST), yang diperkenalkan oleh Fielding (2000). REST menggunakan metode HTTP untuk operasi seperti GET, POST, PUT, DELETE, dan biasanya mengembalikan data dalam format JSON atau XML. Keuntungan RESTful API adalah kesederhanaan dan ekstensibilitasnya, sehingga memudahkan integrasi antaraplikasi. Di sisi lain, SOAP (Simple Object Access Protocol) adalah protokol yang lebih kompleks yang menggunakan XML untuk pertukaran data dan menyediakan standar yang lebih ketat untuk keamanan dan transaksi (Pautasso, Zimmermann, & Leymann, 2008).

Dalam hal keamanan API, sifat RESTful API yang ringan dan fleksibel membuatnya sering menjadi target serangan jika tidak dilengkapi dengan mekanisme keamanan yang memadai. Misalnya, autentikasi yang kuat, enkripsi data melalui HTTPS, dan perlindungan terhadap serangan pemutaran ulang serta manipulasi data penting untuk mengamankan komunikasi antara klien dan server. Selain itu, REST API sering kali menggunakan token untuk autentikasi, seperti: B.Token Web JSON (JWT) atau OAuth. Memerlukan pengelolaan dan verifikasi yang cermat untuk mencegah penyalahgunaan.

REST API

Representational State Transfer (REST) adalah arsitektur yang dimaksudkan untuk mendukung pengembangan sistem berbasis web melalui penggunaan metode HTTP seperti GET, POST, PUT, dan DELETE. Metode ini memungkinkan Anda melakukan operasi berikut pada sumber daya web: Mendapatkan data (GET), mengirim data (POST), memperbarui data (PUT), dan menghapus data (DELETE) membuat komunikasi antar sistem menjadi lebih efisien dan mudah. REST API menggunakan protokol HTTP sebagai media utama untuk memungkinkan interaksi antara klien dan server (Masse, 2011).

REST pertama kali diperkenalkan pada tahun 2000 oleh Roy Fielding dalam tesis doktoralnya, dimana REST digambarkan sebagai gaya arsitektur berdasarkan protokol HTTP. Salah satu prinsip utama REST adalah sifatnya yang "tanpa kewarganegaraan". Ini berarti bahwa setiap permintaan klien harus berisi semua informasi yang diperlukan, terlepas dari keadaan sebelumnya. Pendekatan ini membuat komunikasi antar sistem menjadi lebih sederhana dan bebas konteks (Fielding, 2000).

Jurnal Minfo Polgan
Volume 13, Nomor 2, Januari 2025 e-ISSN : 2797-3298
DOI : https://doi.org/10.33395/jmp.v13i2.14406 p-ISSN : 2089-9424

Fleksibilitas REST API menyediakan akses ke berbagai platform dan bahasa pemrograman yang mendukung HTTP. Hal ini memungkinkan pengembang untuk membuat aplikasi web, seluler, atau desktop yang dapat berinteraksi dengan REST API yang sama tanpa membuat perubahan besar pada antarmuka API (Masse, 2011). REST juga mendukung interoperabilitas aplikasi yang dibangun menggunakan teknologi berbeda.

Selain fleksibilitas, REST API juga memiliki keunggulan skalabilitas. Pemisahan klien dan server memungkinkan pengembangan sistem yang dapat berkembang seiring waktu tanpa mengkhawatirkan detail pengelolaan data di server (Pauley, 2017). Sifat REST yang ringan dan integrasi yang mudah dengan layanan mikro menjadikannya pilihan populer untuk sistem modern (Nielsen, 2016).

Dalam konteks keamanan REST API, penting untuk menerapkan autentikasi, enkripsi melalui HTTPS, dan keamanan token untuk mencegah serangan seperti man-in-the-middle dan penyalahgunaan API. Dengan mematuhi prinsip REST dan memastikan komunikasi yang aman antar sistem, REST API adalah solusi yang andal dan efisien untuk desain sistem berbasis web modern

HMAC SHA-256

HMAC SHA-256 (Hash-based Message Authentication Code dengan fungsi hash SHA-256) adalah metode kriptografi yang menggabungkan kunci rahasia dan fungsi hash untuk memastikan autentikasi dan integritas data. Teknik ini diperkenalkan oleh Krawczyk, Bellare, dan Canetti (1997) dan dikembangkan untuk menghasilkan kode otentikasi aman yang dapat memverifikasi bahwa pesan yang diterima belum diubah oleh pihak yang tidak berwenang selama transmisi.

Proses HMAC dimulai dengan menggabungkan kunci rahasia dengan data pesan. Kombinasi ini di-hash menggunakan fungsi hash pilihan Anda, dalam hal ini SHA-256. Hasil hash awal kemudian digabungkan kembali dengan kunci pribadi dan fungsi hash diterapkan lagi untuk menghasilkan nilai akhir yang disebut intisari. Intisari ini berfungsi sebagai tanda tangan digital untuk pesan dan dikirimkan bersama pesan untuk memungkinkan penerima memverifikasi keaslian dan integritas pesan (Krawczyk, Bellare, & Canetti, 1997).

HMAC sangat efektif dalam memastikan integritas dan autentikasi data. Penerima pesan kemudian dapat menghitung ulang nilai hash menggunakan data yang sama dan kunci pribadi mereka dan membandingkannya dengan intisari yang mereka terima. Jika kedua nilai cocok, pesan dianggap asli dan belum diubah dengan cara apa pun. Sebaliknya, jika hasil hash tidak cocok, pesan tersebut mungkin telah dirusak atau mungkin berasal dari sumber yang salah (Menezes, van Oorschot, & Vanstone, 1996).

Keamanan HMAC bergantung pada panjang dan keacakan kunci privat serta kekuatan algoritma hashing yang digunakan. SHA-256 memberikan ketahanan tinggi terhadap tabrakan dan serangan pra-gambar sebagai dasar HMAC (NIST, 2015). Selain itu, penggunaan kunci privat meningkatkan ketahanan terhadap manipulasi data dan serangan replay, menjadikan HMAC alat yang ideal untuk protokol keamanan seperti TLS dan IPsec (Paar & Pelzl, 2010).

Dalam konteks autentikasi REST API dan integritas data, Anda dapat membuat token autentikasi menggunakan HMAC SHA-256. Token ini meningkatkan keamanan komunikasi antara klien dan server dengan memastikan bahwa setiap permintaan dan respons API tidak hanya berasal dari sumber tepercaya tetapi juga tetap tidak berubah selama proses pengiriman.

METODE PENELITIAN

Penelitian ini menggunakan pendekatan penelitian pengembangan sistem, yang bertujuan untuk merancang, mengimplementasikan, dan menguji sistem autentikasi dan integritas data berbasis API-Key yang dipadukan dengan HMAC SHA-256. Adapun langkah-langkah dalam penelitian ini adalah:

1. Analisis Kebutuhan Sistem

Pada tahap awal, dilakukan analisis kebutuhan sistem untuk mengidentifikasi masalah yang dihadapi dalam sistem autentikasi API saat ini terkait dengan kemungkinan serangan menggunakan ZAP, dan bagaimana HMAC SHA-256 serta API-Key dapat digunakan untuk meningkatkan keamanan. Penelitian ini akan mengkaji literatur terkait metode autentikasi yang ada dan menilai celah keamanan yang dapat diatasi dengan pendekatan yang diusulkan.

Jurnal Minfo Polgan
Volume 13, Nomor 2, Januari 2025 e-ISSN : 2797-3298
DOI : https://doi.org/10.33395/jmp.v13i2.14406 p-ISSN : 2089-9424

2. Perancangan Sistem

Tahap berikutnya adalah merancang formula pembangkitan token yang terdiri dari beberapa komponen utama:

a. API Key dan API Secret: Merancang pasangan identifikasi yang digunakan untuk autentikasi klien dalam sistem.
b. Request Token Generation: Merancang formulasi pembangkit token dengan HMAC SHA-256 untuk menghasilkan request token dan response token yang mengamankan setiap permintaan yang dikirimkan oleh klien dan tanggapan yang dibalas ke klien.
c. Keamanan Kanal Komunikasi: Merancang penggunaan HTTPS untuk memastikan keamanan dalam transmisi data antara klien dan server.
d. Validasi Token: Merancang prosedur validasi di sisi server dan disisi klien untuk memastikan bahwa data yang diterima tidak dimodifikasi.
e. Pencatatan dan Audit: Mendesain sistem untuk pencatatan transaksi API yang dapat membantu dalam audit dan pencegahan replay attack.

3. Implementasi Sistem

Tahap implementasi melibatkan pembangunan prototipe berdasarkan desain yang telah direncanakan. Implementasi ini akan mencakup pengembangan aplikasi backend dan frontend yang berfungsi untuk menghasilkan request token, memvalidasi token, dan mengelola komunikasi melalui API yang aman.

4. Pengujian Sistem

Pengujian atas prototipe untuk mendapatkan efektifitas dari sistem sesuai dengan tujuan perancangan. Pengujian meliputi uji fungsionalitas yang menguji apakah prototipe hasil rancangan dapat menvalidasi dan memeriksa integritas data selama transmisi, dan uji Keamanan yang menguji apakah prototipe hasil rancangan dapat mendeteksi serangan replay attack.

5. Evaluasi dan Analisis Hasil

Setelah pengujian, hasil dari implementasi dan uji coba sistem dievaluasi. Evaluasi ini akan melibatkan analisis apakah sistem yang dikembangkan berhasil memenuhi tujuan utama, yaitu meningkatkan keamanan dan memastikan integritas data selama transmisi API.

HASIL DAN PEMBAHASAN

Perancangan

a. Akses API

Akses ke API harus dilakukan melalui **Transport Layer Security (TLS)** untuk melindungi proses pengiriman dan penerimaan data. Tanpa penggunaan TLS, data yang dipertukarkan melalui REST API akan terkirim dalam format teks biasa (plaintext), sehingga rentan terhadap penyadapan dan serangan oleh pihak ketiga. Dengan TLS, komunikasi antara klien dan server dienkripsi, memastikan kerahasiaan dan integritas data selama transmisi.

Format URL yang digunakan untuk mengakses API secara aman adalah sebagai berikut:
https://[hostname]:8089/[application]/api/[versioning]/[module]/[operation]

Tabel 1. Komponen URL

Komponen	Nilai	Keterangan
Protocol	https://	Untuk aspek keamanan data yang dipertukarkan adalah wajib menggunakan kanal HTTPS
Hostname	Hostname	Sesuai nama host + domain web server dimana API di hosting.
Application	Hotel	Sesuai dengan nama aplikasi
Versioning	v1.0	Berfungsi sebagai penanda versi API untuk keperluan pengendalian versi pada pengembangan kedepan
Modul	Modul	Berfungsi sebagai nama dari modul layanan yang dapat diakses oleh klien seperti: *vacant*, *booking*, *payment*
Operation	Operation	Berfungsi sebagai akses atas operasi yang tersedia pada modul layanan seperti: *read*, *create*, *update*, *delete*, *cancel*

Jurnal Minfo Polgan
Volume 13, Nomor 2, Januari 2025 e-ISSN : 2797-3298
DOI : https://doi.org/10.33395/jmp.v13i2.14406 p-ISSN : 2089-9424

b. Pembangkitan Token

Komponen utama pembangkitan token adalah API-Key dan API-Secret yang nantinya dikombinasikan dengan komponen lainnya seperti timestamp dan payload pada setiap request ataupun response.

Tabel 2. Kunci akses klien

Key	Contoh	Keterangan
API-Key	key-live:hsW9224mdN01	Identitas masing-masing klien
API-Secret	secret-live:Sdw1974513490	Share secret untuk membangkitkan request dan response *token* pada setiap sesinya

Token akan dibangkitkan melalui dua tahapan sebagai berikut:

1. **Tahapan Pertama: Pembuatan Intermediate Key**
   Pada tahap ini, tiga komponen utama digunakan yaitu API-Key, API-Secret, dan X-Request-Time. X-Request-Time. Ketiga komponen digabung menjadi satu string kemudian diolah dengan menggunakan algoritma SHA-256 untuk menghasilkan sebuah intermediate key sepanjang 32 byte.
2. **Tahapan Kedua: Pembuatan Token**
   Intermediate key yang dihasilkan dari tahap pertama digunakan pada tahap berikutnya bersamaan dengan payload untuk membangkitkan token unik dengan menggunakan algoritma HMAC SHA-256.

Formulasi untuk pembangkitan token adalah:

$$\text{token}=\text{HMAC}_{\text{SHA-256}}(\text{request payload},\text{SHA-256}(\text{API-Key}+\text{API-Secret}+\text{X-Request-Time}))$$

c. Request Header dan Validasi Sisi Server

Komponen X-Request-Time, API-Key, dan Request-Token akan dikirim pada request header. Server perlu melakukan pemeriksaan kadaluarasa atas Request-Token berdasarkan selisih waktu antara X-Request-Time dan timestamp penerimaan di server yang tidak melebihi batas waktu tertentu.

Tabel 3. HTTP *Request Header*

Header key	Contoh Value	Keterangan
Content-Type	application/json	*Request* body menggunakan *format* JSON
Accept	application/json	*Response* menggunakan *format* JSON
X-Request-Time	1705989228.6644	Titik waktu komputer klien dalam mikro detik sejak UNIX EPOCH
API-Key	key-live:hsW9224mdN01	Sesuai dengan API-Key identitas klien yang mengakses API
Request-Token	50f2ccf67fc28191637e159 221d34390ef8d15cccefe81 f6de88a1044fa93feb	Hasil hashing dalam hexadecimal algoritma hash_hmac("sha256", request payload, hash("sha256", API-Key+API-Secret+X-Request-Time))

Server mengambil API-Secret yang tersimpan pada sisi server untuk membangkitkan token validasi dengan formulasi pembangkit token. Jika token validasi sama dengan Request-Token yang diterima, maka dapat dipastikan keabsahan klien dan integritas data yang diterima.

d. Response Header dan Validasi Sisi Klien

Komponen X-Response-Time, API-Key, dan Response-Token akan dikirim pada response header. X-Response-Time menunjukkan waktu server saat membangkitkan respons, API-Key mengidentifikasi klien yang menjadi target response.

Jurnal Minfo Polgan
Volume 13, Nomor 2, Januari 2025
DOI : https://doi.org/10.33395/jmp.v13i2.14406
e-ISSN : 2797-3298
p-ISSN : 2089-9424

Tabel 4. HTTP *Response Header*

Header key	Contoh Value	Keterangan
Content-Type	application/json	*Response* body menggunakan *format* JSON
X-Response-Time	1712732030.6744	Titik waktu komputer server dalam mikro detik sejak UNIX EPOCH
Response-Token	50f2ccf67fc28191637e159221d34390ef8d15cccefe81f6de88a1044fa93feb	Hasil hashing dalam hexadecimal algoritma HASH_HMAC("sha256", response payload, SHA256(API-Key+API-Secret+X-ResponseTime))

Klien mengambil API-Key dan dan API-Secret untuk membangkitkan token validasi dengan formulasi pembangkit token. Jika token validasi sama dengan Request-Token yang diterima, maka dapat dipastikan keabsahan server dan integritas data yang diterima.

e. Request Body

Payload yang dikirim dalam body permintaan (request body) menggunakan format JSON dan dikirim dengan jenis konten (Content-Type) application/json. Untuk metode POST, data dikirim dalam format raw.

Tabel 5. *Request Body*

https://[hostname]/hotel/api/v1.0/vacant/read

```
{
        "JenisKamar":"DELUXE",
        "DariTanggal": "2024-04-10",
        "SampaiTanggal": "2024-04-20"
}
```

https://[hostname]/hotel/api/v1.0/booking/add

```
{
        "JenisId": "KTP",
        "NomorId": "1234567890123456",
        "Nama": "Hendra Soewarno",
        "JenisKamar": "DELUXE",
        "DariTanggal": "2024-04-11",
        "SampaiTanggal": "2024-04-12",
        "NamaTamu": "Hendra Soewarno",
        "ContactNo" : "081xxxxxxxxx"
}
```

f. HTTP Code dan Response Body

Kode status HTTP yang dapat diterima dalam respons API mencakup:
200 OK: Menandakan bahwa permintaan berhasil diproses dan respons yang diinginkan telah dikirim.
404 Not Found: Menandakan bahwa sumber daya yang diminta tidak ditemukan di server.
500 Internal Server Error: Menandakan bahwa terjadi kesalahan di sisi server yang mengakibatkan kegagalan pemrosesan permintaan.

Tabel 6. API Response Status

Field	Nilai Berhasil	Nilai Gagal
status	1	0
message	Ok	Keterangan kegagalan
data	Data	Null

Jurnal Minfo Polgan
Volume 13, Nomor 2, Januari 2025
DOI : https://doi.org/10.33395/jmp.v13i2.14406
e-ISSN : 2797-3298
p-ISSN : 2089-9424

Contoh hasil: https://[hostname]/hotel/api/v1.0/vacant/read

Tabel 7. *Response Body*

Response Keberhasilan	Response Kegagalan	
	Token Kadaluarsa	Token tidak valid

{
"Status": 1,
"Message": "Ok",
"Data": {
"JenisKamar":"DELUXE
",
"AVAILABLE":"5"
}
}
	{
	"Status": 0,
	"Message": "Token
	Expired",
	"Data": null
	}
		HTTP Code 200{
		"Status": 0,
		"Message": "Invalid Request
		Token or Payload",
		"Data": null
		}

Contoh hasil: https://[hostname]/hotel/api/v1.0/booking/add

Tabel 8. *Response Body*

Response Keberhasilan	**Response Kegagalan**	
	Token Kadaluarsa	**Token tidak valid**

HTTP Code 200
{
"Status": 1,
"Message": "Ok",
"Data": {
"BookingId":"12345789"
}
}
	{
	"Status": 0,
	"Message": "Token
	Expired",
	"Data": null
	}
		HTTP Code 200{
		"Status": 0,
		"Message": "Invalid Request
		Token or Payload",
		"Data": null
		}

g. Logging dan Pendeteksian Replay Attack

Setiap permintaan dan jawaban pada server perlu dicatat dalam log. Pencatatan ini bertujuan untuk mendeteksi kemungkinan upaya pemakaian token secara berulang (replay attack), dan juga mendukung tujuan audit dan pembuatan rekam jejak yang komprehensif. Logging membantu dalam pemantauan aktivitas, deteksi anomali, serta analisis keamanan untuk memastikan integritas dan keamanan sistem.

Tabel 9. *Logging fields*

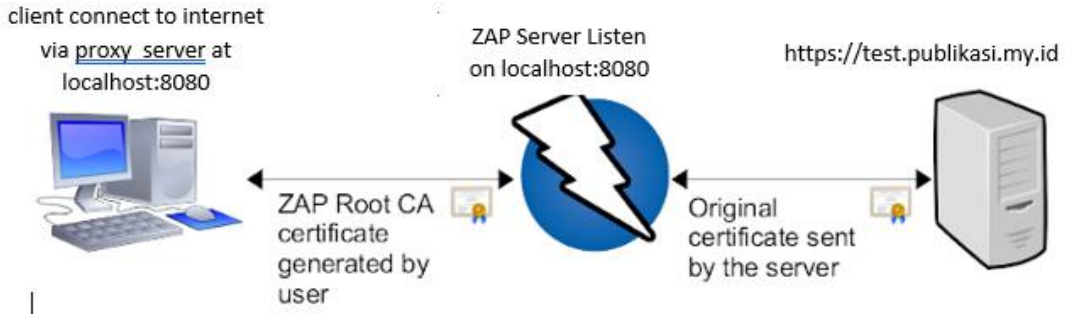
Field	Value	Keterangan
Timestamp	yyyy-mm-dd hh:mm:ss	Timestamp request
X-Request-Time	Mikro detik sejak UNIX EPOCH	Titik waktu request dari komputer klien
URL endpoint	URL API	Berisi URL API yang diakses baik pada
API-Key	API-Key klien	API-Key identitas unik masing-masing klien
Request-Token	Request-Token	Request token unik untuk masing-masing request
Request	JSON request	Request Body
X-Response-Time	Mikro detik sejak UNIX EPOCH	Titik waktu response dari komputer server
Response	JSON response	Response Body
Id	AutoIncrement	Primary Key

API-Key + Request-Token (Unique Key)

Jurnal Minfo Polgan
Volume 13, Nomor 2, Januari 2025 e-ISSN : 2797-3298
DOI : https://doi.org/10.33395/jmp.v13i2.14406 p-ISSN : 2089-9424

Pengujian

Rancangan yang telah dibuat diimplementasikan dalam bentuk dua skrip prototipe, yaitu **sender.php** dan **receiver.php**. Skrip **sender.php** dijalankan pada komputer klien yang menggunakan XAMPP server dan memiliki koneksi internet, sementara **receiver.php** ditempatkan pada web server yang dapat diakses melalui alamat **https://test.publikasi.my.id**. Semua permintaan yang dikirimkan oleh **sender.php** melewati Zed Attack Proxy (ZAP), yang dikonfigurasi untuk memonitor trafik melalui **localhost:8080**, seperti yang ditampilkan pada Gambar 1.



Gambar 1. Lingkungan testing API (sumber: zaproxy.org)

Ketika sender.php dijalankan, pesan kesalahan *"SSL certificate problem: self signed certificate in certificate chain"* muncul. Kesalahan ini terjadi karena sertifikat ZAP Root CA belum ditambahkan ke konfigurasi sistem PHP. Dengan simulasi di mana penyerang memiliki akses ke sistem target, sertifikat ZAP Root CA dapat dimasukkan melalui pengaturan parameter curl.cainfo pada file php.ini. Setelah sertifikat ditambahkan, layanan Apache Web Server perlu di-restart untuk memastikan perubahan diterapkan secara efektif.

Kami menggunakan fitur *"Toggle Breakpoint on All Requests"* pada ZAP untuk mensimulasikan serangan *man-in-the-middle* (MITM) terhadap permintaan HTTPS. Dalam simulasi ini, permintaan dari klien dihentikan, lalu payload dimodifikasi sesuai kebutuhan. Setelah proses modifikasi selesai, tombol **Step** digunakan untuk mengirimkan permintaan yang telah diubah ke server tujuan. Selanjutnya, respons dari server juga dihentikan untuk dilakukan modifikasi payload sebelum diteruskan kembali ke klien. Modifikasi pada respons diselesaikan, dan tombol **Step** kembali digunakan untuk mengirimkan respons yang telah dimodifikasi ke klien. Hasil pengujian tersebut disajikan dalam Tabel 10.

Tabel 10. Hasil Uji Fungsi

Step	Data	Dapat Terbaca	Dapat Diubah	Terdeteksi oleh Receiver.php dan Sender.php
Request	API-Key, X-Request-Time, Request-Token dan Payload	Ya	Ya	Ya (Invalid Token or Payload)
Response	API-Key, X-Response-Time, Response-Token dan Payload	Ya	Ya	Ya (Invalid Token or Payload)

Simulasi pengujian replay attack dilakukan dengan memanfaatkan fitur "Requester" pada ZAP. Dalam pengujian ini, semua data header dan payload dari permintaan sebelumnya diduplikasi secara lengkap. Setelah data tersebut disiapkan, tombol "Send" digunakan untuk mengirimkan ulang permintaan yang sama ke server. Hasil dari pengujian ini disajikan pada Tabel 11.

Jurnal Minfo Polgan
Volume 13, Nomor 2, Januari 2025 e-ISSN : 2797-3298
DOI : https://doi.org/10.33395/jmp.v13i2.14406 p-ISSN : 2089-9424

Tabel 11. Hasil Uji Keamanan

Kanal	Data	Dapat Terbaca	Replay Attack	Eksepsi pada Receiver.php
HTTPS	API-Key, X-Request-Time, Request-Token, Request Body	Ya	Ya	Ya (Token Expired) kalau replay attack diluar bingkai waktu yang ditetapkan Ya (Token reused)

KESIMPULAN

Berdasarkan hasil pengujian fungsional dan keamanan terhadap hasil rancangan yang diimplementasikan dalam bentuk prototipe, dapat disimpulkan bahwa formulasi pembangkitan request token dan response token terbukti efektif dalam mendukung autentikasi klien oleh server, autentikasi server oleh klien, memastikan integritas data yang dipertukarkan, serta mencegah serangan replay attack. Mekanisme ini mampu beroperasi secara andal bahkan dalam lingkungan yang rentan, di mana penyerang memiliki akses ke target melalui penyalahgunaan SSL/TLS interception seperti ZAP.

DAFTAR PUSTAKA

Biehl, M. (2016). *API Design for C++*. Addison-Wesley.

Fielding, R. (2000). *Architectural Styles and the Design of Network-based Software Architectures* [Doctoral dissertation, University of California, Irvine].

Masse, M. (2011). *REST API Design Rulebook: Designing Consistent Web APIs*. O'Reilly Media.

Nielsen, M. (2016). *Microservices Architecture: Make the Architecture Work for You*. Wiley.

Paar, C., & Pelzl, J. (2010). *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer.

Pauley, J. (2017). *RESTful Web Services: A Tutorial*. O'Reilly Media.

Pautasso, C., Zimmermann, O., & Leymann, F. (2008). RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision. *Proceedings of the 17th International Conference on World Wide Web*, 805–814. ACM. https://doi.org/10.1145/1367497.1367606

Pautasso, C., Zimmermann, O., & Leymann, F. (2008). RESTful Web Services vs. Big Web Services: Making the Right Choice. *Proceedings of the International Workshop on Web Services and Formal Methods*, 13–18.

ScienceDirect. (n.d.). Zed Attack Proxy. Retrieved from https://www.sciencedirect.com/topics/computer-science/zed-attack-proxy

Siriwardena, P. (2019). *Advanced API Security: OAuth 2.0 and Beyond*. Apress.

Sullivan, B. (2019). *Network Security Principles and Practices* (2nd ed.). Pearson.