Implementasi Mekanisme Autentikasi API Berbasis HMAC-SHA256 dengan Skema Nonce dan Timestamp untuk Mencegah Replay Attack

Muhammad Dzaki Arta - 13522149
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13522140@std.stei.itb.ac.id, muhammaddzakiarta@gmail.com

***Abstract*—Makalah ini mengusulkan dan mengimplementasikan mekanisme autentikasi REST API berbasis HMAC-SHA256 yang dikombinasikan dengan skema nonce dan timestamp untuk mencegah replay attack. Mekanisme ini dirancang untuk menjamin autentikasi pengirim, integritas pesan, freshness request, serta perlindungan terhadap penggunaan ulang request tanpa menimbulkan overhead komputasi yang signifikan. Sistem terdiri dari modul Request Signer pada sisi client yang membentuk signature dari metode HTTP, path, body request, timestamp, dan nonce, serta modul Request Validator pada sisi server yang melakukan verifikasi signature, validasi timestamp, dan deteksi penggunaan ulang nonce menggunakan cache sementara.**

Implementasi dilakukan menggunakan FastAPI dan Redis, kemudian dievaluasi melalui pengujian fungsional dan pengujian performa. Hasil pengujian fungsional menunjukkan bahwa request yang valid diterima oleh sistem, sedangkan request hasil replay, signature yang dimodifikasi, dan timestamp kadaluarsa berhasil ditolak. Pengujian performa menunjukkan bahwa mekanisme autentikasi menambahkan overhead rata-rata sekitar 0,45 ms per request dibandingkan endpoint tanpa autentikasi, yang masih berada dalam batas yang dapat diterima untuk aplikasi dengan kebutuhan keamanan tinggi. Hasil ini menunjukkan bahwa mekanisme yang diusulkan efektif meningkatkan keamanan REST API terhadap replay attack tanpa menimbulkan degradasi performa yang signifikan.

***Keywords*—HMAC-SHA256, autentikasi API, integritas pesan, nonce, replay attack, REST API, timestamp.**

I. Pendahuluan

Perkembangan web dan aplikasi yang semakin maju membuat REST API sebagai tulang punggung komunikasi antar *client* dan *server*. REST API digunakan untuk mengakses, memodifikasi, dan mengelola data secara terstruktur melalui protokol HTTP, sehingga menjadi komponen kritis dalam arsitektur layanan digital seperti perbankan, e-commerce, dan Internet of Things (IoT). Aspek keamanan komunikasi data pada REST API menjadi sangat penting untuk mencegah penyalahgunaan akses dan manipulasi data oleh pihak yang tidak berwenang.

Salah satu permasalahan keamanan utama pada REST API adalah lemahnya mekanisme autentikasi dan integritas pesan yang berpotensi dimanfaatkan penyerang untuk melakukan serangan seperti impersonasi, *man-in-the-middle*, dan penyadapan. Tanpa skema autentikasi yang kuat, penyerang dapat memalsukan identitas client, memodifikasi payload request, atau memanfaatkan kembali request yang valid untuk mengeksekusi operasi berulang kali di sisi server. Kondisi ini menimbulkan risiko serius, terutama pada API yang menangani transaksi sensitif seperti otorisasi pembayaran, pengelolaan kredensial, atau perubahan data penting.

Replay attack menjadi salah satu ancaman spesifik yang sulit untuk dideteksi oleh sistem yang hanya mengandalkan autentikasi berbasis token statis atau kredensial yang dapat digunakan berulang kali.berbasis token statis atau kredensial yang dapat digunakan berulang kali.

Untuk menjawab permasalahan tersebut, dibutuhkan mekanisme autentikasi yang tidak hanya memverifikasi identitas pengirim dan integritas pesan, tetapi juga mampu mencegah penggunaan ulang request yang sama. Hash-Based Message Authentication Code (HMAC) dengan fungsi hash kriptografis seperti SHA-256 banyak digunakan sebagai solusi ringan dan andal untuk menjamin autentikasi dan integritas pesan pada API, karena sifatnya yang efisien dan tahan terhadap berbagai bentuk serangan pemalsuan pesan. Namun, penggunaan HMAC secara statis tanpa parameter dinamis masih menyisakan celah terhadap replay attack jika nilai autentikasi dapat disalin dan digunakan kembali.

Makalah ini mengusulkan dan mengimplementasikan mekanisme autentikasi API berbasis HMAC-SHA256 yang dikombinasikan dengan skema nonce dan timestamp untuk mencegah replay attack. Fokus utama penelitian adalah merancang format header keamanan, alur pembuatan dan verifikasi signature di sisi client dan server, serta menganalisis kemampuan skema ini dalam mendeteksi dan menolak request yang berusaha di-replay. Diharapkan, hasil implementasi ini dapat menjadi referensi praktis bagi pengembang dalam membangun

REST API yang lebih aman, khususnya pada skenario yang membutuhkan jaminan autentikasi kuat, integritas pesan, dan perlindungan terhadap replay attack.

II. Landasan Teori

A. Autentikasi dan Integritas Pesan

Autentikasi adalah proses verifikasi pengirim pesan untuk memastikan bahwa pesan benar benar berasal dari pengirim.Autentikasi API pada REST API bertujuan untuk memverifikasi bahwa request yang masuk benar benar berasal dari client yang berwenang dan belum di modifikasi selama proses pengiriman [1].

Integritas pesan adalah jaminan bahwa pesan yang diterima sama dengan pesan yang dikirim tanpa memiliki perubahan atau modifikasi dari pihak ketiga. Kombinasi autentikasi dan integritas pesan memastikan bahwa sistem komunikasi API aman dari ancaman impersonasi, tampering, dan *unauthorized modification.*

Autentikasi dan integritas pesan biasanya diwujudkan melalui mekanisme *Message Authentication Code* (MAC) yang melibatkan *shared secret key* di antara *client dan server.* Mekanisme ini memungkinkan verifikasi pesan tanpa memerlukan enkripsi pesan secara keseluruhan, sehingga lebih efisien secara komputasi.

B. *Hash-Based Message Authentication Code* (HMAC)

HMAC merupakan sebuah kode kecil yang berukuran tetap dan dihasilkan dari pesan dan kunci untuk mengotentikasi pengirim dan pemeriksa pesan. HMAC-SHA256 merupakan HMAC dengan menggunakan SHA-256 sebagai *underlying* sebagai *hash function* dengan output 256-bit (32 byte). Keunggulan HMAC adalah efisiensi komputasionalnya yang minimal, fleksibilitas, dan sifat one-way yang menjamin keamanan key [2].

Formula *cryptography* dari HMAC yang akan digunakan adalah sebagai berikut

$$HMAC(K, m) = H((K \oplus opad) \| H((K \oplus ipad) \| m))$$

Dimana:

- $\oplus$ Melambangkan *bitwise exclusive-or operation.*
- $\|$ Melambangkan *concatentaion* operation.
- opad (*outer padding*) dan inpad (*inner padding*) berupa konstanta.
- H melambangkan *hash function.*
- K melambangkan *cryptographic key.*
- m melambangkan pesan orisinil.

Inner padding terdiri dari byte 0x36 yang di ulang beberapa kali sehingga panjangnya sama dengan ukuran blok fungsi hash H. *Outer padding* terdiri dari byte 0x5c yang diulang beberapa kali sehingga panjangnya sama dengan ukuran blok fungsi hash H [3].

C. *Replay Attack*

Replay attack merupakan jenis serangan dimana penyerang melakukan intercept sebuah pesan ataupun request yang valid, kemudian mengirimkannya kembali untuk memicu serangan ilegal ataupun untuk memicu operasi berulang tanpa otorisasi [3].

Penelitian pada keamanan IoT dan API menunjukkan bahwa replay attack dapat melewati mekanisme idempoten standar HTTP jika server tidak menyimpan konteks atau tidak memeriksa identitas pesan secara unik. Contohnya, pengulangan request pembayaran atau perintah kontrol perangkat dapat berhasil dijalankan kembali apabila server hanya memvalidasi signature, tetapi tidak memverifikasi keunikan pesan atau freshness waktu [4].

D. Time-Based Message Authentication Code (TMAC)

Time-Based Message Authentication Code (TMAC) adalah varian MAC yang memasukkan parameter waktu (time counter atau timestamp terkuantisasi) ke dalam proses perhitungan kode autentikasi. Dengan cara ini, nilai MAC hanya berlaku pada interval waktu tertentu, dan menjadi tidak valid begitu waktu bergeser ke periode berikutnya [5].

TMAC memberikan *attack window* yang lebih sempit dibanding HMAC biasa hal ini dikarenakan penyerang hanya dapat memanfaatkan pesan yang dicuri selama interval waktu yang sama. Begitu time counter bertambah, signature yang lama tidak akan bisa dipakai lagi sehingga *replay attack* diluar dari window waktu tersebut akan otomatis ditolak [3].

E. *REST API* dan *API Security*

REST API (*Representational State Transfer Application Programming Interface*) adalah gaya arsitektur layanan web yang memanfaatkan protokol HTTP dan prinsip statelessness untuk menyediakan akses sumber daya melalui operasi seperti GET, POST, PUT, dan DELETE. Sifat *stateless*-nya membuat setiap request membawa seluruh informasi yang dibutuhkan untuk diproses, termasuk kredensial autentikasi dan parameter yang relevan [6].

Keterbukaan REST API dapat menjadi target dari berbagai ancaman keamanan seperti *injection, man-in-the-middle, mass assignment, spoofing,* dan *replay attack.* Laporan dan survei terbaru menunjukkan bahwa kegagalan autentikasi dan otorisasi termasuk ke dalam risiko tertinggi pada API modern, terutama ketika token dan signature tidak dilindungi dengan benar [7].

III. Perancangan Sistem

A. Arsitektur Sistem

Sistem autentikasi yang dirancang beroperasi pada *application layer* dimana setiap permintaan http (request) dari *client* ke *server* akan melalui proses *digital signature* menggunakan HMAC-SHA256. Sistem terdiri dari 2 komponen yaitu:

1. *Request Signer*

Merupakan modul yang berada pada sisi client yang akan berfungsi sebagai menyusun parameter, membuat nonce, mengambil timestamp, dan menghasilkan signature

2. Request Validator
   Merupakan modul middleware yang berada pada sisi server yang akan berfungsi untuk memvalidasi request sebelum diproses oleh controller utama.

Client dan server harus memiliki *secret key* yang sama yang telah dipertukarkan sebelumnya.

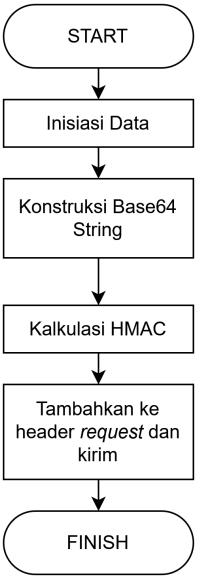
B. Header dan Payload

Sistem akan membuat 3 header yang akan digunakan untuk autentikasi saat mekanisme ini berjalan. Terdapat 3 tambahan format header yang akan dibuat yaitu:

1. X-Auth-Timestamp
   Memiliki nilai *timestamp* saat request tersebut dibuat
2. X-Auth-Nonce
   Merupakan string acak dengan menggunakan modul UUIDv4 yang akan digunakan sebagai nonce.
3. X-Auth-Signature
   Merupakan inti dari mekanisme autentikasi ini yang berupa nilai hasil kalkulasi HMAC-SHA256. Dan hasilnya berupa hexdigest sehingga memudahkan dalam proses komunikasi.

C. Algoritma pembentukan signature

Untuk menjamin integritas data, signature tidak akan dibentuk dari body pesan, tetapi juga mencakup metode HTTP dan URL. Gambar 1 merupakan alur algoritma pembentukan signature yang berada pada sisi client.



Gambar 1. Diagram alir algoritma pembentukan signature

Berikut adalah beberapa tahap dalam pembentukan signature pada sisi client:

1. Inisiasi Data
   Tahap ini akan mengambil method path,serta body yang berada pada request. Nonce serta timestamp akan dibangkitkan pada tahapan ini
2. Konstruktsi Baase64 string
   Pada tahapan ini seluruh data yang nantinya akan digunakan untuk signature akan dilakukan *concatenation* dengan struktur sebagai berikut:

$$BaseString = Method + ":" + Path + ":" +$$ Body + “:” + “Timestamp” + “Nonce”

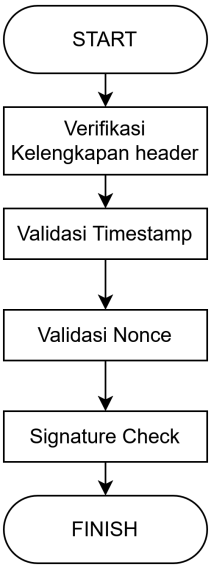
3. Kalkulasi HMAC
   Pada Tahapan ini Signature akan dibuat dengan menggunakan algoritma HMAC-SHA256. Dengan formula sebagai berikut:

$$Signature: HMAC_{SHA256}(SecretKey, BaseString)$$

4. Tambahkan header *request* dan kirim
   Tahapan ini akan menambahkan beberapa header request yang akan digunakan oleh server untuk memverifikasi setiap request yang digunakan dan mengirimkannya ke server.

D. Algoritma Verifikasi dan Pencegahan Replay

Pada sisi server, logika verifikasi dirancang agar request yang diberikan aman terhadap serangan *replay attack.* Gambar 2 merupakan alur dari logika verifikasi signature yang ada pada sisi server.



Gambar 2. Diagram alir proses verifikasi signature

Berikut adalah beberapa tahapan dalam melakukan verifikasi signature pada sisi server:

1. Verifikasi Header
   Akan melakukan verifikasi pada kelengkapan header yang dimana jika salah satu dari header ini X-Auth-Timestamp, X-Auth-Nonce, X-Auth-Signature tidak ada, maka server akan mengembalikan response 400 bad request
2. Validasi Timestamp
   Pada tahapan ini server akan memeriksa apakah

request masih berada pada waktu yang dapat ditoleransi. Formula untuk menentukannya adalah sebagai berikut:
$|ServerTime - ClientTimestamp| \leq Window$

Jika selisih waktu > 60 detik, maka server akan menolak dengan balasan 401 Unauthorized. Hal ini akan mencegah penyerang menggunakan request lama

3. Validasi Nonce
   Pada tahapan ini server akan mengecek apakah nonce berada dalam cache. Jika nonce berada dalam cache maka ini menandakan request tersebut sudah digunakan dalam kurang dari 60 detik sebelumnya dan server akan menolak request tersebut dengan status 401 unauthorized. Jika nonce tidak berada dalam cache ini menandakan request tersebut merupakan nonce baru dan akan disimpan dalam cache dengan waktu kadaluarsa sama dengan 60 detik.
4. Signature Check
   Tahapan ini merupakan inti dari verifikasi signature. Pada tahapan ini server akan menyusun ulang baseString menggunakan parameter yang diterima pada header dan body, lalu menghitung ulang dengan menggunakan algoritma dan key yang sama pada sisi client yaitu HMAC-SHA256. Hasil signature ini akan dibandingkan dengan signature yang berada pada header. Jika signature cocok maka server akan menerima request tersebut dan jika signature tidak cocok maka server akan menolak dengan status 401 Unauthorized.

IV. Implementasi Solusi

A. Lingkungan Implementasi

Implementasi mekanisme autentikasi dilakukan pada lingkungan pengembangan REST API sederhana dengan arsitektur client–server. Tujuan implementasi adalah memverifikasi bahwa skema HMAC-SHA256 dengan nonce dan timestamp dapat dijalankan secara fungsional dan mampu mencegah replay attack.

Lingkungan yang akan digunakan adalah sebagai berikut:

Tabel 1. Komponen dan spesifikasi lingkungan implementasi

Komponen	**Spesifikasi**
Sistem Operasi	Windows 11 + Docker desktop
Bahasa Pemrograman	Python 3.11
Framework WEB	FastAPI
Penyimpanan Nonce	Redis 7 (in-memory)
Protokol	HTTP/1.1
Format Data	JSON

Redis digunakan untuk menyimpan nonce yang telah dipakai dengan waktu kadaluarsa 60 detik sesuai dengan window toleransi timestamp yang telah ditentukan pada bagian perancangan.

B. Implementasi Sisi Client

Pada sisi client diimplementasikan modul Request Signer yang bertanggung jawab untuk membentuk request yang telah diautentikasi sebelum dikirimkan ke server. Alur implementasi client meliput

1. Mengambil HTTP method, path URL, dan body request.
2. Membuat timestamp berupa waktu UNIX.
3. Membuat nonce unik menggunakan UUIDv4.
4. Menyusun base string.
5. Menghitung signature menggunakan HMAC-SHA256.
6. Menambahkan signature ke header HTTP.

Pada implementasi akan dibuat beberapa fungsi diantaranya adalah

1. Fungsi build_base_string

```python
def build_base_string(method, path, body, timestamp, nonce):
    return f"{method}:{path}:{body}:{timestamp}:{nonce}"
```

Gambar 3. Kode fungsi build_base_string

Fungsi ini membentuk representasi string deterministik dari request. Tujuannya adalah memastikan client dan server menghitung signature dari data yang identik. Penyusunan dilakukan secara berurutan dan eksplisit untuk menghindari ambiguities.

2. Fungsi compute_hmac

```python
def compute_hmac(base_string: str) -> str:
    return hmac.new(SECRET_KEY, base_string.encode
        (), hashlib.sha256).hexdigest()
```

Gambar 4. Kode fungsi compute_hmac

Fungsi ini menghasilkan HMAC menggunakan SHA-256 sebagai hash function. Output berupa hexdigest agar mudah ditransmisikan melalui HTTP header.

3. Fungsi send_request

```python
def send_request(data, reuse_headers=None):
    body = json.dumps(data)
    method = "POST"
    path = "/api/data"

    if reuse_headers:
        headers = reuse_headers
    else:
        timestamp = int(time.time())
        nonce = str(uuid.uuid4())
        base_string = build_base_string(method,
            path, body, timestamp, nonce)
        signature = compute_hmac(base_string)

        headers = {
            "X-Auth-Timestamp": str(timestamp),
            "X-Auth-Nonce": nonce,
            "X-Auth-Signature": signature,
            "Content-Type": "application/json",
        }

    response = requests.post(SERVER_URL, headers
        =headers, data=body)
    print("Status:", response.status_code, response
        .text)
    return headers
```

Gambar 5. Kode fungsi send_request

Fungsi send_request bertugas membentuk dan mengirimkan HTTP request dari sisi client ke server dengan atau tanpa mekanisme autentikasi. Payload data terlebih dahulu dikonversi ke format JSON agar sesuai dengan format body HTTP. Jika parameter reuse_headers tidak diberikan, fungsi akan membangkitkan timestamp menggunakan waktu sistem saat ini dan nonce sebagai nilai acak unik menggunakan UUIDv4. Selanjutnya fungsi menyusun base string dari method HTTP, path URL, body request, timestamp, dan nonce, lalu menghitung nilai signature menggunakan algoritma HMAC-SHA256 dengan shared secret key. Nilai timestamp, nonce, dan signature kemudian ditambahkan ke dalam header HTTP sebagai metadata autentikasi sebelum request dikirimkan ke server.

Apabila parameter reuse_headers diberikan, fungsi akan menggunakan kembali header yang lama tanpa membangkitkan timestamp, nonce, dan signature baru. Mekanisme ini digunakan untuk mensimulasikan skenario pengujian seperti replay attack, manipulasi signature, atau penggunaan timestamp kadaluarsa. Setelah request dikirimkan menggunakan metode HTTP POST, fungsi mencetak status response dari server dan mengembalikan header yang digunakan agar dapat dipakai kembali pada skenario pengujian berikutnya. Dengan demikian, fungsi ini berperan sebagai modul Request Signer sekaligus alat bantu pengujian keamanan dari sisi client.

C Implementasi Sisi Server

Pada sisi server diimplementasikan middleware Request Validator yang berfungsi sebagai lapisan keamanan sebelum request diteruskan ke endpoint utama. Tahapan validasinya adalah sebagai berikut:

1. Validasi keberadaan header autentikasi.
2. Validasi timestamp.
3. Validasi nonce menggunakan Redis.
4. Rekonstruksi base string.
5. Verifikasi signature.

Berikut adalah implementasi kode yang digunakan pada sisi server

1. Middleware Autentikasi

```python
@app.middleware("http")
async def auth_middleware(request: Request,
    call_next):
```

Gambar 6. Middleware autentikasi

Middleware dipanggil sebelum request diproses endpoint sehingga dapat mencegah request tidak sah mencapai logika aplikasi.

2. Validasi Timestamp

```python
if abs(int(time.time()) - ts) > TIME_WINDOW:
    return JSONResponse(status_code=401, content
        ={"detail": "Timestamp expired"})
```

Gambar 7. Logika validasi *timestamp*

Kode ini berfungsi untuk menolak request yang sudah terlalu lama atau terlalu jauh dari waktu server. Dengan TIME_WINDOW memiliki nilai 60 detik.

3. Validasi Nonce

```python
if r.exists(nonce):
    return JSONResponse(status_code=401, content
        ={"detail": "Replay detected"})
r.set(nonce, "1", ex=TIME_WINDOW)
```

Gambar 8. Logika pengecekan nonce pada redis

Nonce disimpan sementara agar request yang sama tidak dapat diproses ulang. ‘r’ merupakan redis instance yang digunakan untuk menyimpan nonce sementara dengan waktu expirasinya adalah 30 detik.

4. Verifikasi Signature

```python
base = build_base_string(request.method, request
    .url.path, body, ts, nonce)
if not hmac.compare_digest(compute_hmac(base), sig
    ):
    return JSONResponse(status_code=401, content
        ={"detail": "Invalid signature"})
```

Gambar 9. Logika verifikasi signature

Signature dihitung ulang dan dibandingkan secara constant-time untuk mencegah timing attack. Hal ini merupakan inti serta pertahanan terakhir dari implementasi ini.

V. Pengujian dan Analisis Hasil

A. Pengujian

Pengujian dilakukan untuk mengevaluasi kebenaran fungsional, ketahanan terhadap replay attack, serta dampak performa dari mekanisme autentikasi yang diusulkan. Pengujian dibagi menjadi dua kategori utama, yaitu pengujian fungsional dan pengujian performa.

1. Pengujian Fungsional

Pengujian fungsional bertujuan untuk memverifikasi bahwa sistem dapat:

1. Menerima request yang valid.
2. Menolak request yang tidak sah atau dimodifikasi.
3. Mendeteksi dan menolak replay attack.
4. Menolak request dengan timestamp yang sudah kadaluarsa.

Pengujian dilakukan menggunakan skrip client yang mengirimkan beberapa jenis request berikut:

- Request normal dengan signature, nonce, dan timestamp valid.
- Request replay dengan mengirim ulang header yang sama.
- Request dengan signature yang dimodifikasi.
- Request dengan timestamp yang berada di luar window toleransi waktu.

2. Pengujian Performa

Pengujian performa bertujuan untuk mengukur overhead yang ditimbulkan oleh mekanisme autentikasi. Pengujian dilakukan dengan membandingkan dua endpoint:

- /api/plain, endpoint tanpa mekanisme autentikasi.
- /api/data, endpoint dengan mekanisme autentikasi HMAC-SHA256, nonce, dan timestamp.

Pengujian dilakukan menggunakan skrip load_test.py dengan konfigurasi:

- Jumlah thread: 10
- Jumlah request per thread: 100
- Total request: 1000

Setiap skenario diuji secara terpisah untuk menghindari interferensi antar pengujian.

B. Analisis Hasil

1. Hasil pengujian fungsional

Hasil pengujian fungsional dapat dilihat sebagai berikut:

```
=== Normal request ===
Status: 200 {"status":"ok","data":{"value":123}}

=== Replay attack (reuse same headers) ===
Status: 401 {"detail":"Replay detected"}

=== Tampered signature ===
Status: 401 {"detail":"Invalid signature"}

=== Expired timestamp ===
Status: 401 {"detail":"Timestamp expired"}
```

Gambar 10. Hasil pengujian fungsional

Hasil pengujian menunjukkan bahwa request dengan header autentikasi yang valid berhasil diproses oleh server dan menghasilkan status HTTP 200. Hal ini menandakan bahwa mekanisme pembentukan dan verifikasi signature berbasis HMAC-SHA256 antara client dan server berjalan secara konsisten, serta sistem mampu mengenali request yang sah tanpa mengganggu fungsionalitas normal API.

Pengujian replay attack menunjukkan bahwa request yang dikirim ulang dengan header yang sama ditolak oleh server dengan status HTTP 401. Hal ini membuktikan bahwa mekanisme penyimpanan dan pengecekan nonce pada sisi server efektif dalam mendeteksi penggunaan ulang request dalam window waktu yang sama, sehingga sistem mampu mencegah eksekusi ulang operasi yang seharusnya hanya dijalankan satu kali.

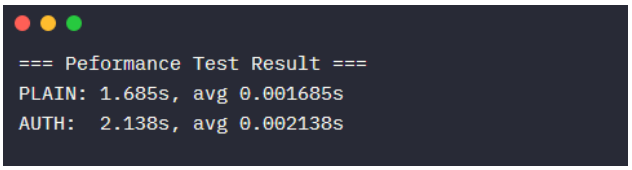
Pengujian terhadap signature yang dimodifikasi menghasilkan penolakan dengan status HTTP 401, yang menunjukkan bahwa sistem mampu mendeteksi perubahan pada komponen request yang dilindungi oleh signature. Hasil ini membuktikan bahwa mekanisme HMAC-SHA256 efektif dalam menjaga integritas pesan dan mencegah manipulasi data selama proses transmisi.

Pengujian terhadap timestamp yang berada di luar window toleransi menunjukkan bahwa request tersebut ditolak oleh server dengan status HTTP 401. Hal ini menunjukkan bahwa penggunaan timestamp berhasil menjamin freshness request dan membatasi attack window bagi penyerang, sehingga request lama tidak dapat digunakan kembali di luar interval waktu yang diizinkan.

Secara keseluruhan, hasil pengujian fungsional menunjukkan bahwa mekanisme autentikasi yang diimplementasikan berhasil memenuhi tujuan utama sistem, yaitu menjamin autentikasi, menjaga integritas pesan, memastikan freshness request, serta mencegah replay attack secara efektif.

2. Hasil Pengujian Performa

Hasil pengujian performa adalah sebagai berikut:



Gambar 11. Hasil pengujian performa

Gambar 11 menunjukan hasil eksekusi kode dalam melakukan performance test. Overhead autentikasi dapat dihitung sebagai:

$$Overhead = 2.138 - 1.685 = 0.453\ ms\ per\ request$$

Hasil pengujian performa menunjukkan bahwa endpoint tanpa mekanisme autentikasi (/api/plain) memiliki waktu eksekusi total sebesar 1.685 detik untuk 1000 request, dengan rata-rata latency sebesar 1.685 ms per request. Sementara itu, endpoint dengan mekanisme autentikasi berbasis HMAC-SHA256 dengan skema nonce dan timestamp (/api/data) memiliki waktu eksekusi total sebesar 2.138 detik untuk jumlah request yang sama, dengan rata-rata latency sebesar 2.138 ms per request. Perbedaan ini menunjukkan adanya overhead performa akibat penerapan mekanisme keamanan.

Overhead yang ditimbulkan oleh mekanisme autentikasi dapat dihitung sebagai selisih rata-rata latency antara kedua endpoint, yaitu sebesar sekitar 0.453 ms per request. Overhead ini berasal dari beberapa operasi tambahan pada sisi server, antara lain perhitungan HMAC-SHA256, validasi timestamp, serta akses ke Redis untuk pengecekan dan penyimpanan nonce. Meskipun demikian, nilai overhead tersebut relatif kecil dibandingkan total waktu pemrosesan request dan masih berada dalam batas yang dapat diterima untuk aplikasi yang membutuhkan tingkat keamanan tinggi.

Dengan demikian, hasil pengujian performa menunjukkan bahwa mekanisme autentikasi yang diusulkan tidak menimbulkan degradasi performa yang signifikan. Sistem tetap mampu menangani request dengan latency rendah sambil tetap memberikan peningkatan keamanan berupa autentikasi kuat, integritas pesan, freshness, dan perlindungan terhadap replay attack.

V. Kesimpulan

Berdasarkan hasil pengujian yang telah dilakukan, mekanisme autentikasi berbasis HMAC-SHA256 dengan skema nonce dan timestamp terbukti mampu meningkatkan keamanan REST API secara efektif. Pengujian fungsional menunjukkan bahwa sistem berhasil menerima request yang valid serta menolak request hasil replay, request dengan signature yang dimodifikasi, dan request dengan timestamp kadaluarsa. Hal ini menunjukkan bahwa mekanisme yang diimplementasikan mampu menjamin autentikasi, menjaga integritas pesan, memastikan freshness request, dan mencegah replay attack sesuai dengan tujuan perancangan sistem.

Hasil pengujian performa menunjukkan bahwa penerapan mekanisme autentikasi menambahkan overhead waktu sekitar 0,45 ms per request dibandingkan endpoint tanpa autentikasi. Overhead ini relatif kecil dan masih berada dalam batas yang dapat diterima, terutama jika dibandingkan dengan peningkatan keamanan yang diperoleh. Dengan demikian, mekanisme yang diusulkan dapat diterapkan pada sistem REST API yang menangani data sensitif tanpa menimbulkan degradasi performa yang signifikan.

Secara keseluruhan, hasil pengujian membuktikan bahwa mekanisme autentikasi yang diusulkan bersifat efektif, efisien, dan layak diterapkan sebagai solusi praktis untuk melindungi REST API dari replay attack dan manipulasi request, khususnya pada sistem yang membutuhkan jaminan keamanan tinggi.

VI. Ucapan Terima Kasih

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga makalah ini dapat disusun dan diselesaikan dengan baik. Makalah ini disusun sebagai salah satu bentuk pemenuhan tugas pada mata kuliah IF4020 Kriptografi.

Pada kesempatan ini, penulis mengucapkan terima kasih yang sebesar-besarnya kepada Bapak Dr. Ir. Rinaldi Munir, MT selaku dosen pengampu mata kuliah IF4020 Kriptografi atas ilmu, arahan, serta bimbingan yang telah diberikan selama proses perkuliahan berlangsung.

Penulis menyadari bahwa makalah ini masih memiliki berbagai keterbatasan dan kekurangan. Oleh karena itu, kritik dan saran yang bersifat membangun sangat diharapkan sebagai bahan perbaikan dan penyempurnaan karya tulis ini di masa yang akan datang.

Referensi

[1] S. Dalimunthe, E. Hasri Putra, and M. A. Fadhly Ridha, “Restful API Security Using JSON Web Token (JWT) With HMAC-Sha512 Algorithm in Session Management,” *IT J. Res. Dev.*, vol. 8, no. 1, pp. 81–94, Dec. 2023, doi: 10.25299/itjrd.2023.12029.

[2] R. Munir, “Message Authentication Code.” [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi-dan-Koding/2023-2024/17-MAC-2024.pdf

[3] B. Gupta, “A replay-attack resistant message authentication scheme using time-based keying hash functions and unique message identifiers,” Feb. 05, 2016, *arXiv*: arXiv:1602.02148. doi: 10.48550/arXiv.1602.02148.

[4] R. Sinaga, S. Samsinar, S. Fatima, and F. Frangky, “ANALYSIS OF SECURITY CHALLENGES IN REST API IN EDGE COMPUTING-BASED IOT ECOSYSTEM: A REVIEW,” *JIKO J. Inform. Dan Komput.*, vol. 8, no. 2, pp. 141–147, July 2025, doi: 10.33387/jiko.v8i2.10097.

[5] Z. Zhai, J. Qian, Y. Tao, L. Zhao, and B. Cheng, “A Lightweight Timestamp-based MAC Detection Scheme for XOR Network Coding in Wireless Sensor Networks,” in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, New Delhi India: ACM, Oct. 2018, pp. 735–737. doi: 10.1145/3241539.3267766.

[6] S. Dalimunthe, E. Hasri Putra, and M. A. Fadhly Ridha, “Restful API Security Using JSON Web Token (JWT) With HMAC-Sha512 Algorithm in Session Management,” *IT J. Res. Dev.*, vol. 8, no. 1, pp. 81–94, Dec. 2023, doi: 10.25299/itjrd.2023.12029.

[7] Z. Mousavi, C. Islam, M. A. Babar, A. Abuadbba, and K. Moore, “Detecting Misuse of Security APIs: A Systematic Review,” May

14, 2025, *arXiv*: arXiv:2306.08869. doi: 10.48550/arXiv.2306.08869.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

Muhammad Dzaki Arta
13522149