Metode autentifikacije IoT uređaja u komunikaciji sa serverom: Implementacija i komparativna analiza

Stefan Antić, dr Nemanja Radosavljević

***Sadržaj* — Rad se sastoji od nekoliko ključnih segmenata koji istražuju teorijske i praktične aspekte autentifikacije IoT uređaja. U uvodnom delu predstavljeni su osnovni koncepti Interneta stvari (IoT), značaj autentifikacije i komunikacije putem HTTP protokola. Centralni deo rada analizira osam metoda autentifikacije, uključujući MAC adresu, osnovnu HTTP autentifikaciju, API ključeve, HMAC, JWT, OAuth 2.0 i TLS sertifikate. Svaka metoda je detaljno obrađena sa aspekta prednosti, mana i konkretne implementacije za klijentske (IoT uređaje) i serverske strane uz korišćenje JavaScript-a i PHP-a (Laravel). Rad analizira i nudi predloge za dalji razvoj u okviru savremenih trendova kao što su kvantna kriptografija, decentralizovane identifikacije (DID), veštačka inteligencija i energetski efikasni algoritmi. U zaključnom delu, rad pruža praktične smernice za izbor odgovarajuće metode autentifikacije i osvetljava mogućnosti daljeg razvoja u oblasti sigurnosti IoT ekosistema.**

***Ključne reči* — API ključ, decentralizovane identifikacije, energetska efikasnost, HTTP, IoT, JWT, kvantna kriptografija, MAC adresa, metode autentifikacije, OAuth 2.0, TLS sertifikati, veštačka inteligencija.**

I. UVOD

UREĐAJI U IOT ekosistemu mogu biti vrlo različiti – od jednostavnih senzora za merenje temperature i vlažnosti, preko pametnih termostata, do naprednih industrijskih robota. Ovi uređaji često imaju ograničene hardverske resurse, uključujući procesorsku snagu, memoriju i energetsku efikasnost, što

[1]

S. A. Autor, Računarski fakultet, Srbija (e-mail: stefanantiic7@gmail.com).
N. R. Autor, Računarski fakultet, Srbija (e-mail: nradosavljevic@raf.rs).

predstavlja izazov za implementaciju kompleksnih bezbednosnih protokola.

Jedan od osnovnih načina komunikacije IoT uređaja sa centralnim serverom jeste HTTP protokol. HTTP (HyperText transfer protocol) je protokol koji se koristi za razmenu informacija između klijenta (u ovom slučaju IoT uređaja) i servera. Iako je HTTP tradicionalno razvijen za razmenu podataka između pretraživača i veb servera, njegova jednostavnost, fleksibilnost i široka podrška u različitim aplikacijama i uređajima čine ga pogodnim za IoT okruženje. IoT uređaji koriste HTTP za slanje podataka ka centralnom serveru gde se ti podaci analiziraju, obrađuju i skladište.

Međutim, komunikacija putem HTTP-a otvara niz bezbednosnih pitanja, posebno u kontekstu autentifikacije IoT uređaja. IoT uređaji često prenose osetljive podatke, kao što su podaci o zdravstvenom stanju, lokaciji, ili proizvodnim procesima, zbog čega je važno osigurati da samo autorizovani uređaji imaju pristup serveru i da se podaci ne kompromituju. Autentifikacija je prvi korak u osiguravanju poverljivosti i integriteta podataka, jer omogućava serveru da potvrdi identitet uređaja koji pokušava da uspostavi komunikaciju. U ovom radu će biti analizirane različite metode autentifikacije koje se mogu primeniti u komunikaciji IoT uređaja sa serverom putem HTTP protokola.

II. METODE AUTENTIFIKACIJE IOT UREĐAJA

Autentifikacija IoT uređaja predstavlja ključni aspekt bezbednosti u savremenim informacionim sistemima. Ona podrazumeva postupak verifikacije identiteta uređaja pre uspostavljanja komunikacije sa serverom ili drugim komponentama mreže. U kontekstu IoT, gde veliki broj uređaja razmenjuje osetljive podatke preko različitih mreža, autentifikacija služi kao prva linija odbrane od neovlašćenog pristupa i potencijalnih napada. U ovom poglavlju biće analizirane različite metode autentifikacije, sa ciljem da se identifikuju njihove primene, ograničenja i potencijal za unapređenje sigurnosti u IoT okruženjima.

A. Autentifikacija zasnovana na MAC adresi

Autentifikacija zasnovana na MAC adresi je tehnika koja koristi jedinstvenu MAC (Media Access Controll) adresu uređaja za kontrolu pristupa mreži. MAC adrese, koje su hardverski identifikatori dodeljeni mrežnim interfejsima (mrežnim karticama), mogu da pomognu u daljoj proveri da li je neki uređaj ovlašćen za pristup mreži ili određenom servisu. Međutim, iako je ovaj metod jednostavan i ima specifične namene, on ima značajna bezbednosna ograničenja.

```php
public function handle(Request $request, Closure $next)
{
    // Get MAC address from request headers
    $macAddress = $request->header( key: 'Device-MAC');

    // Get the list of approved MAC addresses from config
    $approvedMacAddresses = [
        'AA:BB:CC:DD:EE:FF', // Device 1
        '11:22:33:44:55:66', // Device 2
        // ...
    ];

    // Check if the MAC address is in the approved list
    if (!in_array($macAddress, $approvedMacAddresses)) {
        return response()->json(['error' => 'Unauthorized device'], status: 403);
    }

    return $next($request);
}
```

Sl. 1. primer implementacije autentifikacije zasnovane na MAC adresi

Iz navedenog koda vidimo da server od klijenta očekuje ”Device-MAC” parametar u zaglavlju zahteva. Ukoliko je poslata vrednost jedna od poznatih adresa, zahtev se šalje dalje na obradu, u suprotnom se klijentu vraća odgovor o neovlašćenom pristupu.

Potvrda autentičnosti zasnovana na MAC adresi je jednostavna za podešavanje i zahteva minimalnu konfiguraciju i na uređaju i na serveru. Ovaj pristup funkcioniše na većini uređaja, iz razloga što svi oni imaju pristup internetu uz pomoć mrežnog interfejsa koji na svom hardverskom nivou poseduje jedinstvenu MAC adresu.

U slučaju da se napadač nađe na istoj lokalnoj mreži kao i uređaj sa ovlašćenim pristupom, napadač relativno lako može saznati MAC adresu žrtve korišćenjem nekog od alata za posmatranje komunikacije na mreži. Nakon toga, neovčašćenom korisniku omogućeno je da zaobiđe ovaj metod, oponašanjem MAC adrese sa odobrenim pristupom.

Autentifikacija zasnovana na MAC adresi je jednostavna i obezbeđuje minimalnu kontrolu pristupa na nivou uređaja, ali njene bezbednosne slabosti, kao što je podložnost lažiranju, ograničavaju njenu efikasnost. Najbolje se koristi u kombinaciji sa drugim merama autentifikacije u okruženjima gde su jednostavnost i lakoća upravljanja važniji od visoke bezbednosti.

B. Osnovna autentifikacija sa HTTP zaglavljem (korisničko ime i lozinka)

Ovaj tip predstavlja jednostavan metod autentifikacije koji se obično koristi u sistemima zasnovanim isključivo na HTTP protokolu. Omogućava klijentu da se autentifikuje na serveru uključivanjem korisničkog imena i lozinke u zaglavlje svakog HTTP zahteva. Uzevši to u obzir, ova metoda zahteva da se svaki uređaj posmatra kao zaseban korisnik sa svojim parom kredencijala.

Cilj je da se kredencijali pre samog slanja kodiraju, ne enkriptuju, i tako kodirani šalju u sklopu HTTP zahteva. Ovo kodiranje se vrši pomoću Base64 enkodovanja, koji transformiše korisničko ime i lozinku u ASCII tekst.

Postavlja se pitanje, na koji način čuvati kredencijale na samom uređaju? Kako izbeći da ukoliko neovlašćeno lice dođe u posed uređaja ne dođe u posed i samih kredencijala tog uređaja? Na našem primeru, koji se bazira na Android uređaju, oslonićemo se na sistemsku funkcionalnost za skladištenje osetljivih podataka - Android Keystore. Biblioteka koja nam ovo omogućava je Expo SecureStore. Ova biblioteka pruža način za šifrovanje i bezbedno skladištenje parova ključ/vrednost lokalno na samom uređaju. Ovaj sistem bezbedno skladišti podatke, čineći ih dostupnim samo aplikaciji koja ih je kreirala. Podaci sačuvani pomoću SecureStore-a su šifrovani, a sistem obezbeđuje pristup tim podacima korišćenjem različitih metoda zaštite samog uređaja, kao što su PIN, lozinka, prepoznavanje lica ili biometrijska brava.

```
async function storeCredentials(username, password) : Promise<void> {
    await SecureStore.setItemAsync('username', username);
    await SecureStore.setItemAsync('password', password);
}
```

Sl. 2. Funkcija za bezbedno čuvanje kredencijala

```
async function storeCredentials(username, password) : Promise<void> {
    await SecureStore.setItemAsync('username', username);
    await SecureStore.setItemAsync('password', password);
}

const username : string = await SecureStore.getItemAsync( key: 'username');
const password : string = await SecureStore.getItemAsync( key: 'password');
// Concatenate username and password with a colon
const credentials : string = `${username}:${password}`;
// Encode the credentials in Base64
const base64Credentials : string = btoa(credentials);
// Add the encoded credentials to the Authorization header
const headers : {Authorization: string} = {
    "Authorization": `Basic ${base64Credentials}`
};
const response : Response = await fetch( input: "https://cloudcities.co/api/sensor-events", init: {
    method: "GET",
    headers: headers
})
```

Sl. 3. Pristup zaštićenim kredencijalima i slanje u okviru HTTP zahteva

Treba uzeti u obzir način slanja ovih kredencijala. Base64 je vrsta kodiranja koja se lako dekodira. Ako veza nije zaštićena HTTPS protokolom, kredencijali uređaja su izloženi presretanju, što ovaj metod čini ranjivim na napade kao što je “man in the middle”.

```
public function handle(Request $request, Closure $next)
{
    // Retrieve the Authorization header
    $header = $request->header( key: 'Authorization');

    // Check if the header is present and uses Basic Authentication
    if (!$header || !str_starts_with($header, 'Basic ')) {
        return Response::json(['error' => 'Unauthorized'], status: 401);
    }

    // Decode credentials (e.g., "Basic dXNlcjpwYXNzd29yZA==")
    $encodedCredentials = substr($header, offset: 6);
    $decodedCredentials = base64_decode($encodedCredentials);

    // Split the decoded string into username and password
    [$username, $password] = explode( separator: ':', $decodedCredentials, limit: 2);

    // Look up the user in the database by username
    $user = User::where( column: 'email', $username)->first();

    // Check if the user exists and the password is correct
    if (!$user || !Hash::check($password, $user->password)) {
        return Response::json(['error' => 'Unauthorized'], status: 401);
    }

    // Allow the request to proceed if authentication is successful
    return $next($request);
}
```

Sl. 4. Implementacija ovog metoda autentifikacije na strani servera

C. *Potvrda identiteta korišćenjem API ključa*

Autentifikacija API ključem je široko korišćen metod za autentifikaciju IoT uređaja sa serverima. Ovaj metod se oslanja na jedinstveni API ključ — token koji služi kao akreditiv za određeni uređaj. Kada neki uređaj uputi zahtev serveru, zahtev uključuje API ključ u zaglavlje ili parametre zahteva, omogućavajući serveru da verifikuje identitet uređaja.

API ključ je jedinstveni identifikator, obično dugačak niz alfanumeričkih znakova, koji klijentu (u ovom slučaju, IoT uređaju) izdaje server. API ključevi su obično vezani za određene dozvole pristupa, omogućavajući serveru da kontroliše kojim radnjama ili podacima API uređaj može da pristupi.

```php
function register(Request $request) {
    $apiKey = generateUniqueApiKey(); // Generate a unique API key
    $client = Client::query()->create([
        'name' => $request->get( key: 'name'),
        'api_key' => $apiKey, // Store the generated API key
    ]);
    return $client;
}

Codeium: Refactor | Explain | Document | ×
1 usage
function generateUniqueApiKey()
{
    do {
        $apiKey = Str::random( length: 40); // Generate a 40-character API key
    } while (Client::query()->where( column: 'api_key', $apiKey)->exists());


    return $apiKey;
}
```

Sl. 5. Obrada zahteva za registraciju novog uređaja

Dobijeni API ključ mora biti bezbedno uskladišten na samom uređaju (npr. korišćenjem već pomenutog SecureStore-a iz prethodnog primera) kako bi se sprečio neovlašćeni pristup u slučaju da je uređaj kompromitovan. Zatim se isti ključ šalje pri svakom sledećem zahtevu.

```javascript
// Retrieve the API key from SecureStore
const apiKey : string  = await SecureStore.getItemAsync( key: 'api_key');
// Send request with API key in Authorization header
const response : Response  = await fetch( input: 'https://cloudcities.co/api/sensor-events',  init: {
    method: 'GET',
    headers: {
        'Authorization': `ApiKey ${apiKey}`,
        'Content-Type': 'application/json',
    },
});
```

Sl. 6. Slanje API ključa u zaglavlje zahteva

```php
public function handle(Request $request, Closure $next) {
    $apiKey = $request->header( key: 'Authorization');
    // Check for the 'ApiKey' prefix and get the key
    if (str_starts_with($apiKey, 'ApiKey ')) {
        $apiKey = substr($apiKey, offset: 7);
    } else {
        return response()->json(['error' => 'Unauthorized'], status: 401);
    }
    // Validate the API key
    $client = Client::query()->where( column: 'api_key', $apiKey)->first();


    if (!$client) {
        return response()->json(['error' => 'Unauthorized'], status: 401);
    }
    return $next($request);
}
```

Sl. 7. Provera ovlašćenja prispelog zahteva na serveru

Ključne prednosti ovog metoda autentifikacije su:

- Jednostavnost: autentikacija API ključem je jednostavna za implementaciju i ne zahteva složene kriptografske operacije na IoT uređajima sa ograničenim resursima.
- Kontrola i fleksibilnost: API ključevi omogućavaju serveru da kontroliše pristup definisanjem posebnih dozvola povezanih sa svakim ključem.
- Kompatibilnost: Ovaj metod je kompatibilan sa mnogim tipovima protokola, kao što su HTTP, AMQP, MQTT, itd., što ga čini pogodnim za različite komunikacione protokole.

Nedostaci ovog metoda autentifikacije na koje je potrebno obratiti pažnju u implementaciji:

- Bezbednosni rizici pri izlaganju: API ključevi su osetljivi i mogu se lako zloupotrebiti ako su izloženi. Bez šifrovanja u transportu (HTTPS), ranjivi su na presretanje.
- Bez roka trajanja: Obično se generišu bez datuma isteka, što povećava šansu od zloupotrebe prilikom izlaganja ključa.
- Ograničene bezbednosne kontrole: API ključevi sami po sebi ne pružaju zaštitu od napada ponovne reprodukcije, napada grubom silom (Brute-force) ili drugih neovlašćenih radnji.

D. MAC adresa u kombinaciji sa API ključem

Ovaj način autentifikacije kombinuje dva faktora autentikacije kako bi

poboljšao bezbednost IoT uređaja koji komuniciraju sa HTTP serverom. Ovaj metod koristi jedinstveni identifikator hardvera IoT uređaja (MAC adresu) zajedno sa API ključem koji se izdaje tokom kreiranja uređaja ili registracije. Ovaj pristup dodaje dodatni sloj sigurnosti, specifične za sam uređaj, uz zadržavanje jednostavnosti u implementaciji.

Kao što je već pomenuto u jednom od prethodnih primera MAC adresa je jedinstvena hardverska adresa dodeljena mrežnom interfejsu. Služi kao identifikator uređaja unutar lokalne mreže. API ključ deluje kao akreditiv za autentifikaciju zasnovan na softveru, koji se izdaje IoT uređaju tokom registracije. Osigurava da samo ovlašćeni uređaji sa ispravnim API ključem mogu pristupiti serveru. Kombinacija ova dva pristupa se zasniva na tome da kada IoT uređaj pošalje zahtev serveru, on uključuje svoju MAC adresu i API ključ u zahtev. Server potvrđuje oba parametra: proverava da li je MAC adresa na registrovana, a zatim verifikuje API ključ u odnosu na svoju bazu podataka. Oba faktora moraju da se podudaraju da bi zahtev bio prihvaćen.

Izdavanje API ključa se radi na identični način kao kod prethodnog metoda (Potvrda identiteta korišćenjem API ključa).

```
function getMacAddress() : any | undefined  {
    const networkInterfaces : {}  = os.networkInterfaces();
    for (const interfaceName in networkInterfaces) {
        const networkInterface = networkInterfaces[interfaceName];
        for (const details : any  of networkInterface) {
            if (!details.internal && details.mac) {
                return details.mac; // Return the first non-internal address
            }
        }
    }
    throw new Error("Unable to retrieve MAC address");
}

// Retrieve the API key from SecureStore
const API_KEY : string  = await SecureStore.getItemAsync( key: 'API_KEY');
const macAddress = getMacAddress();
const headers : {...}  = {
    "Authorization": `ApiKey ${API_KEY}`,
    "Device-MAC": macAddress,
    "Content-Type": "application/json",
};
const response : Response  = await fetch( input: 'https://cloudcities.co/api/sensor-events', init: {
    method: "GET",
    headers,
});
```

Sl. 8. Dobijanje MAC adrese uređaja i njeno slanje uz zahtev serveru

```php
public function handle(Request $request, Closure $next)
{
    $apiKey = $request->header( key: 'Authorization');
    $macAddress = $request->header( key: 'Device-MAC');


    // Validate API Key exists in the request
    if (!$apiKey || !str_starts_with($apiKey, 'ApiKey ')) {
        return response()->json(['error' => 'Unauthorized'], status: 401);
    }
    $apiKey = substr($apiKey, offset: 7);


    // Check API Key and MAC Address in database
    $device = Device::where('api_key', $apiKey)
        ->where('mac_address', $macAddress)
        ->first();


    if (!$device) {
        return response()->json(['error' => 'Unauthorized'], status: 401);
    }

    return $next($request);
}
```

Sl. 9. Obrada dolaznog zahteva na serveru

Ključne prednosti ovog metoda autentifikacije su:

- Poboljšana bezbednost: Zahtevajući i MAC adresi i API ključu, ovaj metod obezbeđuje dva sloja verifikacije. Čak i ako je API ključ kompromitovan, napadač takođe mora da lažira tačnu MAC adresu.
- Autentifikacija specifična za sam uređaj: MAC adrese povezuju autentifikaciju sa fizičkim uređajem, što otežava lažno predstavljanje uređaja bez fizičkog pristupa.

Izazovi i ograničenja ovog metoda autentifikacije na koje je potrebno obratiti pažnju u implementaciji:

- Lažiranje MAC adrese: MAC adrese se mogu lažirati korišćenjem lako dostupnih alata, smanjujući njihovu efikasnost kao jedinog identifikatora. U kombinaciji sa API ključevima, površina napada je smanjena, ali ne i eliminisana.
- Vidljivost mreže: MAC adresa je vidljiva samo uređajima na istoj lokalnoj mreži ili podmreži, što ograničava njenu korisnost u komunikaciji širom interneta. Zbog toga je neophodno eksplicitno slanje MAC adrese u sklopu samog zahteva.
- Upravljanje ključem: Upravljanje API ključevima (rotacija, opoziv, obnavljanje) za veliku flotu IoT uređaja može postati složeno bez

automatizacije.

- Čuvanje API ključeva na uređajima: API ključevi moraju biti bezbedno uskladišteni na IoT uređajima.

E. Autentifikacija bazirana na heširanoj vrednosti zahteva - HMAC

HMAC (Hash-Based Message Authentication Code) Autentifikacija je metod gde se kriptografska heš funkcija koristi za proveru i integriteta i autentičnosti poruke. HMAC se obično koristi u IoT i API autentifikaciji kako bi se osigurala bezbedna komunikacija između klijenta (uređaja) i servera.

Bazira se na tome da klijent i server dele tajni ključ poznat samo njima. Klijent pri svakom zahtevu kreira HMAC heširajući poruku (npr. podatke zahteva i vremensku oznaku) sa deljenim tajnim ključem. HMAC se šalje zajedno sa porukom u HTTP zahtevu.

Server, pri dolasku zahteva, ponovo kreira HMAC koristeći primljenu poruku i deljeni tajni ključ. Ako se HMAC kreiran na serverskoj strani podudara sa primljenim HMAC-om, zahtev je validan, a podaci u zahtevu autentični.

```
const SECRET_KEY : string  = await SecureStore.getItemAsync( key: 'secret_key');
Codeium: Refactor | Explain | Docstring | ×
Show usages
function generateHmac(payload, secretKey) {
    return crypto.createHmac( sha256: 'sha256', secretKey)
        .update(payload)
        .digest( algorithm: 'hex');
}
const timestamp : string  = Date.now().toString(); // Timestamp for replay protection
const payload : string  = JSON.stringify( value: { data: 'example_data' });
// Create the HMAC signature
const hmac = generateHmac( payload: `${timestamp}:${payload}`, SECRET_KEY);

// Send the request
const response : Response  = await fetch( input: 'https://cloudcities.co/api/sensor-events',  init: {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
        'Authorization': `HMAC ${hmac}`,
        'Timestamp': timestamp // Timestamp for server verification
    },
    body: payload
});
```

Sl. 10. Primer koda na strani klijenta za slanje zahteva korišćenjem HMAC autentifikacije

```php
public function handle(Request $request, Closure $next)
{
    $secret = env( key: 'HMAC_SECRET_KEY'); // Shared secret key
    $authorizationHeader = $request->header( key: 'Authorization');
    $timestamp = $request->header( key: 'Timestamp');

    if (!$authorizationHeader || !str_starts_with($authorizationHeader, 'HMAC ')){
        return response()->json(['error' => 'Unauthorized'], status: 401);
    }

    $receivedHmac = substr($authorizationHeader, offset: 5); // 'HMAC ' prefix

    // Check for replay attacks
    if (abs( num: time() - $timestamp / 1000) > 120) {
        // Allow 2-minute window due to time skew and delay in processing
        return response()->json(['error' => 'Request expired'], status: 401);
    }

    // Recreate the HMAC using the same payload and secret
    $payload = $request->getContent();
    $calculatedHmac = hash_hmac( algo: 'sha256', data: "$timestamp:$payload", $secret);

    if (!hash_equals($calculatedHmac, $receivedHmac)) {
        return response()->json(['error' => 'Invalid HMAC'], status: 401);
    }

    return $next($request);
}
```

Sl. 11. Generisanje i provera HMAC-a na strani servera, kako bi se uverili da je zahtev validan, a podaci autentični

Ključne prednosti ovog metoda autentifikacije su:

- Integritet i autentičnost: Osigurava da poruka nije neovlaštena i da dolazi od pouzdanog izvora.
- Bez razmene ključeva: Nema potrebe za prenosom tajnog ključa tokom komunikacije, smanjujući rizik od presretanja ključa.
- Nezahtevna: Pogodna za IoT uređaje sa ograničenim resursima jer ne zahteva asimetričnu kriptografiju.
- Bez zadržavanja stanja na serveru: Ne zahteva skladištenje tokena ili podataka o sesiji na strani servera.

Ograničenja koje ovaj metod autentifikacije nosi:

- Ranjivost deljenog ključa: Neophodno je da i klijent i server bezbedno čuvaju tajni ključ. Ako je ključ ugrožen, bezbednost sistema je ugrožena potencijalno lažiranim zahtevima.
- Ponovni napadi: Bez vremenske oznake, napadači mogu ponovo da koriste važeće HMAC-ove. Ovo se ublažava dodavanjem vremenskih oznaka ili jedinstvenih identifikatora zahteva.
- Upravljanje ključem: Upravljanje i rotiranje zajedničkih ključeva

može biti izazov za velike sisteme.

F. JWT (JSON Web Token) autentifikacija

JWT (JSON Web Token) je otvoreni standard koji definiše kompaktan i samostalan način za bezbedno prenošenje informacija između klijenta i servera u okviru JSON objekta. Naširoko se koristi za autentifikaciju i razmenu podataka. Server izdaje token klijentu nakon uspešne autentifikacije. Klijent koristi ovaj token za autentifikaciju narednih zahteva.

Ovaj token se sastoji iz tri dela (Base64) enkodovanog teksta odvojenih tačkom:

- Zaglavlje, Header (npr. { "alg": "HS256", "typ": "JWT" }).
  - Metapodaci o tokenu i algoritam heširanja potpisa ovog tokena.
- Telo, Payload (npr. { "sub": "12345", "name": "Device 1", "exp": 1680000000 })
  - Dodatni podaci o samom klijentu, urešđaju: identifikator uređaja, uloga, vreme isteka tokena, itd.
- Potpis: tekst dobijen korišćenjem funkcije koja kreira HMAC sa SHA-256 tipom heširanja (HS256) zaglavlja i tela tokena, potpisanog tajnim ključem.
  - Kriptografski potpis za proveru integriteta tokena.

Izdavanje tokena se inicira autentifikacijom klijenta korišćenjem svojih kredencijala (npr. identifikator klijenta i tajni ključ). Server generiše JWT koji uključuje: zaglavlje, telo tokena i potpis. Nakon toga, u sklopu svakog narednog zahteva klijent šalje dobijeni token.Server proverava potpis tokena pomoću tajnog ključa kako bi se uverio da su podaci u tokenu validni i da njima nije manipulisano.

```
const credentials : {client_id: string, client_secret: string}  = {
    client_id: "client_id",
    client_secret: "client_secret",
};
const response : Response  = await fetch( input: 'https://cloudcities.co/api/login', init: {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify(credentials),
});
const data = await response.json();
const jwt = data.token;
```

Sl. 12. Inicijalni zahtev za dobijanje tokena na uređaju

```
        $clientId = $request->get( key: 'client_id');
        $clientSecret = $request->get( key: 'client_secret');

        $client = Client::query()->where( column: 'client_id', $clientId)
            ->where( column: 'client_secret', $clientSecret)->firstOrFail();
        $payload = [ // Generate JWT payload
            'sub' => $clientId,
            'iat' => time(), // Issued at
            'exp' => time() + 3600,// Expiration (1 hour)
        ];
        $jwt = JWT::encode($payload, env( key: 'JWT_SECRET'),  string: 'HS256');

        return response()->json(['token' => $jwt]);
```

Sl. 13. Obrada zahteva za izdavanje tokena

```
public function handle(Request $request, Closure $next) {
    $authorizationHeader = $request->header( key: 'Authorization');
    if(!$authorizationHeader || !str_starts_with($authorizationHeader, 'Bearer '))
    {
        return response()->json(['error' => 'Unauthorized'],  status: 401);
    }

    $token = substr($authorizationHeader,  offset: 7); // Remove 'Bearer ' prefix

    try {
        $decoded = JWT::decode($token, new Key(env( key: 'JWT_SECRET'),  alg: 'HS256'));
        Client::query()->where( column: 'client_id', $decoded->client_id)->firstOrFail();
    } catch (\Exception $e) {
        return response()->json(['error' => 'Invalid or expired token'], status: 401);
    }
    return $next($request);
}
```

Sl. 14. Server proverava svakih zahtev ka zaštićenim putanjama

Ključne prednosti ovog metoda autentifikacije su:

- Bez zadržavanja stanja na serveru: tokeni su samostalni i ne zahtevaju skladištenje na strani servera.
- Kompaktnost: tokeni su mali i mogu se poslati kao deo parametara URL-a ili zaglavlja.
- Integritet: Potpis obezbeđuje da se token ne može menjati nakon slanja.
- Skalabilnost: Pogodno za distribuirane sisteme jer sam token sadrži sve potrebne informacije.

Izazovi i ograničenja ovog metoda autentifikacije na koje je potrebno obratiti pažnju u implementaciji:

- Invalidacija: Dizajn bez zadržavanja tokena na serveru otežava invalidaciju tokena bez dodatnih mehanizama kao što je crna lista.
- Veličina tokena: Tokeni mogu postati veliki ako telo tokena sadrži previše informacija.
- Upravljanje istekom: Tokeni moraju imati kratak životni vek da bi se smanjili rizici u slučaju krađe.

G. Metoda OAuth 2.0 sa dozvolom na osnovu kredencijala klijenta

OAuth 2.0 je definisan kao otvoreni standard za autorizaciju. On definiše mehanizme za siguran delegirani pristup, omogućavajući klijentu (npr. IoT uređaju, aplikaciji) da pristupi resursima u ime vlasnika resursa (npr. korisnika, sistema). OAuth 2.0 je radni okvir, a ne protokol, što znači da pruža samo smernice za primenu bezbedne autorizacije, ostavljajući specifične detalje inženjeru.

OAuth 2.0 definiše više tipova odobrenja za različite scenarije:

- Autorizacioni kod: Za aplikacije koje zahtevaju interakciju korisnika.
- Implicitno odobrenje: Za male aplikacije na strani klijenta.
- Kredencijali klijenta: Za komunikaciju između mašina (npr. IoT uređaji).
- Lozinka vlasnika resursa: Za pouzdane aplikacije.

U skladu sa temom rada, fokusiraćemo se na odobrenje sa kredencijalima klijenta (client credentials).

Klijent (IoT uređaj) se autentifikuje sa serverom koristeći svoj identifikator klijenta i tajni ključ. Ove kredencijale klijent šalje serveru kako bi zatražio pristupni token. Server verifikuje klijenta i izdaje token za pristup. Nakon ovoga, klijent uključuje pristupni token u zaglavlje autorizacije za pristup zaštićenim resursima.

```php
Route::post( uri: '/oauth/token', function (Request $request) {
    $request->validate([
        'client_id' => 'required|string',
        'client_secret' => 'required|string',
        'grant_type' => 'required|in:client_credentials',
    ]);
    // Find the client by client_id
    $client = DB::table( table: 'clients')
        ->where( column: 'client_id', $request->get( key: 'client_id'))
        ->firstOrFail();
    // Validate the client secret
    if (!password_verify($request->get( key: 'client_secret'), $client->client_secret)) {
        return response()->json(['error' => 'Invalid credentials'], status: 401);
    }
    $token = Str::random( length: 60); // Generate access token
    $expiresAt = now()->addHours( value: 1); // Token expiration time


    // Save token in the database
    DB::table( table: 'tokens')->insert([
        'client_id' => $client->id,
        'token' => hash( algo: 'sha256', $token),
        'expires_at' => $expiresAt,
    ]);


    return response()->json([
        'access_token' => $token,
        'token_type' => 'Bearer',
        'expires_in' => $expiresAt->timestamp - now()->timestamp,
    ]);
});
```

Sl. 15. Obrada zahteva za izdavanje tokena

```php
public function handle(Request $request, Closure $next)
{
    $header = $request->header( key: 'Authorization');


    if (!$header || !str_starts_with($header, 'Bearer ')) {
        return response()->json(['error' => 'Unauthorized'], status: 401);
    }

    $token = substr($header, offset: 7);
    $hashedToken = hash( algo: 'sha256', $token);

    $tokenRecord = DB::table( table: 'tokens')
        ->where( column: 'token', $hashedToken)
        ->where( column: 'expires_at', operator: '>', now())
        ->first();

    if (!$tokenRecord) {
        return response()->json(['error' => 'Unauthorized or expired'], status: 401);
    }

    return $next($request);
}
```

Sl. 16. Osigurati željene resurse proverom tokena

Ključne prednosti ovog metoda autentifikacije su:

- Bezbedna autentifikacija uređaj-server: idealno za scenarije u kojima klijent od poverenja komunicira direktno sa serverom.
- Nema interakcije korisnika: pojednostavljuje autentifikaciju jer nisu uključeni nikakvi korisnički kredencijali ili interakcija.
- Kratkotrajni tokeni: tokeni za pristup imaju kratak vek trajanja, smanjujući uticaj krađe tokena.
- Centralizovano upravljanje pristupom: omogućava centralizovanu kontrolu pristupa izdavanjem i opozivom tokena.

Izazovi i ograničenja ovog metoda autentifikacije na koje je potrebno obratiti pažnju u implementaciji:

- Tajni ključ klijenta mora biti bezbedno uskladištena na IoT uređaju.
- Ovaj tip granta nije pogodan za scenarije koji zahtevaju autentifikaciju ili autorizaciju specifičnu za korisnika.
- Zahteva efikasne mehanizme za isticanje i obnavljanje tokena na samom klijentu. Klijent je dužan da vodi računa da li je token istekao ili ga je server okarakterisao kao nevažeći.

III. METODA AUTENTIFIKACIJE UPOTREBOM TLS SERTIFIKATA

TLS sertifikati predstavljaju metod autentifikacije koji koristi X.509 sertifikate za verifikaciju i klijenta (npr. IoT uređaja) i servera. Za razliku od tradicionalnog TLS-a, gde je samo server autentifikovan, mTLS zahteva od klijenta da predstavi sopstveni sertifikat tokom procesa uspostavljanja konekcije. Ovaj metod se široko koristi za obezbeđenje IoT uređaja, API-ja i osetljivih komunikacija zbog svojih jakih mogućnosti autentifikacije.

U ovom odeljku biće opisani koraci potrebni za generisanje i upravljanje X.509 sertifikatima za IoT uređaje, uključujući kreiranje autoriteta za sertifikate (eng. Certified Autority, CA), generisanje sertifikata uređaja i proces potpisivanja. Ovo osigurava bezbednu autentifikaciju i komunikaciju između uređaja i servera.

Koraci za kreiranje autoriteta za izdavanje sertifikata i kreiranje sertifikata uređaja:

1. Kreiranje autoriteta za izdavanje sertifikata (CA). Za kreiranje IoT aplikacija, nije neophodno koristiti neko zvanično autorizaciono telo za potpisivanje sertifikata, već se možemo osloniti na samopotpisane sertifikate, kao u ovom primeru.
    a. Generisanje CA privatnog ključa koji se koristi za potpisivanje sertifikata samog uređaja: *openssl genrsa -out ca.key 2048*

   b. Koristeći privatni ključ, generisaćemo samopotpisani sertifikat:
      openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.crt -subj "/CN=MyIoTCA"
2. Kreiranje sertifikata uređaja. Svaki IoT uređaj zahteva jedinstveni sertifikat da bi se obezbedila sigurna komunikacija i doslednost.
   a. Svaki uređaj generiše sopstveni privatni ključ: *openssl genrsa -out device1.key 2048*
   b. Sledeći korak je kreiranje zahteva za potpisivanje sertifikata (Certificate Signing Request - CSR) koji sadrži javni ključ uređaja i identifikacione informacije (Common Name - CN). Za kreiranje ovog zahteva koristimo prethodno kreirani privatni ključ:
      openssl req -new -key device1.key -out device1.csr -subj "/CN=Device1"
3. Potpisivanje sertifikata uređaja. Autoritet za izdavanje sertifikata (Certificate Authority - CA) potpisuje prethodno kreirani zahtev (CSR) uređaja i time kreira pouzdani X.509 sertifikat: *openssl x509 -req -in device1.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out device1.crt -days 365 -sha256*

   Rezultat komande:
   - device1.crt: Potpisani sertifikat za uređaj.
   - ca.srl: Jedinstvena datoteka serijskog broja koja osigurava da svi uređaji imaju jedinstvene serijske brojeve.
4. Distribucija sertifikata.
   a. Na urećaju: urećaj je dužan da skladišti privatni ključ (device1.key) i potpisani sertifikat (device1.crt). ove datoteke nikada ne bi trebalo da budu izložene ili nebezbedno prenesene.
   b. Na serveru: server mora imati pristup CA sertifikatu (ca.crt) koji koristi za validaciju dolaznih sertifikata uređaja.

```php
public function handle(Request $request, Closure $next)
{
    $deviceCert = $request->header( key: 'Device-Certificate');
    // Previously generated certificate for Certificate Authority.
    $caCertPath = storage_path( path: 'app/ca.crt');

    $isValid = openssl_x509_checkpurpose($deviceCert, purpose: X509_PURPOSE_SSL_CLIENT, [$caCertPath]);

    if (!$isValid) {
        return response()->json(['error' => 'Invalid certificate'], status: 403);
    }

    return $next($request);
}
```

Sl. 17. Validacija sertifikata uređaja korišćenjem CA sertifikata

Ključne prednosti koje ovaj metod čine najpouzdanijim:

- Jaka autentifikacija, bez zajedničkih ključeva
    - TLS sertifikati se oslanjaju na asimetričnu kriptografiju (parovi javni/privatni ključevi), koja obezbeđuje snažan mehanizam za proveru identiteta klijenata (uređaja) i servera.
    - Eliminiše potrebu za zajedničkim ključevima kao što su lozinke, smanjujući rizik od napada grubom silom.
- Međusobna autentikacija (eng. mutual TLS - mTLS)
    - Uzajamni TLS može potvrditi autentičnost i klijenta i servera.
    - Osigurava dvosmerno poverenje, što ga čini idealnim za visoko bezbedna okruženja.
- Interoperabilnost
    - TLS sertifikati prate X.509 standard, čineći ih kompatibilnim sa širokim spektrom uređaja, platformi i protokola.
    - Obezbeđuje kompatibilnost preko heterogenih IoT sistema.
- Identitet uređaja
    - Svaki sertifikat jedinstveno identifikuje uređaj.
    - Omogućava pojedinačno praćenje, reviziju i odgovornost za uređaje.

LITERATURA

[1] Kumar Donta, P., Hazra, A., Lovén, L., "Learning Techniques for the Internet of Things", Springer, 2024.
[2] Perera, C., Zaslavsky, A., Christen, P., Georgakopoulos, G. Context aware computing for the Internet of Things: A survey. IEEE Communications Surveys Tuto- rials 16 (1): 414–454, 2014.
[3] Liyanage, M., Braeken, A., Kumar, P., Ylianttila, M., "IoT Security: Advances in Authentication", Wiley, 2020.
[4] Richardson, L., Amundsen, M., Ruby, S., RESTful Web APIs: Services for a Changing World, O'Reilly Media, 2013.
[5] Kurose, J. F., Ross, K. W., Computer Networking: A Top-Down Approach, Pearson, 2016.
[6] Russell, B., Van Duren, D. Practical IoT Security, Packt Publishing, 2016.
[7] Boyd, R., Getting Started with OAuth 2.0, O'Reilly Media, 2012.
[8] Bennett, C. H., Brassard, G., Quantum cryptography: Public key distribution and coin tossing, Theoretical Computer Science, Volume 560, 7-11, 2014.
[9] Mosca, M. Cybersecurity in an era with quantum computers: Will we be ready?, IEEE Security & Privacy, Volume 16, 38–41, 2018.
[10] Zyskind, G., Nathan, O., Pentland, A. S. Decentralizing privacy: Using blockchain to protect personal data. IEEE Security and Privacy Workshops, 180–184, 2015.
[11] Stamp, M., Visaggio C. A., Mercaldo, F., Troia, F., Artificial Intelligence for Cybersecurity, Springer, 2022.

ABSTRACT

The Internet of Things (IoT) is revolutionizing everyday life and industrial processes by enabling connected devices to communicate, analyze, and act autonomously. As IoT devices increasingly transmit sensitive data, secure communication becomes paramount, with authentication serving as the foundation for ensuring data confidentiality and integrity. This paper explores various authentication methods tailored to IoT devices communicating with servers via the HTTP protocol, offering theoretical insights and practical implementation guidelines.

Each method is thoroughly examined concerning its advantages, disadvantages, and suitability for different IoT environments. Implementations are provided for both client-side (IoT devices) and server-side using technologies like JavaScript and PHP (Laravel). Particular attention is given to the resource constraints typical of IoT devices, such as limited processing power and memory.

AUTHENTICATION METHODS OF IOT DEVICES IN COMMUNICATION WITH THE SERVER: IMPLEMENTATION AND COMPARATIVE ANALYSIS

Stefan Antić, dr Nemanja Radosavljević