readme.md for @push.rocks/smartradius

A TypeScript RADIUS server and client library with full RFC 2865/2866 compliance for network authentication, authorization, and accounting (AAA).

Issue Reporting and Security

For reporting bugs, issues, or security vulnerabilities, please visit community.foss.global/. This is the central community hub for all issue reporting. Developers who sign and comply with our contribution agreement and go through identification can also get a code.foss.global/ account to submit Pull Requests directly.

Install

Install via npm or pnpm:

```
npm install @push.rocks/smartradius
# or
pnpm add @push.rocks/smartradius
```

Usage

This module provides a TypeScript-based RADIUS (Remote Authentication Dial-In User Service) server and client implementation. RADIUS is a networking protocol that provides centralized Authentication, Authorization, and Accounting (AAA) management for users who connect and use a network service.

What is RADIUS?

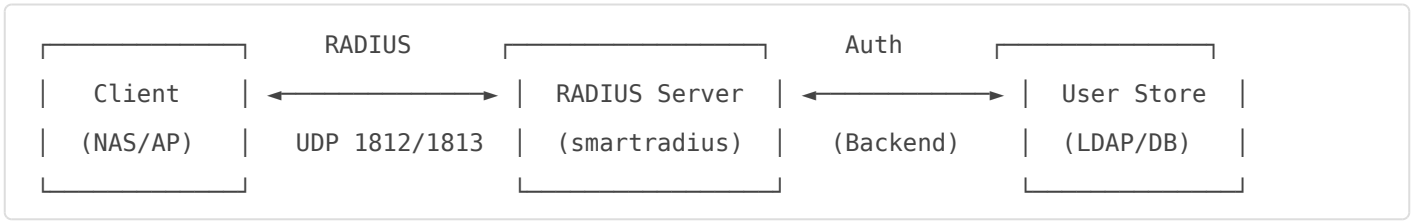
RADIUS is an industry-standard protocol used for:

- □□ **Authentication** - Verifying user credentials (username/password via PAP or CHAP)
- □□ **Authorization** - Determining what resources authenticated users can access
- □□ **Accounting** - Tracking network usage for billing, auditing, and statistics

Common Use Cases

- □□ **Enterprise Wi-Fi** - 802.1X authentication for wireless networks
- □□ **VPN Access** - Authenticating remote users connecting via VPN
- □□ **Network Access Control** - Managing who can access network resources
- □□ **ISP Authentication** - Authenticating dial-up and broadband users
- □□ **Device Authentication** - IoT and network device access control

Architecture



Creating a RADIUS Server

```
import { RadiusServer, ERadiusCode } from '@push.rocks/smartradius';

// Create a RADIUS server
const server = new RadiusServer({
  authPort: 1812,        // Authentication port (default)
  acctPort: 1813,        // Accounting port (default)
  bindAddress: '0.0.0.0', // Listen on all interfaces
  defaultSecret: 'shared-secret-with-clients',

  // Authentication handler
  authenticationHandler: async (request) => {
    console.log(`Auth request from ${request.username}`);

    // PAP authentication
    if (request.password !== undefined) {
```

```
      if (request.username === 'testuser' && request.password === 'testpass') {
        return {
          code: ERadiusCode.AccessAccept,
          replyMessage: 'Welcome!',
          sessionTimeout: 3600,
          framedIpAddress: '10.0.0.100',
        };
      }
    }

    // CHAP authentication
    if (request.chapPassword && request.chapChallenge) {
      const { RadiusAuthenticator } = await import('@push.rocks/smartradius');
      const isValid = RadiusAuthenticator.verifyChapResponse(
        request.chapPassword,
        request.chapChallenge,
        'expected-password'
      );
      if (isValid) {
        return { code: ERadiusCode.AccessAccept };
      }
    }

    return {
      code: ERadiusCode.AccessReject,
      replyMessage: 'Invalid credentials',
    };
  },

  // Accounting handler
  accountingHandler: async (request) => {
    console.log(`Accounting ${request.statusType}: session ${request.sessionId}`);
    // Record accounting data to database
    return { success: true };
  },
});

// Start the server
await server.start();
```

```
console.log('RADIUS server running on ports 1812/1813');

// Per-client secrets (optional)
server.setClientSecret('192.168.1.100', 'client-specific-secret');

// Get statistics
const stats = server.getStats();
console.log(`Processed ${stats.authRequests} auth requests`);

// Stop the server
await server.stop();
```

Creating a RADIUS Client

```
import { RadiusClient, EAcctStatusType } from '@push.rocks/smartradius';

// Create a RADIUS client
const client = new RadiusClient({
  host: '192.168.1.1',
  authPort: 1812,
  acctPort: 1813,
  secret: 'shared-secret',
  timeout: 5000,    // 5 second timeout
  retries: 3,       // Retry 3 times
  nasIdentifier: 'my-nas-device',
});

// Connect (binds UDP socket)
await client.connect();

// PAP Authentication
const papResponse = await client.authenticatePap('username', 'password');
if (papResponse.accepted) {
  console.log('PAP Auth successful!');
  console.log('Session timeout:', papResponse.sessionTimeout);
  console.log('Assigned IP:', papResponse.framedIpAddress);
}
```

```
// CHAP Authentication
const chapResponse = await client.authenticateChap('username', 'password');
if (chapResponse.accepted) {
  console.log('CHAP Auth successful!');
}

// Authentication with custom attributes
const customResponse = await client.authenticate({
  username: 'user',
  password: 'pass',
  nasPort: 1,
  calledStationId: 'AA-BB-CC-DD-EE-FF',
  callingStationId: '11-22-33-44-55-66',
});

// Accounting: Session Start
await client.accountingStart('session-001', 'username');

// Accounting: Interim Update
await client.accountingUpdate('session-001', {
  username: 'username',
  sessionTime: 300,       // 5 minutes
  inputOctets: 1024000,   // 1 MB received
  outputOctets: 2048000,  // 2 MB sent
});

// Accounting: Session Stop
await client.accountingStop('session-001', {
  username: 'username',
  sessionTime: 600,
  inputOctets: 2048000,
  outputOctets: 4096000,
  terminateCause: 1,  // User-Request
});

// Disconnect
await client.disconnect();
```

Working with RADIUS Packets Directly

```
import {
  RadiusPacket,
  RadiusAuthenticator,
  RadiusAttributes,
  ERadiusCode,
  ERadiusAttributeType,
} from '@push.rocks/smartradius';

// Create an Access-Request packet
const packet = RadiusPacket.createAccessRequest(1, 'secret', [
  { type: ERadiusAttributeType.UserName, value: 'testuser' },
  { type: ERadiusAttributeType.UserPassword, value: 'testpass' },
  { type: ERadiusAttributeType.NasIpAddress, value: '192.168.1.1' },
]);

// Decode a received packet
const decoded = RadiusPacket.decodeAndParse(receivedBuffer);
console.log('Code:', RadiusPacket.getCodeName(decoded.code));
console.log('Attributes:', decoded.parsedAttributes);

// Verify response authenticator
const isValid = RadiusAuthenticator.verifyResponseAuthenticator(
  responseBuffer,
  requestAuthenticator,
  secret
);

// Encrypt/decrypt PAP password
const encrypted = RadiusAuthenticator.encryptPassword(password, authenticator, secret);
const decrypted = RadiusAuthenticator.decryptPassword(encrypted, authenticator, secret);

// Calculate CHAP response
const chapResponse = RadiusAuthenticator.calculateChapResponse(chapId, password, challenge);

// Create Vendor-Specific Attribute
const vsaAttr = RadiusAttributes.createVendorAttribute(
  9,          // Cisco vendor ID
  1,          // Vendor-specific type
  Buffer.from('cisco-av-pair=value')
```

```
);
```

Supported Features

Authentication (RFC 2865)

- ✅ PAP (Password Authentication Protocol)
- ✅ CHAP (Challenge-Handshake Authentication Protocol)
- ✅ Access-Request, Access-Accept, Access-Reject, Access-Challenge packets
- ✅ Message-Authenticator (HMAC-MD5) for EAP support
- ✅ All standard attributes (1-63) plus EAP attributes (79, 80)

Accounting (RFC 2866)

- ✅ Accounting-Request, Accounting-Response packets
- ✅ Status types: Start, Stop, Interim-Update, Accounting-On/Off
- ✅ Session tracking: time, octets, packets
- ✅ All termination cause codes

Protocol Features

- ✅ Duplicate request detection with response caching
- ✅ Response authenticator verification
- ✅ Configurable timeout and retry with exponential backoff
- ✅ Per-client shared secret management
- ✅ Vendor-Specific Attributes (VSA) support

Enums and Types

```
// Packet codes
enum ERadiusCode {
  AccessRequest = 1,
  AccessAccept = 2,
  AccessReject = 3,
  AccountingRequest = 4,
  AccountingResponse = 5,
  AccessChallenge = 11,
}

// Accounting status types
```

```
enum EAcctStatusType {
  Start = 1,
  Stop = 2,
  InterimUpdate = 3,
  AccountingOn = 7,
  AccountingOff = 8,
}

// Termination causes
enum EAcctTerminateCause {
  UserRequest = 1,
  LostCarrier = 2,
  IdleTimeout = 4,
  SessionTimeout = 5,
  AdminReset = 6,
  // ... and more
}
```

Module Structure

This library is organized into sub-modules for clean separation of concerns:

Module	Description
`ts_shared`	Protocol definitions - RFC enums and core interfaces
`ts_server`	Server implementation - RadiusServer, packet handling, crypto
`ts_client`	Client implementation - RadiusClient with retry logic
`ts`	Main entry point - re-exports everything

RFC Compliance

This implementation follows:

- **RFC 2865** - Remote Authentication Dial In User Service (RADIUS)
- **RFC 2866** - RADIUS Accounting

The RFC specification files are included in the `./spec/` directory for reference.

License and Legal Information

This repository contains open-source code licensed under the MIT License. A copy of the license can be found in the LICENSE file.

Please note: The MIT License does not grant permission to use the trade names, trademarks, service marks, or product names of the project, except as required for reasonable and customary use in describing the origin of the work and reproducing the content of the NOTICE file.

Trademarks

This project is owned and maintained by Task Venture Capital GmbH. The names and logos associated with Task Venture Capital GmbH and any related products or services are trademarks of Task Venture Capital GmbH or third parties, and are not included within the scope of the MIT license granted herein.

Use of these trademarks must comply with Task Venture Capital GmbH's Trademark Guidelines or the guidelines of the respective third-party owners, and any usage must be approved in writing. Third-party trademarks used herein are the property of their respective owners and used only in a descriptive manner, e.g. for an implementation of an API or similar.

Company Information

Task Venture Capital GmbH Registered at District Court Bremen HRB 35230 HB, Germany

For any legal inquiries or further information, please contact us via email at hello@task.vc.

By using this repository, you acknowledge that you have read this section, agree to comply with its terms, and understand that the licensing of the code does not imply endorsement by Task Venture Capital GmbH of any derivative works.

Revision #3
Created 2026-03-28 11:13:43 UTC by foss.global Team
Updated 2026-03-28 12:20:29 UTC by foss.global Team