OWASP Top 10 Security Guide for Node.js and Express.js Applications

The current OWASP Top 10 (2021) presents critical security risks that require specific attention in Node.js and Express.js applications. owasp +2 This comprehensive guide provides practical, production-ready solutions with code examples covering both traditional callbacks and modern async/await patterns. **Broken Access Control has emerged as the #1 vulnerability**, owasp OWASP Foundation making proper authentication and authorization essential for modern applications.

Current OWASP Top 10 vulnerabilities and Node.js implementations

The OWASP Top 10 2021 identifies the most critical web application security risks, with significant changes from previous versions including three new categories addressing modern threats. owasp +2

A01 Broken Access Control (Previously #5)

How it manifests in Express.js: Direct object references, missing authorization checks, and privilege escalation through route manipulation.

Vulnerable Code Example:

javascript

```javascript
// VULNERABLE - No authorization check
app.get('/user/:id', (req, res) => {
  const userId = req.params.id;
  User.findById(userId, (err, user) => {
    res.json(user); // Any authenticated user can access any profile
  });
});
```

Secure Implementation:

javascript

```javascript
// SECURE - Resource-based authorization
const checkResourceAccess = async (req, res, next) => {
  const requestedUserId = req.params.id;
  const currentUserId = req.user.id;

  if (requestedUserId !== currentUserId && !req.user.isAdmin) {
    return res.status(403).json({ error: 'Access denied' });
  }
  next();
};

app.get('/user/:id', authenticateToken, checkResourceAccess, async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    res.json({ id: user.id, name: user.name, email: user.email });
  } catch (error) {
    res.status(500).json({ error: 'Server error' });
  }
});
```

A02 Cryptographic Failures (Previously #3 - Sensitive Data Exposure)

Secure password hashing with bcrypt:

```javascript
const bcrypt = require('bcryptjs');

// Registration with secure hashing
app.post('/register', async (req, res) => {
  const { password } = req.body;
  const saltRounds = 12;

  try {
    const hashedPassword = await bcrypt.hash(password, saltRounds);
    const user = await User.create({
      ...req.body,
      password: hashedPassword
    });
    res.json({ message: 'User created successfully' });
  } catch (error) {
    res.status(500).json({ error: 'Registration failed' });
  }
});
```

A03 Injection (Previously #1)

SQL Injection Prevention:

```javascript
```

```
const mysql = require('mysql2/promise');

// VULNERABLE - String concatenation
app.get('/users', async (req, res) => {
  const name = req.query.name;
  const query = `SELECT * FROM users WHERE name = '${name}'`; // DANGEROUS
});

// SECURE - Prepared statements
app.get('/users', async (req, res) => {
  const name = req.query.name;
  try {
    const [rows] = await connection.execute(
      'SELECT id, name, email FROM users WHERE name = ?',
      [name]
    );
    res.json(rows);
  } catch (error) {
    res.status(500).json({ error: 'Query failed' });
  }
});
```

NoSQL Injection Prevention: Stack Overflow +3

javascript

```
const mongoSanitize = require('express-mongo-sanitize');

app.use(mongoSanitize()); // Apply sanitization globally

// SECURE - Input validation and casting
app.post('/login', async (req, res) => {
  const { username, password } = req.body;

  try {
    const user = await User.findOne({
      username: String(username),  // Explicit type casting
      password: String(password)
    });
    res.json(user);
  } catch (error) {
    res.status(500).json({ error: 'Login failed' });
  }
});
```

A04 Insecure Design (NEW category)

This addresses architectural flaws that cannot be fixed with perfect implementation. owasp Focus on threat modeling and secure design patterns from the beginning.

Secure Design Pattern:

```
javascript
```

```
// Implement proper workflow validation
const validateWorkflow = (currentState, requestedAction, userRole) => {
  const allowedTransitions = {
    'draft': ['submit', 'delete'],
    'pending': userRole === 'admin' ? ['approve', 'reject'] : [],
    'approved': userRole === 'admin' ? ['archive'] : []
  };

  return allowedTransitions[currentState]?.includes(requestedAction) || false;
};

app.post('/documents/:id/action', async (req, res) => {
  const { action } = req.body;
  const document = await Document.findById(req.params.id);

  if (!validateWorkflow(document.status, action, req.user.role)) {
    return res.status(403).json({ error: 'Action not allowed' });
  }

  // Process valid action
});
```

A05 Security Misconfiguration (Previously #6)

Secure Express.js configuration: `npm`

```
javascript
```

```
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');

// Comprehensive security setup
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'", "'nonce-{random}'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      imgSrc: ["'self'", "data:", "https:"]
    }
  },
  hsts: {
    maxAge: 31536000,
    includeSubDomains: true,
    preload: true
  }
}));

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100,
  message: "Too many requests from this IP"
});

app.use('/api/', limiter);
app.use(express.json({ limit: '1kb' })); // Restrict payload size
app.disable('x-powered-by'); // Hide server information
```

Essential security libraries and middleware

Helmet.js for security headers

Helmet sets crucial security headers automatically and is essential for every Express.js application. github +3

javascript

```javascript
const helmet = require('helmet');

app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      "script-src": ["'self'", "example.com"],
      "style-src": ["'self'", "'unsafe-inline'"]
    }
  },
  crossOriginEmbedderPolicy: true
}));
```

Input validation with Joi

Joi provides the most powerful schema-based validation for Node.js applications. GeeksforGeeks DEV Community

javascript

```
const Joi = require('joi');

const userSchema = Joi.object({
  email: Joi.string().email().required(),
  password: Joi.string().min(8).pattern(/^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#$%^&*])/).required(),
  age: Joi.number().integer().min(18)
});

const validateUser = (req, res, next) => {
  const { error } = userSchema.validate(req.body);
  if (error) {
    return res.status(400).json({
      error: error.details[0].message
    });
  }
  next();
};

app.post('/users', validateUser, createUser);
```

CORS configuration

```javascript
```

```
const cors = require('cors');

app.use(cors({
  origin: ['https://yourdomain.com'],
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
  credentials: true,
  allowedHeaders: ['Content-Type', 'Authorization']
}));
```

Advanced authentication and session management

JWT authentication with refresh tokens

```
javascript
```

```
const jwt = require('jsonwebtoken');
const bcrypt = require('bcryptjs');

// Secure JWT implementation
const generateTokens = (user) => {
  const accessToken = jwt.sign(
    { id: user._id, email: user.email },
    process.env.JWT_SECRET,
    { expiresIn: '15m' }
  );

  const refreshToken = jwt.sign(
    { id: user._id },
    process.env.REFRESH_TOKEN_SECRET,
    { expiresIn: '7d' }
  );

  return { accessToken, refreshToken };
};

app.post('/login', async (req, res) => {
  const { email, password } = req.body;

  try {
    const user = await User.findOne({ email });
    if (!user || !await bcrypt.compare(password, user.password)) {
      return res.status(401).json({ message: 'Invalid credentials' });
    }

    const tokens = generateTokens(user);

    // Store refresh token securely
    res.cookie('refreshToken', tokens.refreshToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'strict',
      maxAge: 7 * 24 * 60 * 60 * 1000 // 7 days
    });

    res.json({ accessToken: tokens.accessToken });
  } catch (error) {
    res.status(500).json({ error: 'Login failed' });
  }
});
```

GitHub Stack Overflow

Secure session configuration

javascript

```
const session = require('express-session');
const MongoStore = require('connect-mongo');

app.use(session({
  secret: process.env.SESSION_SECRET,
  name: 'sessionId',
  resave: false,
  saveUninitialized: false,
  store: MongoStore.create({
    mongoUrl: process.env.DATABASE_URL
  }),
  cookie: {
    secure: process.env.NODE_ENV === 'production',
    httpOnly: true,
    maxAge: 24 * 60 * 60 * 1000,
    sameSite: 'strict'
  }
}));
```

npm expressjs

XSS prevention and Content Security Policy

Input sanitization and output encoding

javascript

```javascript
const { body, validationResult } = require('express-validator');
const createDOMPurify = require('isomorphic-dompurify');

// Comprehensive input validation
const validateInput = [
  body('content').trim().escape(),
  body('title').isLength({ min: 1, max: 100 }).trim().escape(),
  (req, res, next) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    next();
  }
];

// HTML sanitization for rich content
app.post('/posts', validateInput, (req, res) => {
  const clean = createDOMPurify.sanitize(req.body.content, {
    ALLOWED_TAGS: ['b', 'i', 'em', 'strong', 'p', 'br'],
    ALLOWED_ATTR: []
  });

  const post = new Post({
    title: req.body.title,
    content: clean
  });

  res.json({ message: 'Post created successfully' });
});
```

OWASP Cheat Sheet Series

Content Security Policy implementation

```javascript
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'", "'nonce-{random}'", "'strict-dynamic'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      imgSrc: ["'self'", "data:", "https:"],
      connectSrc: ["'self'"],
      fontSrc: ["'self'"],
      objectSrc: ["'none'"],
      mediaSrc: ["'self'"],
      frameSrc: ["'none'"],
      baseUri: ["'self'"],
      formAction: ["'self'"]
    },
    reportOnly: false
  }
}));

// CSP violation reporting
app.post('/csp-report', express.json({ type: 'application/csp-report' }), (req, res) => {
  console.log('CSP Violation:', req.body);
  res.status(204).end();
});
```

Medium +4

Rate limiting and brute force protection

Advanced rate limiting with Redis

javascript

```javascript
const { RateLimiterRedis } = require('rate-limiter-flexible');

const loginLimiter = new RateLimiterRedis({
  storeClient: redisClient,
  keyPrefix: 'login_attempts',
  points: 5, // Number of attempts
  duration: 900, // Per 15 minutes
  blockDuration: 900, // Block for 15 minutes
});

app.post('/login', async (req, res) => {
  try {
    await loginLimiter.consume(req.ip);
    // Process login
  } catch (rejRes) {
    const timeToWait = Math.ceil(rejRes.msBeforeNext / 1000);
    return res.status(429).json({
      error: 'Too many login attempts',
      retryAfter: timeToWait
    });
  }
});
```

npm +2

Secure file upload handling

Comprehensive file upload security with Multer

javascript

```
const multer = require('multer');
const crypto = require('crypto');
const path = require('path');

const storage = multer.diskStorage({
  destination: './uploads/secure/',
  filename: (req, file, cb) => {
    const uniqueName = crypto.randomBytes(16).toString('hex');
    const sanitizedName = file.originalname.replace(/[^a-zA-Z0-9.-]/g, '');
    cb(null, `${uniqueName}-${sanitizedName}`);
  }
});

const fileFilter = (req, file, cb) => {
  const allowedTypes = ['image/jpeg', 'image/png', 'application/pdf'];
  const allowedExtensions = ['.jpg', '.jpeg', '.png', '.pdf'];
  const ext = path.extname(file.originalname).toLowerCase();

  if (allowedTypes.includes(file.mimetype) && allowedExtensions.includes(ext)) {
    cb(null, true);
  } else {
    cb(new Error('Invalid file type'), false);
  }
};

const upload = multer({
  storage,
  limits: {
    fileSize: 5 * 1024 * 1024, // 5MB
    files: 3
  },
  fileFilter
});

app.post('/upload', upload.array('files', 3), async (req, res) => {
  if (!req.files?.length) {
    return res.status(400).json({ error: 'No files uploaded' });
  }

  // File signature verification
  for (const file of req.files) {
    const isValid = await verifyFileSignature(file.path, file.mimetype);
    if (!isValid) {
      await fs.unlink(file.path);
      return res.status(400).json({ error: 'File validation failed' });
    }
  }

  res.json({ message: 'Files uploaded successfully' });
});
```

Express.js +4

Dependency vulnerability management

Automated scanning with multiple tools

bash

```bash
```

```
# Built-in npm audit
npm audit --production
npm audit fix --force

# Professional vulnerability scanning
npm install -g snyk
snyk test
snyk monitor

# Generate security reports
npm audit --json > audit-report.json
snyk test --json > snyk-report.json
```

Continuous security monitoring

```javascript
// package.json security scripts
{
  "scripts": {
    "security-check": "npm audit && snyk test",
    "security-monitor": "snyk monitor",
    "update-check": "npm outdated"
  }
}
```

CI/CD security integration

```yaml
```

```yaml
# .github/workflows/security.yml
name: Security Scan
on: [push, pull_request]
jobs:
 security:
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v3
   - name: Install dependencies
    run: npm ci
   - name: Run npm audit
    run: npm audit --audit-level high
   - name: Run Snyk test
    run: npx snyk test --severity-threshold=high
    env:
     SNYK_TOKEN: ${{ secrets.SNYK_TOKEN }}
```

Secrets management and environment security

Secure environment configuration

```
javascript
```

```
// config/index.js - Centralized secure configuration
const requiredEnvVars = [
  'DB_HOST', 'DB_USER', 'DB_PASS', 'JWT_SECRET', 'SESSION_SECRET'
];

for (const envVar of requiredEnvVars) {
  if (!process.env[envVar]) {
    throw new Error(`Missing required environment variable: ${envVar}`);
  }
}

const config = {
  env: process.env.NODE_ENV || 'development',
  port: parseInt(process.env.PORT) || 3000,
  database: {
    host: process.env.DB_HOST,
    user: process.env.DB_USER,
    password: process.env.DB_PASS
  },
  jwt: {
    secret: process.env.JWT_SECRET,
    expiresIn: process.env.JWT_EXPIRES_IN || '1h'
  }
};

module.exports = config;
```

Medium SitePoint

Using encrypted secrets with dotenv-vault

```bash
# Setup encrypted environment files
npx dotenv-vault@latest new
npx dotenv-vault@latest push
npx dotenv-vault@latest build

# Deploy with single environment variable
heroku config:set DOTENV_KEY=dotenv://:key_1234...@dotenv.org/vault/.env.vault?environment=production
```

Medium

Production-ready complete security setup

Comprehensive Express.js security stack

```javascript
```

```
const express = require('express');
const helmet = require('helmet');
const cors = require('cors');
const rateLimit = require('express-rate-limit');
const mongoSanitize = require('express-mongo-sanitize');
const compression = require('compression');
require('dotenv').config();

const app = express();

// Security middleware stack
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'", "'nonce-{random}'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      imgSrc: ["'self'", "data:", "https:"]
    }
  }
}));

app.use(cors({
  origin: process.env.ALLOWED_ORIGINS?.split(',') || ['http://localhost:3000'],
  credentials: true
}));

app.use(rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100
}));
```

```
app.use(express.json({ limit: '10mb' }));
app.use(mongoSanitize());
app.use(compression());
app.disable('x-powered-by');

// Global error handler
app.use((err, req, res, next) => {
  console.error('Error:', err);

  if (process.env.NODE_ENV === 'production') {
    res.status(500).json({ error: 'Internal server error' });
  } else {
    res.status(500).json({ error: err.message, stack: err.stack });
  }
});

app.listen(process.env.PORT || 3000, () => {
  console.log('Secure server running');
});
```

expressjs

Security testing and monitoring

Key security practices for ongoing protection

- **Regular dependency updates**: Use npm audit npm and **Snyk** for continuous monitoring npm +2
- **Input validation**: Always validate with **Joi** or **express-validator** GeeksforGeeks
- **Authentication security**: Implement proper JWT handling with refresh tokens and secure sessions

- **Rate limiting**: Use **express-rate-limit** and **rate-limiter-flexible** for advanced protection owasp
- **Security headers**: **Helmet.js** is essential for all applications expressjs LogRocket
- **Database security**: Always use prepared statements and parameterized queries Snyk +7
- **File uploads**: Validate file types, sizes, and signatures with **Multer** Express.js +2
- **Environment security**: Use **dotenv-vault** or similar for encrypted configuration npm Medium
- **Monitoring**: Implement comprehensive logging and security event tracking owasp

The OWASP Top 10 2021 represents the current standard, with a 2025 update in development. owasp Medium **Broken Access Control remains the most critical vulnerability**, owasp OWASP Foundation requiring robust authentication and authorization patterns. Modern Node.js applications benefit from a defense-in-depth approach using multiple security layers, with particular attention to the Express.js middleware ecosystem and npm dependency management. Medium +2