Contents

1 SEC-001: Authentication Bypass

1.1 Severity

Critical □

1.2 OWASP Reference

A07:2021 - Identification and Authentication Failures

1.3 CWE References

- CWE-287: Improper Authentication
- CWE-306: Missing Authentication for Critical Function

1.4 Description

Authentication bypass vulnerabilities occur when an attacker can access protected resources or perform privileged actions without providing valid credentials. In MCP servers, this vulnerability allows unauthorized users to invoke protected functions, access sensitive data, or manipulate server configuration without proper identity verification.

This is particularly critical in MCP implementations because:

- MCP servers often handle sensitive operations like code execution, file system access, and data transformation
- Authentication is the first line of defense in securing server-to-client interactions
- Bypass vulnerabilities can lead to complete compromise of the MCP server and its connected resources

1.5 Compliance Mapping

1.5.1 SOC2

- **CC6.1**: Logical and Physical Access Controls - The organization implements logical access security measures to protect against unauthorized access, including authentication and authorization.
- **CC6.2**: Access Control - User identities and credentials are managed through defined access control procedures.

1.5.2 HIPAA

- **§164.312(a)(1)**: User Identification and Authentication - Information system login credentials and control mechanisms are implemented.
- **§164.312(d)**: Encryption and Decryption - Appropriate encryption mechanisms are used to protect electronic protected health information.

1.5.3 PCI DSS

- **6.2.4**: Security patches must be installed within one month of release
- **8.3.1**: Multi-factor authentication must be implemented for all non-console access
- **8.4.1**: Access must be controlled using unique user IDs and strong authentication mechanisms

1.6 Vulnerable Code Example

```
// □ INSECURE: No authentication middleware
const express = require('express');
const app = express();

// Direct access to sensitive functions without any authentication check
app.post('/api/execute-tool', (req, res) => {
  const { toolName, params } = req.body;

  // This endpoint is completely exposed - anyone can call it
  const result = executeTool(toolName, params);

  res.json({ success: true, result });
});

app.delete('/api/config', (req, res) => {
  // No verification that the user has permission to delete configuration
  deleteServerConfig();
  res.json({ success: true, message: 'Configuration deleted' });
});

app.listen(3000);
```

1.7 Secure Implementation

```
// □ SECURE: Proper authentication middleware with JWT tokens
const express = require('express');
const jwt = require('jsonwebtoken');
const app = express();

const SECRET_KEY = process.env.JWT_SECRET;

// Authentication middleware
const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1]; // Bearer TOKEN

  if (!token) {
    return res.status(401).json({ error: 'Access token required' });
  }

  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) {
      return res.status(403).json({ error: 'Invalid or expired token' });
    }
    req.user = user;
    next();
  });
};

// Authorization middleware for role-based access
const authorizeRole = (requiredRole) => {
  return (req, res, next) => {
    if (!req.user || req.user.role !== requiredRole) {
      return res.status(403).json({ error: 'Insufficient permissions' });
    }
    next();
  };
};

// Protected endpoint with authentication
app.post('/api/execute-tool', authenticateToken, (req, res) => {
  const { toolName, params } = req.body;

  // User identity is verified via JWT token
  console.log(`Tool execution requested by user: ${req.user.id}`);

  const result = executeTool(toolName, params);
  res.json({ success: true, result });
});
```

```
// Protected endpoint with role-based authorization
app.delete('/api/config',
  authenticateToken,
  authorizeRole('admin'),
  (req, res) => {
    // Only authenticated admin users can delete configuration
    console.log(`Config deleted by admin: ${req.user.id}`);
    deleteServerConfig();
    res.json({ success: true, message: 'Configuration deleted' });
  }
);

app.listen(3000);
```

1.8 Remediation Steps

1. **Implement Token-Based Authentication**: Use industry-standard JWT (JSON Web Tokens) or OAuth 2.0 for all API endpoints. Ensure tokens are generated only after successful credential validation and include expiration times (typically 15-60 minutes for access tokens).
2. **Enforce Authentication Middleware**: Create middleware that validates authentication tokens on every protected endpoint. This middleware should check token validity, expiration, and signature before allowing access to sensitive functions.
3. **Implement Role-Based Access Control (RBAC)**: Define user roles with specific permissions and validate that authenticated users have the required role/permissions for each operation. Document which roles can access which endpoints.
4. **Use HTTPS and Secure Token Storage**: Always transmit authentication credentials over HTTPS to prevent interception. Store tokens securely (encrypted) and implement proper token rotation policies. Never store sensitive credentials in code or configuration files.

1.9 References

- OWASP Authentication Cheat Sheet
- NIST SP 800-63B: Authentication and Lifecycle Management
- CWE-287: Improper Authentication
- JWT.io: Introduction to JSON Web Tokens
- PCI DSS Requirement 8: Identification and Authentication

Last Updated: January 2026
Status: Published
Language: English