

0xJacky / nginx-ui Public

&lt;&gt; Code Issues 88 Pull requests 4 Discussions Actions Projects

# Race Condition Leads to Persistent Data Corruption and Service Collapse

**High** 0xJacky published GHSA-m468-xcm6-fxg4 3 days ago

## Package

[/0xJacky/nginx-ui](#) (Go)

## Affected versions

&lt; = 2.3.3

## Patched versions

2.3.4

[github.com/uozitech/cosy](#) (Go)

&lt; 1.30.0

1.30.1

## Description

### Summary

The `nginx-ui` application is vulnerable to a **Race Condition**. Due to the complete absence of synchronization mechanisms (Mutex) and non-atomic file writes, concurrent requests lead to the severe corruption of the primary configuration file ( `app.ini` ). This vulnerability results in a persistent Denial of Service (DoS) and introduces a non-deterministic path for **Remote Code Execution (RCE)** through configuration cross-contamination.

### Details

The vulnerability exists because the settings update pipeline does not implement any synchronization primitives. When multiple requests reach the handler simultaneously:

- Memory Corruption:** `ProtectedFill()` modifies shared global singleton pointers without thread-safety, leading to inconsistent states in memory.
- File Corruption:** The underlying library ( `gopkg.in/ini.v1` ) performs direct overwrites. Concurrent write operations interleave at the OS level, resulting in `app.ini` files with empty leading lines, truncated fields, or partially overwritten configuration keys.
- State Persistent Failure:** Depending on which bytes are corrupted, the application either fails its "is-installed" check (redirecting to `/install` ) or encounters a fatal error during boot/runtime that





```

2026-03-14 09:36:39.650 FATAL github.com/0xJacky/Nginx-UI/settings/server_v1.go:110 setting.init, fail to parse 'app.ini': key-value delimiter not found: nginx]
NGINX_UI_WORKING_DIR=/var/run/
HOSTNAME=047dcd55b7df
HOME=/root
PKG_RELEASE=1-trixie
NGINX_UI_OFFICIAL_DOCKER=true
DYNPKG_RELEASE=1-trixie
ACME_VERSION=0.3.1
TERM=xterm
NGINX_VERSION=1.29.5
PATH=/command:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
NJS_VERSION=0.9.5
NJS_RELEASE=1-trixie
DEBIAN_FRONTEND=noninteractive
PWD=/run/s6-rc:s6-rc-init:eiEiFk/servicedirs/nginx-ui
]
#
root@047dcd55b7df:/etc/nginx-ui# cat app.ini
[cert]
Email = admin@admin.com
CADir =
RenewalInterval = 7
RecursiveNameservers =
HTTPChallengePort = 9180

[app]

[server]

[database]

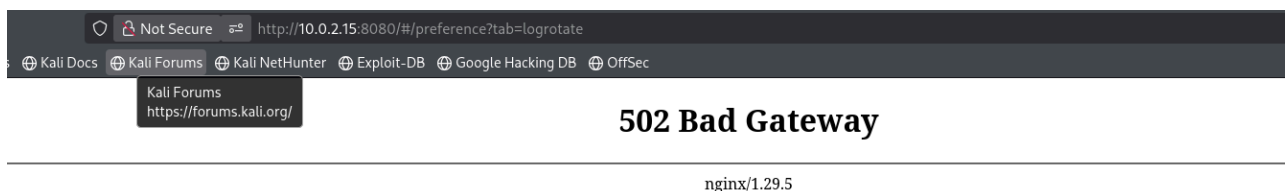
[log]

[sls]

[auth]

```

- Observation (Recovery Failure):** If the service redirects to `/install`, attempting to complete the setup again often fails because the underlying configuration state is too corrupted to be reconciled by the installer logic.
- Observation (Total Service Collapse):** When the corruption in `app.ini` becomes so severe, the Go runtime or the INI parser encounters a fatal error, causing the Nginx-UI service to stop responding entirely (Hard DoS).



- Observation (Cross-Section Contamination):** During testing, it was observed that sometimes INI sections become interleaved. For example, fields belonging to the `[nginx]` section (like `ConfigDir` or `ReloadCmd`) were erroneously written under the `[webauthn]` section.

**Example of corrupted output observed:**

```

[webauthn]
RPDisplayName =
RPID =
RPOrigins =

```



```
gDirWhiteList =
ConfigDir      = /etc/nginx
ConfigPath    =
PIDPath       = /run/nginx.pid
SbinPath      =
TestConfigCmd =
ReloadCmd     = nginx -s reload
RestartCmd    = nginx -s stop
StubStatusPort = 51820
ContainerName =
```

## Impact

This is a **High** security risk (CWE-362: Race Condition).

- **Integrity:** Permanent corruption of application settings and system-level configuration.
- **Availability:** High. The attack results in a persistent Denial of Service that cannot be recovered via the web UI.
- **Remote Code Execution (RCE) Risk:** Since the application allows updating certain fields (like Node Name) and uses others as shell commands (like ReloadCmd or RestartCmd), the observed "cross-contamination" of INI values means an attacker could potentially force a user-controlled string into a command execution field. If ReloadCmd is overwritten with a malicious payload provided in another field, the next nginx reload will execute that payload. While highly impactful, this specific exploit path is non-deterministic and depends on the precise interleaving of thread execution, making targeted exploitation difficult.

## Recommended Mitigation

1. **Implement Mutex Locking:** Wrap the `ProtectedFill` and `settings.Save()` calls in a `sync.Mutex` to serialize access to global settings.
2. **Atomic File Writes:** Implement a "write-then-rename" strategy. Write the new configuration to `app.ini.tmp` and use `os.Rename()` to replace the original file atomically, ensuring the configuration file is always in a valid state.

### Severity

High 7.1 / 10

#### CVSS v4 base metrics

#### Exploitability Metrics

Attack Vector	Network
Attack Complexity	Low
Attack Requirements	None

Privileges Required High

User interaction None

**Vulnerable System Impact Metrics**

Confidentiality Low

Integrity High

Availability High

**Subsequent System Impact Metrics**

Confidentiality None

Integrity None

Availability None

[Learn more about base metrics](#)

CVSS:4.0/AV:N/AC:L/AT:N/PR:H/UI:N/VC:L/VI:H/VA:H/SC:N/SI:N/SA:N


**CVE ID**

CVE-2026-33028

**Weaknesses**

▶ CWE-362

**Credits**

 **dapickle**

Reporter