Talos Vulnerability Report

TALOS-2022-1545

WWBN AVideo password hash improper authentication vulnerability

AUGUST 16, 2022

CVE NUMBER

CVE-2022-32282

SUMMARY

An improper password check exists in the login functionality of WWBN AVideo 11.6 and dev master commit 3f7c0364. An attacker that owns a users' password hash will be able to use it to directly login into the account, leading to increased privileges.

CONFIRMED VULNERABLE VERSIONS

The versions below were either tested or verified to be vulnerable by Talos or confirmed to be vulnerable by the vendor.

WWBN AVideo 11.6
WWBN AVideo dev master commit 3f7c0364

PRODUCT URLS

AVideo - https://github.com/WWBN/AVideo

CVSSV3 SCORE

7.2 - CVSS:3.0/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:H

CWE

CWE-836 - Use of Password Hash Instead of Password for Authentication

DETAILS

AVideo is a web application, mostly written in PHP, that can be used to create an audio/video sharing website. It allows users to import videos from various sources, encode and share them in various ways. Users can sign up to the website in order to share videos, while viewers have anonymous access to the publicly-available contents. The platform provides plugins for features like live streaming, skins, YouTube uploads and more.

AVideo, by default, saves a user's password as a salted hash in the database. When a user tries to login, the function `login`, defined in the `User` class in `objects/user.php`, is called:

```
public function login($noPass = false, $encodedPass = false,
$ignoreEmailVerification = false) {
    if (User::isLogged()) {
        return false;
    }
    ...
    if (strtolower($encodedPass) === 'false') {
        $encodedPass = false;
    }
    //_error_log("user::login: noPass = $noPass, encodedPass = $encodedPass, this-
>user, $this->user " . getRealIpAddr());
    if ($noPass) {
        $user = $this->find($this->user, false, true);
    } else {
        $user = $this->find($this->user, $this->password, true, $encodedPass);
    }
    ...
```

Password check is relayed to the `find` function, passing user and password as arguments:

```
private function find($user, $pass, $mustBeactive = false, $encodedPass = false) {
    ...
    $sql = "SELECT * FROM users WHERE user = ? ";
    ...
    $sql .= " LIMIT 1";
    $res = sqlDAL::readSql($sql, $formats, $values, true);
    $result = sqlDAL::fetchAssoc($res);
    sqlDAL::close($res);
    if (!empty($result)) {
        if ($pass !== false) {
            if (!encryptPasswordVerify($pass, $result['password'], $encodedPass)) {
// [1]
                if (!empty($advancedCustom) && $advancedCustom-
>enableOldPassHashCheck) {
                    _error_log("Password check new hash pass does not match, trying
MD5");
                    return $this->find_Old($user, $pass, $mustBeactive,
$encodedPass);
                } else {
                    return false;
                }
            }
        }
        $user = $result;
    } else {
        _error_log("Password check new hash user not found");
        //check if is the old password style
        $user = false;
        //$user = false;
    }
    return $user;
```

The function fetches the user record from the database, and uses `encryptPasswordVerify` to check if the password matches:

```
function encryptPasswordVerify($password, $hash, $encodedPass = false) {
    global $advancedCustom, $global;
    if (!$encodedPass || $encodedPass === 'false') {
        //_error_log("encryptPasswordVerify: encrypt");
        $passwordSalted = encryptPassword($password);
        // in case you enable the salt later
        $passwordUnSalted = encryptPassword($password, true);
    } else {
        //_error_log("encryptPasswordVerify: do not encrypt");
        $passwordSalted = $password;
        // in case you enable the salt later
        $passwordUnSalted = $password;
    }
    //_error_log("passwordSalted = $passwordSalted,  hash=$hash,
passwordUnSalted=$passwordUnSalted");
    return $passwordSalted === $hash || $passwordUnSalted === $hash || $password ===
$hash; // [2]
}
```

The password is encrypted in the same way it was stored in the database (using `encryptPassword`), however this function allows 3 different passwords to match: the salted hash of `$password`, the unsalted hash of `$password`, and whatever hash is stored in the database (the `$password === $hash`). This means that the hash as stored in the database can be used as a password to login. Any SQL injection vulnerability can thus be used to dump the administrator's password hash and use that exact hash to login via the web interface.

VENDOR RESPONSE

Vendor confirms issues fixed on July 7th 2022

TIMELINE

2022-06-29 - Initial Vendor Contact
2022-07-05 - Vendor Disclosure
2022-07-07 - Vendor Patch Release
2022-08-16 - Public Release

CREDIT

Discovered by Claudio Bozzato of Cisco Talos.
