WPScan report bash script

Created by Tom Carrigan

Created on 10 September 2021

Assisted and documentation by Regan van Eijk

Tested by Tom Carrigan and Regan van Eijk

Created and tested in Ubuntu subsystem 20.04.3 LTS

This bash script was created to provide easier report creation from WPScans, allowing further automation of information formatting, reducing redundant data and displaying required output information from a WPScan.

When using WPScan, the current results output varies depending on if an output type was selected, but the information not displayed in the most user-friendly way regardless. It typically looks something like this:

And in json format, it looks like this:

```
{
  "banner": {
    "description": "WordPress Security Scanner by the WPScan Team",
    "version": "3.8.18",
    "authors": [
      "@_WPScan_",
      "@ethicalhack3r",
      "@erwan_lr",
      "@firefart"
    ],
    "sponsor": "Sponsored by Automattic - https://automattic.com/"
  },
  "start_time": 1631843162,
  "start_memory": 48766976,
  "target_url": "http://www.         .com/",
  "target_ip": "           ",
  "effective_url": "https://www.         .com/",
  "interesting_findings": [
    {
      "url": "https://www.         .com/",
      "to_s": "Headers",
      "type": "headers",
      "found_by": "Headers (Passive Detection)",
      "confidence": 100,
      "confirmed_by": {

      },
      "references": {

      },
      "interesting_entries": [
        "referrer-policy: no-referrer-when-downgrade",
        "x-platform: DM",
        "cf-cache-status: DYNAMIC",
        "expect-ct: max-age=604800, report-uri=\"https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct\"",
        "server: cloudflare",
        "cf-ray: 68fea8a1dbd7559f-SYD"
      ]
    },
    {
      "url": "https://www.         .com/xmlrpc.php",
      "to_s": "XML-RPC seems to be enabled: https://www.         .com/xmlrpc.php",
      "type": "xmlrpc",
      "found_by": "Link Tag (Passive Detection)",
      "confidence": 100,
      "confirmed_by": {
        "Direct Access (Aggressive Detection)": {
          "confidence": 100
        }
      },
      "references": {
        "url": [
          "http://codex.wordpress.org/XML-RPC_Pingback_API"
        ],
        "metasploit": [
          "auxiliary/scanner/http/wordpress_ghost_scanner",
          "auxiliary/dos/http/wordpress_xmlrpc_dos",
          "auxiliary/scanner/http/wordpress_xmlrpc_login",
          "auxiliary/scanner/http/wordpress_pingback_access"
        ]
      },
      "interesting_entries": [

      ]
    },
    {
      "url": "http://www.         .com/wp-content/mu-plugins/",
      "to_s": "This site has 'Must Use Plugins': http://www.         .com/wp-content/mu-plugins/",
      "type": "mu_plugins",
      "found_by": "URLs In Homepage (Passive Detection)",
      "confidence": 100,
      "confirmed_by": {
        "Direct Access (Aggressive Detection)": {
          "confidence": 80
        }
      },
      "references": {
        "url": [
          "http://codex.wordpress.org/Must_Use_Plugins"
        ]
      },
      "interesting_entries": [

      ]
    }
  ],
```


And that is only some of the information provided. Most of it is helpful information as a whole but also contains little bits of useless information everywhere. So this bash script was created to help reduce redundant information output.

Dependencies:

- WPScan

The WPScan CLI tool is a free, for non-commercial use, black box WordPress security scanner written for security professionals and blog maintainers to test the security of their sites.

- jq

jq is a tool for processing JSON inputs, applying the given filter to its JSON text inputs and producing the filter's results as JSON on standard output.

The Bash

```
#!/bin/bash
args=("$@")
sudo wpscan --ignore-main-redirect --rua --url ${args[0]} -e vp --api-token ${args[1]} --output ${args[2]} --format json
jq -r '.target_url, .target_ip' ${args[2]} | tee >> apireport${args[2]}
echo "Enumerated Plugins and their Vulnerabilities" | tee >> apireport${args[2]}
jq '.plugins[] | [.slug,.vulnerabilities]' ${args[2]} | tee >> apireport${args[2]}
sed -i '/wpvulndb.*/,+2d' apireport${args[2]}
sed -i '/references.*/d' apireport${args[2]}
```

The breakdown

First, This line is required for the OS to recognize the following script as a bash:

```
#!/bin/bash
```

This line establishes output arguments (used as variables) will be required for this script:

```
args=(“$@”)
```

This line has multiple functions used;

The --ignore-main-redirect tells the WPScan to ignore automatic redirects, such as redirects from http to https.

- The --rua (--random-user-agent) tells the WPScan to use a random user-agent for each scan.
- --url is the target to be scanned.
- The ${arg[0]} immediately after --url will turn the input target url into variable 0.
- e to enumerate (add vp to state enumeration for vulnerable plugins or ap for all plugins).
- --api-token to add an api-token the user has generated so they can list vulnerabilities.
- The ${args[1]} immediately after will turn the api-token input into variable 1.
- --output to save results as a separate file.
- The ${args[2]} turns the name of the output file into variable 2.
- --format json after --output(variable) means the output format will be in json format.

```
sudo wpscan --ignore-main-redirect --rua --url ${args[0]} -e vp --api-token ${args[1]} --output ${args[2]} --format json
```

This line uses jq to read output data (‘.target_url, .target_ip’ as raw strings from ${args[2]} using -r and extract the data to an output file (tee >> apireprt${args[2]}):

```
jq -r ‘.target_url, .target_ip’ ${args[2]} | tee >> apireport${args[2]}
```

This line adds the string to the previously created output file using an echo command:

```
echo “Enumerated Plugins and their Vulnerabilities” | tee >> apireport${args[2]}
```

This line uses jq to extract from the plugins list (.plugins[]) the specified plugins ([.slug,vulnerabilities]) and insert them into the previously created output file (apireport${args[2]}):

```
jq '.plugins[] | [slug,.vulnerabilities]' ${args[2]} | tee >> apireport${args[2]}
```

These next 2 lines use sed to remove unnecessary information from the created output file, the first line finds the requested data and deletes it plus 2 lines (+2d), the second line finds the requested data and deletes it for that line only:

```
sed -i ‘/wpvulndb.*/,+2d’ apireport${args[2]}
sed -i ‘/references.*/d’ apireport${args[2]}
```

Why the variables are as they are

- The ${args[]} can be numbered however the user wants, as long as the relative positioning stays the same. ${args[0] can be ${args[3]} as long as all relative args calls are changed with it etc.
- The file created which contains the requested data will be named apireport, as this was specified at the apireport${args[2]} variable. The name can be changed to the users choice by changing the word before ${args[2]}, example; endreport${args[2]} makes the filename endreport, scan${args[2]} makes the filename scan etc.
- Because ${args[0]} is the target url, this cannot be changed mid report, hence why it is only used at the beginning of the script. The same applies to ${args[1]} as this is the api-token used by the user. Changing these will disrupt the bash and cause errors EXCEPT when using the scan without an api-token.

If the user wanted to use the bash WITHOUT an api-token so they only see plugins with issues, but NOT see specific vulnerabilities, follow these steps:

1. Remove --api-token ${args[1] from line 9,
2. For EVERY instance of ${args[2]}, change them to ${args[1]}

Output Result

The end result is two files, one WPScan in json format with the full scan details, and the other containing just information specified in the bash script.

The filtered output file jumps straight into required information and should look something like this (Target URL and IP information has been removed):

Summary

This bash can be altered on a use-by-use basis if required to search for specific sections of data to suit individual user needs. The bash can also be expanded to suit individual needs as well.