## CONTENTS:

1 **CVE Binary Tool quick start / README**  
1.1 Installing CVE Binary Tool ........................................ 4  
1.2 Most popular usage options ......................................... 4  
1.3 Output Options ...................................................... 5  
1.4 Full option list ...................................................... 5  
1.5 Configuration ....................................................... 7  
1.6 Using CVE Binary Tool in GitHub Actions ......................... 7  
1.7 Binary checker list ................................................ 8  
1.8 Language Specific checkers ....................................... 8  
1.9 Limitations ......................................................... 9  
1.10 Requirements ..................................................... 9  
1.11 Feedback & Contributions ...................................... 10  
1.12 Security Issues .................................................. 10  

2 **CVE Binary Tool User Manual**  
2.1 How it works ....................................................... 14  
2.2 Installing .......................................................... 15  
2.3 Fixing Known Issues / What should I do if it finds something?  15  
2.4 Limitations ......................................................... 16  
2.5 Optional Arguments ............................................... 16  
2.6 CVE Data Download Arguments ................................ 16  
2.7 Checkers Arguments .............................................. 17  
2.8 Input Arguments .................................................. 17  
2.9 Output Arguments ................................................ 23  
2.10 Merge Report Arguments ....................................... 26  
2.11 Deprecated Arguments ......................................... 28  
2.12 Feedback & Contributions .................................... 28  
2.13 Security Issues .................................................. 28  

3 **CSV2CVE**  
3.1 Running the tool: ................................................ 29  
3.2 Additional Options: ............................................... 29  
3.3 Example .csv file: ............................................... 30  
3.4 Example output: .................................................. 30  

4 **How To Guides**  
4.1 How do I create a custom theme for the HTML Report? ........ 31  
4.2 How to scan a docker image? .................................... 39  
4.3 How do I use CVE Binary Tool in an offline environment? .... 41  
4.4 Best practices for running multiple scans at once ............... 42
The CVE Binary Tool helps you determine if your system includes known vulnerabilities. You can scan binaries for a number of common, vulnerable components (openssl, libpng, libxml2, expat and others), or if you know the components used, you can get a list of known vulnerabilities associated with a list of components and versions.
The CVE Binary Tool is a free, open source tool to help you find known vulnerabilities in software, using data from the National Vulnerability Database (NVD) list of Common Vulnerabilities and Exposures (CVEs).

The tool has two main modes of operation:

1. A binary scanner which helps you determine which packages may have been included as part of a piece of software. There are around 100 checkers which focus on common, vulnerable open source components such as openssl, libpng, libxml2 and expat.

2. Tools for scanning known component lists in various formats, including .csv, several linux distribution package lists, language specific package scanners and several Software Bill of Materials (SBOM) formats.

It is intended to be used as part of your continuous integration system to enable regular vulnerability scanning and give you early warning of known issues in your supply chain.

For more details, see our documentation or this quickstart guide

- **CVE Binary Tool quick start / README**
  - Installing CVE Binary Tool
  - Most popular usage options
    - Finding known vulnerabilities using the binary scanner
    - Finding known vulnerabilities in a list of components
    - Scanning an SBOM file for known vulnerabilities
    - Using the tool offline
  - Output Options
  - Full option list
  - Configuration
  - Using CVE Binary Tool in GitHub Actions
  - Binary checker list
  - Language Specific checkers
    - Java
    - Javascript
    - Python
  - Limitations
1.1 Installing CVE Binary Tool

CVE Binary Tool can be installed using pip:

```
pip install cve-bin-tool
```

You can also do `pip install --user -e .` to install a local copy which is useful if you're trying the latest code from the cve-bin-tool github or doing development. The Contributor Documentation covers how to set up for local development in more detail.

1.2 Most popular usage options

1.2.1 Finding known vulnerabilities using the binary scanner

To run the binary scanner on a directory or file:

```
cve-bin-tool <directory/file>
```

Note that this option will also use any language specific checkers to find known vulnerabilities in components.

1.2.2 Finding known vulnerabilities in a list of components

To scan a comma-delimited (CSV) or JSON file which lists dependencies and versions:

```
cve-bin-tool --input-file <filename>
```

Note that the `--input-file` option can also be used to add extra triage data like remarks, comments etc. while scanning a directory so that output will reflect this triage data and you can save time of re-triaging (Usage: `cve-bin-tool -i=test.csv /path/to/scan`). A VEX file (which may be created using the `--vex` command line option) can also be used as a triage file. A VEX file is detected if the file suffix is `.vex`.

1.2.3 Scanning an SBOM file for known vulnerabilities

To scan a software bill of materials file (SBOM):

```
cve-bin-tool --sbom <sbom_filetype> --sbom-file <sbom_filename>
```

Valid SBOM types are SPDX, CycloneDX, and SWID.
1.2.4 Using the tool offline

Specifying the \texttt{--offline} option when running a scan ensures that cve-bin-tool doesn’t attempt to download the latest database files or to check for a newer version of the tool.

Note that you will need to obtain a copy of the vulnerability data before the tool can run in offline mode. The offline how-to guide contains more information on how to set up your database.

1.3 Output Options

The CVE Binary Tool provides console-based output by default. If you wish to provide another format, you can specify this and a filename on the command line using \texttt{--format}. The valid formats are CSV, JSON, console, HTML and PDF. The output filename can be specified using the \texttt{--output-file} flag.

The reported vulnerabilities can additionally be reported in the Vulnerability Exchange (VEX) format by specifying \texttt{--vex} command line option. The generated VEX file can then be used as an \texttt{--input-file} to support a triage process.

If you wish to use PDF support, you will need to install the reportlab library separately.

If you intend to use PDF support when you install cve-bin-tool you can specify it and report lab will be installed as part of the cve-bin-tool install:

\begin{verbatim}
pip install cve-bin-tool[PDF]
\end{verbatim}

If you’ve already installed cve-bin-tool you can add reportlab after the fact using pip:

\begin{verbatim}
pip install --upgrade reportlab
\end{verbatim}

Note that reportlab was taken out of the default cve-bin-tool install because it has a known CVE associated with it (CVE-2020-28463). The cve-bin-tool code uses the recommended mitigations to limit which resources added to PDFs, as well as additional input validation. This is a bit of a strange CVE because it describes core functionality of PDFs: external items, such as images, can be embedded in them, and thus anyone viewing a PDF could load an external image (similar to how viewing a web page can trigger external loads). There’s no inherent “fix” for that, only mitigations where users of the library must ensure only expected items are added to PDFs at the time of generation.

Since users may not want to have software installed with an open, unfixable CVE associated with it, we’ve opted to make PDF support only available to users who have installed the library themselves. Once the library is installed, the PDF report option will function.

1.4 Full option list

Usage: cve-bin-tool <directory/file to scan>

\begin{verbatim}
optional arguments:
-h, --help               show this help message and exit
-e, --exclude            exclude path while scanning
-V, --version            show program's version number and exit
--disable-version-check  skips checking for a new version
--disable-validation-check skips checking xml files against schema
--offline                operate in offline mode
\end{verbatim}

(continues on next page)
CVE Data Download:
- `n {json,api}`, `--nvd {json,api}`
  choose method for getting CVE lists from NVD
- `u {now,daily,never,latest}`, `--update {now,daily,never,latest}`
  update schedule for NVD database (default: daily)
- `--nvd-api-key NVD_API_KEY`
  specify NVD API key (used to improve NVD rate limit)

Input:
- `directory` directory to scan
- `-i INPUT_FILE`, `--input-file INPUT_FILE`
  provide input filename
- `-C CONFIG`, `--config CONFIG`
  provide config file
- `-L PACKAGE_LIST`, `--package-list PACKAGE_LIST`
  provide package list
- `--sbom {spdx,cyclonedx,swid}`
  specify type of software bill of materials (sbom)
  (default: spdx)
- `--sbom-file SBOM_FILE`
  provide sbom filename

Output:
- `-q`, `--quiet`
  suppress output
- `-l {debug,info,warning,error,critical}`, `--log {debug,info,warning,error,critical}`
  log level (default: info)
- `--html-theme HTML_THEME`
  provide custom theme directory for HTML Report
- `--f {csv,json,console,html,pdf}`, `--format {csv,json,console,html,pdf}`
  update output format (default: console)
- `-c CVSS`, `--cvss CVSS`
  minimum CVSS score (as integer in range 0 to 10) to report (default: 0)
- `-S {low,medium,high,critical}`, `--severity {low,medium,high,critical}`
  minimum CVE severity to report (default: low)
- `--report` Produces a report even if there are no CVE for the respective output format
- `-A [<distro_name>-<distro_version_name>]`, `--available-fix [<distro_name>-<distro_version_name>]`
  Lists available fixes of the package from Linux distribution
- `-b [<distro_name>-<distro_version_name>]`, `--backport-fix [<distro_name>-<distro_version_name>]`
  Lists backported fixes if available from Linux distribution
- `--affected-versions` Lists versions of product affected by a given CVE (to facilitate upgrades)
- `--vex VEX` Provide vulnerability exchange (vex) filename

Merge Report:
- `--append INTERMEDIATE_PATH` provide path for saving intermediate report
- `--reports` provide a tag to differentiate between multiple intermediate reports
For further information about all of these options, please see the CVE Binary Tool user manual.

Note: For backward compatibility, we still support csv2cve command for producing CVEs from csv but we recommend using the --input-file command going forwards.

-L or --package-list option runs a CVE scan on installed packages listed in a package list. It takes a python package list (requirements.txt) or a package list of packages of systems that has dpkg, pacman or rpm package manager as an input for the scan. This option is much faster and detects more CVEs than the default method of scanning binaries.

You can get a package list of all installed packages in

- a system using dpkg package manager by running `dpkg-query -W -f '${binary:Package}\n' > pkg-list`
- a system using pacman package manager by running `pacman -Qqe > pkg-list`
- a system using rpm package manager by running `rpm -qa --queryformat '%{NAME}\n' > pkg-list`

in the terminal and provide it as an input by running `cve-bin-tool -L pkg-list` for a full package scan.

### 1.5 Configuration

You can use --config option to provide configuration file for the tool. You can still override options specified in config file with command line arguments. See our sample config files in the test/config

### 1.6 Using CVE Binary Tool in GitHub Actions

If you want to integrate cve-bin-tool as a part of your github action pipeline. You can checkout our example github action.
1.7 Binary checker list

The following checkers are available for finding components in binary files:

<table>
<thead>
<tr>
<th>Accountsservice</th>
<th>avahi</th>
<th>bash</th>
<th>bind</th>
<th>binutils</th>
<th>bolt</th>
<th>bubblewrap</th>
</tr>
</thead>
<tbody>
<tr>
<td>busybox</td>
<td>bzip2</td>
<td>cronie</td>
<td>cryptsetup</td>
<td>cups</td>
<td>curl</td>
<td>dbus</td>
</tr>
<tr>
<td>dnsmasq</td>
<td>dovecot</td>
<td>dpkg</td>
<td>enscript</td>
<td>expat</td>
<td>ffmpeg</td>
<td>freeradius</td>
</tr>
<tr>
<td>ftp</td>
<td>gcc</td>
<td>gimp</td>
<td>glibc</td>
<td>gnomeshell</td>
<td>gnupg</td>
<td>gnuuts</td>
</tr>
<tr>
<td>gpgme</td>
<td>gstreamer</td>
<td>gnutp</td>
<td>haproxy</td>
<td>hdf5</td>
<td>hostapd</td>
<td>hunspell</td>
</tr>
<tr>
<td>icecast</td>
<td>icu</td>
<td>irssi</td>
<td>kbd</td>
<td>kerberos</td>
<td>kexectools</td>
<td>libarchive</td>
</tr>
<tr>
<td>libbpg</td>
<td>libdb</td>
<td>libebml</td>
<td>libgcrypt</td>
<td>libical</td>
<td>libjpeg_turbo</td>
<td>liblas</td>
</tr>
<tr>
<td>libnss</td>
<td>librsyg</td>
<td>libsec-comp</td>
<td>libnsndfile</td>
<td>libsvolv</td>
<td>libsoup</td>
<td>libsrtp</td>
</tr>
<tr>
<td>libssh2</td>
<td>libtiff</td>
<td>libvirt</td>
<td>libvncserver</td>
<td>libxslt</td>
<td>lightpd</td>
<td>logrotate</td>
</tr>
<tr>
<td>lua</td>
<td>mariadb</td>
<td>mdadm</td>
<td>memcached</td>
<td>mtr</td>
<td>mysql</td>
<td>nano</td>
</tr>
<tr>
<td>ncurses</td>
<td>nessus</td>
<td>netpbm</td>
<td>nginx</td>
<td>node</td>
<td>ntp</td>
<td>open_vm_tools</td>
</tr>
<tr>
<td>openafs</td>
<td>openjpeg</td>
<td>openldap</td>
<td>openssl</td>
<td>openswan</td>
<td>openswan</td>
<td>openvpn</td>
</tr>
<tr>
<td>p7zip</td>
<td>pcsc_lite</td>
<td>pizg</td>
<td>png</td>
<td>polarssl_fedora</td>
<td>poppler</td>
<td>postgresql</td>
</tr>
<tr>
<td>pppp</td>
<td>python</td>
<td>qt</td>
<td>radare2</td>
<td>rsyslog</td>
<td>samba</td>
<td>sane_backends</td>
</tr>
<tr>
<td>sqlite</td>
<td>strongswan</td>
<td>subversion</td>
<td>sudo</td>
<td>syslogng</td>
<td>systemd</td>
<td>tcpcpump</td>
</tr>
<tr>
<td>trousers</td>
<td>varnish</td>
<td>webkitgtk</td>
<td>wireshark</td>
<td>wpa_supplicant</td>
<td>xerces</td>
<td>xml2</td>
</tr>
<tr>
<td>zsh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All the checkers can be found in the checkers directory, as can the instructions on how to add a new checker. Support for new checkers can be requested via GitHub issues.

1.8 Language Specific checkers

A number of checkers are available for finding vulnerable components in specific language packages.

1.8.1 Java

The scanner examines the pom.xml file within a Java package archive to identify Java components. The package names and versions within the archive are used to search the database for vulnerabilities. JAR, WAR and EAR archives are supported.
1.8.2 Javascript

The scanner examines the package-lock.json file within a javascript application to identify components. The package names and versions are used to search the database for vulnerabilities.

1.8.3 Python

The scanner examines the PKG-INFO and METADATA files for an installed Python package to extract the component name and version which are used to search the database for vulnerabilities.

The tool supports the scanning of the contents of any Wheel package files (indicated with a file extension of .whl) and egg package files (indicated with a file extension of .egg).

The --package-list option can be used with a Python dependencies file requirements.txt to find the vulnerabilities in the list of components.

1.9 Limitations

This scanner does not attempt to exploit issues or examine the code in greater detail; it only looks for library signatures and version numbers. As such, it cannot tell if someone has backported fixes to a vulnerable version, and it will not work if library or version information was intentionally obfuscated.

This tool is meant to be used as a quick-to-run, easily-automatable check in a non-malicious environment so that developers can be made aware of old libraries with security issues that have been compiled into their binaries.

The tool does not guarantee that any vulnerabilities reported are actually present or exploitable, neither is it able to find all present vulnerabilities with a guarantee.

Users can add triage information to reports to mark issues as false positives, false negatives, indicate that the risk has been mitigated by configuration/usage changes, and so on.

Triage details can be re-used on other projects so, for example, triage on a Linux base image could be applied to multiple containers using that image.

For more information and usage of triage information with the tool kindly have a look here.

If you are using the binary scanner capabilities, be aware that we only have a limited number of binary checkers (see table above) so we can only detect those libraries. Contributions of new checkers are always welcome! You can also use an alternate way to detect components (for example, a bill of materials tool such as tern) and then use the resulting list as input to cve-bin-tool to get a more comprehensive vulnerability list.

The tool uses a vulnerability database in order to detect the present vulnerabilities, in case the database is not frequently updated (specially if the tool is used in offline mode), the tool would be unable to detect any newly discovered vulnerabilities. Hence it is highly advised to keep the database updated.

1.10 Requirements

To use the auto-extractor, you may need the following utilities depending on the type of file you need to extract. The utilities below are required to run the full test suite on Linux:

- file
- strings
- tar
• unzip
• rpm2cpio
• cpio
• ar
• cabextract

Most of these are installed by default on many Linux systems, but cabextract and rpm2cpio in particular might need to be installed.

On windows systems, you may need:
• ar
• 7z
• Expand
• pdftotext

Windows has ar and Expand installed by default, but 7z in particular might need to be installed. If you want to run our test-suite or scan a zstd compressed file, We recommend installing this 7-zip-zstd fork of 7zip. We are currently using 7z for extracting jar, apk, msi, exe and rpm files.

If you get an error about building libraries when you try to install from pip, you may need to install the Windows build tools. The Windows build tools are available for free from https://visualstudio.microsoft.com/visual-cpp-build-tools/

If you get an error while installing brotlipy on Windows, installing the compiler above should fix it.

pdftotext is required for running tests. (users of cve-bin-tool may not need it, developers likely will.) The best approach to install it on Windows involves using conda (click here for further instructions).

You can check our CI configuration to see what versions of python we’re explicitly testing.

1.11 Feedback & Contributions

Bugs and feature requests can be made via GitHub issues. Be aware that these issues are not private, so take care when providing output to make sure you are not disclosing security issues in other products.

Pull requests are also welcome via git.
• New contributors should read the contributor guide to get started.
• Folk who already have experience contributing to open source projects may not need the full guide but should still use the pull request checklist to make things easy for everyone.

1.12 Security Issues

Security issues with the tool itself can be reported to Intel’s security incident response team via https://intel.com/security.

If in the course of using this tool you discover a security issue with someone else’s code, please disclose responsibly to the appropriate party.
CHAPTER TWO

CVE BINARY TOOL USER MANUAL

• CVE Binary Tool User Manual
  – How it works
  – Installing
  – Fixing Known Issues / What should I do if it finds something?
  – Limitations
  – Optional Arguments
    * -e EXCLUDE, –exclude EXCLUDE
    * -h, –help
    * -V, –version
    * –disable-version-check
    * –disable-validation-check
  – CVE Data Download Arguments
    * -u {now,daily,never,latest}, –update {now,daily,never,latest}
    * -n [json,api], –nvd [json,api]
    * –nvd-api-key NVD_API_KEY
  – Checkers Arguments
    * -s SKIPS, –skips SKIPS
    * -r CHECKERS, –runs CHECKERS
  – Input Arguments
    * directory (positional argument)
    * -i INPUT_FILE, –input-file INPUT_FILE
    * -L PACKAGE_LIST, –package-list PACKAGE_LIST
    * -C CONFIG, –config CONFIG
      · Yaml example file
      · Toml example file
  – Output Arguments
    * -o OUTPUT_FILE, –output-file OUTPUT_FILE
The CVE Binary Tool scans for a number of common, vulnerable open source components like openssl, libpng, libxml2, expat etc. to let you know if a given directory or binary file includes common libraries with known vulnerabilities, known as CVEs (Common Vulnerabilities and Exposures).

Usage: cve-bin-tool

You can also do python -m cve_bin_tool.cli which is useful if you’re trying the latest code from the cve-bin-tool github.

optional arguments:
-\(h\), --help show this help message and exit
-\(e\), --exclude exclude path while scanning
-\(V\), --version show program's version number and exit
-\(x\), --extract autoextract compressed files
--disable-version-check skips checking for a new version
--disable-validation-check skips checking xml files against schema

CVE Data Download:
-\(n\) {json,api}, --nvd {json,api} choose method for getting CVE lists from NVD
-\(u\) {now,daily,never,l latest}, --update {now,daily,never,l latest}

(continues on next page)
update schedule for NVD database (default: daily)
--nvd-api-key NVD_API_KEY
    specify NVD API key (used to improve NVD rate limit)

Input:
    directory directory to scan
    -i INPUT_FILE, --input-file INPUT_FILE
        provide input filename
    -L PACKAGE_LIST, --package-list PACKAGE_LIST
        provide package list
    -C CONFIG, --config CONFIG
        provide config file

Output:
    -q, --quiet suppress output
    -l {debug,info,warning,error,critical}, --log {debug,info,warning,error,critical}
        log level (default: info)
    -o OUTPUT_FILE, --output-file OUTPUT_FILE
        provide output filename (default: output to stdout)
    --html-theme HTML_THEME
        provide custom theme directory for HTML Report
    -f {csv,json,console,html}, --format {csv,json,console,html}
        update output format (default: console)
    -c CVSS, --cvss CVSS minimum CVSS score (as integer in range 0 to 10) to report (default: 0)
    -S {low,medium,high,critical}, --severity {low,medium,high,critical}
        minimum CVE severity to report (default: low)
    --report
        Produces a report even if there are no CVE for the respective output format
    -A [<distro_name>-<distro_version_name>], --available-fix [<distro_name>-<distro_version_name>]
        Lists available fixes of the package from Linux distribution
    -b [<distro_name>-<distro_version_name>], --backport-fix [<distro_name>-<distro_version_name>]
        Lists backported fixes if available from Linux distribution
    --affected-versions
        Lists versions of product affected by a given CVE (to facilitate upgrades)
    --vex VEX
        Provide vulnerability exchange (vex) filename

Merge Report:
    -a INTERMEDIATE_PATH, --append INTERMEDIATE_PATH
        provide path for saving intermediate report
    -t TAG, --tag TAG
        provide a tag to differentiate between multiple intermediate reports
    -m INTERMEDIATE_REPORTS, --merge INTERMEDIATE_REPORTS
        comma separated intermediate reports path for merging
    -F TAGS, --filter TAGS
        comma separated tags to filter out intermediate reports

Checkers:
    -s SKIPS, --skips SKIPS
        comma-separated list of checkers to disable
    -r RUNS, --runs RUNS
        comma-separated list of checkers to enable
**2.1 How it works**

This scanner looks at the strings found in binary files to see if they match vulnerable versions of a small set of popular open source libraries.

It does not attempt to exploit issues or examine code in greater detail. As such, it cannot tell if someone has backported fixes to an otherwise vulnerable version, it merely provides a mapping between strings, versions, and known CVEs.

A list of currently available checkers can be found in the checkers directory or using cve-bin-tool --help command, as can the instructions on how to add a new checker. Support for new checkers can be requested via GitHub issues. (Please note, you will need to be logged in to add a new issue.)

This tool gives a list of CVE numbers. For those not familiar with the process, these can be looked up using a number of different tools, such as the vulnerability search on the CVE Details website. Each CVE field contains a short summary of the issue, a set of severity scores that are combined to make a CVSS score, a list of products known to be affected, and links to more information (which may include links to sample exploits as well as patches to fix the issue).
2.2 Installing

cve-bin-tool can be installed via pip. If your PATH environment variable is properly configured, installation will result in cve-bin-tool being accessible globally. If not you can treat cve-bin-tool as python -m cve_bin_toolcli.

```bash
pip install -U cve-bin-tool
```

If you want the latest and greatest between releases you can grab from GitHub.

```bash
pip install -U git+https://github.com/intel/cve-bin-tool
```

CVE Binary Tool relies on a few command line utilities which are usually present on GNU/Linux systems but you may need to install.

- `file`
- `strings`
- `tar`
- `unzip`
- `rpm2cpio`
- `cpio`
- `ar`
- `cabextract`

On Windows, it requires

- `ar`
- `7z`
- `Expand`

Windows has ar and Expand installed in default, but 7z in particular might need to be installed. If you want to run our test-suite or scan a zstd compressed file, We recommend installing this 7-zip-zstd fork of 7zip. We are currently using 7z for extracting jar, apk, msi, exe and rpm files.

2.3 Fixing Known Issues / What should I do if it finds something?

The most recommended way to fix a given CVE is to upgrade the package to a non-vulnerable version. Ideally, a CVE is only made public after a fix is available, although this is not always the case.

If this is not possible for some reason, search for the CVE number to get information on possible workarounds and patches that could be backported to other versions. Note that neither workarounds nor backported fixes can be detected by this tool, so your binary will continue to show up as vulnerable even though it may now be safely mitigated and result in a false positive. To avoid this problem, we recommend classifying CVE as Mitigated as explained in the Input section.
2.4 Limitations

The last release of this tool to support python 2.7 is 0.3.1. Please use python 3.7+ for development and future versions. Linux and Windows are supported, as is usage within cygwin on windows. This tool does not scan for all possible known public vulnerabilities, it only scans for specific commonly vulnerable open source components. A complete list of currently supported library checkers can be found in the checkers directory. As the name implies, this tool is intended for use with binaries. If you have access to a known list of product names and versions, we do have an option --input-file that can be used to look up known vulnerabilities given a CSV or JSON file. See the detailed description of --input-file for more details.

2.5 Optional Arguments

2.5.1 -e EXCLUDE, --exclude EXCLUDE

This option allows one the skip a comma-separated lists of paths. This can be useful for excluding certain files and directories from the scan which will also decrease the scanning time.

2.5.2 -h, --help

This option shows a help message and exits.

2.5.3 -V, --version

This option shows program’s version number and exits.

2.5.4 --disable-version-check

This option skips checking for a new version of the program.

2.5.5 --disable-validation-check

This option skips validating XML files (e.g. within an SBOM) against a schema.

2.6 CVE Data Download Arguments

2.6.1 -u {now,daily,never,latest}, --update {now,daily,never,latest}

This option controls the frequency of updates for the CVE data from the National Vulnerability Database. By default, the tool checks the staleness of the data with every run, and if the data is more than one day old, it gets an update from NVD. You may also choose to update the data now (in which case all cached data is deleted and a full new download is done) or never in which case the staleness check is not done and no update is requested. The now and never modes can be combined to produce alternative update schedules if daily is not the desired one.
2.6.2 -n {json,api}, –nvd {json,api}

This option selects how CVE data is downloaded from the National Vulnerability Database. The default api option uses the NVD CVE Retrieval API. The results from this API are updated as quickly as the NVD website. A major benefit of using this NVD API is incremental updates which basically means you won’t have to download the complete feed again in case you want the latest CVE entries from NVD. See the detailed guide on incremental updates for more details.

You may also choose to update the data using json option which uses the JSON feeds available on this page. These per-year feeds are updated once per day. This mode was the default for CVE Binary Tool prior to the 3.0 release.

2.6.3 –nvd-api-key NVD_API_KEY

An NVD API key allows registered users to make a greater number of requests to the API. At this time, the NVD API documentation says, “The public rate limit (without an API key) is 10 requests in a rolling 60 second window; the rate limit with an API key is 100 requests in a rolling 60 second window.”

CVE Binary tool by default queries the NVD database once per day and caches the results to help alleviate load on the NVD servers. Users who update more regularly or who are running the tool in shared environments (such as cloud providers or GitHub Actions) may find themselves hitting the rate limits despite those precautions and should obtain and use an NVD API key with CVE Binary Tool.

To get an API key, users should visit the NVD API key request page.

Related: NVD API key announcement

2.7 Checkers Arguments

2.7.1 -s SKIPS, –skips SKIPS

This option allows one to skip (disable) a comma-separated list of checkers. This can be useful for improving the performance of the tool when you have some prior knowledge about what checkers may apply to the binary you are scanning.

2.7.2 -r CHECKERS, –runs CHECKERS

This option allows one to enable a comma-separated list of checkers.

2.8 Input Arguments

2.8.1 directory (positional argument)

Specify path to directory you want to scan.
2.8.2 -i INPUT_FILE, –input-file INPUT_FILE

This option extends functionality of csv2cve for other formats like JSON and allow cve-bin-tool to accept some form of triage data and incorporate that into the output so that people could spend less time re-triaging.

You can provide either CSV or JSON file as input_file with vendor, product and version fields. You can also add optional fields like remarks, comments, cve_number, severity. Here’s the detailed description and usecase of each fields:

1. **vendor, product, version** - To query locally stored CVE database and give you a list of CVEs that affect each vendor, product, version listed.

2. **remarks** - remarks help you categorized different CVEs into different categories like:
   - NewFound (1, n, N)
   - Unexplored (2, u, U)
   - Confirmed (3, c, C)
   - Mitigated, (4, m, M)
   - Ignored (5, i, I)
   - All the characters denoted in parenthesis are aliases for that specific value. Output will be displayed in the same order as priority given to the remarks.

3. **comments** - You can write any comments you want to write in this field. This will be ignored in the console output but will be propagated as it is in CSV, JSON or HTML formats.

4. **severity** - This field allows you to adjust severity score of specific product or CVE. This can be useful in the case where CVE affects a portion of the library that you aren’t using currently but you don’t want to ignore it completely. In that case, you can reduce severity for this CVE.

5. **cve_number** - This field give you fine grained control over output of specific CVE. You can change remarks, comments and severity for specific CVE instead of whole product.

You can also provide a Vulnerability Exchange (VEX) file which contains the reported vulnerabilities for components within a product. The supported format is the CycloneDX VEX format which can be generated using the --vex option. A VEX file is identified with a file extension of .vex. For the triage process, the state value in the analysis section of each CVE should have one of the following values:

```
"under_review" - this is the default state and should be used to indicate the vulnerability is to be reviewed
"in_triage" - this should be used to indicate that the vulnerability is being reviewed
"exploitable" - this should be used to indicate that the vulnerability is known to be exploitable
"not_affected" - this should be used to indicate that the vulnerability has been mitigated
```

The detail value in the analysis section can be used to provide comments related to the state

You can use -i or --input-file option to produce list of CVEs found in given vendor, product and version fields (Usage: cve-bin-tool -i=test.csv) or supplement extra triage data like remarks, comments etc. while scanning directory so that output will reflect this triage data and you can save time of re-triaging (Usage: cve-bin-tool -i=test.csv /path/to/scan).

Note that --input-file, unlike cve-bin-tool directory scan, will work on *any* product known in the National Vulnerability Database, not only those that have checkers written.

Note: For backward compatibility, we still support csv2cve command for producing CVEs from csv but we recommend using new --input-file command instead.
For Example if input_file contains following data:

You can test it using our test input file with following command:

cve-bin-tool -i="test/json/test_triage.json"

The output will look like following:

```
CVE BINARY TOOL


| NewFound CVEs |

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Product</th>
<th>Version</th>
<th>CVE Number</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2019-5443</td>
<td>HIGH</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2019-5481</td>
<td>CRITICAL</td>
</tr>
</tbody>
</table>

| Unexplored CVEs |

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Product</th>
<th>Version</th>
<th>CVE Number</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>haxx</td>
<td>libcurl</td>
<td>7.59.0</td>
<td>CVE-2018-14618</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>haxx</td>
<td>libcurl</td>
<td>7.59.0</td>
<td>CVE-2018-16890</td>
<td>HIGH</td>
</tr>
<tr>
<td>mit</td>
<td>kerberos</td>
<td>1.15.1</td>
<td>CVE-2000-0547</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>libjpeg-turbo</td>
<td>libjpeg-turbo</td>
<td>2.0.1</td>
<td>CVE-2018-20330</td>
<td>HIGH</td>
</tr>
</tbody>
</table>

| Confirmed CVEs |

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Product</th>
<th>Version</th>
<th>CVE Number</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>mit</td>
<td>kerberos_5</td>
<td>5-1.15.1</td>
<td>CVE-2018-5729</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>mit</td>
<td>kerberos_5</td>
<td>5-1.15.1</td>
<td>CVE-2018-5730</td>
<td>LOW</td>
</tr>
<tr>
<td>libjpeg-turbo</td>
<td>libjpeg-turbo</td>
<td>2.0.1</td>
<td>CVE-2018-19664</td>
<td>CRITICAL</td>
</tr>
</tbody>
</table>

| Mitigated CVEs |

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Product</th>
<th>Version</th>
<th>CVE Number</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssh</td>
<td>ssh2</td>
<td>2.0</td>
<td>CVE-1999-1029</td>
<td>HIGH</td>
</tr>
<tr>
<td>ssh</td>
<td>ssh2</td>
<td>2.0</td>
<td>CVE-1999-1231</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>sun</td>
<td>sunos</td>
<td>5.4</td>
<td>CVE-1999-0008</td>
<td>HIGH</td>
</tr>
</tbody>
</table>
```
2.8.3 -L PACKAGE_LIST, --package-list PACKAGE_LIST

This option runs a CVE scan on installed packages listed in a package list. It takes a python package list (requirements.txt) or a package list of packages of systems that has dpkg, pacman or rpm package manager as an input for the scan. This option is much faster and detects more CVEs than the default method of scanning binaries.

An example of the package list for Linux systems:

bash
unzip
firefox
sed
python3

Note: The packages in the package list should be installed in the system before the scan. Run

• pip install -r requirements.txt to install python packages
• sudo apt-get install $(cat package-list) for packages in a Debian based system
• sudo yum install $(cat package-list) for packages in a CentOS/Fedora system
• sudo pacman -S $(cat package-list) for packages in a system that uses pacman package manager (Arch Linux, Manjaro etc.)

Note: Don’t use invalid package names in the package list, as it may throw errors.

You can test it using our test package list with following command:

cve-bin-tool -L test/txt/test_ubuntu_list.txt

You can get a package list of all installed packages in

• a system using dpkg package manager by running dpkg-query -W -f '${binary:Package}\n' > pkg-list
• a system using pacman package manager by running pacman -Qqe > pkg-list
• a system using rpm package manager by running rpm -qa --queryformat '%{NAME}\n' > pkg-list

in the terminal and provide it as an input by running cve-bin-tool -L pkg-list for a full package scan.

2.8.4 -C CONFIG, --config CONFIG

We currently have number of command line options and we understand that it won’t be feasible to type all the option everytime you want to run a scan. You can use --config option to provide configuration file for the tool. You can still override options specified in config file with command line arguments. We support 2 most popular config file format:

1. TOML which is popular amongst Python developer and very similar to INI file. If you are not familiar with TOML checkout official TOML documentation
2. YAML which is popular amongst devops community and since many of our users are devops. We also support YAML as config file format. You can find out more about YAML at yaml.org

You can see our sample TOML config file here and sample YAML config file here.

You have to specify either a directory to scan and/or an input file containing vendor, product and version fields either in JSON or CSV format.
Yaml example file

```yaml
input:
  directory: test/assets
  # To supplement triage data of previous scan or run standalone as csv2cve
  # Currently we only support csv and json file.
  input_file: test/csv/triage.csv

checker:
  # list of checkers you want to skip
  skips:
    - python
    - bzip2
  # list of checkers you want to run
  runs:
    - curl
    - binutils

output:
  # specify output verbosity from [debug, info, warning, error, critical]
  # verbosity will decreases as you go left to right (default: info)
  log_level: debug
  # if true then we don't display any output and
  # only exit-code with number of cves get returned
  # overwrites setting specified in log_level
  # Note: it's lowercase true or false
  quiet: false
  # specify one of an output format: [csv, json, html, console] (default: console)
  format: console
  # provide output filename (optional)
  # if not specified we will generate one according to output format specified
  output_file: '
  # specify minimum CVE severity level to report from [low, medium, high, critical]
  # (default: low)
  severity: low
  # specify minimum CVSS score to report from integer range 0 to 10 (default: 0)
  cvss: 0

other:
  # set true if you want to skip checking for newer version
  disable_version_check: false
  # update schedule for NVD database (default: daily)
  update: daily
  # set true if you want to autoextract archive files. (default: true)
  extract: true
```

2.8. Input Arguments
Toml example file

```toml
[input]
# Directory to scan
directory = "test/assets"

# To supplement triage data of previous scan or run standalone as csv2cve
# Currently we only support csv and json file.
input_file = "test/csv/triage.csv"

[checker]
# list of checkers you want to skip
skips = ["python", "bzip2"]

# list of checkers you want to run
runs = ["curl", "binutils"]

[output]
# specify output verbosity from ["debug", "info", "warning", "error", "critical"]
# verbosity will decreases as you go left to right (default: "info")
log_level = "debug"

# if true then we don't display any output and
# only exit-code with number of cves get returned
# overwrites setting specified in log_level
# Note: it's lowercase true or false
quiet = false

# specify one of an output format: ["csv", "json", "html", "console"] (default: "console")
format = "console"

# provide output filename (optional)
# if not specified we will generate one according to output format specified
output_file = ""

# specify minimum CVE severity level to report from ["low", "medium", "high", "critical"] (default: "low")
severity = "low"

# specify minimum CVSS score to report from integer range 0 to 10 (default: 0)
cvss = 0

[other]
# set true if you want to skip checking for newer version
disable_version_check = false

# update schedule for NVD database (default: daily)
update = "daily"
```

(continues on next page)
2.9 Output Arguments

Although the examples in this section show results for a single library to make them shorter and easier to read, the tool was designed to be run on entire directories and will scan all files in a directory if one is supplied.

2.9.1 -o OUTPUT_FILE, --output-file OUTPUT_FILE

This option allows you to specify the filename for the report, rather than having CVE Binary Tool generate it by itself.

2.9.2 --html-theme HTML_THEME

This option specifies the theme directory to be used in formatting the HTML report.

2.9.3 -f {csv,json,console,html}, --format {csv,json,console,html}

This option allows the CVE Binary Tool to produce a report in an alternate format. This is useful if you have other tools which only take a specific format. The default is console which prints category wise beautiful tables of CVEs on terminal.

1. --format csv - write output file in csv (comma separated) format.

```
vendor,product,version,cve_number,severity,remarks,comments
haxx,curl,7.34.0,CVE-2014-0015,MEDIUM,Mitigated,
haxx,curl,7.34.0,CVE-2014-0138,MEDIUM,NewFound,
haxx,curl,7.34.0,CVE-2014-0139,MEDIUM,Unexplored,
```

2. --format json - write output file in json (javascript object notation) format.

```
[  
  {  
    "vendor": "haxx",
    "product": "curl",
    "version": "7.34.0",
    "cve_number": "CVE-2014-0015",
    "severity": "MEDIUM",
    "remarks": "Mitigated",
    "comments": ""
  },
  {  
    "vendor": "haxx",
    "product": "curl",
    "version": "7.34.0",
    "cve_number": "CVE-2014-0138",
    "severity": "MEDIUM",
    "remarks": "NewFound",
    "comments": ""
  }
]
```
3. **--format console** - prints in nice colored tabular format.

4. **--format html** - creates a report in html format according to the specified HTML theme.

5. **--format pdf** - creates a report in PDF format.

If you wish to use PDF support, you will need to install the `reportlab` library separately.

If you intend to use PDF support when you install cve-bin-tool you can specify it and report lab will be installed as part of the cve-bin-tool install:

```bash
pip install cve-bin-tool[PDF]
```

If you’ve already installed cve-bin-tool you can add reportlab after the fact using pip:

```bash
pip install --upgrade reportlab
```

Note that reportlab was taken out of the default cve-bin-tool install because it has a known CVE associated with it (CVE-2020-28463). The cve-bin-tool code uses the recommended mitigations to limit which resources added to PDFs, as well as additional input validation. This is a bit of a strange CVE because it describes core functionality of PDFs: external items, such as images, can be embedded in them, and thus anyone viewing a PDF could load an external image (similar to how viewing a web page can trigger external loads). There’s no inherent “fix” for that, only mitigations where users of the library must ensure only expected items are added to PDFs at the time of generation.

Since users may not want to have software installed with an open, unfixable CVE associated with it, we’ve opted to make PDF support only available to users who have installed the library themselves. Once the library is installed, the PDF report option will function.

### 2.9.4 -c CVSS, --cvss CVSS

This option specifies the minimum CVSS score (as integer in range 0 to 10) of the CVE to report. The default value is 0 which results in all CVEs being reported.

### 2.9.5 -S {low,medium,high,critical}, --severity {low,medium,high,critical}

This option specifies the minimum CVE severity to report. The default value is low which results in all CVEs being reported.

Note that this option is overridden by **--cvss** parameter if this is also specified.
2.9.6 –report

This option produces a report for all output formats even if there are 0 CVEs. By default CVE Binary tool doesn’t produce an output when there are 0 CVEs.

2.9.7 -A [〈distro_name›-〈distro_version_name›], –available-fix [〈distro_name›-〈distro_version_name›]

This option lists the available fixes of the package from Linux distribution if there are any.

The currently supported Linux distributions are:

```plaintext
debian-bullseye
debian-stretch
debian-buster
ubuntu-hirsute
ubuntu-groovy
ubuntu-focal
ubuntu-eoan
ubuntu-disco
ubuntu-cosmic
ubuntu-bionic
ubuntu-artful
ubuntu-zesty
ubuntu-yakkety
ubuntu-xenial
```

2.9.8 -b [〈distro_name›-〈distro_version_name›], –backport-fix [〈distro_name›-〈distro_version_name›]

This option outputs the available backported fixes for the packages with CVEs if there are any.

By default CVE Binary tool checks for backported fixes according to the Linux distribution of the local machine. You can specify the distribution information explicitly in 〈distro_name›-〈distro_version_name› fashion.

```
cve-bin-tool <path-to-binary> --backport-fix ubuntu-focal
```

Currently supported options

```plaintext
debian-bullseye
debian-stretch
debian-buster
ubuntu-hirsute
ubuntu-groovy
ubuntu-focal
ubuntu-eoan
ubuntu-disco
ubuntu-cosmic
ubuntu-bionic
ubuntu-artful
ubuntu-zesty
ubuntu-yakkety
ubuntu-xenial
```
2.9.9 –affected-versions

This option reports the versions of a product affected by a given CVE.

2.9.10 –vex VEX_FILE

This option allows you to specify the filename for a Vulnerability Exchange (VEX) file which contains all the reported vulnerabilities detected by the scan. This file is typically updated (outside of the CVE Binary tool) to record the results of a triage activity and can be used as a file with --input-file parameter.

2.9.11 Output verbosity

As well as the modes above, there are two other output options to decrease or increase the number of messages printed:

1. Quiet mode (-q) suppresses all output but exits with an error number indicating the number of files with known CVEs. This is intended for continuous integration and headless tests, while the other modes are all more human-friendly.

2. Log mode (-l log_level) prints logs of the specified log_level and above. The default log level is info. The logs can be suppressed by using quiet mode.

Quiet Mode

As the name implies, quiet mode has no console output, and one must check the return code to see if any issues were found. The return value will be the number of files that have been found to have CVEs.

Below is what it returns on bash when one file is found to have CVEs:

```
terri@sandia:~/Code/cve-bin-tool$ cve-bin-tool -q ~/output_test_quiet/openssl
terri@sandia:~/Code/cve-bin-tool$ echo $?
1
```

Note that errors are returned as negative numbers. Any positive number indicates that CVEs may be present in the code. A good result here is 0.

Logging modes

The logging modes provide additional fine-grained control for debug information.

2.10 Merge Report Arguments

Users may wish to create and combine multiple cve-bin-tool reports to track how vulnerability changes over time, how long it takes to fix issues, or other changes between different reports. We have a number of options related to merging report data.
2.10.1 -a INTERMEDIATE_PATH, –append INTERMEDIATE_PATH

This option allows you to save the output in form of an intermediate report which you can later merge with other reports of the same format.
See the detailed guide on intermediate reports for more details.

Intermediate report format

```json
{
    "metadata": {
        "timestamp": "2021-06-17.00-00-30",
        "tag": "backend",
        "scanned_dir": "/home/path/",
        "products_with_cve": 139,
        "products_without_cve": 2,
        "total_files": 49
    },
    "report": [
        {
            "vendor": "gnu",
            "product": "gcc",
            "version": "9.0.1",
            "cve_number": "CVE-2019-15847",
            "severity": "HIGH",
            "score": "7.5",
            "cvss_version": "3",
            "paths": "/home/path/glib.tar.gz,/home/path/gcc.tar.gz",
            "remarks": "NewFound",
            "comments": ""
        }
    ]
}
```

2.10.2 -t TAG, –tag TAG

This option allows you to save a tag inside the metadata of intermediate reports. By default the value is empty ""

2.10. Merge Report Arguments
2.10.3 -m INTERMEDIATE_REPORTS, –merge INTERMEDIATE_REPORTS

This option allows you to merge intermediate reports created using -a or --append. The output from the merged report produces a report on the console. But you can also use it along with -f --format and -o --output-file to produce output in other formats. It takes a list of comma-separated filepaths.

2.10.4 -F TAGS, –filter TAGS

This allows you to filter out intermediate reports based on the tag. This can be useful while merging multiple intermediate reports from a single path. See detailed guide on filter intermediate reports for more information.

2.11 Deprecated Arguments

2.11.1 -x, –extract

This option allows the CVE Binary Tool to extract compressed files into a temporary directory so the contents can be scanned. If the quiet flag is not used, the list of extracted files will be printed.

CVE Binary Tool by default auto-extract all compressed files inside the directory path. You can always exclude certain paths by using -e --exclude

2.12 Feedback & Contributions

Bugs and feature requests can be made via GitHub issues. Be aware that these issues are not private, so take care when providing output to make sure you are not disclosing security issues in other products.

Pull requests are also welcome via git.

2.13 Security Issues

Security issues with the tool itself can be reported to Intel’s security incident response team via https://intel.com/security.

If in the course of using this tool you discover a security issue with someone else’s code, please disclose responsibly to the appropriate party.
CSV2CVE

This tool takes a comma-delimited file (.csv) with the format `vendor,product,version` and queries the locally stored CVE data (the same data used by the CVE Binary Tool) to give you a list of CVEs that affect each version listed.

This is meant as a helper tool for folk who know the list of product being used in their software, so that you don’t have to rely on binary detection heuristics. There exist other tools that do this, but it seemed potentially useful to provide both in the same suite of tools, and it also saves users from having to download two copies of the same data.

At the moment, you must use the exact vendor and product strings used in the National Vulnerability Database. You can read more on how to find the correct string in the checker documentation. Future work could extend this to use the mappings already in the CVE Binary Tool or to use other mappings such as common linux package names for a given distribution. (Contributions welcome!)

Note: For backward compatibility, we still support `csv2cve` command for producing CVEs from csv but we recommend using new `--input-file` command instead.

3.1 Running the tool:

`csv2cve`

If you are trying to run a local copy from source, you can also use `python -m cve_bin_tool.csv2cve`

3.2 Additional Options:

<table>
<thead>
<tr>
<th>Output options:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-l {debug,info,warning,error,critical}, --log {debug,info,warning,error,critical}</code></td>
</tr>
<tr>
<td>log level. The default log level is info</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Functional options:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-u {now,daily,never}, --update {now,daily,never}</code></td>
</tr>
<tr>
<td>update schedule for NVD database. Default is daily.</td>
</tr>
</tbody>
</table>
### 3.3 Example .csv file:

Note that this *does* require that the first row be `vendor,product,version` so that the csv parser can do the right thing. You can have the columns in a different order and/or include other information, but it needs those 3 columns to work.

<table>
<thead>
<tr>
<th>vendor</th>
<th>product</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>libjpeg-turbo</td>
<td>libjpeg-turbo</td>
<td>2.0.1</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
</tr>
<tr>
<td>haxx</td>
<td>libcurl</td>
<td>7.59.0</td>
</tr>
<tr>
<td>wontwork</td>
<td>no</td>
<td>7.7</td>
</tr>
</tbody>
</table>

### 3.4 Example output:

```bash
$ python -m cve_bin_tool.csv2cve test.csv

CVE BINARY TOOL

• cve-bin-tool Report Generated: 2020-08-03 10:14:11

<table>
<thead>
<tr>
<th>NewFound CVEs</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Product</th>
<th>Version</th>
<th>CVE Number</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>wontwork</td>
<td>no</td>
<td>7.7</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Unexplored CVEs</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Product</th>
<th>Version</th>
<th>CVE Number</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>haxx</td>
<td>libcurl</td>
<td>7.59.0</td>
<td>CVE-2018-14618</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>haxx</td>
<td>libcurl</td>
<td>7.59.0</td>
<td>CVE-2018-16890</td>
<td>HIGH</td>
</tr>
<tr>
<td>haxx</td>
<td>libcurl</td>
<td>7.59.0</td>
<td>CVE-2019-3822</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>haxx</td>
<td>libcurl</td>
<td>7.59.0</td>
<td>CVE-2019-3823</td>
<td>HIGH</td>
</tr>
<tr>
<td>haxx</td>
<td>libcurl</td>
<td>7.59.0</td>
<td>CVE-2019-5436</td>
<td>HIGH</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2018-0500</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2018-1000300</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2018-1000301</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2018-16839</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2018-16840</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2018-16842</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2019-5443</td>
<td>HIGH</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2019-5481</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>haxx</td>
<td>curl</td>
<td>7.59.0</td>
<td>CVE-2019-5482</td>
<td>CRITICAL</td>
</tr>
<tr>
<td>libjpeg-turbo</td>
<td>libjpeg-turbo</td>
<td>2.0.1</td>
<td>CVE-2018-19664</td>
<td>MEDIUM</td>
</tr>
<tr>
<td>libjpeg-turbo</td>
<td>libjpeg-turbo</td>
<td>2.0.1</td>
<td>CVE-2018-20330</td>
<td>HIGH</td>
</tr>
</tbody>
</table>
4.1 How do I create a custom theme for the HTML Report?

CVE Binary Tool provides HTML Report as an output type and although the Report design will work for most of the users it might be the case that we want to update the Report Design according to our needs. So in this tutorial, we will be discussing on how to customise the HTML Report.

CVE Binary Tool provides the functionality to edit report components which we can use to redesign or change the report. CVE Binary Tool uses Object Oriented Programming approach for redesigning that means we need to overwrite files to update the design.

Example: If we want to update the dashboard then we only need to provide dashboard.html file. It will update the dashboard design as specified in the given file. All the remaining design will be as it is.

Before we start let’s discuss the requirements. So first and the most important requirement is that we must know Jinja because cve-bin-tool uses that as its templating engine. We don’t need the advance concepts but knowledge of jinja variables {{ }} and jinja block {% %} is a must. For more info see Jinja Basics. Second we must setup a directory to work with. So let’s start by creating a new directory.

This guide assumes that you are using the version 1.1 or above of cve-bin-tool.

4.1.1 Setting up the directory

Before we start customising the report we have to set up our work directory. Work directory or root directory will contain all the configuration files for components we want to update.

We first need to set up our root directory and then we need to create config folders inside this root directory.

**steps to setup directory:**

1) Create a root directory anywhere on your computer. We can name this directory whatever we want. I will create a new directory named my_theme because I want to store my config files there.

```
user@ubuntu: ~$ mkdir my_theme
user@ubuntu: ~$ cd my_theme/
```

2) Create a new config folder templates. Config folders are simple folders but with a specified name. Note that the config folder name must match with templates otherwise it will not overwrite components.

```
user@ubuntu: ~/my_theme$ mkdir templates
user@ubuntu: ~/my_theme$ cd templates/
```
# Actual Path After the Steps

```
user@ubuntu: ~/my_theme/templates$
```

This will set up our templates config folder (templates directory) and now we are ready to start customising the HTML Report.

## 4.1.2 Setting up the Files

Now as we have set up our templates folder we will create files that are needed for the report redesign. CVE Binary Tool allows customisation of 4 basic templates. We only need to implement only those templates that we want to overwrite.

1. base.html
2. dashboard.html
3. row_cve.html
4. row_product.html

These files will control the structure and design of the report. We’ll discuss in detail on what things we need to consider while creating these files and we’ll also discuss which files correspond to which part of the report. But before that let’s create those files.

I’ll create those files using terminal but you can also create them with the help any File Manager or Text Editor.

```
user@ubuntu: ~/my_theme/templates$ touch base.html
user@ubuntu: ~/my_theme/templates$ touch dashboard.html
user@ubuntu: ~/my_theme/templates$ touch row_cve.html
user@ubuntu: ~/my_theme/templates$ touch row_product.html
```

We can confirm that using `ls` in the current directory

```
user@ubuntu: ~/my_theme/templates$ ls
base.html
dashboard.html
row_cve.html
row_product.html
```

HTML Report with a custom theme can be generated after creating these files and providing the `--html-theme` argument with the root directory path.

Now we’ll discuss in detail to see what is the role of each file and how to customise that according to our needs.
row_cve.html

Role:

Each CVE has a unique number associated with it called CVE Number and a severity level measuring the level of severity (Critical, High, Medium, Low). Apart from that a small description of CVE is also present. The row_cve.html handles all the information about CVE and design. Design info includes component design for a single cve and it must not be a full HTML. Full html include <html>, <head> <body> tags.

Example of half html

<!-- example html file -->
<div>
  <h2> I'm not a full html because I don't have head and body </h2>
  <p> This is a example code for half-html </p>
</div>

How to customise?

For customising we need to overwrite the row_cve.html. We need to provide this file inside the templates directory. We have already created this file inside our templates directory if you are following along. We need to include some jinja variables in row_cve.html to include the cve details.

For more help, you can take a look at cve-bin-tool’s own row_cve.html template implementation.

<!-- CVE Binary Tool's row_cve.html -->

<!-- for each cve in CVE list we will have this row-->
<div class="card listCVE bg-{{ severity }} text-color-main shadow-sm">
  <div class="row text-left m-t-5 m-b-5">
    <div class="col-12 col-lg-5 p-t-10 ">
      <h6 class="m-l-10">{{ cve_number }}</h6>
    </div>
    <div class="col-7 col-lg-4 p-t-10 ">
      <h6 class="m-l-10">Severity: {{ severity }}</h6>
    </div>
    <div class="col-5 col-lg-3 text-center">
      <button class="btn borderButton text-color-main" data-toggle="collapse" data-target="#info{{ var_id }}">more info</button>
    </div>
  </div>
  <!-- Hidden Data That we want to show -->
  <div id="info{{ var_id }}" class="collapse bg-white" data-parent="#accord{{ fix_id }}">
    <p class="summary">{{ description }} <a href="https://nvd.nist.gov/vuln/detail/{{ cve_number }}" target="_blank" rel="noopener noreferrer">..read more</a></p>
  </div>
</div>

4.1. How do I create a custom theme for the HTML Report?
Role:

Each product has one or more CVE associated with it. It also contains information about the Vendor, Version and the
name of the Product along with the cve count and a Product Analysis graph based on CVEs. The list of CVEs contains
the data rendered with the template row_cve.html.

How to customise?

For customising we need to overwrite the row_product.html. We need to provide this file inside the templates
directory. We have already created this file inside our templates directory if you are following along. We need to
handles the following jinja variables in the row_product.html. Again this should not be a full-html.

For more help, you can take a look at cve-bin-tool’s own row_product.html template implementation.

<!-- CVE Binary Tool's row_product.html -->
<!-- Header for the Product Row [VENDOR, PRODUCT, VERSION, NUMBER_OF_CVES] -->

<div class="card text-center pHeading text-color-main p-t-5 product">
  <div class="row">
    <a class="stretched-link" data-toggle="collapse" href="#div{{ fix_id }}"></a>
    <!-- Vendor -->
    <div class="col-6 col-lg">
      <h5 class="font-weight-light">Vendor: {{ vendor }}</h5>
    </div>
    <!-- Product -->
    <div class="col-6 col-lg">
      <h5 class="font-weight-light">Product: {{ name }}</h5>
    </div>
    <!-- Version -->
    <div class="col-6 col-lg">
      <h5 class="font-weight-light">Version: {{ version }}</h5>
    </div>
    <!-- Total number of Known vulnerability -->
    <div class="col-6 col-lg">
      <h5 class="font-weight-light">Number of CVE's: {{ cve_count }}</h5>
    </div>
  </div>
</div>

</div>

<!-- Product CVEs start from here -->

<!-- cve row contains ListCVES for each product of specific version and analysis Chart-->
dashboard.html

Role:

The dashboard is the main showcase area with two graphs. One with the details of the number of products that were found and the other contains the information about the number of CVEs in the products that were found.

How to customise?

The dashboard.html is to be present in our work directory under the templates folder. It must not be a full-html and must handle the following jinja variable.

For more help, you can take a look at cve-bin-tool’s own dashboard.html template implementation.

4.1. How do I create a custom theme for the HTML Report?
base.html

Role:

As the name suggests base.html is the actual base for all other templates and the rendered results of each template are included in this base template. It also holds all the scripts and CSS files and is required to be a full-html.

How to customise?

As such, there is no restriction on how to customise base.html but we need to handle some jinja variables otherwise it will not render properly. Also, we must make sure that plotly.js is included at the top otherwise the graphs will not render.

Here is the list of jinja variables that we need to provide in the template. For more help, you can take a look at cve-bin-tool’s own base.html template implementation.
4.1. How do I create a custom theme for the HTML Report?
4.1.3 Adding and updating custom CSS and JavaScript files

CVE Binary Tool uses the bootstrap 4 to style the templates but we might want to use the latest bootstrap version available (bootstrap 5 is in its early stage at the time when I’m writing this tutorial). It might also be the case that we want to include our custom CSS files and even js files.

Updating CSS and JS files

CVE Binary Tool allows us to update the CSS and JS files in the same manner as we update the templates. So we just need to create a new config folder inside our work directory (In our case my_theme) named css to update the CSS files and other named js to update the javascript files.

Here is the list of files that we can update.

CSS FILES

<table>
<thead>
<tr>
<th>File name</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap.css</td>
<td>Bootstrap.css file for the report</td>
</tr>
<tr>
<td>main.css</td>
<td>Custom CSS file for the report</td>
</tr>
</tbody>
</table>

Example: If we want to update the main.css file we’ll create a main.css file inside the css folder under the work directory (my_theme).
JavaScript FILES

<table>
<thead>
<tr>
<th>File name</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>bootstrap.js</td>
<td>Bootstrap.js file for the report</td>
</tr>
<tr>
<td>plotly.js</td>
<td>Plotly.js is used for graph generation</td>
</tr>
<tr>
<td>jquery.js</td>
<td>JQuery file for the report</td>
</tr>
<tr>
<td>main.js</td>
<td>Custom JavaScript for the report</td>
</tr>
</tbody>
</table>

Example: If we want to update the `main.js` file we’ll create a `main.js` file inside the `js` folder under the work directory(`my_theme`).

Adding CSS and JavaScript

Apart from the given files, we might want to include other Popular JS and CSS files like we might want to add Font Awesome, Popper.js or any other custom CSS or js file.

So to add custom styles we need to include them in the `base.html` inside the templates directory. So the implementation of `base.html` is a must. If we don’t want to update the cve-bin-tool’s `base.html` then we can copy the template and paste that in the templates folder of our work directory and then we can include our CSS and js files.

There are two methods to include CSS and js

1) Use CDNs to include js and CSS files
2) Copy the contents of CSS and JS files directly inside the `base.html`. This method is recommended as it will allow the report to work even in complete offline mode (No Internet).

We know that we must maintain different HTML, CSS and js files but because we want to generate a single report file so need to include everything in a single file.

For more help or suggestion you can contact our community.

4.2 How to scan a docker image?

There are couple of ways to scan a docker image using cve-bin-tool.

1. You can scan a docker image by installing and running cve-bin-tool inside a docker container.
2. You can export the directory on host and scan it on host.

We are going to scan `/usr/bin` directory of ubuntu:latest docker image for demonstration purpose but you can use same recipe to scan directory of your interest.

4.2.1 Install and run CVE Binary Tool inside container

Let's first create a docker instance of the image we want to scan using following command:

```
docker run -it -d --name cve_scan ubuntu --entrypoint bash
```

This will create new instance of ubuntu image and run it in the background. You can check if your container is running or not using docker `ps`.

Note: you may need to use sudo if current user isn’t in the docker group.

Now let’s go inside the container using the docker `exec` command and install python in it.
In this example we have defined container name as cve_scan. You will get a random name if you have not defined while running the container initially.

Update the container and install python3 on it.

```
apt-get update
apt-get install python3
apt-get install python3-pip
```

Note: this step is distro specific if your container is based on different distro (Ex: centos) checkout official documentation of that specific distro for installing python3 in it.

Now let's install cve-bin-tool in our container.

```
pip3 install cve-bin-tool
```

This will install latest version of cve-bin-tool from PyPI.

You can also install latest development version from our github repository using following command

```
pip3 install git+https://github.com/intel/cve-bin-tool
```

After all the things done check the version of cve-bin-tool using the command.

```
cve-bin-tool -V
```

If there is output then You have installed cve-bin-tool successfully in the docker container.

Now let's scan /usr/bin directory and export report to the host using following command.

```
cve-bin-tool /usr/bin -f csv -o usr_bin_cve.csv
```

This will take sometime and after generation of the report, you have to export it to the host. You first need to exit current docker session by typing `exit` in the container terminal. Now let's copy report from container to the host.

```
docker cp cve_scan:~/usr_bin_cve.csv ~/Documents/usr_bin_cve.csv
```

This will save CVE report of scanned docker directory in the ~/Documents/usr_bin_cve.csv.

### 4.2.2 Export directory from container and scan

Assuming, you already have created docker instance named cve_scan as mentioned above. You can export directory you want to scan to the host and scan there.

```
docker cp cve_scan:/usr/bin/ ~/scan
```

This will copy all files and directories from /usr/bin to /scan and now you can scan /scan directory with cve-bin-tool normally.

Note: You may want to use docker cp -a if you want to copy all uid/gid info.

```
cve-bin-tool scan [OPTIONS]
```

Note: This method assumes you already have installed cve-bin-tool on host. If you haven’t install it with pip3 install cve-bin-tool.
Both of the above mentioned methods will help you scan a docker image and you can choose one over another. Second method is comparatively easier than first but has overhead of copying all data from container to host while first method requires you to install cve-bin-tool in docker container which can take around 10 minutes. You can automate both processes with simple bash script.

4.3 How do I use CVE Binary Tool in an offline environment?

The cve-bin-tool can be used in offline environments which do not have direct access to the internet to download the latest vulnerability databases.

4.3.1 Prepare the vulnerability database for offline use

To download the vulnerability database for use in an offline environment, ensure that cve-bin-tool is installed on an internet-connected system.

Run the tool to obtain the latest version of the vulnerability database

```
$ cve-bin-tool --update now
```

NOTE The tool will error with InsufficientArgs because no directory was specified for a scan. This is expected behaviour.

4.3.2 Transfer the vulnerability database file into a directory in the offline environment

The way of transfer depends on the environment. The files to be transferred are in “~/.cache/cve-bin-tool”

4.3.3 Import the vulnerability database file on the offline system

The vulnerability database should be copied into ~/.cache/cve-bin-tool.

The cve-bin-tool will fail to operate in offline mode if a vulnerability database is not present on the system.

4.3.4 Run cve-bin-tool with –offline option

In an offline environment, specify the --offline option when running a scan so that cve-bin-tool doesn’t attempt to download the latest database files or check for a newer version of the tool. The --offline option is equivalent to specifying --update never and --disable-version-check options.

4.3.5 Maintenance Updates

In an offline environment, it is important to update the vulnerability database on a regular basis as often as you feel appropriate, so that the scanner can continue to detect recently-identified vulnerabilities. If any changes to CVE data is required (e.g. to remove false positives), you might also want to create and copy over a triage data file for usage. The time of the latest database update is reported whenever a scan is performed.

It is important to periodically check if the cve-bin-tool has also been updated as this check cannot be performed within an offline environment.
4.4 Best practices for running multiple scans at once

If you’re running multiple instances of cve-bin-tool at once, you could potentially cause a race condition where multiple processes are trying to update the database from nvd at the same time. This is not ideal.

To avoid this, you should use a single command to run the nvd update, then turn off the updater in all other copies.

4.4.1 Step 1: Update

To update (without scanning) you can use the following command:

```
cve-bin-tool -u now
```

We recommend once per day, but this can be more frequently or less frequently depending on your needs. Ideally, you want to be sure this completes before you kick off any other scans, so that you aren’t checking against a partial database.

4.4.2 Step 2: Scan

Each parallel instance of cve-bin-tool can then be invoked as follows:

```
cve-bin-tool -u never $path_to_directory_or_file
```

4.5 NVD CVE Retrieval API

The CVE API is the next stage in providing up to date vulnerability information for NVD data consumers. The results from this API are updated as quickly as NVD website (unlike the traditional feeds which have explicit update interval of once per day).

This can be also used as a backup if the current JSON feed retrieval interface is removed from the NVD website.

Note: This API retrieval is slower in comparison to the traditional method.

You can read more about this [here](#).

The NVD API is enabled by default or you can explicitly use `-n api` or `--nvd api`:

```
python -m cve_bin_tool.cli -n api
```

A major benefit of using this NVD API is incremental updates.

4.5.1 What are Incremental Updates?

With the help of this REST API, we can fetch just the newly added/modified NVD data using the timestamp of your current local database copy. This will fetch only the CVE entries for which any vulnerability or product string was modified or published later than the above-mentioned timestamp. This can save users a lot of time and internet bandwidth.
4.5.2 How to use Incremental Updates?

You can use pre-existing `-u latest` parameter along with the `-n api`. This will simply update your local database and cache copy with the newly published and modified NVD entries.

```python
python -m cve_bin_tool.cli -u latest -n api
```

4.6 How to use intermediate reports

Let's consider a case where multiple groups have done triage separately and want to merge their outputs in a single report. We can do this by saving scans in form of intermediate reports and merge them whenever required.

4.6.1 Create Intermediate reports

To create an intermediate report on a scan for path `/home/code/backend/`, you can use:

```python
python -m cve_bin_tool.cli -a /home/reports/backend.json /home/code/backend/
```

Here we are saving the intermediate report in `/home/reports/backend.json`

Alternatively, you can just use the directory path omitting the filename. Example:

```python
python -m cve_bin_tool.cli -a /home/reports/ -t frontend /home/code/frontend/
```

CVE-Binary Tool will generate a filename with the default naming convention which is: "append.YYYY-MM-DD. hh-mm-ss.json"

Note: You can also use `-t --tag` if you want to add a unique tag inside your intermediate report. By default it is empty and stored as "".

Intermediate report format

```json
{
   "metadata": {
      "timestamp": "2021-06-17.00-00-30",
      "tag": "",
      "scanned_dir": "/home/code/backend",
      "products_with_cve": 139,
      "products_without_cve": 2,
      "total_files": 49
   },
   "report": [
      {
         "vendor": "gnu",
         "product": "gcc",
         "version": "9.0.1",
         "cve_number": "CVE-2019-15847",
         "severity": "HIGH",
         "score": "7.5",
         "cvss_version": "3",
         "paths": "/home/code/backend/glib.tar.gz,/home/code/backend/gcc.tar.gz",
         "remarks": "NewFound",
         "comments": ""
      }
   ]
}
```

(continues on next page)
4.6.2 Merge intermediate reports

You can merge multiple intermediate reports created using `-m --merge`

```
python -m cve_bin_tool.cli -m /home/reports/
```

`-m --merge` takes a comma-separated string. So, you can also pass filename(s) directly:

```
python -m cve_bin_tool.cli -m /home/reports/backend.json,/home/reports/append.2021-06-17.00-00-30.json
```

If you want to save the output in some other format (By default, it is console). You can also use `-f --format` and `-o --output-file` while merging intermediate reports. For example, If you want to generate an HTML report:

```
python -m cve_bin_tool.cli -m /home/reports/ -f html -o /home/reports/merged_intermediate.html
```

4.7 Filter Intermediate Reports

Let’s consider a case where a user has multiple intermediate reports stored at a single path. But the user wants to merge only selected reports. We can do this by filtering intermediate reports based on the tag.

4.7.1 How to filter Intermediate reports?

Suppose you have multiple intermediate reports stored at a single path.

```
path
  └── intermediate.cve-bin-tool.2021-06-02.01-17-48.json[weekly]
  └── intermediate.cve-bin-tool.2021-06-09.02-39-43.json[weekly]
  └── intermediate.cve-bin-tool.2021-06-16.23-44-35.json[weekly]
  └── intermediate.cve-bin-tool.2021-06-23.23-44-49.json[weekly]
  └── intermediate.cve-bin-tool.2021-06-24.23-45-08.json[daily]
  └── intermediate.cve-bin-tool.2021-06-25.23-45-08.json[daily]
  └── intermediate.cve-bin-tool.2021-06-26.23-45-08.json[daily]

0 directories, 7 files
```

Note: You can specify tags while generating intermediate reports using `-t --tag`. For this example, we have assumed that the intermediate reports contain tag `{weekly or daily}`

We want to merge the weekly generated report to plot the week-wise timeline trace as an HTML report. We can have some other use cases as well. To filter out the intermediate reports, use:
This will generate a merged report from these files -

```
path
  ├── intermediate.cve-bin-tool.2021-06-02.01-17-48.json[weekly]
  ├── intermediate.cve-bin-tool.2021-06-09.02-39-43.json[weekly]
  └── intermediate.cve-bin-tool.2021-06-16.23-44-35.json[weekly]
```

Alternatively, users can filter out intermediate reports which are generated by multiple teams if they have specified distinguishable tags. If we have intermediate reports (backend and frontend). We can simply use:

```
python -m cve_bin_tool.cli -F backend -m /path
```

This will generate a vulnerability report for all the backend related binaries.
5.1 CVE Binary Tool 3.0

The CVE Binary Tool 3.0 release includes improved tools for checking known lists of packages including Linux distributions, improved methods of communication with NVD to get vulnerability data, additional checkers, and significant refactoring to streamline the output.

5.1.1 New feature highlights:

- **SBOM Scanning**: CVE Binary Tool can now take Software Bill of Materials (SBOM) files to help users improve their supply chain security data for all known dependencies. The initial feature can handle some versions of SPDX, CycloneDX and SWID formats. More information on SBOM scanning can be found here: https://github.com/intel/cve-bin-tool/blob/main/doc/how_to_guides/sbom.md

- **Known vulnerability information**: Users scanning some linux distro packages can now get additional information about fixes available for those platforms.

- **Vulnerability Data**: The default method for getting NVD vulnerability lists has been changed. Previously we downloaded full yearly JSON files if anything in the year had changed, the new API allows us to get only the latest changes. Users may see a speedup during the update phase as a result.

- **(Breaking change) Return codes**: The return codes used by CVE Binary Tool have changed.
  - A 0 will be returned if no CVEs are found, a 1 will be returned if any CVEs were found (no matter how many), and codes 2+ indicate operational errors. A full list of error codes is available here: https://github.com/intel/cve-bin-tool/blob/main/cve_bin_tool/error_handler.py
  - Previously we returned the number of CVEs found, but this could exceed the expected range for return codes and cause unexpected behaviour.

Thanks especially to our 2021 GSoC students, @BreadGenie, @imsahil007 and @peb-peb whose final GSoC contributions are part of this release.

A full list of changes is available in GitHub. https://github.com/intel/cve-bin-tool/releases/tag/v3.0

Commit messages use the Conventional Commits format.
5.2 CVE Binary Tool 2.2.1

Release date: 04 Aug 2021

The 2.2.1 release relaxes the behaviour when file extraction fails, which was causing problems for some users scanning files with .exe and .apk file extensions using the previous release. In 2.2 all extraction fails caused the tool to halt and throw an exception, in 2.2.1 the tool will log a warning and continue.

5.3 CVE Binary Tool 2.2

Release date: 08 Jul 2021

The 2.2 release contains a number of bugfixes and improvements thanks to the many students who contributed as part of our Google Summer of Code selection process. Congratulations to @BreadGenie, @imsahil007 and @peb-peb who will be continuing to work with us for the next few months!

New feature highlights:

- CVE Binary Tool can now be used to get lists of vulnerabilities affecting a python requirements.txt file, as well as lists of packages installed on .deb or .rpm based systems (Thanks to @BreadGenie)
- Scan reports can now be merged (Thanks to @imsahil007)
- Reports can now be generated in PDF format (Thanks to @anthonyharrison)
- A new helper script is available to help new contributors find appropriate patterns for new checkers (Thanks to @peb-peb)
- Reports can now be generated even if no CVEs are found (Thanks to @BreadGenie)
- We’ve added rate limiting for our NVD requests (Thanks to @nisamson, @param211, @bhargav)

There are also a number of new checkers and bug fixes.

Thanks also to @jerinjtitus, @Molkree, @alt-glitch, @CabTheProgrammer, @Romi-776, @chaitanyamogal, @Rahul2044, @utkarsh147-del, @SinghHrmn, @SaurabhK122, @pdxjohnny and @terriko for their contributions to this release.

5.4 CVE Binary Tool 2.1.post1

Release date: 27 Apr 2021

Rate limiting temporary fix in response to NVD API update

5.5 CVE Binary Tool 2.1

Release date: 07 Dec 2020

This release fixes an issue with jinja2 autoescape breaking the HTML reports and includes some updates to tests.
5.6 CVE Binary Tool 2.0

Release date: 12 Nov 2020

This release features code from our three successful Google Summer of Code students!

- @SinghHrmn made improvements to our output formats, including adding a new HTML human-readable report format. You can try out a demo at https://intel.github.io/cve-bin-tool/
  - Read Harmandeep’s final GSoC report for more details.
- @Niraj-Kamdar improved the performance of cve-bin-tool and its tests, provided significant code modernization and added input modes so you can now add and re-use triage data with your scans.
  - Read Niraj’s final GSoC report for more details
- @SaurabhK122 added a huge number of new checkers to the tool, both in this release and the previous one.
  - Read Saurabh’s final GSoC report for more details

Thanks also to the mentors who worked with our students this year: @terriko, @pdxjohnn, @meflin, @mdwood-intel and unofficial mentor @anthonyharrison who helped us considerably with real-world feedback.

This release also includes contributions from the following new contributors:

- @anthonyharrison
- @imsahil007
- @chaitanyamogal
- @Rahul2044
- @Wicked7000
- @willmcgugan
- @kritirikhi
- @sakshatshinde

5.7 CVE Binary Tool 1.1.1

Release Date: 9 Nov 2020

This point release includes fixes so documentation will build and display correctly on readthedocs. There are no functional changes to the code.

5.8 CVE Binary Tool 2.0 alpha release

Release Date: 29 Oct 2020

This is an alpha release for people interested in trying out an early preview of 2.0. Major features include performance improvements, triage options, new output modes, and many new checkers thanks to our Google Summer of Code students @Niraj-Kamdar, @SinghHrmn and @SaurabhK122. Thanks for an incredibly productive summer!

We are expecting to make some documentation improvements before the final release, which we hope to have out next week.
5.9 CVE Binary Tool 1.1

Release Date: 29 Oct 2020

This is an alpha release for people interested in trying out an early preview of 2.0. Major features include performance improvements, triage options, new output modes, and many new checkers thanks to our Google Summer of Code students @Niraj-Kamdar, @SinghHrmn and @SaurabhK122. Thanks for an incredibly productive summer!

We are expecting to make some documentation improvements before the final release, which we hope to have out next week.

5.10 CVE Binary Tool 1.0

Release Date: 20 Apr 2020

This release includes major improvements to the way NVD data is used and stored. If you have tried the development tree from GitHub, you may wish to run cve-bin-tool -u now after you upgrade to remove old data.

There are now three output formats:

- Console (like before only prettier)
- CSV (comma-delimited text, suitable for import into spreadsheets)
- JSON (suitable for machine parsing)

And 17 new checkers (as well as improved tests for some of the old):

- binutils
- bluez
- bzip2
- ffmpeg
- gnutls
- gstreamer
- hostapd
- libcurl
- libdb
- ncurses
- nginx
- openssh
- python
- rsyslog
- strongswan
- syslogng
- varnish

Thanks to our many new and returning contributors for this 1.0 release. We have 21 new contributors since I last thanked people in 0.3.0:
And I’d like to make a quick list of our previous contributors, some of whom have continued to be active for this release:

- @bksahu
- @CaptainDaVinci
- @GiridharPrasath
- @pdxjohnny
- @PrajwalM2212
- @rossburton
- @sanketsaurav
- @sannanansari
- @terriko
- @wzao1515

Thanks also to the many people who reported bugs and helped us make things better!

I want to particularly thank all those involved with Google Summer of Code – not only have you made our code better, but you’ve also helped us improve our onboarding process and just brought a huge amount of energy to this project in 2020.
5.11 CVE Binary Tool 0.3.1

Release Date: 27 Nov 2019

This release contains fixes so the CVE Binary Tool handles the new CVSS 3.1 data correctly.

You may also notice some new checkers thanks to our Hacktoberfest participants! We’re still working on more robust tests before they’re fully supported, but we figured it was more fun to give you the preview than specifically withhold them. Have fun, and please file bugs if anything doesn’t work for you so we know how to best to target our testing.

5.12 CVE Binary Tool 0.3.0

Release date: 13 Aug 2019

The 0.3.0 release adds Windows support to the cve-bin-tool, with many thanks to @wzao1515 who has been doing amazing work as our Google Summer of Code Student!

New checkers in this release:

- icu
- kerberos
- libgcrypt
- libjpeg
- sqlite
- systemd

New flags:

- `-s / --skip`
  - allows you to disable a list of checkers

- `-m / --multithread`
  - lets the scanner run in multithreaded mode for improved performance

- `-u / --update`
  - allows you to choose if the CVE information is updated. Default is daily.

This release also contains a number of bugfixes and improved signatures.

Many thanks to our new contributors in this release: @wzao1515 @PrajwalM2212 @rossburton @GiridharPrasath @sannanansari @sanketsaurav @bksahu @CaptainDaVinci As well as the many people who reported bugs and helped us make things better!
5.13 CVE Binary Tool 0.2.0

Initial release, 18 Jan 2019.
The CVE Binary Tool team participates in a few events every year that are aimed at new people in open source. This guide is meant to help people get over the initial hurdle of figuring out how to use git and make a contribution.

If you’ve already contributed to other open source projects, contributing to the CVE Binary Tool project should be pretty similar and you can probably figure it out by guessing. Experienced contributors might want to just skip ahead to the checklist for a great pull request. But if you’ve never contributed to anything before, or you just want to see what we consider best practice before you start, this is the guide for you!

• CVE Binary Tool Contributor Guide
  – Imposter syndrome disclaimer
  – Development Environment
  – Getting and maintaining a local copy of the source code
  – Setting up a virtualenv
  – Installing dependencies
  – Running your local copy of CVE Binary Tool
  – Help, my checkers aren’t loading
  – Running tests
  – Running linters
    * Using pre-commit to run linters automatically
    * Running isort by itself
    * Running black by itself
    * Running bandit by itself
    * Other linting tools
  – Making a new branch & pull request
    * Commit message tips
    * Sharing your code with us
    * Checklist for a great pull request
  – Code Review
  – Style Guide for cve-bin-tool
    * String Formatting
  – Making documentation
6.1 Imposter syndrome disclaimer

We want your help. No really, we do.

There might be a little voice inside that tells you you’re not ready; that you need to do one more tutorial, or learn another framework, or write a few more blog posts before you can help with this project.

I assure you, that’s not the case.

This document contains some contribution guidelines and best practices, but if you don’t get it right the first time we’ll try to help you fix it.

The contribution guidelines outline the process that you’ll need to follow to get a patch merged. By making expectations and process explicit, we hope it will make it easier for you to contribute.

And you don’t just have to write code. You can help out by writing documentation, tests, or even by giving feedback about this work. (And yes, that includes giving feedback about the contribution guidelines.)

If you have questions or want to chat, we have a gitter chat room where you can ask questions, or you can put them in GitHub issues too.

Thank you for contributing!

This section is adapted from this excellent document from @adriennefriend

6.2 Development Environment

Linux is the preferred operating system to use while contributing to CVE Binary Tool. If you’re using Windows, we recommend setting up Windows Subsystem for Linux.

6.3 Getting and maintaining a local copy of the source code

There are lots of different ways to use git, and it’s so easy to get into a messy state that there’s a comic about it. So… if you get stuck, remember, even experienced programmers sometimes just delete their trees and copy over the stuff they want manually.

If you’re planning to contribute, first you’ll want to get a local copy of the source code (also known as “cloning the repository”)

$ git clone git@github.com:intel/cve-bin-tool.git

Once you’ve got the copy, you can update it using

$ git pull

You’re also going to want to have your own “fork” of the repository on GitHub. To make a fork on GitHub, read the instructions at Fork a repo. A fork is a copy of the main repository that you control, and you’ll be using it to store and share your code with others. You only need to make the fork once.

Once you’ve set up your fork, you will find it useful to set up a git remote for pull requests:

$ git remote add myfork git@github.com/MYUSERNAME/cve-bin-tool.git

Replace MYUSERNAME with your own GitHub username.
6.4 Setting up a virtualenv

This section isn’t required, but many contributors find it helpful, especially for running tests using different versions of python.

`virtualenv` is a tool for setting up virtual python environments. This allows you to have all the dependencies for cve-bin-tool set up in a single environment, or have different environments set up for testing using different versions of Python.

To install it:

```
pip install virtualenv
```

To make a new venv using python 3.9:

```
virtualenv -p python3.9 ~/Code/venv3.9
```

Each time you want to use a virtualenv, you “activate” it using the activate script:

```
source ~/Code/venv3.9/bin/activate
```

And when you’re done with the venv, you can deactivate it using the `deactivate` command.

While you’re in a venv, the `python` command will point to whatever version you specified when the venv was created, and pip command will install things only in that venv so you don’t have to worry about conflicts with other versions or system packages.

6.5 Installing dependencies

The packages you need to run CVE Binary Tool are listed in the `requirements.txt` file in the main cve-bin-tool directory. You can install all of them using the following pip command:

```
pip install -U -r requirements.txt
```

The `-U` in that line above will update you to the latest versions of packages as needed, which we recommend because people running security tools generally want to have all the latest updates if possible. The `-r requirements.txt` specifies the file with all the requirements.

We also have a recommended list of dependencies just for developers that include things like the flake8 linter. You probably want to install them too if you’re intending to be a developer.

```
pip install -r dev-requirements.txt
```

6.6 Running your local copy of CVE Binary Tool

One of the reasons we suggest virtualenv is that it makes it easier to do this section.

If you want to run a local copy of cve-bin-tool, the recommended way is to install it locally. From the cve-bin-tool main directory, run:

```
python3 -m pip install --user -e .
```
Then you can type `cve-bin-tool` on the command line and it will do the right thing. This includes some special code intended to deal with adding new checkers to the list on the fly so things should work seamlessly for you while you’re building new contributions.

### 6.7 Help, my checkers aren’t loading

CVE Binary Tool uses the installed egg file to figure out which checkers are installed. If you run it directly without installing it (e.g. you try to use `python -m cve_bin_tool.cli`), it will usually work fine but you may occasionally find that checkers aren’t loading properly. Typically this happens with new checkers you are adding, but sometimes if you `git pull` it will cause a similar effect. If you get into this state, you can fix it by running the following command from the main `cve-bin-tool` directory:

```
python setup.py egg_info
```

We recommend that you switch to having a local install (e.g. run `pip install --user -e .` in the main `cve-bin-tool` directory) to avoid this problem in the future.

### 6.8 Running tests

The CVE Binary Tool has a set of tests that can be run using `pytest` command. Typically you want to run `pytest` in the `cve-bin-tool` directory to run the short test suite and make sure tests pass.

There is a README file in the tests directory which contains more info about how to run specific tests, or how to run the longer tests.

We have done our best to make tests stable and ensure that they pass at all times, but occasionally tests may fail due to factors outside your control (common causes: internet connectivity, rate limiting by NVD or new vulnerability data changing our test expectations). If a test doesn’t pass, you should look at it to see if any changes you made caused the failure. If you’re not sure, submit your code as a pull request and mention the issue and someone will try to help you sort it out.

When you submit your code as a pull request, the whole test suite will be run on windows and linux using the versions of python we support, including longer tests. We don’t expect you to do all that yourself; usually trying for one version of python on whatever local OS you have is good enough and you can let GitHub Actions do the rest!

### 6.9 Running linters

CVE Binary Tool uses a few tools to improve code quality and readability:

- `isort` sorts imports alphabetically and by type
- `black` provides automatic style formatting. This will give you basic PE8 compliance. (PE8 is where the default python style guide is defined.)
- `flake8` provides additional code “linting” for more complex errors like unused imports.
- `pyupgrade` helps us be forward compatible with new versions of python.
- `bandit` is more of a static analysis tool than a linter and helps us find potential security flaws in the code.
- `gitlint` helps ensure that the commit messages follow Conventional Commits.

We provide a `dev-requirements.txt` file which includes all the precise versions of tools as they’ll be used in GitHub Actions. You can install them all using pip:
pip install -r dev-requirements.txt

6.9.1 Using pre-commit to run linters automatically

We've provided a pre-commit hook (in .pre-commit.config.yaml) so if you want to run isort/Black locally before you commit, you can install the hook as follows from the main cve-bin-tool directory:

```bash
pre-commit install --hook-type pre-commit --hook-type commit-msg
```

Once this is installed, all of those commands will run automatically when you run `git commit` and it won't let you commit until any issues are resolved. (You can also run them manually using `pre-commit` with no arguments.) This will only run on files staged for commit (e.g. things where you've already run `git add`). If you want to run on arbitrary files, see below:

6.9.2 Running isort by itself

To format the imports using isort, you run `isort --profile black` followed by the filename. You will have to add `--profile black` when calling isort to make it compatible with Black formatter. For formatting a particular file name `filename.py`.

```bash
isort --profile black filename.py
```

Alternatively, you can run isort recursively for all the files by adding . instead of filename

```bash
isort --profile black .
```

6.9.3 Running black by itself

To format the code, you run `black` followed by the filename you wish to reformat. For formatting a particular file name `filename.py`.

```bash
black filename.py
```

In many cases, it will make your life easier if you only run black on files you've changed because you won't have to scroll through a pile of auto-formatting changes to find your own modifications. However, you can also specify a whole folder using ./

6.9.4 Running bandit by itself

We have a configuration file for bandit called `bandit.conf` that you should use. This disables a few of the checkers. To run it on all the code we scan, use the following:

```bash
bandit -c bandit.conf -r cve_bin_tool/ test/
```

You can also run it on individual files:

```bash
bandit -c bandit.conf filename.py
```
If you run it without the config file, it will run a few extra checkers, so you'll get additional warnings.

Bandit helps you target manual code review, but bandit issues aren't always things that need to be fixed, just reviewed. If you have a bandit finding that doesn't actually need a fix, you can mark it as reviewed using a `# nosec` comment. If possible, include details as to why the bandit results are ok for future reviewers. For example, we have comments like `#nosec uses static https url above` in cases where bandit prompted us to review the variable being passed to `urlopen()`.

### 6.9.5 Other linting tools

As well as black for automatically making sure code adheres to the style guide, we use `flake8` to help us find things like unused imports. The `flake8` documentation covers what you need to know about running it.

We use `pyupgrade` to make sure our syntax is updated to fit new versions of python.

We also have a spell checker set up to help us avoid typos in documentation. The [spelling actions readme file](https://github.com/intel/cve-bin-tool) gives more information including how to add new words to the dictionary if needed.

We also have a tool to help make sure that new checkers are added to the tables in our documentation and relevant words associated with checker names are put in allow dictionary for spelling checks, this is done automatically with GitHub actions. The [format_checkers code is here](https://github.com/intel/cve-bin-tool), if you’re curious.

You can view all the config files for GitHub Actions (what we use for Continuous Integration (CI)) in the `.github/workflows` directory.

### 6.10 Making a new branch & pull request

Git allows you to have “branches” with variant versions of the code. You can see what’s available using `git branch` and switch to one using `git checkout branch_name`.

To make your life easier, we recommend that the main branch always be kept in sync with the repo at [https://github.com/intel/cve-bin-tool](https://github.com/intel/cve-bin-tool), as in you never check in any code to that branch. That way, you can use that “clean” main branch as a basis for each new branch you start as follows:

```
git checkout main
git pull
```

```
git checkout -b my_new_branch
```

Note: If you accidentally check something in to main and want to reset it to match our main branch, you can save your work using `checkout -b` and then do a `git reset` to fix it:

```
git checkout -b saved_branch
```

```
git reset --hard origin/main
```

You do not need to do the `checkout` step if you don’t want to save the changes you made.

When you’re ready to share that branch to make a pull request, make sure you’ve checked in all the files you’re working on. You can get a list of the files you modified using `git status` and see what modifications you made using `git diff`.

Use `git add FILENAME` to add the files you want to put in your pull request, and use `git commit` to check them in. Try to use a clear commit message and use the [Conventional Commits format](https://www.conventionalcommits.org/).
6.10.1 Commit message tips

We usually merge pull requests into a single commit when we accept them, so it’s fine if you have lots of commits in your branch while you figure stuff out, and we can fix your commit message as needed then. But if you make sure that at least the title of your pull request follows the Conventional Commits format that you’d like for that merged commit message, that makes our job easier!

GitHub also has some keywords that help us link issues and then close them automatically when code is merged. The most common one you’ll see us use looks like fixes: #123456. You can put this in the title of your PR (what usually becomes the commit message when we merge your code), another line in the commit message, or any comment in the pull request to make it work. You and read more about linking a pull request to an issue in the GitHub documentation.

6.10.2 Sharing your code with us

Once your branch is ready and you’ve checked in all your code, push it to your fork:

```bash
git push myfork
```

From there, you can go to our pull request page to make a new pull request from the web interface.

6.10.3 Checklist for a great pull request

Here’s a quick checklist to help you make sure your pull request is ready to go:

1. Have I run the tests locally on at least one version of Python?
   - Run the command `pytest` (See also Running Tests)
   - GitHub Actions will run the tests for you, but you can often find and fix issues faster if you do a local run of the tests.

2. Have I run the code linters and fixed any issues they found?
   - We recommend setting up pre-commit so these are run automatically (See also Running Linters)
   - GitHub Actions will run the linters for you too if you forget! (And don’t worry, even experienced folk forget sometimes.)
   - You will be responsible for fixing any issue found by the linters before your code can be merged.

3. Have I added any tests I need to prove that my code works?
   - This is especially important for new features or new checkers.

4. Have I added or updated any documentation if I changed or added a feature?
   - New features are often documented in MANUAL.md. (See Making documentation for more information.)

5. Have I used Conventional Commits to format the title of my pull request?

6. If I closed a bug, have I linked it using one of GitHub’s keywords? (e.g. include the text fixed #1234)

7. Have I checked on the results from GitHub Actions?
   - GitHub Actions will run all the tests, linters and a spell check for you. If you can, try to make sure everything is running cleanly with no errors before leaving it for a human code reviewer!
   - As of this writing, tests take less than 20 minutes to run once they start, but they can be queued for a while before they start. Go get a cup of tea or work on something else while you wait!
6.11 Code Review

Once you have created a pull request (PR), GitHub Actions will try to run all the tests on your code. If you can, make any modifications you need to make to ensure that they all pass, but if you get stuck a reviewer will see if they can help you fix them. Remember that you can run the tests locally while you’re debugging; you don’t have to wait for GitHub to run the tests (see the Running tests section above for how to run tests).

Someone will review your code and try to provide feedback in the comments on GitHub. Usually it takes a few days, sometimes up to a week. The core contributors for this project work on it as part of their day jobs and are usually on US Pacific time, so you might get an answer a bit faster during their work week.

If something needs fixing or we have questions, we’ll work back and forth with you to get that sorted. We usually do most of the chatting directly in the pull request comments on GitHub, but if you’re stuck you can also stop by our Gitter chat room to talk with folk outside of the bug.

Another useful tool is git rebase, which allows you to change the “base” that your code uses. We most often use it as git rebase origin/main which can be useful if a change in the main tree is affecting your code’s ability to merge. Rebasing is a bit much for an intro document, but there’s a git rebase tutorial here that you may find useful if it comes up.

Once any issues are resolved, we’ll merge your code. Yay!

In rare cases, the code won’t work for us and we’ll let you know. Sometimes this happens because someone else has already submitted a fix for the same bug, (Issues marked good first issue can be in high demand!) or because you worked on a checker that didn’t have a good signature. Don’t worry, these things happens, no one thinks less of you for trying!

6.12 Style Guide for cve-bin-tool

Most of our “style” stuff is caught by the black and flake8 linters, but we also recommend that contributors use f-strings for formatted strings:

6.12.1 String Formatting

Python provides many different ways to format the string (you can read about them here) and we use f-string formatting in our tool.

Note: f-strings are only supported in python 3.6+.

• Example: Formatting string using f-string

```python
#Program prints a string containing name and age of person
taxname = "John Doe"
age = 23
print(f"Name of the person is {name} and his age is {age}"

#Output
# "Name of the person is John Doe and his age is 23"
```

Note that the string started with the f followed by the string. Values are always added in the curly braces. Also we don’t need to convert age into string. (we may have used str(age) before using it in the string) f-strings are useful as they provide many cool features. You can read more about features and the good practices to use f-strings here.
6.13 Making documentation

The documentation for CVE Binary Tool can be found in the doc/ directory (with the exception of the README.md file, which is stored in the main directory but linked in the documentation directory for convenience).

Like many other Python-based projects, CVE Binary Tool uses Sphinx and ReadTheDocs to format and display documentation. If you're doing more than minor typo fixes, you may want to install the relevant tools to build the docs. There's a requirements.txt file available in the doc/ directory you can use to install sphinx and related tools:

```
cd doc/
pip install -r requirements.txt
```

Once those are installed, you can build the documentation using make in the docs directory:

```
make docs
```

or use sphinx-build directly with the following options:

```
sphinx-build -b html . _build
```

That will build the HTML rendering of the documentation and store it in the _build directory. You can then use your web browser to go to that directory and see what it looks like.

Note that you don’t need to commit anything in the _build directory. Only the .md and .rst files should be checked in to the repository.

If you don’t already have an editor that understands Markdown (.md) and RestructuredText (.rst) files, you may want to try out Visual Studio Code, which is free and has a nice Markdown editor with a preview.

6.14 Where should I start?

Many beginners get stuck trying to figure out how to start. You’re not alone!

Here’s three things we recommend:

1. Try something marked as a “good first issue” We try to mark issues that might be easier for beginners.
2. Add tests to an existing checker This will give you some practice with the test suite.
3. Add a new checker This will give you some deeper understanding of how the tool works and what a signature looks like. We have a few new checker requests listed in the “good first issue” list, or any linux library that has known CVEs (preferably recent ones) is probably interesting enough.
4. Suggest fixes for documentation. If you try some instruction and it doesn’t work, or you notice a typo, those are always easy first commits! One place we’re a bit weak is instructions for Windows users.

If you get stuck or find something that you think should work but doesn’t, ask for help in an issue or stop by the cve-bin-tool gitter to ask questions.

Note that our “good first issue” bugs are in high demand during the February-April due to the start of Google Summer of Code. It’s totally fine to comment on a bug and say you’re interested in working on it, but if you don’t actually have any pull request with a tentative fix up within a week or so, someone else may pick it up and finish it. If you want to spend more time thinking, the new checkers (especially ones no one has asked for) might be a good place for a relaxed first commit.
CHAPTER
SEVEN

INDICES AND TABLES

• genindex
• modindex
• search