OSS Index Nuget Vulnerability Auditor

Requirements

This document enumerates the requirements for the **OSS Index Nuget Vulnerability Auditor** (v1.0).

Overview

OSS Index (https://ossindex.net) provides a REST API that allows the user to search for and locate Nuget packages/artifacts, locate the source code for said package, and report on any known security vulnerabilities that affect the package. The goal of the project is to build a tool that will read the Nuget "package.config" file which identifies a project's package dependencies, and inform the user of any vulnerabilities affecting any of these dependencies.

This project may be broken down into three separate, but related, components:

1. **Common audit library:** reads dependency information from a package.config file, then cross references the information with the OSS Index database through a REST API to identify packages with vulnerabilities.
2. **Command line wrapper:** reports on all dependencies, highlighting vulnerabilities in a command window.
3. **Visual Studio extension**:
   - 3.1 Highlights dependencies in the packages.config file(s) that have vulnerabilities
   - 3.2 List vulnerabilities and affected packages in the "Error List"

This document enumerates the requirements for all three of these components.

Supplementary Documentation

There are several supplementary documents which provide further details on the REST API (and how best to use it), UI mock ups, etc. These documents are included below.

- **OSS Index API User Guide**. Indicates not only the API calls to use, but suggests best practices.
- **Visual Studio UI Mockup**
- **Command Line Mockup**

Requirements preamble

The requirements are broken into individual sections, one general and the rest on a per “component” basis. We have created these requirements based on our understanding of C#, Nuget, and Visual Studio. Please let us know if the requirements are not possible due to a misunderstanding of the technologies in play. We can adjust the requirements accordingly.

1. General Requirements

Req. ID	Description
1.1	Code should be written in C#
1.2	Project should be created using Visual Studio
1.3	Solution must be readable and compile using Visual Studio Community 2015
1.4	Solution must not use GPLed or commercial libraries (with the exception of those provided by Visual Studio that can be released in an open source product). Libraries with the following licenses are acceptable: • BSD • Apache • MIT • Mozilla (MPL) To use code with another license please contact Vör Security.
1.5	The final product code will be released under the BSD 3-clause license. Author may put their name and information in the code, but by Vör Security will be applying their copyright information and license at the top of each file. Of course, as BSD code you can do whatever you want with the final product once it is released.

2. Common Audit Library Requirements

Req. ID	Description
2.1	The audit library should compile and link as a static library that is subsequently used by the command line wrapped and visual studio extension.
2.2	The library will have a function/method that takes as input a packages.config path, parse out the package dependency information, pass to the second function/method defined in requirement 2.3, and return the resulting list of packages with their associated version and vulnerabilities.
2.2.1	a packages.config with invalid contents (not XML, missing packages) should throw an exception.
2.2.2	The program should collect all package dependency information from the packages.config file. This includes the package names and associated version number (or range).
2.2	The library will have a function/method that takes as input a list of packages (see 2.2.1 for details) and return said list of packages with associated vulnerabilities (see 2.2.7)
2.2.1	The library will take as input a list of packages, defined by NAME and a version number or range. More description of version ranges can be found here: https://docs.nuget.org/create/versioning#Specifying-Version-Ranges-in-.nuspec-Files
2.2.2	The library will do simple parsing of the range to figure out the **minimum** version number the range is applicable to. This will simplify the REST queries.
2.2.3	The library will perform a bulk request to the OSS Index “package search” API and will be returned a list of found packages with associated information. See **api.pdf** for details.
2.2.4	The library will perform a bulk request to the OSS Index “SCM details” API to get details on the source packages associated with the aforementioned packages. See **api.pdf** for details.
2.2.5	For each “SCM details” returned that has associated vulnerabilities, perform a “vulnerability” request to get the list of vulnerabilities. See **api.pdf** for details.
2.2.6	Identify which vulnerabilities affect the minimum version, identified above. See **api.pdf** for details.
2.2.7	Return a list of packages associated with their associated version range and identified vulnerabilities
2.3	All results to REST API requests should be cached to disk, so that if the same request is made multiple times the REST API (web service) is not called for subsequent calls
2.4	Cached results should be purged (ignored) after 12 hours, so that new data can be retrieved from the web service (in case the data on the server has changed).

3. Command Line Wrapper Requirements

Req. ID	Description
3.1	The command line wrapper code should compile and link with the common library to produce a single executable (audit-net.exe)
3.2	Execution of the program with no arguments should present usage text and exit cleanly. The usage text would be something like this: **`Usage: audit-net.exe <packages.config>`**
3.2	Execution of the program with invalid arguments should result in an error message and the usage text.
3.3	Execution of the program with a packages.config with invalid contents (not XML, missing packages) should result in an error message.
3.5	The packages.config file path will be passed to the common audit library which will return a set of results for each dependency.
3.6	Each dependency will have some text printed to the screen indicating the dependency and vulnerability situation. See the **cmd-mockup.pdf** file for possible scenarios.
3.7	Each scenario described in the **cmd-mockup.pdf** file will be handled.
3.8	If vulnerabilities are found affecting any package in the packages.config file, the command will exit (on completion) with a non-zero error code indicating the number of packages with “active” vulnerabilities.

4. Visual Studio Extension Requirements

Req. ID	Description
4.1	Extension should be installable and usable in as large a range of products as possible.
4.1.1	Extension should be installable and usable in Visual Studio Professional 2012 (if possible)
4.1.2	Extension should be installable and usable in Visual Studio Enterprise 2012 (if possible)
4.1.3	Extension should be installable and usable in Visual Studio Professional 2013
4.1.4	Extension should be installable and usable in Visual Studio Enterprise 2013
4.1.5	Extension should be installable and usable in Visual Studio Community 2013
4.1.6	Extension should be installable and usable in Visual Studio Professional 2015
4.1.7	Extension should be installable and usable in Visual Studio Enterprise 2015
4.1.8	Extension should be installable and usable in Visual Studio Community 2015
4.2	The extension operates on solutions once they are loaded into Visual Studio
4.3	Identify all package.config files in the solution
4.4	Each packages.config file path will be passed to the common audit library which will return a set of results for each dependency.
4.5	All vulnerabilities returned as a result of the common audit library call will be added to the Visual Studio “Error List” in a similar manner to that found in the **vs-mockup.pdf** file.
4.6	Double clicking an associated error in the “Error list” should open the associated packages.config file in the default editor (XML Editor, from the looks of it).
4.7	All packages with “active” vulnerabilities found will be highlighted/annotated in the XML editor that is used when the packages.config file is opened.
4.8	If the packages.config file is directly edited the file should be parsed and checked again for vulnerabilities (go back to requirement 4.4)
4.9	If the packages.config file is edited indirectly through use of the Nuget Package Manager the file should be parsed and checked again for vulnerabilities (go back to requirement 4.4)

OSS Index API User Guide

(Specifically for Nuget Auditor project)

The following document discusses the OSS Index API as it relates to the Nuget Auditor project.

Overview

The OSS Index REST API provides information about packages, source code, and vulnerabilities.

There are three main calls that can be performed on the OSS Index API which are applicable to the Nuget Auditor project:

1. Package search: Which allows the user to search for nuget packages and artifacts by name and version.
2. SCM details: Many results from the “package search” will provide information about an “SCM” resource. The SCM details request provides information about the source code for the package.
3. Vulnerabilities: If the “SCM details” indicates there are vulnerabilities identified for the package, the “Vulnerability” request will return a list of vulnerabilities.

Package search

Type	POST
URL	https://ossindex.net/v1.0/search/nuget/

Body	`[`
	`  {"pm": "nuget", "name": ":package name", "version": ":version"},`
	`  {"pm": "nuget", "name": ":package name", "version": ":version"},`
	`  ...`
	`]`

Example	`[`
	`  {"pm": "nuget", "name": "AjaxControlToolkit", "version": "7.1213.0"}`
	`]`

Response	`[`
	`{`
	`  "name": "AjaxControlToolkit",`
	`  "version": "7.1213.0",`
	`  ...`
	
	`  "scm_id": 7098995671,`
	`  ...`
	`  "search": [`
	`  "nuget",`
	`  "AjaxControlToolkit",`
	`  "https://www.nuget.org/api/v2/package/AjaxControlToolkit/7.1213.0",`
	`  "1.2.1"`
	`  ]`
	`},`
	`...`
	`]`

The package search should be done as a “batch” search. All package dependencies within a file can be queried at the same time (to a maximum of 100 dependencies). If there are more than 100 dependencies they need to be queried in multiple requests.

Note that not every package will have an associated SCM. It is also possible that some packages will not be returned in the search.

The pertinent response fields are shown above. The “scm_id” is used as input for the “SCM details” query. The “search” field is the user's search parameters that caused the match. This information should be enough to identify which package request resulted in which scm_id.

SCM details

Type	GET
URL	https://ossindex.net/v1.0/scm/:id[,:id]
Example	`https://ossindex.net/v1.0/scm/284089289,8322029565`

Response	`[`
	`  {`
	`  "id": 284089289,`
	`  ...`
	`  "hasVulnerability": true,`
	`  "vulnerabilities": "https://ossindex.net/v1.0/scm/284089289/vulnerabilities",`
	`  ...`
	`  },`
	`  {`
	`  "id": 8322029565,`
	`  ...`
	`  "hasVulnerability": false,`
	`  ...`
	`  }`
	`]`

The package search should be done as a “batch” request. All package dependencies within a file can be queried at the same time, to a maximum of 100 IDs. If there are more than 100 IDs the request needs to be broken up into multiple requests.

The response includes parameters that will be interesting. Note that the results IDs match the scm_id in the request.

Some responses will indicate that there are vulnerabilities, while some indicate that there are no vulnerabilities. This is important information for the down stream tools.

Vulnerabilities

Type	POST
URL	https://ossindex.net/v1.0/scm/:id/vulnerabilities
Example	`https://ossindex.net/v1.0/scm/284089289/vulnerabilities`
Response	`[{ "uri": "cve:/CVE-2015-4670", "id": 7163420065, "title": "[CVE-2015-4670] Improper Limitation of a Pathname to a Restricted Directory ("Path Traversal")", "summary": "Directory traversal vulnerability in the AjaxFileUpload control in DevExpress AJAX Control Toolkit (aka AjaxControlToolkit) before 15.1 allows remote attackers to write to arbitrary files via a .. (dot dot) in the fileId parameter to AjaxFileUploadHandler.axd.", "cve-id": "CVE-2015-4670", "details": "https://ossindex.net/v1.0/cve/7163420065", "versions": ["15.0"] }]`

The “vulnerability” request must be performed on a case-by-case basis, as there is no batch version at this time. For every “SCM details” that returns a true for “hasVulnerability” a vulnerability request should be performed. This returns details for each vulnerability for the package (SCM).

Note that the “versions” attribute may be a list of versions, or a list of version ranges (defined as semantic version ranges). The supported ranges are quite simple and minimal, some examples include:

- 15.0.0
- <15.0.0
- <=15.0.0
- >0

These versions are used in the “Common audit library” for comparison with the user's installed package version. If the user's version is contained within the specified version list or range (list) then the vulnerability affects the specified package version (is considered “active”).

There are “semantic version” packages for .NET (available through Nuget). It is possible that one or more of them support the range definitions as described above. Alternatively some simple parsing could change these simple ranges to that supported by these libraries.

OSS Index Nuget Vulnerability Auditor

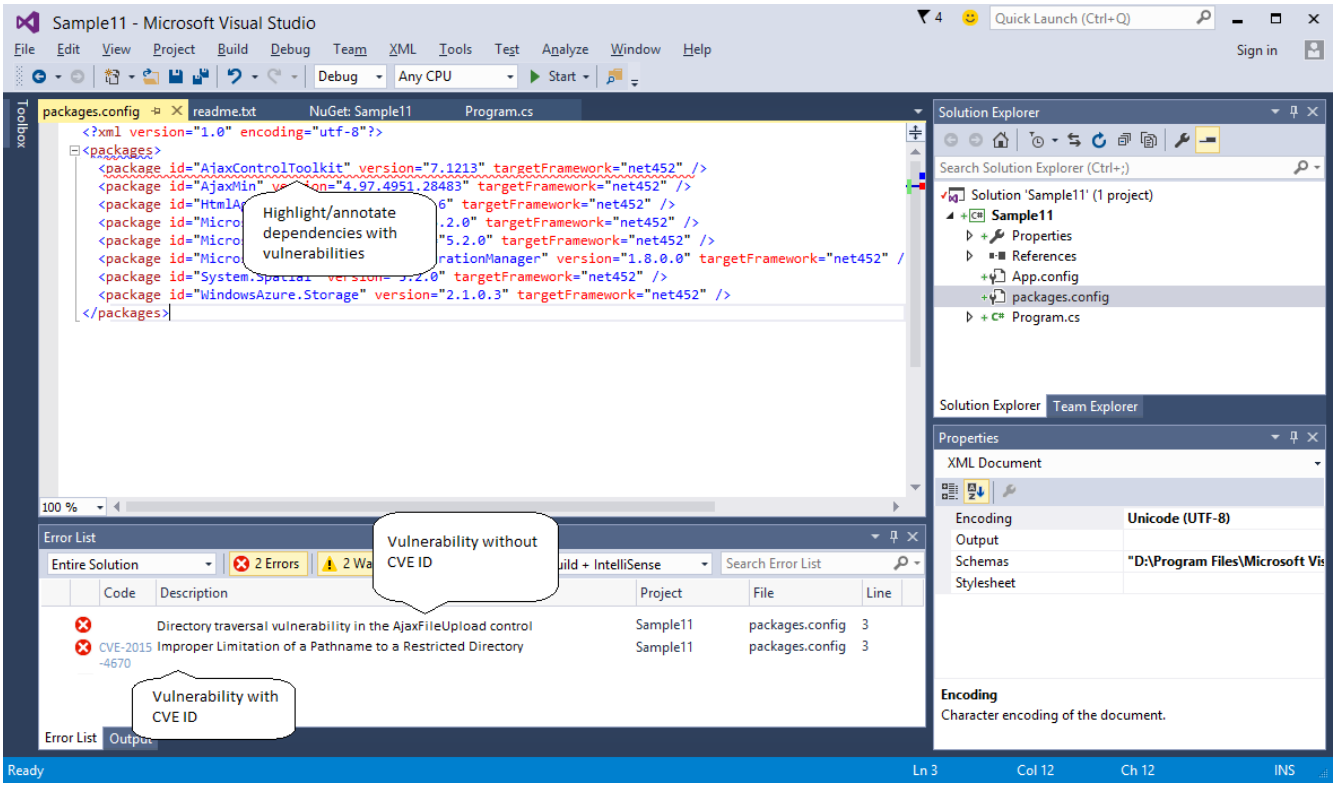
Visual Studio UI Mockup

The OSS Index Nuget Vulnerability Auditor Visual Studio extension uses the common audit library to identify dependencies and their vulnerabilities, and displays them as annotations in the editor and as errors in the “Error List”.

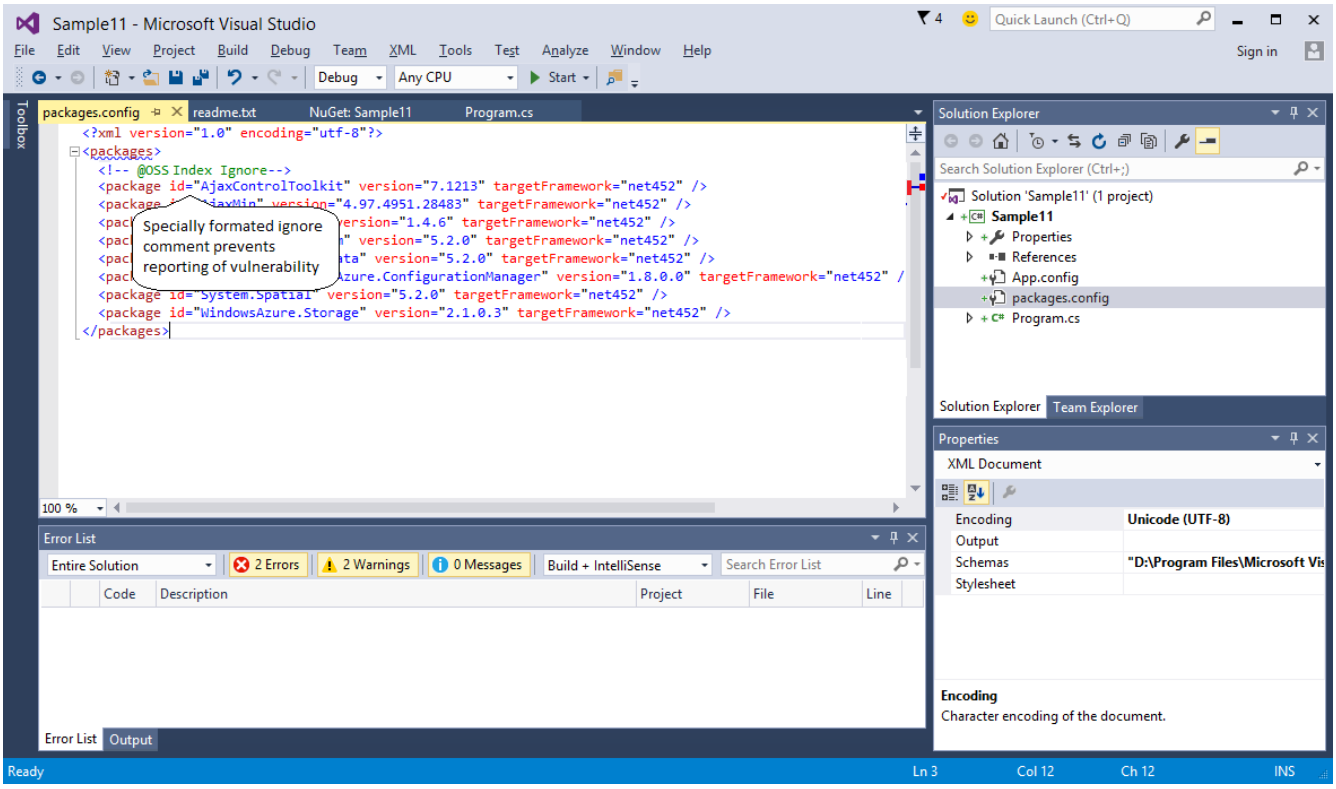
Dependencies should be checked whenever the a packages.config file is changed. It may be edited by hand, but is more likely edited using the built in Nuget Package Manager.

Bear in mind that for complex solutions there may be multiple packages.config files. They should each be checked.

The following mock up illustrates how these annotations and errors may appear. From my understanding, the package dependencies can be found within the package.config files, so this is likely the easiest place to highlight vulnerabilities.



As mentioned in the requirements, users can prevent vulnerabilities from being reported using a simple comment in the package.config preceding the package with the vulnerability.



OSS Index Nuget Vulnerability Auditor

Command Line Mockup

The OSS Index Nuget Vulnerability Auditor command line tool uses the common audit library to identify dependencies and their vulnerabilities, and prints the dependency and vulnerability information to a command window.

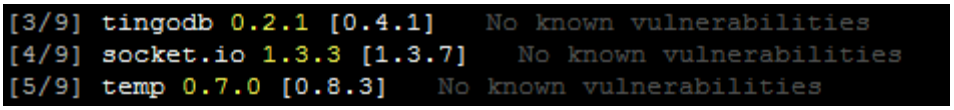
The user provides a “packages.config” file through a command line argument.

The following screenshots are from the “auditjs” command which is available for node.js, the functionality will be quite similar.

Scenario 1

- Package: Known
- SCM: Known
- Vulnerabilities: None

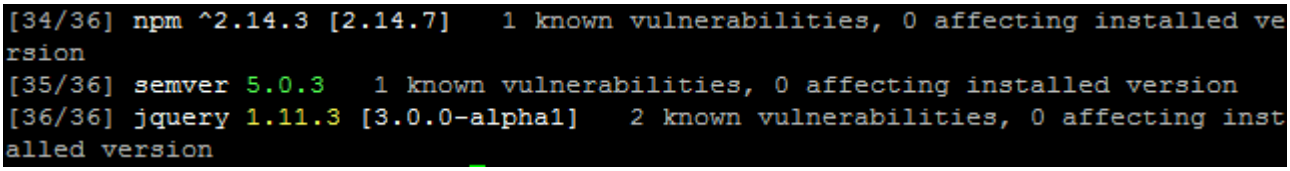
Starts with a number showing [package num/total packages]. List the package name and version number. Don't worry about the version number in the brackets, this is not a requirement for this project. Trailing text indicates there are no known vulnerabilities.



Scenario 2

- Package: Known
- SCM: Known
- Vulnerabilities: Yes, but none applicable to this version

Starts with a number showing [package num/total packages]. List the package name and version number or version range (whatever is specified in the package.config file). Trailing text indicates number of known vulnerabilities and that none affect this version.



Scenario 3

- Package: Known
- SCM: Known
- Vulnerabilities: Yes, with one or more applicable to the minimum version for the package

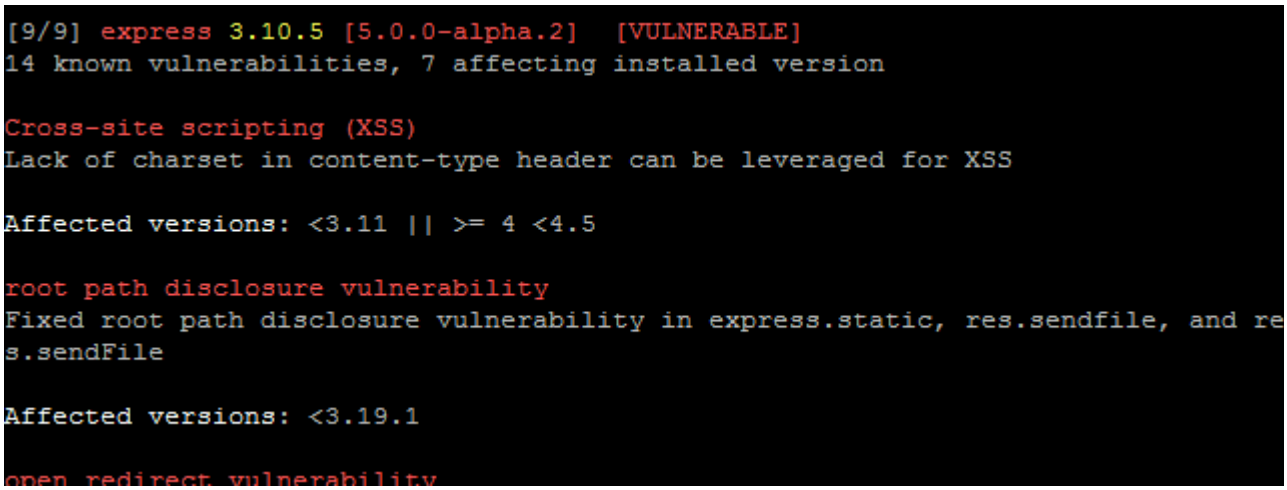
First line: Starts with a number showing [package num/total packages]. List the package name and version number or version range (whatever is specified in the package.config file). Trailing text indicates the package is vulnerable.

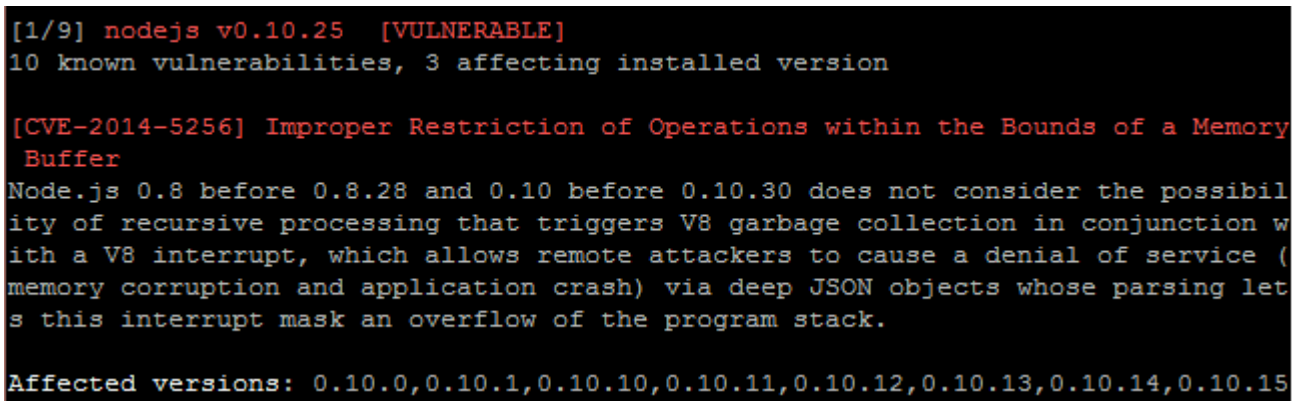
Second line: Indicate total number of vulnerabilities, followed by number affecting the installed version

For each vulnerability affecting the installed version, print:

- Blank line
- [CVE ID] Title
- Description

If the vulnerability has a CVE ID, print the CVE ID in brackets before the title, otherwise do not.





Scenario 4

- Package: Known
- SCM: Unknown
- Vulnerabilities: Unknown

Starts with a number showing [package num/total packages] (similar to earlier examples). List the package name and version number or version range (whatever is specified in the package.config file). Trailing text should say “Unknown source for package”.

Scenario 5

- Package: Unknown
- SCM: Unknown
- Vulnerabilities: Unknown

Starts with a number showing [package num/total packages] (similar to earlier examples). List the package name and version number or version range (whatever is specified in the package.config file). Trailing text should say “Unknown package”.