MONTANA STATE UNIVERSITY

Hierarchical Software Quality Assurance

Montana State University

https://www.montana.edu/cyber/

SSP_Manager: OSCAL-based Automation of Security Control Compliance

User's Manual (Draft)

Funding Agency:

Science and Technology

CYBERSECURITY & INFRASTRUCTURE SECURITY AGENCY

SSP Manager

This research project is currently under development. This user manual will be updated as new functionalities are added. A separate “Documentation.pdf” will be created to guide maintainers through the source code.

IThis project runs on Linux (In particular, it is being developed on Ubuntu 22.04).

Installation

Make sure you have installed Git, Docker and Docker-compose. For autocompletion while editing OSCAL documents and highlighting of key or value errors, use VS Code. The project has the json schemas for the OSCAL documents, and we set VS Code to find them.

If you don't, Git can be installed with:

```
sudo apt install git
```

Docker can be installed with:

```
sudo apt install docker.io
```

Docker-compose can be installed with:

```
sudo apt install docker-compose
```

Add yourself to the docker group:

```
sudo usermod -aG docker yourUserName
```

(Restart your machine)

To install VS Code in Ubuntu:

```
sudo snap install code --classic
```

This project uses the BRON database developed by Hemberg et al. at MIT as a submodule. The original research for the database can be found as:

Hemberg, Erik, Jonathan Kelly, Michal Shlapentokh-Rothman, Bryn Reinstadler, Katherine Xu, Nick Rutar, and Una-May O'Reilly. "Linking threat tactics, techniques, and patterns with defensive weaknesses, vulnerabilities and affected platform configurations for cyber hunting." arXiv preprint arXiv:2010.00533 (2020).

To clone this repository, including the BRON submodule use:

```
git clone --recurse-submodules https://github.com/MSUSEL/msusel-ssp-manager.git
```

To create the graph database containing the different collections of cybersecurity data:

```
cd BRON
docker-compose up
```

This command will create two containers. One is an arangodb container that will host our database. The second is a bootstrap container that will populate the database with the data from different cybersecurity collections. (The bootstrap process can take up to 45 mins)

Once the bootstrap finishes, you can see the database in your browser at localhost:8529. The username is root and the password is changeme. Select the BRON database.

Go back to the msusel-ssp-manager directory and go into the oscal-processing directory:

```
cd ..
cd oscal-processing
docker build -t oscalprocessing .
```

This command will create a docker image for NIST's OSCAL validation tool. When a file is submitted for validation on the UI, the flask container will spin up a container for the validation tool using this docker image.

Go back to the msusel-ssp-manager directory and run the generate-env.sh scritp. This script stores the current working directory path in a .env file that will be created. The docker-compose command will read this file and inform the UI container of its location in the host file system. Run the script with: :

```
cd ..
./generate-env.sh
```

Now we'll add new collections to the BRON database. These are mappings from MITRE ATT&CK Techniques to NIST SP 800-53 security controls. These mapping where done by MITRE Engenuity Center for Threat-Informed Defense (see:https://github.com/center-for-threat-informed-defense/mappings-explorer/).

```
cd flask
docker-compose up
```

We call the first container that this command creates "driver" and it adds the new collections to the database. This command will also create a Python flask container that we'll be using as a provisional UI to test the tool. The driver container will take some time to complete (up to 30 minutes). When it finishes, the new collections will have been added to the database. You can see them at localhost:8529
The flask UI can be accessed at localhost:5000.

Set Schemas in VS Code

Open the project on VS Code and press Ctrl+Shift+P on the keyboard. On the search bar, type "Worspace json settings". Open the file and copy this content to it and save the changes:

```
{
   "json.schemas": [
       { "fileMatch": ["/flask/oscal_schemas/assessment-plans/*"],
       "url": "./flask/oscal_schemas/oscal_assessment-plan_schema.json" },
       { "fileMatch": ["/flask/oscal_schemas/assessment-results/*"],
       "url": "./flask/oscal_schemas/oscal_assessment-results_schema.json" },
       { "fileMatch": ["/flask/oscal_schemas/catalogs/*"],
       "url": "./flask/oscal_schemas/oscal_catalog_schema.json" },
       { "fileMatch": ["/flask/oscal_schemas/components/*"],
       "url": "./flask/oscal_schemas/oscal_component_schema.json" },
       { "fileMatch": ["/flask/oscal_schemas/POAMs/*"],
       "url": "./flask/oscal_schemas/oscal_poam_schema.json" },
       { "fileMatch": ["/flask/oscal_schemas/profiles/*"],
       "url": "./flask/oscal_schemas/oscal_profile_schema.json" },
       { "fileMatch": ["/flask/oscal_schemas/system-security-plans/*"],
       "url": "./flask/oscal_schemas/oscal_ssp_schema.json" }
   ],
   "yaml.schemas": {
       "./flask/oscal_schemas/oscal_assessment-plan_schema.json": ["/flask/oscal_schemas/assessment-plans/*"],
       "./flask/oscal_schemas/oscal_assessment-results_schema.json": ["/flask/oscal_schemas/assessment-results/*"],
       "./flask/oscal_schemas/oscal_catalog_schema.json": ["/flask/oscal_schemas/catalogs/*"],
       "./flask/oscal_schemas/oscal_component_schema.json": ["/flask/oscal_schemas/components/*"],
       "./flask/oscal_schemas/oscal_poam_schema.json": ["/flask/oscal_schemas/POAMs/*"],
       "./flask/oscal_schemas/oscal_profile_schema.json": ["/flask/oscal_schemas/profiles/*"],
       "./flask/oscal_schemas/oscal_ssp_schema.json": ["/flask/oscal_schemas/system-security-plans/*"]
   }
}
```

The application is now ready.

Going forward, to restart the application, you only need to restart the aragodb container:
docker start containerID
And the flask container with:
docker start containerID
You can stop them with:
docker stop containerID

Project Structure

After installation, the project will have two constantly running Docker containers: the BRON security collections database, and the flask web user interface. These two containers communicate through the host's network. The oscalprocessing container performs operations on OSCAL documents. This is an ephemeral container. It is created when the operation is to be performed and it is removed immediately after completion. The project has two volumes or directories for communication between the Docker containers and the host. The docker_socket volume is used to communicate with the host's Docker engine. The flask container uses this docker_socket volume to tell the Docker engine when it needs to spin up an instance of the oscalprocessing container. The flask_shared volume is the directory through which the flask container and the host exchange files.

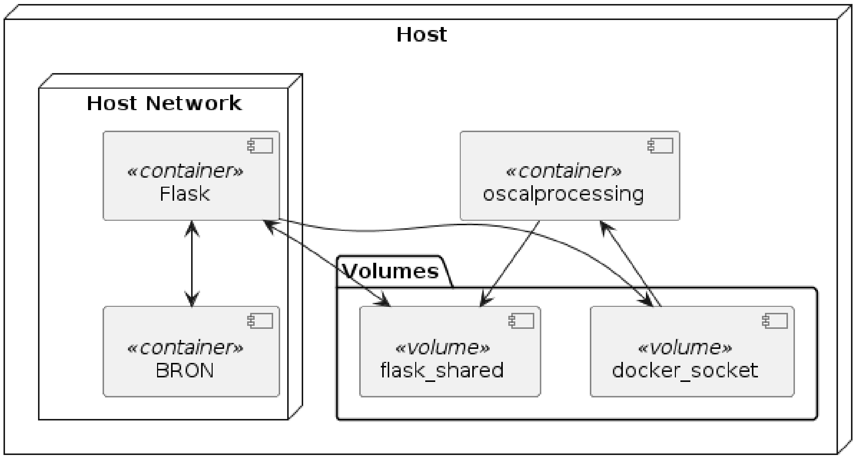
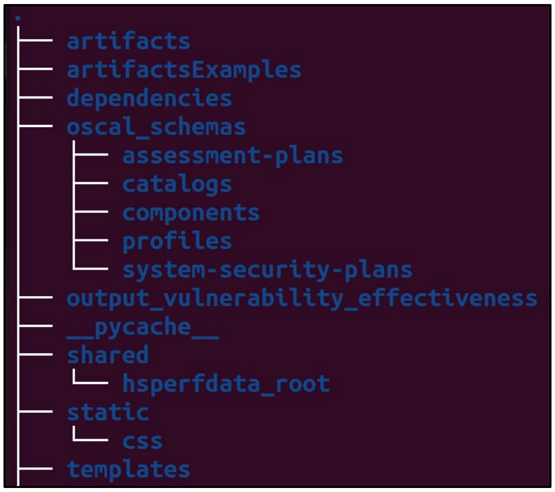


Figure 1. The containers at work when the application is running.

Most of the functionalities for the tool are implemented in the flask container. The image below shows the directories that contain the source code for these functionalities.



```
.
├── artifacts
├── artifactsExamples
├── dependencies
├── oscal_schemas
│   ├── assessment-plans
│   ├── catalogs
│   ├── components
│   ├── profiles
│   └── system-security-plans
├── output_vulnerability_effectiveness
├── __pycache__
├── shared
│   └── hsperfdata_root
├── static
│   └── css
├── templates
```

Figure 2. Directory structure of the flask container.

/artifacts: holds files that are generated as the program runs, but that are not final output files.

/artifactsExamples: contains examples of these files, and a description file. The description file contains brief descriptions of each of the artifacts that may be found in the artifacts directory. /artifactsExamples is not part of the functionality of the tool and may be removed if desired.

/dependencies: contains the program to be analyzed for vulnerabilities in its dependencies. Should include the requirements.txt file for the program. Once the vulnerability effectiveness test is performed, the container's dependencies directory will contain the source code for all the packages that the project installs and all the Python Stdlib files that the files in the project or it's dependencies import.

/oscal_schemas: contains the json schemas for the OSCAL documents. Each type of OSCAL document has it's own subdirectory where it's schema is configured. Each document type should be place in its directory when editing.

/output_vulnerability_effectiveness: after vulnerability effectiveness analysis is performed and the files are moved from the shared directory, the weaknesses found in the dependencies will be in a cwe.json file that will be placed in this directory.

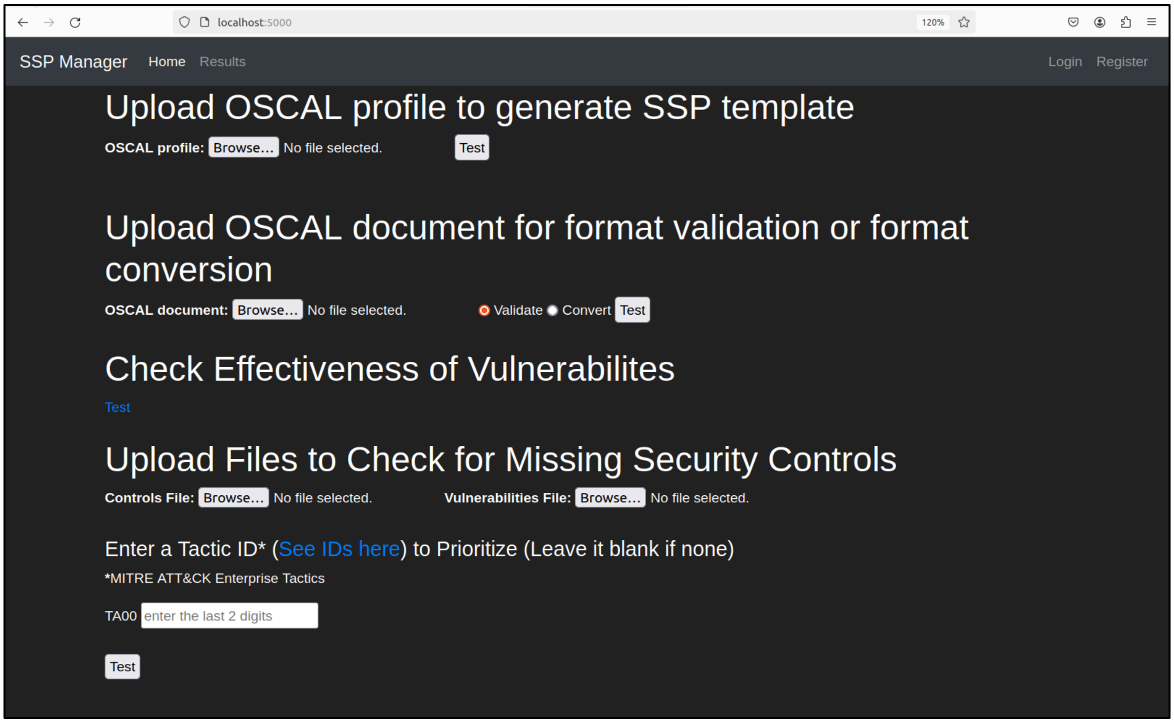
/_pycache_: a Python folder used to place Python bytecode from previous runs, and thus speeding up later executions of the program.

/shared: as mentioned above, is a shared directory with the host file system. The flask container and the host can exchange files through this directory.

/static: a directory for holding CSS files for the UI views.

/templates: contains the html and Jinja2 files that the application renders.

Provisional user interface:



Upload OSCAL profile to generate SSP template

OSCAL profile: Browse... No file selected. Test

Upload OSCAL document for format validation or format conversion

OSCAL document: Browse... No file selected. ● Validate ○ Convert Test

Check Effectiveness of Vulnerabilites

Test

Upload Files to Check for Missing Security Controls

Controls File: Browse... No file selected. **Vulnerabilities File:** Browse... No file selected.

Enter a Tactic ID* (See IDs here) to Prioritize (Leave it blank if none)

*MITRE ATT&CK Enterprise Tactics

TA00 enter the last 2 digits

Test

Figure 3. Home view (localhost:5000)

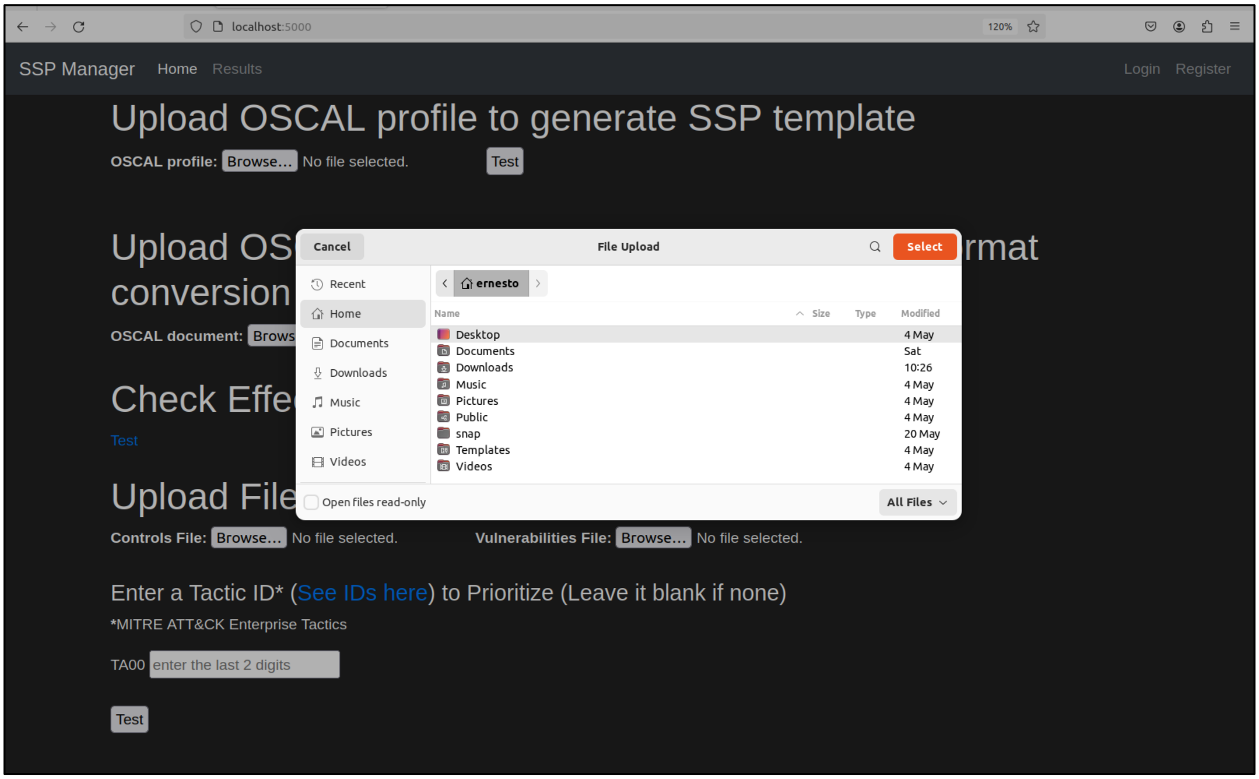
The tool has the following functionalities:

- Generating a SSP template.
- Validating the format of OSCAL documents.
- Converting OSCAL documents from YAML to JSON and vice-versa.
- Vulnerability effectiveness analysis for Python programs.
- Mapping CVEs/CWEs to NIST SP 800-53 security controls.

Generating, validating and converting are simple, straight-forward operations and will be presented first.

Generating SSP Template

This functionality allows users to generate an OSCAL System Security Plan (SSP) template from an OSCAL profile. The SSP template has the minimum required fields for an SSP in OSCAL format. Most of the values will need to be filled by the user as it is impossible to anticipate them. The json schemas that have been loaded into the VS Code IDE will indicate for the user additional fields that can be used.



Upload OSCAL profile to generate SSP template

OSCAL profile: Browse... No file selected. Test

Upload OS... rmat conversion

OSCAL document: Brows

Check Effe

Test

Upload File

Controls File: Browse... No file selected. **Vulnerabilities File:** Browse... No file selected.

Enter a Tactic ID* (See IDs here) to Prioritize (Leave it blank if none)

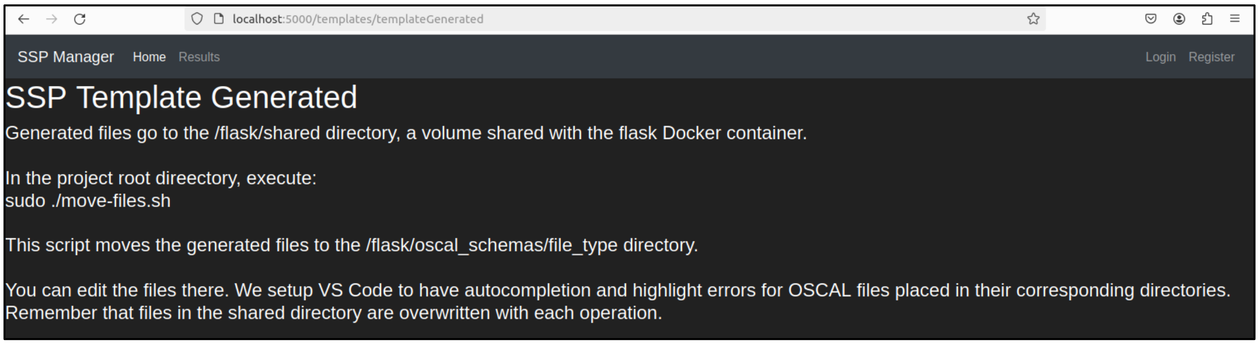
*MITRE ATT&CK Enterprise Tactics

TA00 enter the last 2 digits

Test

Name	Size	Type	Modified
Desktop			4 May
Documents			Sat
Downloads			10:26
Music			4 May
Pictures			4 May
Public			4 May
snap			20 May
Templates			4 May
Videos			4 May

Figure 4. The 'Browse' button has access to the user's file system. A profile.yaml file must be selected.



SSP Template Generated

Generated files go to the /flask/shared directory, a volume shared with the flask Docker container.

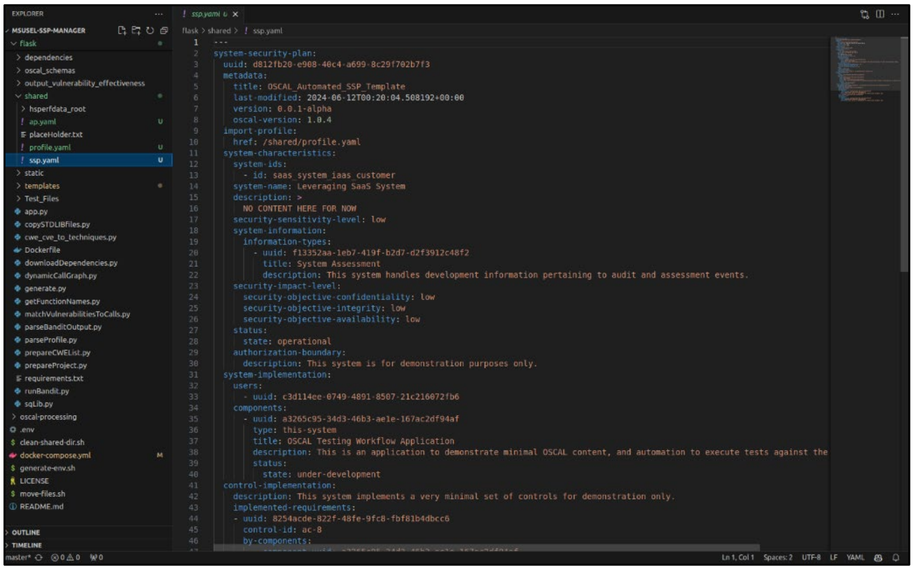
In the project root direectory, execute:
sudo ./move-files.sh

This script moves the generated files to the /flask/oscal_schemas/file_type directory.

You can edit the files there. We setup VS Code to have autocompletion and highlight errors for OSCAL files placed in their corresponding directories.
Remember that files in the shared directory are overwritten with each operation.

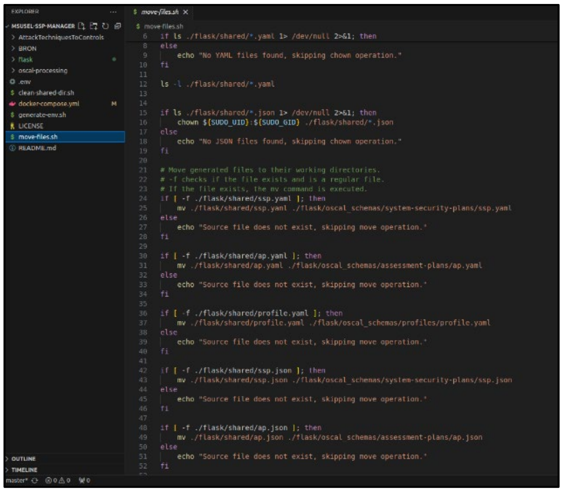
Figure 5. View after generating the SSP template. The user will edit the documents outside the container where the tool runs. The files must be moved as described.

Moving the files

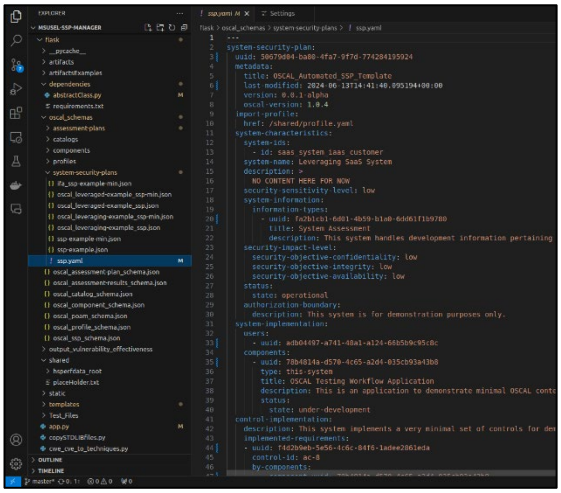


```yaml
---
system-security-plan:
  uuid: d812fb20-e908-40c4-a699-8c29f702b7f3
  metadata:
    title: OSCAL_Automated_SSP_Template
    last-modified: 2024-06-12T00:20:04.508192+00:00
    version: 0.0.1-alpha
    oscal-version: 1.0.4
  import-profile:
    href: /shared/profile.yaml
  system-characteristics:
    system-ids:
      - id: saas_system_iaas_customer
    system-name: Leveraging SaaS System
    description: >
      NO CONTENT HERE FOR NOW
    security-sensitivity-level: low
    system-information:
      information-types:
        - uuid: f13352aa-1eb7-419f-b2d7-d2f3912c48f2
          title: System Assessment
          description: This system handles development information pertaining to audit and assessment events.
    security-impact-level:
      security-objective-confidentiality: low
      security-objective-integrity: low
      security-objective-availability: low
    status:
      state: operational
    authorization-boundary:
      description: This system is for demonstration purposes only.
  system-implementation:
    users:
      - uuid: c3d114ee-0749-4891-8507-21c216072fb6
    components:
      - uuid: a3265c95-34d3-46b3-ae1e-167ac2df94af
        type: this-system
        title: OSCAL Testing Workflow Application
        description: This is an application to demonstrate minimal OSCAL content, and automation to execute tests against the
        status:
          state: under-development
  control-implementation:
    description: This system implements a very minimal set of controls for demonstration only.
    implemented-requirements:
    - uuid: 8254acde-822f-48fe-9fc8-fbf81b4dbcc6
      control-id: ac-8
      by-components:
```

Figure 6. The SSP is generated by the flask container and is stored in the /shared directory, which is shared by the container and the host computer.



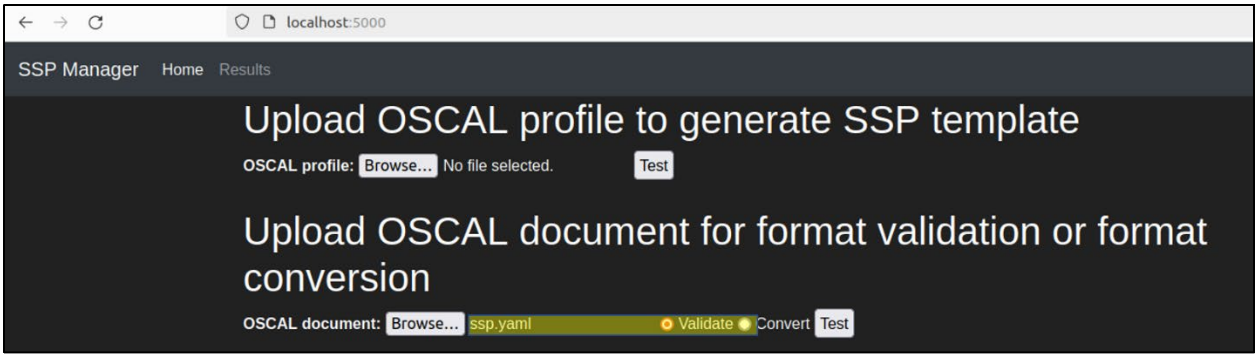
```bash
if ls ./flask/shared/*.yaml 1> /dev/null 2>&1; then
else
    echo "No YAML files found, skipping chown operation."
fi

ls -l ./flask/shared/*.yaml


if ls ./flask/shared/*.json 1> /dev/null 2>&1; then
    chown ${SUDO_UID}:${SUDO_GID} ./flask/shared/*.json
else
    echo "No JSON files found, skipping chown operation."
fi

# Move generated files to their working directories.
# -f checks if the file exists and is a regular file.
# If the file exists, the mv command is executed.
if [ -f ./flask/shared/ssp.yaml ]; then
    mv ./flask/shared/ssp.yaml ./flask/oscal_schemas/system-security-plans/ssp.yaml
else
    echo "Source file does not exist, skipping move operation."
fi

if [ -f ./flask/shared/ap.yaml ]; then
    mv ./flask/shared/ap.yaml ./flask/oscal_schemas/assessment-plans/ap.yaml
else
    echo "Source file does not exist, skipping move operation."
fi

if [ -f ./flask/shared/profile.yaml ]; then
    mv ./flask/shared/profile.yaml ./flask/oscal_schemas/profiles/profile.yaml
else
    echo "Source file does not exist, skipping move operation."
fi

if [ -f ./flask/shared/ssp.json ]; then
    mv ./flask/shared/ssp.json ./flask/oscal_schemas/system-security-plans/ssp.json
else
    echo "Source file does not exist, skipping move operation."
fi

if [ -f ./flask/shared/ap.json ]; then
    mv ./flask/shared/ap.json ./flask/oscal_schemas/assessment-plans/ap.json
else
    echo "Source file does not exist, skipping move operation."
fi
```



```yaml
---
system-security-plan:
  uuid: 50679d04-ba80-4fa7-9f7d-774284195924
  metadata:
    title: OSCAL_Automated_SSP_Template
    last-modified: 2024-06-13T14:41:40.895194+00:00
    version: 0.0.1-alpha
    oscal-version: 1.0.4
  import-profile:
    href: /shared/profile.yaml
  system-characteristics:
    system-ids:
      - id: saas_system_iaas_customer
    system-name: Leveraging SaaS System
    description: >
      NO CONTENT HERE FOR NOW
    security-sensitivity-level: low
    system-information:
      information-types:
        - uuid: fa2b1cb1-6d01-4b59-b1a0-6dd61f1b9780
          title: System Assessment
          description: This system handles development information pertaining
    security-impact-level:
      security-objective-confidentiality: low
      security-objective-integrity: low
      security-objective-availability: low
    status:
      state: operational
    authorization-boundary:
      description: This system is for demonstration purposes only.
  system-implementation:
    users:
      - uuid: adb04497-a741-48a1-a124-66b5b9c95c8c
    components:
      - uuid: 78b4814a-d570-4c65-a2d4-035cb93a43b8
        type: this-system
        title: OSCAL Testing Workflow Application
        description: This is an application to demonstrate minimal OSCAL conte
        status:
          state: under-development
  control-implementation:
    description: This system implements a very minimal set of controls for dem
    implemented-requirements:
    - uuid: f4d2b9eb-5e56-4c6c-84f6-1adee2861eda
      control-id: ac-8
      by-components:
```

Figure 7. On the left, the script to move the files out of the shared volume. On the right, the location of the file after executing the move script.

Validating OSCAL documents



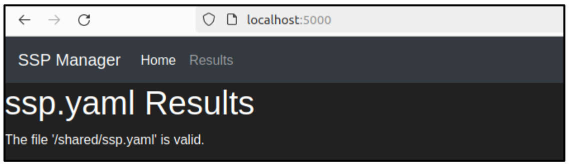
Upload OSCAL profile to generate SSP template

OSCAL profile: No file selected.

Upload OSCAL document for format validation or format conversion

OSCAL document: ssp.yaml ○ Validate ○ Convert

Figure 8. OSCAL documents in YAML or JSON format can be verified for correct OSCAL format. The tool integrates NIST's OSCAL-CLI validation tool. The validation checks that all required fields are present in the document, that any additional fields are part of the format definition.

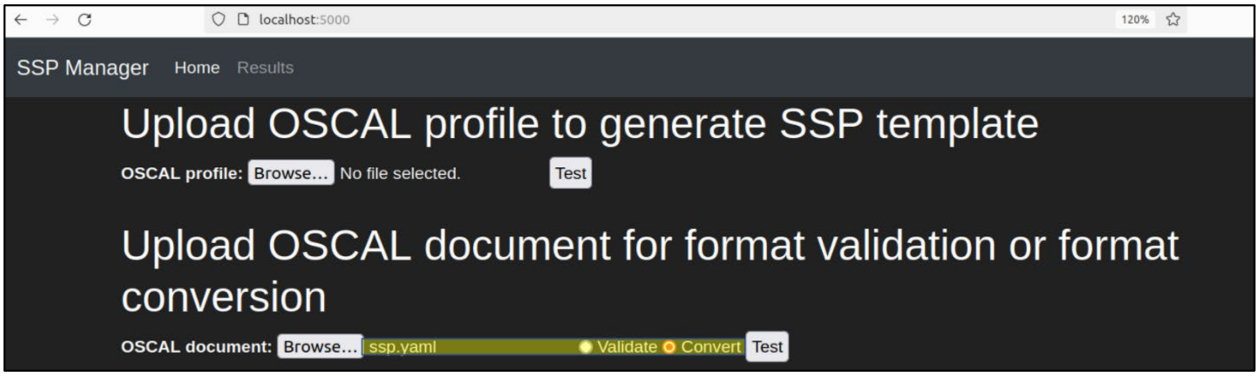


ssp.yaml Results

The file '/shared/ssp.yaml' is valid.

Figure 9. If the format of the file is valid, you'll see a short message. If there are issues with the file, text will be displayed which points out the issue so that the format can be corrected.

Converting between formats



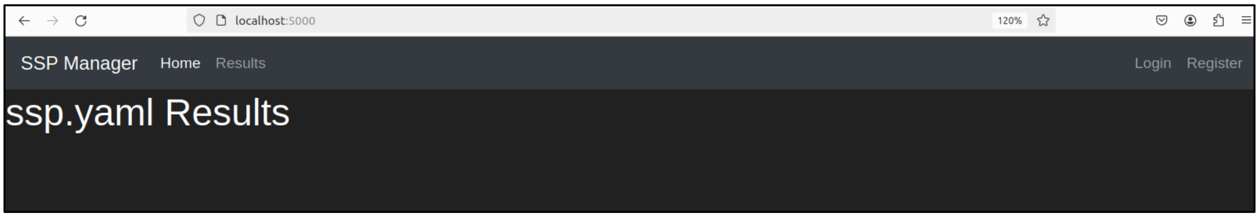
Upload OSCAL profile to generate SSP template

OSCAL profile: No file selected.

Upload OSCAL document for format validation or format conversion

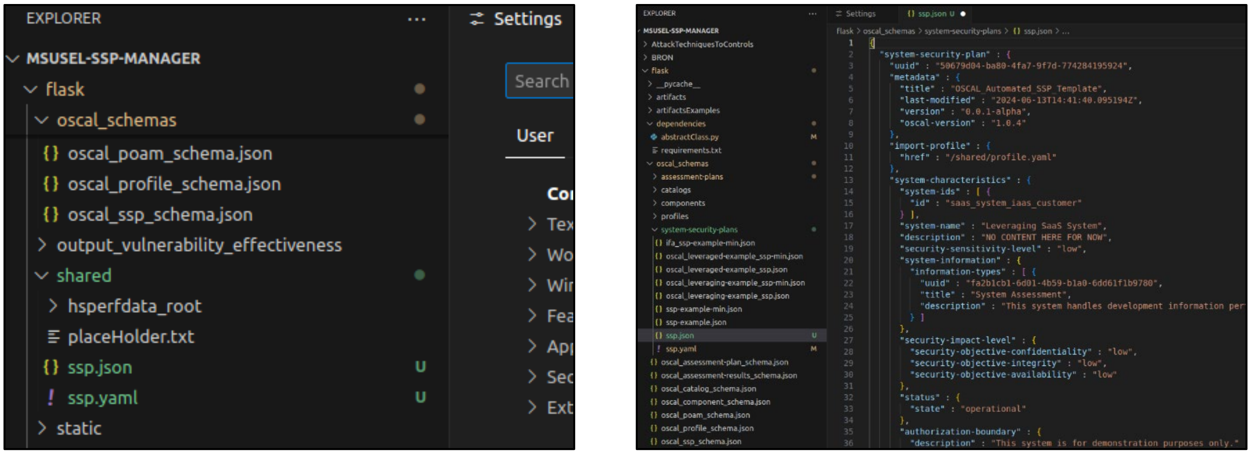
OSCAL document: ssp.yaml ○ Validate ○ Convert

Figure 10. Converting between formats refers to having OSCAL documents go from json to yaml, and vice-versa.



ssp.yaml Results

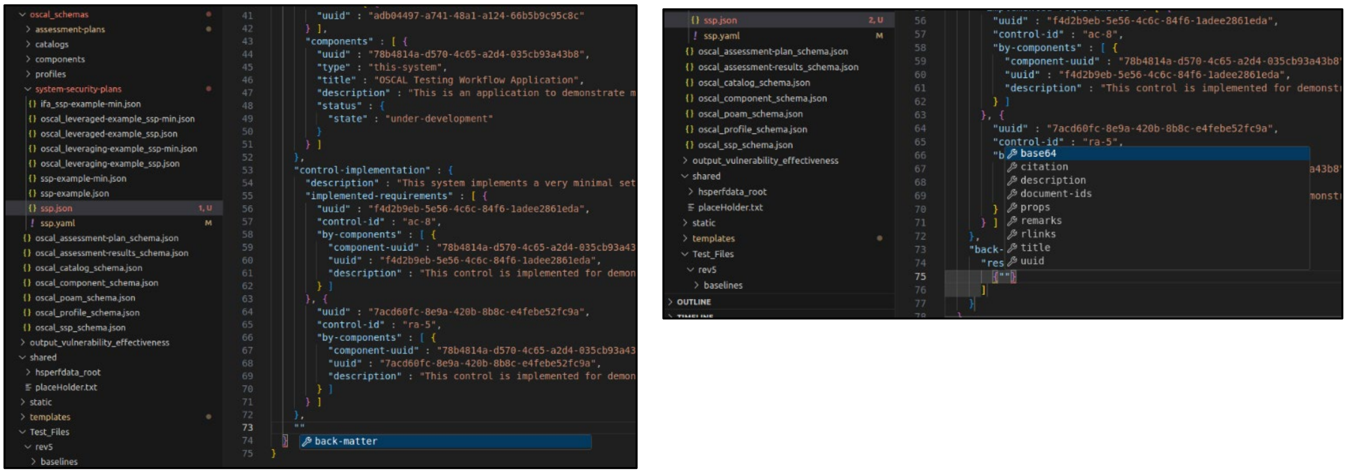
Figure 11. View after conversion.



```
{
  "system-security-plan" : {
    "uuid" : "50679d04-ba80-4fa7-9f7d-774284195924",
    "metadata" : {
      "title" : "OSCAL Automated SSP Template",
      "last-modified" : "2024-06-13T14:41:40.095194Z",
      "version" : "0.0.1-alpha",
      "oscal-version" : "1.0.4"
    },
    "import-profile" : {
      "href" : "/shared/profile.yaml"
    },
    "system-characteristics" : {
      "system-ids" : [ {
        "id" : "saas_system_iaas_customer"
      } ],
      "system-name" : "Leveraging SaaS System",
      "description" : "NO CONTENT HERE FOR NOW",
      "security-sensitivity-level" : "low",
      "system-information" : {
        "information-types" : [ {
          "uuid" : "fa2b1cb1-6d01-4b59-b1a0-6dd61f1b9780",
          "title" : "System Assessment",
          "description" : "This system handles development information per
        } ]
      },
      "security-impact-level" : {
        "security-objective-confidentiality" : "low",
        "security-objective-integrity" : "low",
        "security-objective-availability" : "low"
      },
      "status" : {
        "state" : "operational"
      },
      "authorization-boundary" : {
        "description" : "This system is for demonstration purposes only."
```

Figure 12. On the left: Since the submitted file is copied to the flask container, and the converted file is generated in the container, the files are written to the shared directory. Again, we need to move these files from the host side and take them out of this shared volume. On the right: The move-files.sh script, by default, will move the files to a directory where the IDE is configured to apply the OSCAL schemas for easy editing.

Editing the SSP

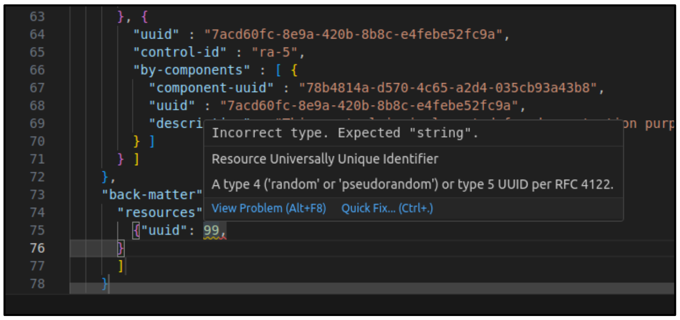


oscal_schemas
assessment-plans
catalogs
components
profiles
system-security-plans
ifa_ssp-example-min.json
oscal_leveraged-example_ssp-min.json
oscal_leveraged-example_ssp.json
oscal_leveraging-example_ssp-min.json
oscal_leveraging-example_ssp.json
ssp-example-min.json
ssp-example.json
ssp.json 1, U
ssp.yaml M
oscal_assessment-plan_schema.json
oscal_assessment-results_schema.json
oscal_catalog_schema.json
oscal_component_schema.json
oscal_poam_schema.json
oscal_profile_schema.json
oscal_ssp_schema.json
output_vulnerability_effectiveness
shared
hsperfdata_root
placeHolder.txt
static
templates
Test_Files
rev5
baselines
back-matter

```
          "uuid" : "adb04497-a741-48a1-a124-66b5b9c95c8c"
        } ],
        "components" : [ {
          "uuid" : "78b4814a-d570-4c65-a2d4-035cb93a43b8",
          "type" : "this-system",
          "title" : "OSCAL Testing Workflow Application",
          "description" : "This is an application to demonstrate m
          "status" : {
            "state" : "under-development"
          }
        } ]
      },
      "control-implementation" : {
        "description" : "This system implements a very minimal set
        "implemented-requirements" : [ {
          "uuid" : "f4d2b9eb-5e56-4c6c-84f6-1adee2861eda",
          "control-id" : "ac-8",
          "by-components" : [ {
            "component-uuid" : "78b4814a-d570-4c65-a2d4-035cb93a43
            "uuid" : "f4d2b9eb-5e56-4c6c-84f6-1adee2861eda",
            "description" : "This control is implemented for demon
          } ]
        }, {
          "uuid" : "7acd60fc-8e9a-420b-8b8c-e4febe52fc9a",
          "control-id" : "ra-5",
          "by-components" : [ {
            "component-uuid" : "78b4814a-d570-4c65-a2d4-035cb93a43
            "uuid" : "7acd60fc-8e9a-420b-8b8c-e4febe52fc9a",
            "description" : "This control is implemented for demon
          } ]
        } ]
      },
      ""
    }
}
```

```
          "uuid" : "f4d2b9eb-5e56-4c6c-84f6-1adee2861eda",
          "control-id" : "ac-8",
          "by-components" : [ {
            "component-uuid" : "78b4814a-d570-4c65-a2d4-035cb93a43b8"
            "uuid" : "f4d2b9eb-5e56-4c6c-84f6-1adee2861eda",
            "description" : "This control is implemented for demonstr
          } ]
        }, {
          "uuid" : "7acd60fc-8e9a-420b-8b8c-e4febe52fc9a",
          "control-id" : "ra-5",
          "b
                                                                  a43b8'
                                                                  monstr
          }
        } ]
      },
      "back-
        "res
          {""}
        ]
      }
```

Figure 13. On the left, auto-completion will show allowed keys for any new field. On the right, key suggestions.



```
        }, {
          "uuid" : "7acd60fc-8e9a-420b-8b8c-e4febe52fc9a",
          "control-id" : "ra-5",
          "by-components" : [ {
            "component-uuid" : "78b4814a-d570-4c65-a2d4-035cb93a43b8",
            "uuid" : "7acd60fc-8e9a-420b-8b8c-e4febe52fc9a",
            "descri                                          on purp
          } ]
        } ]
      },
      "back-matter"
        "resources"
          {"uuid": 99,
          }
        ]
      }
```

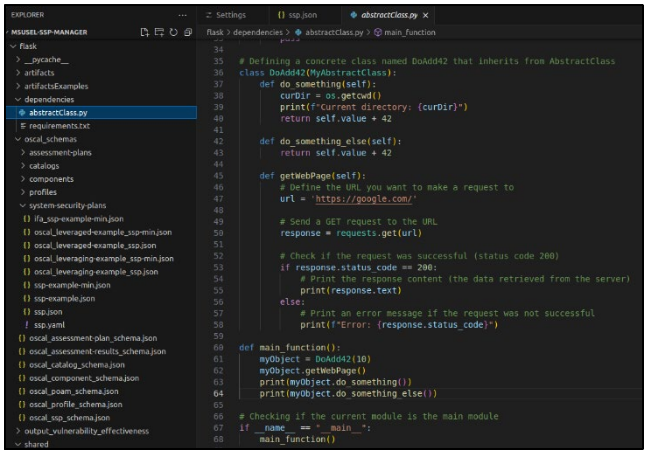
Incorrect type. Expected "string".
Resource Universally Unique Identifier
A type 4 ('random' or 'pseudorandom') or type 5 UUID per RFC 4122.

Figure 14. The IDE will point out incorrect data types for the values.

Once you finish editing, you can convert or validate your document as before. Don't forget to move your files afterwards so that they don't accidentally get overwritten.

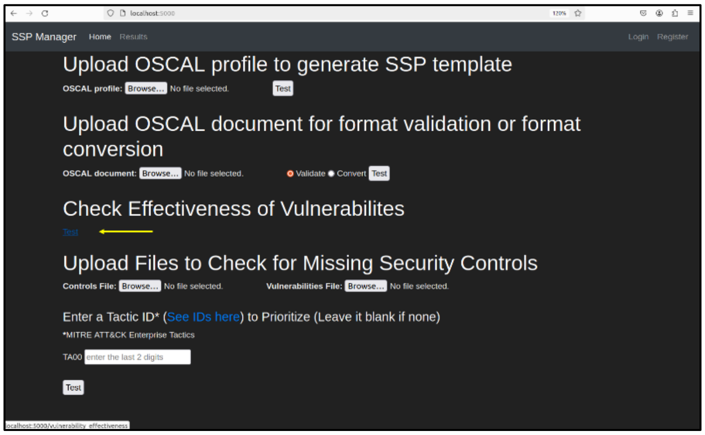
Vulnerability Effectiveness Analysis

The purpose of vulnerability effectiveness analysis is to get rid of some of the false positives in the vulnerabilities identified by static analysis tools. In particular, we want to get rid of the vulnerabilities in the program's dependencies. That is, packages that the program needs to have installed, as well as the modules that it is importing from Python's standard library. The tool downloads the source code for all the packages that were installed, which includes the programs direct dependencies, and its indirect dependencies; that is, the dependencies of its dependencies. It also copies the source code files for any standard library modules used by the program or any of the dependencies. The bandit static analyzer is then run on all these source code files to identify security vulnerabilities. Each vulnerability is then matched to the precise function where it resides. After this is done, a call graph for the program is created. If a function in a dependency file is in the call graph and has a vulnerability, then that vulnerability is effective. Vulnerable functions that are not in the call graph cannot be reached for exploitation from the program. Thus, for the particular program, any vulnerability in an unreachable function is a false positive, and won't be prioritized in the tool report.


EXPLORER
MSUSEL-SSP-MANAGER
flask
__pycache__
artifacts
artifactsExamples
dependencies
abstractClass.py
requirements.txt
oscal_schemas
assessment-plans
catalogs
components
profiles
system-security-plans
ifa_ssp-example-min.json
oscal_leveraged-example_ssp-min.json
oscal_leveraged-example_ssp.json
oscal_leveraging-example_ssp-min.json
oscal_leveraging-example_ssp.json
ssp-example-min.json
ssp-example.json
ssp.json
ssp.yaml
oscal_assessment-plan_schema.json
oscal_assessment-results_schema.json
oscal_catalog_schema.json
oscal_component_schema.json
oscal_poam_schema.json
oscal_profile_schema.json
oscal_ssp_schema.json
output_vulnerability_effectiveness
shared
Settings
ssp.json
abstractClass.py
flask > dependencies > abstractClass.py > main_function

```python
# Defining a concrete class named DoAdd42 that inherits from AbstractClass
class DoAdd42(MyAbstractClass):
    def do_something(self):
        curDir = os.getcwd()
        print(f"Current directory: {curDir}")
        return self.value + 42

    def do_something_else(self):
        return self.value + 42

    def getWebPage(self):
        # Define the URL you want to make a request to
        url = 'https://google.com/'

        # Send a GET request to the URL
        response = requests.get(url)

        # Check if the request was successful (status code 200)
        if response.status_code == 200:
            # Print the response content (the data retrieved from the server)
            print(response.text)
        else:
            # Print an error message if the request was not successful
            print(f"Error: {response.status_code}")

def main_function():
    myObject = DoAdd42(10)
    myObject.getWebPage()
    print(myObject.do_something())
    print(myObject.do_something_else())

# Checking if the current module is the main module
if __name__ == "__main__":
    main_function()
```

Figure 15. The project to be analyzed needs to be in the dependencies directory, along with its requirements.txt file.



Upload OSCAL profile to generate SSP template

OSCAL profile: Browse... No file selected. Test

Upload OSCAL document for format validation or format conversion

OSCAL document: Browse... No file selected. ● Validate ○ Convert Test

Check Effectiveness of Vulnerabilites

Test

Upload Files to Check for Missing Security Controls

Controls File: Browse... No file selected. Vulnerabilities File: Browse... No file selected.

Enter a Tactic ID* (See IDs here) to Prioritize (Leave it blank if none)

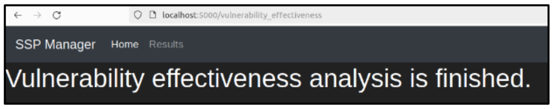
*MITRE ATT&CK Enterprise Tactics

TA00 enter the last 2 digits

Test

localhost:5000/vulnerability_effectiveness

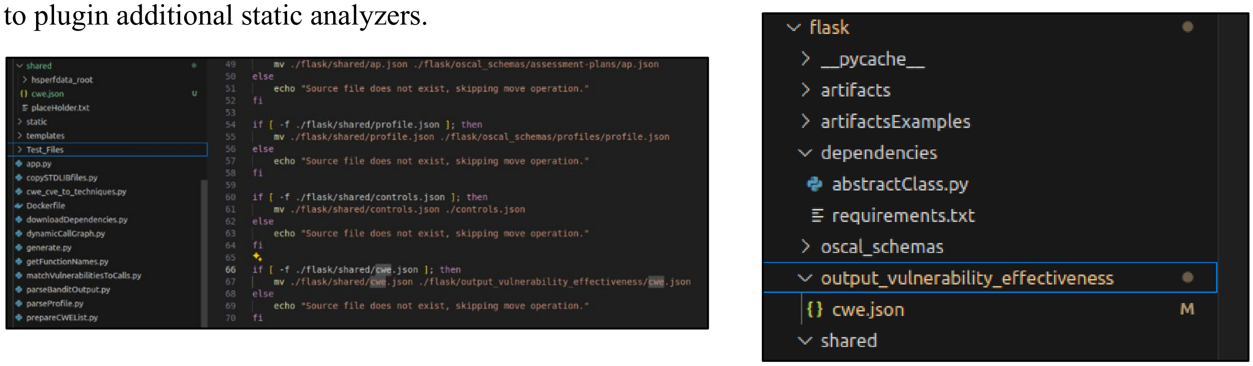
Figure 16. Just hit the test button and the analysis will run.



Vulnerability effectiveness analysis is finished.

Figure 17. You will see the message when the run ends.

The output of the vulnerability effectiveness run is a cwe.json file, as well as other artifacts that are used to display the vulnerable files and functions. These are displayed in table after doing the mapping from CVEs/CWEs to MITRE ATT&CK Tactics and Techniques and NIST SP 800-53 security controls (we describe this functionality below). The bandit static analysis tool checks for CWEs, but the mapping functionality can handle either CWEs or CVEs. In the future, we want to create an interface to make it easy to plugin additional static analyzers.



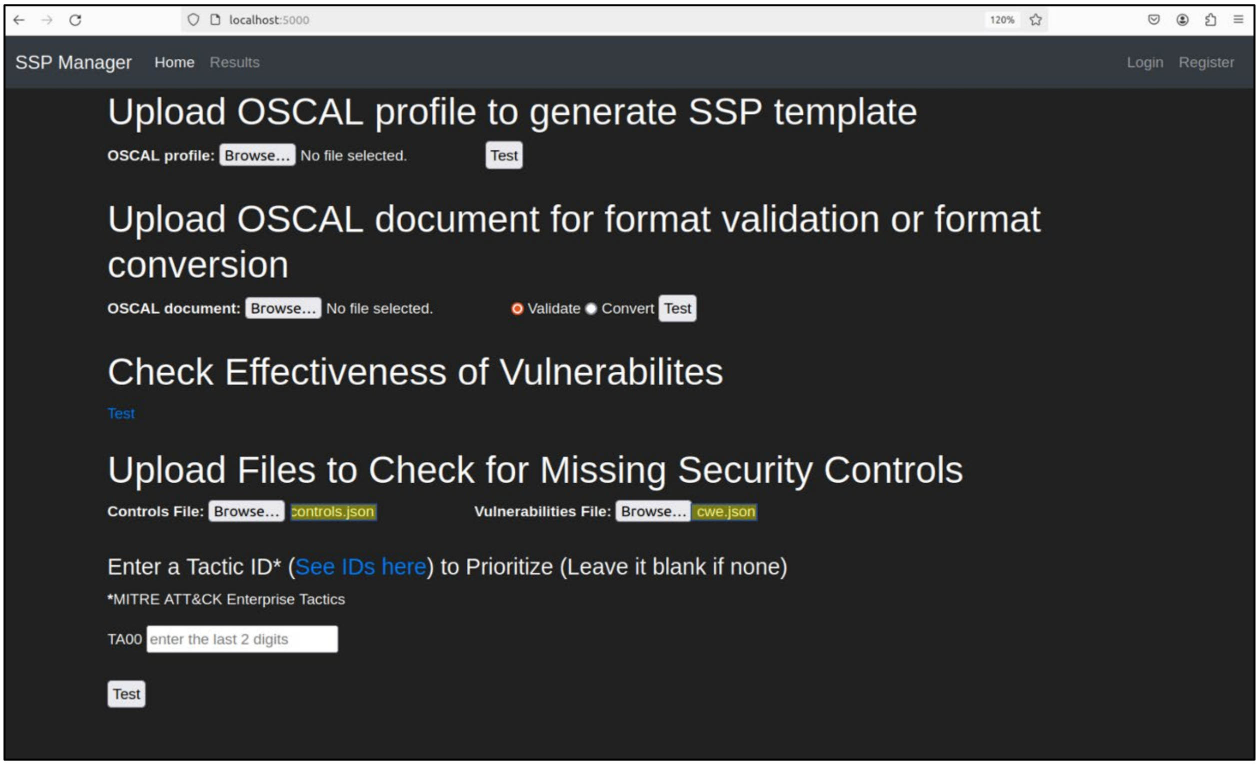
```
    mv ./flask/shared/ap.json ./flask/oscal_schemas/assessment-plans/ap.json
else
    echo "Source file does not exist, skipping move operation."
fi

if [ -f ./flask/shared/profile.json ]; then
    mv ./flask/shared/profile.json ./flask/oscal_schemas/profiles/profile.json
else
    echo "Source file does not exist, skipping move operation."
fi

if [ -f ./flask/shared/controls.json ]; then
    mv ./flask/shared/controls.json ./controls.json
else
    echo "Source file does not exist, skipping move operation."
fi

if [ -f ./flask/shared/cwe.json ]; then
    mv ./flask/shared/cwe.json ./flask/output_vulnerability_effectiveness/cwe.json
else
    echo "Source file does not exist, skipping move operation."
fi
```

Figure 18. On the left, the cwe.json file in the shared directory, and the code snippet from move-files.sh that moves it. On the right, Location of the cwe.json file after moving.

Mapping CVEs/CWEs to NIST SP 800-53 security controls

This functionality maps CVEs/CWEs to MITRE ATT&CK Tactics and Techniques and NIST SP 800-53 security controls. It uses these mappings to illustrate attack tactics and techniques available to an attacker. That is, the presence of the weaknesses/vulnerabilities that were found provide the attacker with some exploitation techniques that would enable him to complete some tactical steps in his attack. These attack techniques are mapped to NIST SP 800-53 rev. 5 security controls that can be used to mitigate their effectiveness. These security controls are the output of this functionality.

The process begins with either a cwe.json or a cve.json file as input. These files contain the weaknesses/vulnerabilities in the project's dependencies that were found to be effective. It also receives a controls.json file which contains the NISP SP 800-53 rev. 5 security controls that the system already has implemented. If the controls that mitigate a particular vulnerability are in place, then that vulnerability is not prioritized.



Upload OSCAL profile to generate SSP template

OSCAL profile: Browse... No file selected. Test

Upload OSCAL document for format validation or format conversion

OSCAL document: Browse... No file selected. ● Validate ○ Convert Test

Check Effectiveness of Vulnerabilites

Test

Upload Files to Check for Missing Security Controls

Controls File: Browse... controls.json Vulnerabilities File: Browse... cwe.json

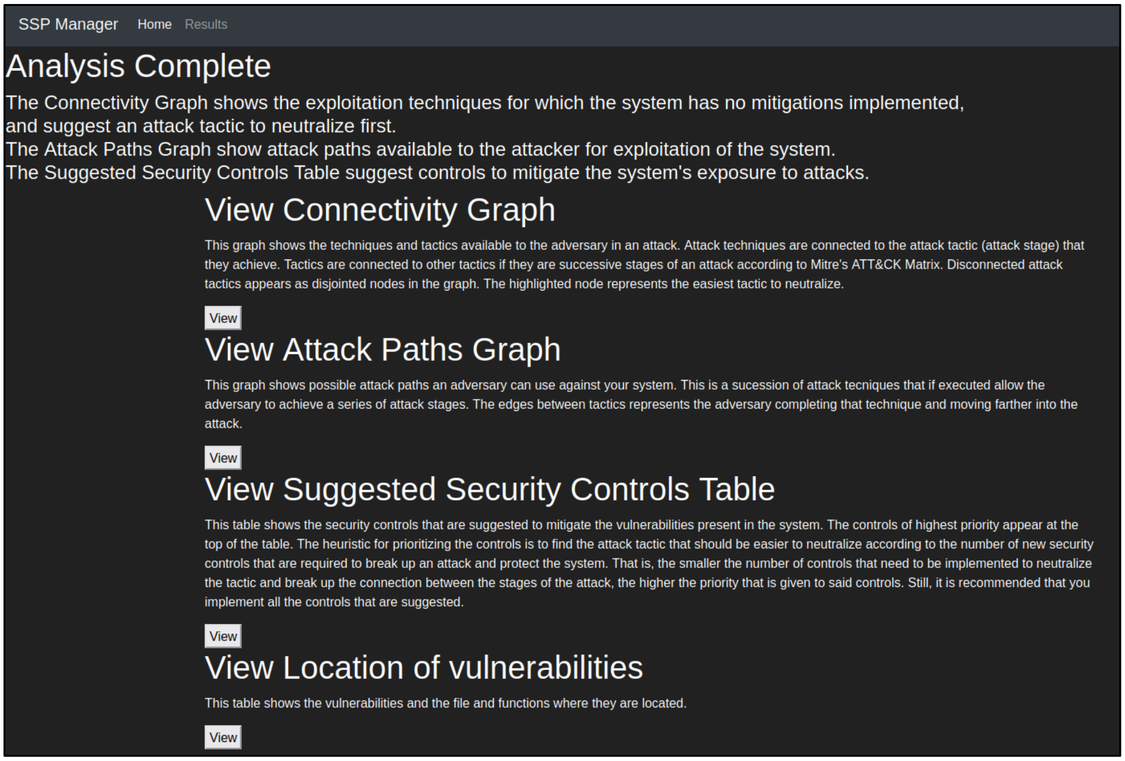
Enter a Tactic ID* (See IDs here) to Prioritize (Leave it blank if none)

*MITRE ATT&CK Enterprise Tactics

TA00 enter the last 2 digits

Test

Figure 19. The input files for the mapping functionality.



Analysis Complete

The Connectivity Graph shows the exploitation techniques for which the system has no mitigations implemented, and suggest an attack tactic to neutralize first.
The Attack Paths Graph show attack paths available to the attacker for exploitation of the system.
The Suggested Security Controls Table suggest controls to mitigate the system's exposure to attacks.

View Connectivity Graph

This graph shows the techniques and tactics available to the adversary in an attack. Attack techniques are connected to the attack tactic (attack stage) that they achieve. Tactics are connected to other tactics if they are successive stages of an attack according to Mitre's ATT&CK Matrix. Disconnected attack tactics appears as disjointed nodes in the graph. The highlighted node represents the easiest tactic to neutralize.

View

View Attack Paths Graph

This graph shows possible attack paths an adversary can use against your system. This is a sucession of attack tecniques that if executed allow the adversary to achieve a series of attack stages. The edges between tactics represents the adversary completing that technique and moving farther into the attack.

View

View Suggested Security Controls Table

This table shows the security controls that are suggested to mitigate the vulnerabilities present in the system. The controls of highest priority appear at the top of the table. The heuristic for prioritizing the controls is to find the attack tactic that should be easier to neutralize according to the number of new security controls that are required to break up an attack and protect the system. That is, the smaller the number of controls that need to be implemented to neutralize the tactic and break up the connection between the stages of the attack, the higher the priority that is given to said controls. Still, it is recommended that you implement all the controls that are suggested.

View

View Location of vulnerabilities

This table shows the vulnerabilities and the file and functions where they are located.

View

Figure 20. The results view for the mapping functionality.

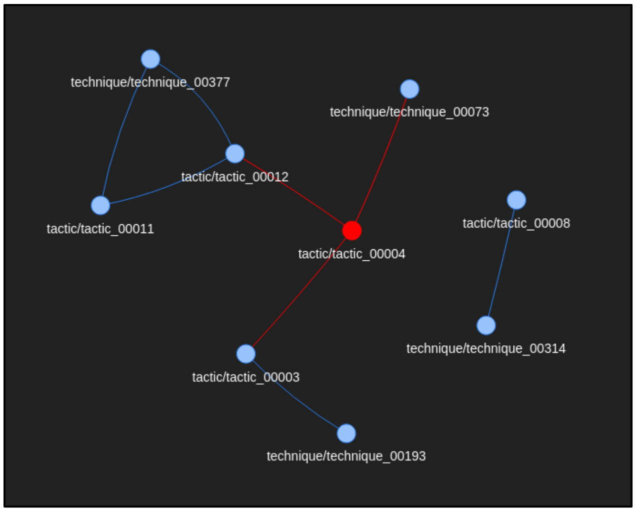

technique/technique_00377
technique/technique_00073
tactic/tactic_00012
tactic/tactic_00011
tactic/tactic_00008
tactic/tactic_00004
technique/technique_00314
tactic/tactic_00003
technique/technique_00193

Figure 21. Adjacent attack stages.

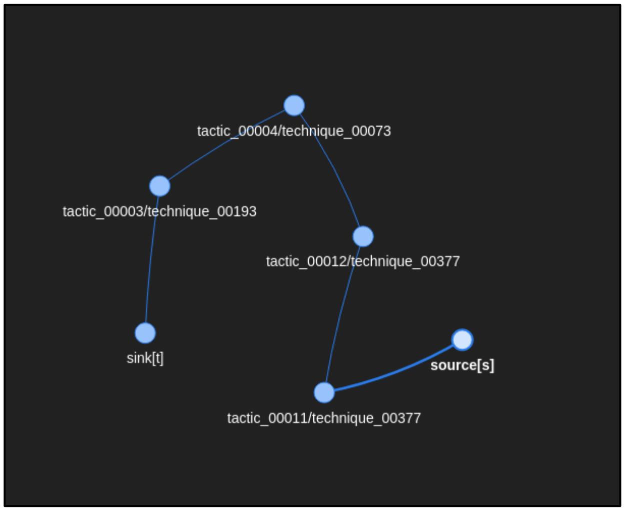

tactic_00004/technique_00073
tactic_00003/technique_00193
tactic_00012/technique_00377
sink[t]
source[s]
tactic_00011/technique_00377

Figure 22. Attack techniques that can be used to achieve different attack stages.

In MITRE ATT&CK, the stages of an attack are classified as tactics. A sequence of tactical achievements allows an attacker to advance or complete his attack strategy. Attack stages refer to actions that an attacker must complete to achieve his objective. For example, the attacker achieves initial access, lateral movement, privilege escalation, command and control, exfiltration, among others. If an attacker cannot complete a required stage of the attack, the attack will stall. MITRE ATT&CK lists techniques that can be used to achieve each tactical objective. MITRE provides mappings from these attack techniques to NIST SP 800-53 rev. 5 security controls. We added this data to the BRON database which maps CVEs/CWEs to MITRE ATT&CK Tactics and Techniques. When we map CVEs/CWEs to attack techniques we are also mapping them to attack tactics (stages).

Security controls are not all implemented at the same time, they are implemented sequentially. We want to prioritize the security controls that provide the most immediate security to the system. To do this, we suggest prioritizing the set of security controls that mitigate all the attack techniques found available for an attacker to complete one particular attack stage. This way the progression of a possible attack could be broken, even if the rest of the security controls are still in the process of being implemented.

In figure 21, the vulnerabilities in the dependencies of the project under analysis were mapped to five different tactics or attack stages: tactic_00003 (credential-access), tactic_00004 (defense evasion), tactic_00008 (impact), tactic_00011 (persistence), and tactic_00012 (privilege-escalation). If you go to localhost:8529, you will find the BRON database web interface (user: root, password:changeme). After entering the credentials, select BRON. Initially, you will find all the collections in the database (fig. 23 left). If you select the tactics collection, you will be able to see the individual documents (fig. 23 right). If you select for example document tactic_00004, you will see:

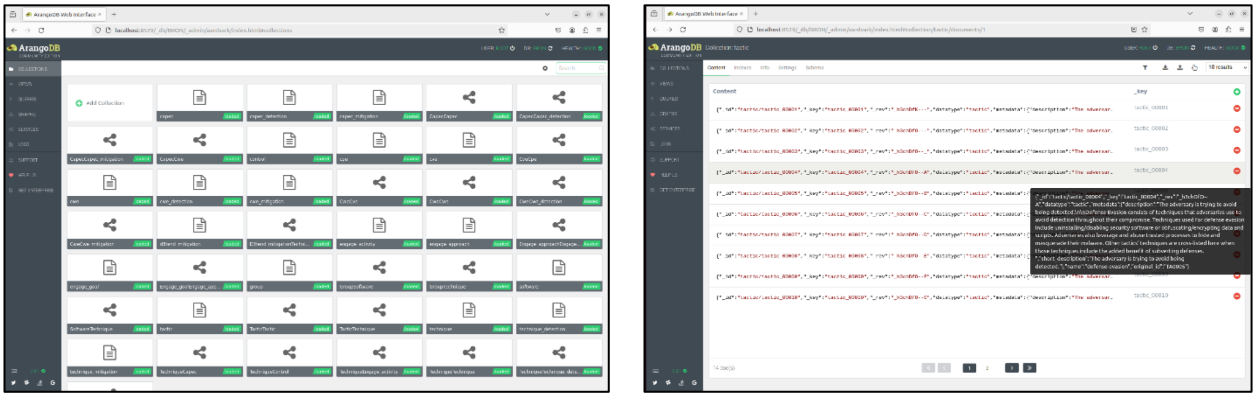
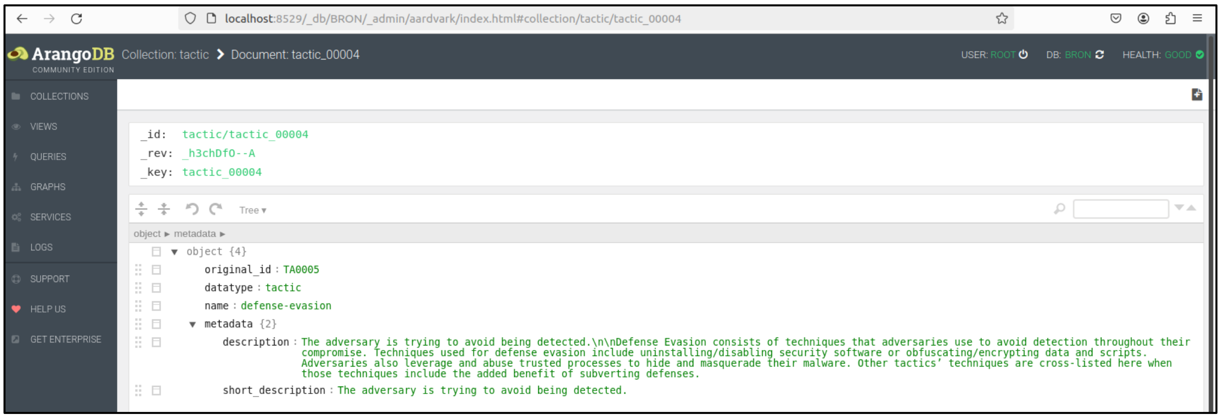


Figure 23. On the left, the collections in the BRON database. On the right, documents in the tactics collection on the BRON .



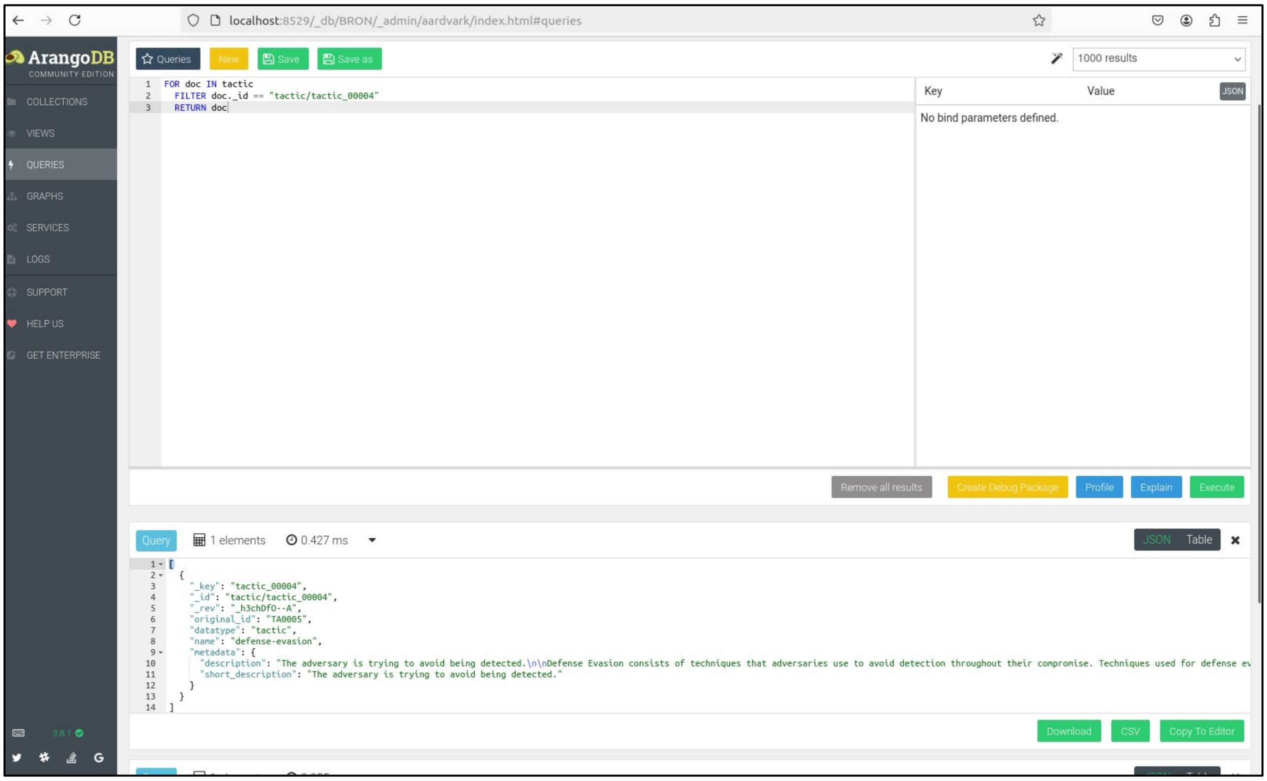
```
_id:  tactic/tactic_00004
_rev:  _h3chDfO--A
_key:  tactic_00004
```

```
▾ object {4}
    original_id : TA0005
    datatype : tactic
    name : defense-evasion
  ▾ metadata {2}
      description : The adversary is trying to avoid being detected.\n\nDefense Evasion consists of techniques that adversaries use to avoid detection throughout their
                    compromise. Techniques used for defense evasion include uninstalling/disabling security software or obfuscating/encrypting data and scripts.
                    Adversaries also leverage and abuse trusted processes to hide and masquerade their malware. Other tactics' techniques are cross-listed here when
                    those techniques include the added benefit of subverting defenses.
      short_description : The adversary is trying to avoid being detected.
```

Figure 24. A document in the tactics collection.

The document includes a description for the particular tactic. The document id ('_id') that the database uses to identify documents is not the same as the ids that MITRE uses for its tactics and techniques. The original_id field stores the id that MITRE ATT&CK uses. To find a particular document, you can run the following query:

FOR doc IN yourCollection
FILTER doc._id == ‘documentID’
RETURN doc



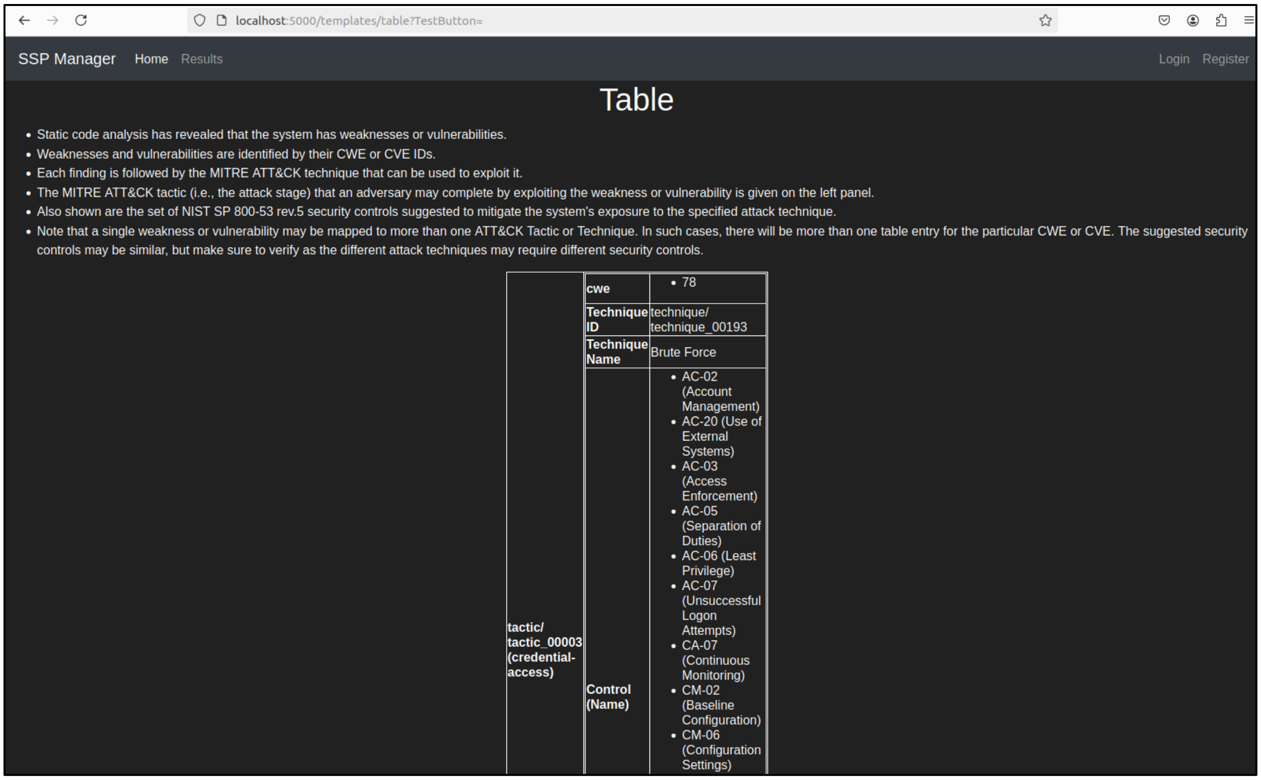
```
FOR doc IN tactic
  FILTER doc._id == "tactic/tactic_00004"
  RETURN doc
```

```
[
  {
    "_key": "tactic_00004",
    "_id": "tactic/tactic_00004",
    "_rev": "_h3chDfO--A",
    "original_id": "TA0005",
    "datatype": "tactic",
    "name": "defense-evasion",
    "metadata": {
      "description": "The adversary is trying to avoid being detected.\n\nDefense Evasion consists of techniques that adversaries use to avoid detection throughout their compromise. Techniques used for defense ev
      "short_description": "The adversary is trying to avoid being detected."
    }
  }
]
```

Figure 25. Query for a single document in a collection.

Going back to figure 21, we have a set of tactics or attack stages. tactic_00004, defense-evasion, is linked to tactic_00003, credential access, and tactic_00012 privilege-escalation. These tactics or stages are not connected to tactic_00008, impact, which can be thought of one of the final stages of the attack where integrity, confidentiality or availability is compromised. In this example, tactic_00004, defense-evasion has been selected to be neutralized first since just one attack technique was identified as available to exploit this weakness. The idea is to neutralize this technique so that any possible attack stalls, even if the mitigations for attack techniques used to complete other stages haven’t been implemented yet. This is the default behavior of the tool: break the attack at the stage where there are the least number of attack techniques available. The heuristic is that it would be faster to neutralize less techniques than more. The user can also select to prioritize any other attack tactic, according to his knowledge and possible concerns about his system.

Figure 22 illustrates attack techniques that could potentially be execute in a sequence allowing a progression of attack stages to be completed. Each node’s identifier specifies the attack technique and the tactic that it achieves. In the sequence shown in the figure, the attacker has techniques available for defense-evasion, privilege-escalation, credential access and persistence. As security controls are implemented to mitigate each of these attack techniques, the attacker’s tactical sequences are broken up, and the system becomes more secure.



Table

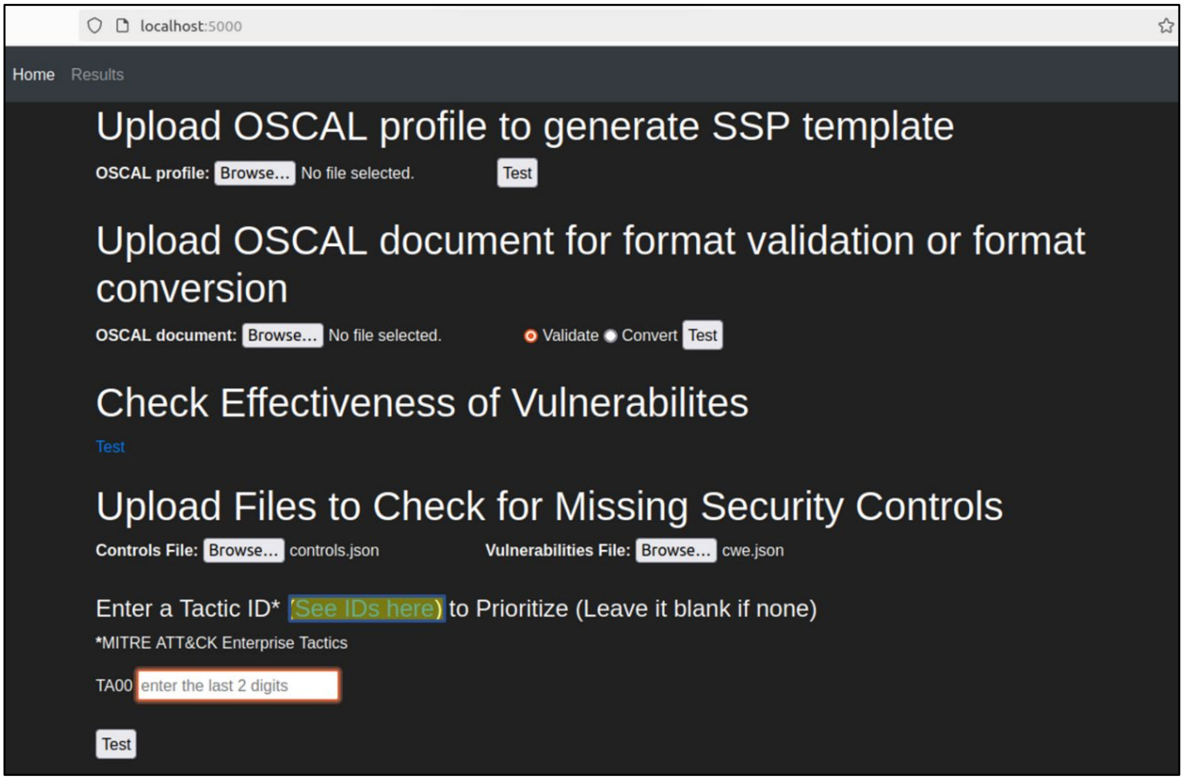
- Static code analysis has revealed that the system has weaknesses or vulnerabilities.
- Weaknesses and vulnerabilities are identified by their CWE or CVE IDs.
- Each finding is followed by the MITRE ATT&CK technique that can be used to exploit it.
- The MITRE ATT&CK tactic (i.e., the attack stage) that an adversary may complete by exploiting the weakness or vulnerability is given on the left panel.
- Also shown are the set of NIST SP 800-53 rev.5 security controls suggested to mitigate the system's exposure to the specified attack technique.
- Note that a single weakness or vulnerability may be mapped to more than one ATT&CK Tactic or Technique. In such cases, there will be more than one table entry for the particular CWE or CVE. The suggested security controls may be similar, but make sure to verify as the different attack techniques may require different security controls.

tactic/ tactic_00003 (credential-access)	**cwe**	• 78
	Technique ID	technique/ technique_00193
	Technique Name	Brute Force
	Control (Name)	• AC-02 (Account Management) • AC-20 (Use of External Systems) • AC-03 (Access Enforcement) • AC-05 (Separation of Duties) • AC-06 (Least Privilege) • AC-07 (Unsuccessful Logon Attempts) • CA-07 (Continuous Monitoring) • CM-02 (Baseline Configuration) • CM-06 (Configuration Settings)

Figure 26. Security controls recommendations, based on the weaknesses/vulnerabilities found.

Figure 26 shows some of the recommended security controls that should be implemented to mitigate the weaknesses in the dependencies for the example program. CWE 78 (‘OS Injection Weakness’) could be exploited by attack technique 00193 Brute Force (MITRE T1110) allowing the attacker to get access to credentials (tactic 00003, MITRE TA0006). Therefore, a recommendation is made for the implementation of a set of NIST SP 800-53 rev. 5 security controls that can mitigate this weakness.

Show tactic as priority:



Upload OSCAL profile to generate SSP template

OSCAL profile: Browse... No file selected. Test

Upload OSCAL document for format validation or format conversion

OSCAL document: Browse... No file selected. ● Validate ○ Convert Test

Check Effectiveness of Vulnerabilites

Test

Upload Files to Check for Missing Security Controls

Controls File: Browse... controls.json Vulnerabilities File: Browse... cwe.json

Enter a Tactic ID* (See IDs here) to Prioritize (Leave it blank if none)

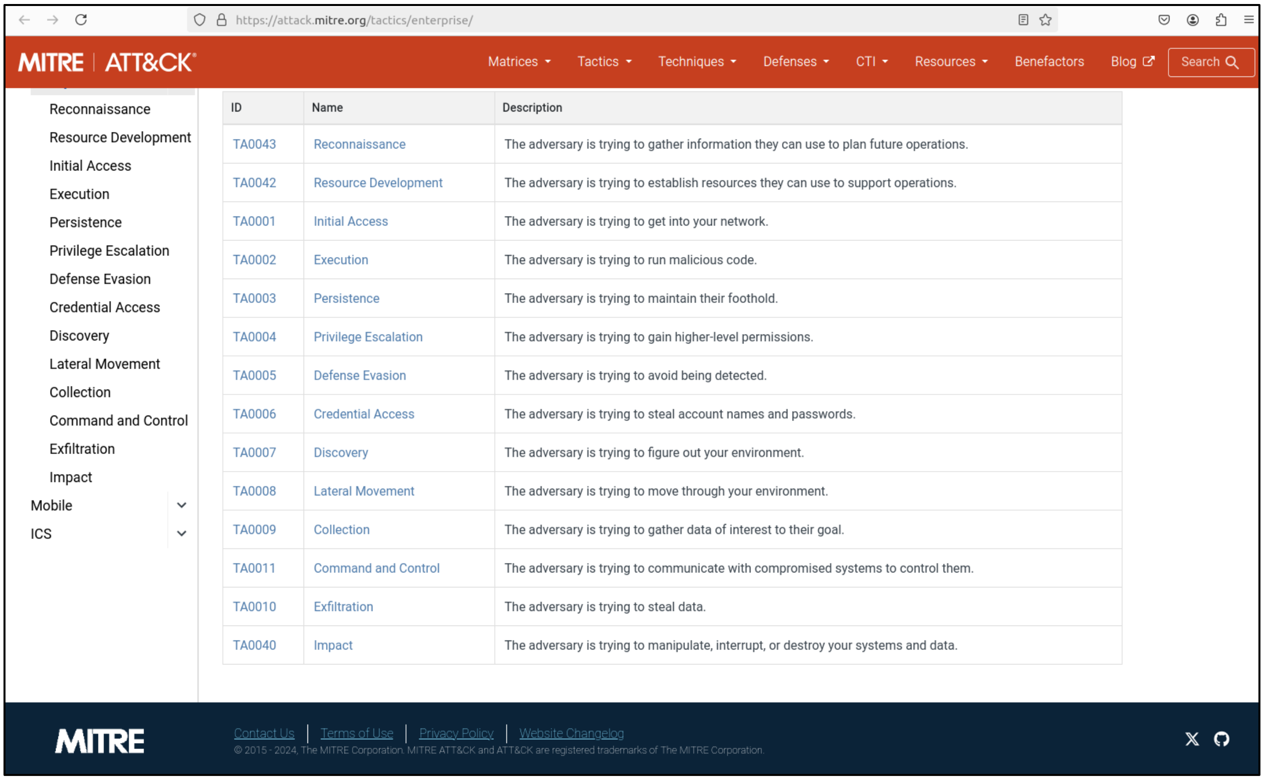
*MITRE ATT&CK Enterprise Tactics

TA00 enter the last 2 digits

Test

Figure 27. There's a link to MITRE ATT&CK Tactics.

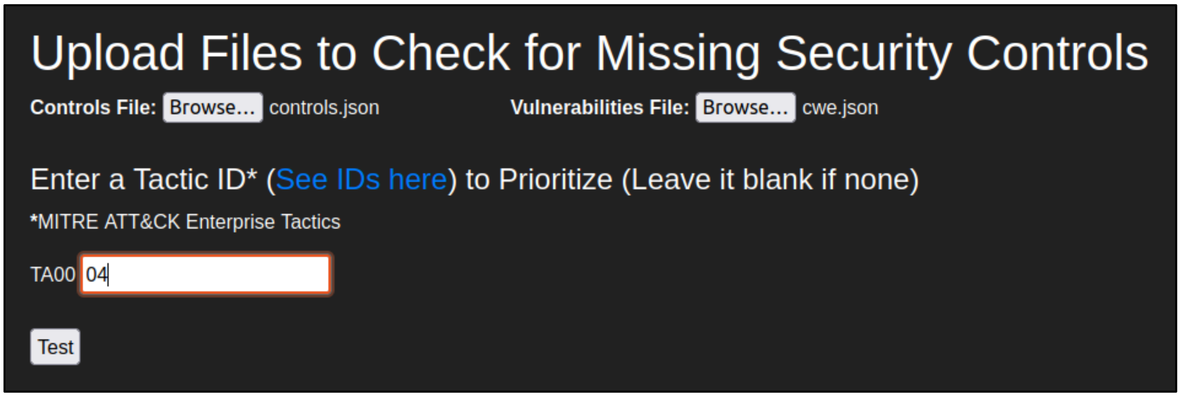
If you want to see the vulnerabilities found to be effective and the mitigating security controls for a particular tactic, like say initial access or lateral movement, you can add to your input (controls.json and cwe.json/cve.json) the MITRE Tactic id for the desired tactic. There's a link in the UI homepage to MITRE's Tactics page so you can quickly look up the tactic id that you need (there are only 14 tactics in MITRE ATT&CK). Just enter the last two digits of the MITRE Tactic id.



ID	Name	Description
TA0043	Reconnaissance	The adversary is trying to gather information they can use to plan future operations.
TA0042	Resource Development	The adversary is trying to establish resources they can use to support operations.
TA0001	Initial Access	The adversary is trying to get into your network.
TA0002	Execution	The adversary is trying to run malicious code.
TA0003	Persistence	The adversary is trying to maintain their foothold.
TA0004	Privilege Escalation	The adversary is trying to gain higher-level permissions.
TA0005	Defense Evasion	The adversary is trying to avoid being detected.
TA0006	Credential Access	The adversary is trying to steal account names and passwords.
TA0007	Discovery	The adversary is trying to figure out your environment.
TA0008	Lateral Movement	The adversary is trying to move through your environment.
TA0009	Collection	The adversary is trying to gather data of interest to their goal.
TA0011	Command and Control	The adversary is trying to communicate with compromised systems to control them.
TA0010	Exfiltration	The adversary is trying to steal data.
TA0040	Impact	The adversary is trying to manipulate, interrupt, or destroy your systems and data.

© 2015 - 2024, The MITRE Corporation. MITRE ATT&CK and ATT&CK are registered trademarks of The MITRE Corporation.

Figure 28. Attack tactics in MITRE's ATT&CK website.



Upload Files to Check for Missing Security Controls

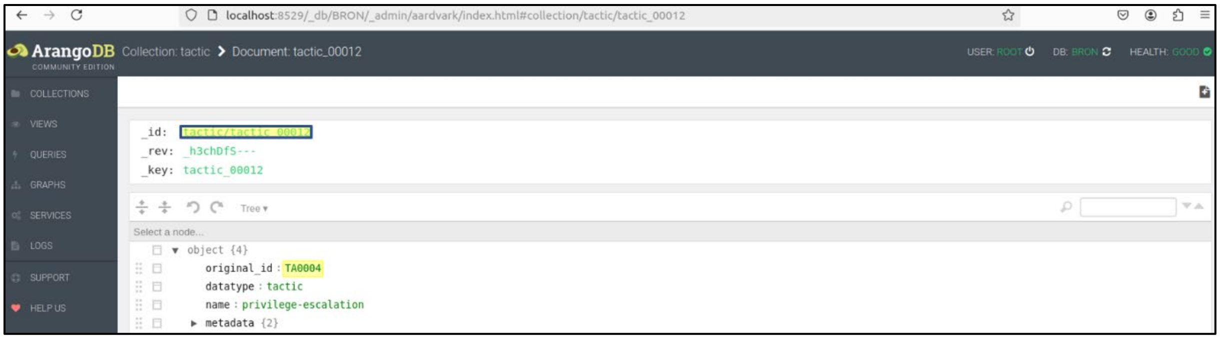
Controls File: Browse… controls.json **Vulnerabilities File:** Browse… cwe.json

Enter a Tactic ID* (See IDs here) to Prioritize (Leave it blank if none)

*MITRE ATT&CK Enterprise Tactics

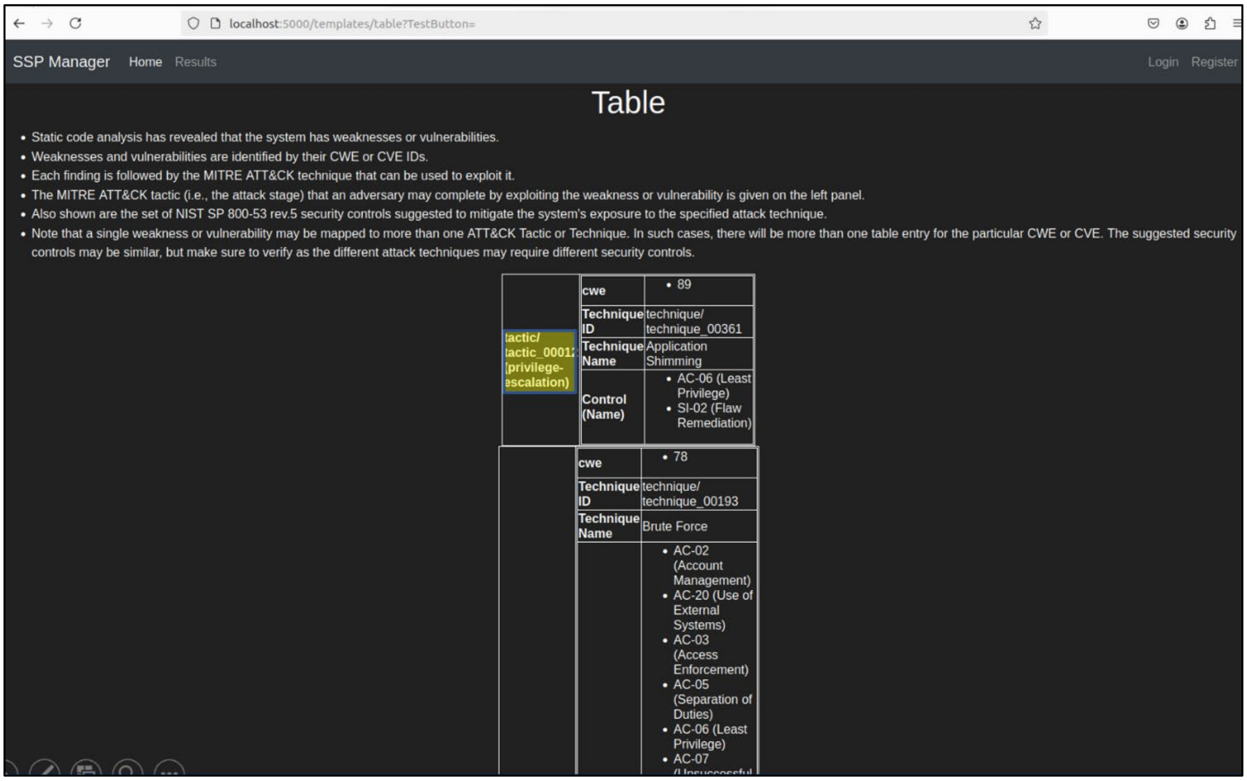
TA00 04

Figure 29. Selecting to view the security controls to prevent priviledge escalation.



```
_id:  tactic/tactic_00012
_rev:  _h3chDfS---
_key:  tactic_00012

▼ object {4}
    original_id : TA0004
    datatype : tactic
    name : privilege-escalation
  ▶ metadata {2}
```

Figure 30. The Ids for tactics or techniques in the local database is different from MITRE ATT&CK Ids.

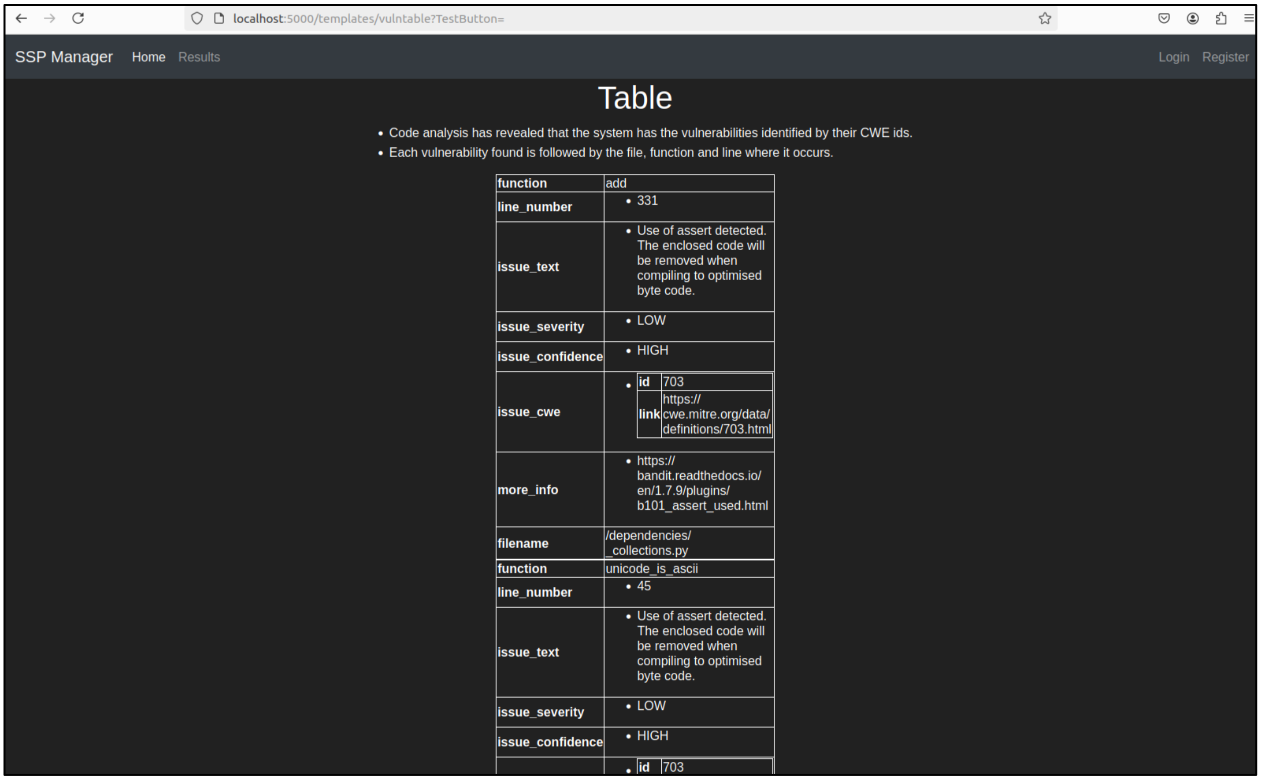


Table

- Static code analysis has revealed that the system has weaknesses or vulnerabilities.
- Weaknesses and vulnerabilities are identified by their CWE or CVE IDs.
- Each finding is followed by the MITRE ATT&CK technique that can be used to exploit it.
- The MITRE ATT&CK tactic (i.e., the attack stage) that an adversary may complete by exploiting the weakness or vulnerability is given on the left panel.
- Also shown are the set of NIST SP 800-53 rev.5 security controls suggested to mitigate the system's exposure to the specified attack technique.
- Note that a single weakness or vulnerability may be mapped to more than one ATT&CK Tactic or Technique. In such cases, there will be more than one table entry for the particular CWE or CVE. The suggested security controls may be similar, but make sure to verify as the different attack techniques may require different security controls.

tactic/ tactic_00012 (privilege-escalation)	**cwe**	• 89
	Technique ID	technique/ technique_00361
	Technique Name	Application Shimming
	Control (Name)	• AC-06 (Least Privilege) • SI-02 (Flaw Remediation)
	cwe	• 78
	Technique ID	technique/ technique_00193
	Technique Name	Brute Force
		• AC-02 (Account Management) • AC-20 (Use of External Systems) • AC-03 (Access Enforcement) • AC-05 (Separation of Duties) • AC-06 (Least Privilege) • AC-07 (Unsuccessful

Figure 31. Selecting to prioritize privilege escalation in the controls recommendations view.



Table

- Code analysis has revealed that the system has the vulnerabilities identified by their CWE ids.
- Each vulnerability found is followed by the file, function and line where it occurs.

function	add
line_number	• 331
issue_text	• Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.
issue_severity	• LOW
issue_confidence	• HIGH
issue_cwe	• **id** 703 **link** https://cwe.mitre.org/data/definitions/703.html
more_info	• https://bandit.readthedocs.io/en/1.7.9/plugins/b101_assert_used.html
filename	/dependencies/_collections.py
function	unicode_is_ascii
line_number	• 45
issue_text	• Use of assert detected. The enclosed code will be removed when compiling to optimised byte code.
issue_severity	• LOW
issue_confidence	• HIGH
	• **id** 703

Figure 32. Detail of an effective vulnerability on a project's dependencies.

The last view, seen in figure 32, lists all the functions in the project's dependencies that are called and are vulnerable. Standard description of the findings are listed: the function name, the filename (at the bottom), the line number, as well as description, initial severity assessment and links to obtain more information.