

Data-Driven Software Security Assessment Report

Stall Statements

File **connection_command.py** contains 2 stall statements:

1. Line number: 255, statement: `print(f'Could not import connection {conn_id}: connection already exists.')`
2. Line number: 259, statement: `print(f'Imported connection {conn_id}')`

File **dag_command.py** contains 2 stall statements:

1. Line number: 310, statement: `print(next_info.logical_date.isoformat())`
2. Line number: 91, statement: `print(f'Task {task.task_id}')`

File **info_command.py** contains 1 stall statement:

1. Line number: 323, statement: `console.print(f'\n[bold][green]{key}[/bold][green]', highlight=False)`

File **kubernetes_command.py** contains 4 stall statements:

1. Line number: 116, statement: `print(f'Inspecting pod {pod_name}')`
2. Line number: 122, statement: `print(f'Deleting pod "{pod_name}" phase "{pod_phase}" and reason "{pod_reason}", restart policy "{pod_restart_policy}")`
3. Line number: 126, statement: `print(f'Can't remove POD: {e}', file=sys.stderr)`
4. Line number: 128, statement: `print(f'No action taken on pod {pod_name}')`

File **standalone_command.py** contains 2 stall statements:

1. Line number: 99, statement: `time.sleep(0.1)`
2. Line number: 140, statement: `print(f'{colorised_name} | {line.strip()}')`

File **task_command.py** contains 3 stall statements:

1. Line number: 483, statement: `already_has_stream_handler = isinstance(handler, logging.StreamHandler)`
2. Line number: 534, statement: `print(textwrap.dedent(f' #
-----\n # property: {attr}\n #
-----\n {getattr(task, attr)}\n '))`
3. Line number: 371, statement: `print(f'{dep.dep_name}: {dep.reason}')`

File **variable_command.py** contains 1 stall statement:

1. Line number: 95, statement: `print(f'Variable import failed: {repr(e)}')`

File **manager.py** contains 1 stall statement:

1. Line number: 836, statement: `time.sleep(0.1)`

File **processor.py** contains 1 stall statement:

1. Line number: 652, statement: `for dag in unpaused_dags:`

File **example_bash_operator.py** contains 1 stall statement:

1. Line number: 52, statement: `task = BashOperator(task_id='runme_' + str(i), bash_command='echo "{{ task_instance_key_str }}" && sleep 1')`

File **example_kubernetes_executor.py** contains 1 stall statement:

1. Line number: 131, statement: `return_code = os.system('cat /shared/test.txt')`

File **example_python_operator.py** contains 2 stall statements:

1. Line number: 60, statement: `time.sleep(random_base)`
2. Line number: 90, statement: `print(Style.DIM + 'Please wait...', flush=True)`

File **celery_executor.py** contains 1 stall statement:

1. Line number: 487, statement: `time.sleep(5)`

File **kubernetes_executor.py** contains 1 stall statement:

1. Line number: 105, statement: `time.sleep(1)`

File **backfill_job.py** contains 1 stall statement:

1. Line number: 806, statement: `time.sleep(self.delay_on_limit_secs)`

File **scheduler_job.py** contains 1 stall statement:

1. Line number: 772, statement: `time.sleep(min(self._scheduler_idle_sleep_time, next_event if next_event else 0))`

File **triggerer_job.py** contains 1 stall statement:

1. Line number: 132, statement: `time.sleep(1)`

File **pod_launcher_deprecated.py** contains 3 stall statements:

1. Line number: 174, statement: `time.sleep(2)`
2. Line number: 135, statement: `time.sleep(1)`
3. Line number: 168, statement: `time.sleep(2)`

File **dag.py** contains 2 stall statements:

1. Line number: 2395, statement: `if dag.is_paused_upon_creation is not None:`
2. Line number: 2396, statement: `orm_dag.is_paused = dag.is_paused_upon_creation`

File **trigger_dagrun.py** contains 1 stall statement:

1. Line number: 178, statement: `time.sleep(self.poke_interval)`

File **airbyte.py** contains 1 stall statement:

1. Line number: 67, statement: `time.sleep(wait_seconds)`

File **batch_client.py** contains 6 stall statements:

1. Line number: 332, statement: `pause = self.exponential_delay(retries)`
2. Line number: 333, statement: `self.log.info('AWS Batch job (%s) status check (%d`

- of %d) in the next %.2f seconds', job_id, retries, self.max_retries, pause)
- 3. Line number: 334, statement: self.delay(pause)
- 4. Line number: 376, statement: pause = self.exponential_delay(retries)
- 5. Line number: 377, statement: self.log.info('AWS Batch job (%s) description retry (%d of %d) in the next %.2f seconds', job_id, retries, self.status_retries, pause)
- 6. Line number: 378, statement: self.delay(pause)

File **datasync.py** contains 1 stall statement:

- 1. Line number: 310, statement: time.sleep(self.wait_interval_seconds)

File **ec2.py** contains 1 stall statement:

- 1. Line number: 192, statement: time.sleep(check_interval)

File **glue.py** contains 1 stall statement:

- 1. Line number: 163, statement: time.sleep(self.JOB_POLL_INTERVAL)

File **sagemaker.py** contains 2 stall statements:

- 1. Line number: 681, statement: time.sleep(check_interval)
- 2. Line number: 769, statement: time.sleep(check_interval)

File **rds.py** contains 1 stall statement:

- 1. Line number: 89, statement: time.sleep(self._await_interval)

File **druid.py** contains 1 stall statement:

- 1. Line number: 107, statement: time.sleep(self.timeout)

File **kylin_cube.py** contains 1 stall statement:

- 1. Line number: 167, statement: time.sleep(self.interval)

File **spark_submit.py** contains 1 stall statement:

- 1. Line number: 553, statement: time.sleep(self._status_poll_interval)

File **pod_manager.py** contains 4 stall statements:

- 1. Line number: 168, statement: time.sleep(1)
- 2. Line number: 219, statement: time.sleep(1)
- 3. Line number: 227, statement: time.sleep(1)
- 4. Line number: 241, statement: time.sleep(2)

File **databricks.py** contains 1 stall statement:

- 1. Line number: 97, statement: time.sleep(operator.polling_period_seconds)

File **dbt.py** contains 1 stall statement:

- 1. Line number: 427, statement: time.sleep(check_interval)

File **example_elasticsearch_query.py** contains 1 stall statement:

- 1. Line number: 36, statement: print(f'table: {table}')

File **ads.py** contains 1 stall statement:

1. Line number: 184, statement: `time.sleep(sleep_time)`

File **__init__.py** contains 3 stall statements:

1. Line number: 29, statement: `logger = logging.getLogger(logger_name)`
2. Line number: 30, statement: `logger.handlers += [handler for handler in logging.getLogger().handlers if handler.name in ['task', 'console']]`
3. Line number: 31, statement: `logger.level = logging.DEBUG`

File **example_bigquery_queries.py** contains 1 stall statement:

1. Line number: 69, statement: `get_data_result = BashOperator(task_id='get_data_result', bash_command=f'echo {get_data.output}')`

File **bigquery.py** contains 1 stall statement:

1. Line number: 1462, statement: `time.sleep(5)`

File **cloud_sql.py** contains 1 stall statement:

1. Line number: 364, statement: `time.sleep(TIME_TO_SLEEP_IN_SECONDS)`

File **compute.py** contains 1 stall statement:

1. Line number: 310, statement: `time.sleep(TIME_TO_SLEEP_IN_SECONDS)`

File **compute_ssh.py** contains 1 stall statement:

1. Line number: 248, statement: `time.sleep(time_to_wait)`

File **dataflow.py** contains 2 stall statements:

1. Line number: 429, statement: `time.sleep(self._poll_sleep)`
2. Line number: 460, statement: `time.sleep(self._poll_sleep)`

File **dataproc.py** contains 1 stall statement:

1. Line number: 773, statement: `time.sleep(wait_time)`

File **datastore.py** contains 1 stall statement:

1. Line number: 262, statement: `time.sleep(polling_interval_in_seconds)`

File **dlp.py** contains 1 stall statement:

1. Line number: 252, statement: `time.sleep(time_to_sleep_in_seconds)`

File **functions.py** contains 1 stall statement:

1. Line number: 232, statement: `time.sleep(TIME_TO_SLEEP_IN_SECONDS)`

File **gcs.py** contains 4 stall statements:

1. Line number: 336, statement: `time.sleep(timeout_seconds)`
2. Line number: 345, statement: `1.0 * 2 ** (num_file_attempts - 1)`
3. Line number: 483, statement: `time.sleep(timeout_seconds)`

4. Line number: 491, statement: `1.0 * 2 ** (num_file_attempts - 1)`

File **kubernetes_engine.py** contains 1 stall statement:

1. Line number: 106, statement: `time.sleep(OPERATIONAL_POLL_INTERVAL)`

File **life_sciences.py** contains 1 stall statement:

1. Line number: 142, statement: `time.sleep(TIME_TO_SLEEP_IN_SECONDS)`

File **looker.py** contains 1 stall statement:

1. Line number: 157, statement: `time.sleep(wait_time)`

File **mlengine.py** contains 5 stall statements:

1. Line number: 59, statement: `time.sleep(2 ** i + random.randint(0, 1000) / 1000)`
2. Line number: 65, statement: `time.sleep(2 ** i + random.randint(0, 1000) / 1000)`
3. Line number: 201, statement: `time.sleep(30)`
4. Line number: 228, statement: `time.sleep(interval)`
5. Line number: 327, statement: `time.sleep(5)`

File **pubsub.py** contains 1 stall statement:

1. Line number: 145, statement: `warnings.warn("The base 64 encoded string as 'data' field has been deprecated. You should pass bytestring (utf-8 encoded).", DeprecationWarning, stacklevel=4)`

File **dataproc.py** contains 2 stall statements:

1. Line number: 567, statement: `time.sleep(time_to_sleep)`
2. Line number: 582, statement: `time.sleep(time_to_sleep)`

File **firestore.py** contains 1 stall statement:

1. Line number: 134, statement: `time.sleep(TIME_TO_SLEEP_IN_SECONDS)`

File **example_influxdb.py** contains 1 stall statement:

1. Line number: 42, statement: `print(record.values)`

File **jenkins_job_trigger.py** contains 2 stall statements:

1. Line number: 164, statement: `time.sleep(self.sleep_time)`
2. Line number: 200, statement: `time.sleep(self.sleep_time)`

File **batch.py** contains 1 stall statement:

1. Line number: 259, statement: `time.sleep(10)`

File **data_factory.py** contains 1 stall statement:

1. Line number: 683, statement: `time.sleep(check_interval)`

File **fileshare.py** contains 2 stall statements:

1. Line number: 97, statement: `warnings.warn('You are using deprecated connection for AzureFileShareHook. Please change it to `Azure FileShare`.', DeprecationWarning)`

2. Line number: 100, statement: warnings.warn('You are using deprecated connection for AzureFileShareHook. Please change it to `Azure FileShare`.', DeprecationWarning)

File **job.py** contains 1 stall statement:

1. Line number: 84, statement: time.sleep(3)

File **qubole.py** contains 1 stall statement:

1. Line number: 173, statement: time.sleep(Qubole.poll_interval)

File **tableau.py** contains 1 stall statement:

1. Line number: 176, statement: time.sleep(check_interval)

File **kerberos.py** contains 1 stall statement:

1. Line number: 186, statement: time.sleep(conf.getint('kerberos', 'reinit_frequency'))

File **base.py** contains 1 stall statement:

1. Line number: 271, statement: time.sleep(self._get_next_poke_interval(started_at, run_duration, try_number))

File **smart_sensor.py** contains 1 stall statement:

1. Line number: 149, statement: print(e)

File **cli_action_loggers.py** contains 2 stall statements:

1. Line number: 71, statement: logging.exception('Failed on pre-execution callback using %s', callback)
2. Line number: 89, statement: logging.exception('Failed on post-execution callback using %s', callback)

File **db.py** contains 1 stall statement:

1. Line number: 668, statement: time.sleep(1)

File **db_cleanup.py** contains 1 stall statement:

1. Line number: 144, statement: print(entry.__dict__)

File **helpers.py** contains 1 stall statement:

1. Line number: 108, statement: print('Please respond with y/yes or n/no.')

File **secrets_masker.py** contains 1 stall statement:

1. Line number: 104, statement: for flt in logging.getLogger('airflow.task').filters:

File **init_appbuilder.py** contains 1 stall statement:

1. Line number: 231, statement: self.register_blueprint(baseview)

File **init_views.py** contains 1 stall statement:

1. Line number: 141, statement: app.register_blueprint(blue_print['blueprint'])

File **views.py** contains 2 stall statements:

1. Line number: 3531, statement: `logging.warning('User %s tried to modify %s without having access.', g.user.username, dag_id)`
2. Line number: 4226, statement: `logging.info('Variable import failed: %s', repr(e))`

File **assign_cherry_picked_prs_with_milestone.py** contains 14 stall statements:

1. Line number: 294, statement: `console.print('-' * 80)`
2. Line number: 295, statement: `console.print(f'\n >>> Retrieving PR#{pr_number}: https://github.com/apache/airflow/pull/{pr_number}')`
3. Line number: 303, statement: `console.print(f'[red]The PR #{pr_number} could not be found[/]')`
4. Line number: 305, statement: `console.print(f'\nPR:{pr_number}: {pr.title}\n')`
5. Line number: 309, statement: `console.print(f'[green]The PR #{pr_number} is already assigned to the milestone: {pr.milestone.title}[/]. Labels: {label_names}')`
6. Line number: 311, statement: `console.print(f'[yellow]It will be classified as doc-only change[/]\n')`
7. Line number: 315, statement: `console.print(f'[yellow]It will be excluded from changelog[/]\n')`
8. Line number: 319, statement: `console.print(f'[green]The change will be included in changelog[/]\n')`
9. Line number: 325, statement: `console.print(f'[yellow]The PR #{pr_number} is already assigned to another milestone: {pr.milestone.title}[/]. Labels: {label_names}')`
10. Line number: 326, statement: `console.print(f'Marking the PR #{pr_number} as {milestone.title}')`
11. Line number: 329, statement: `console.print(f'Adding the PR #{pr_number} to {milestone.title}')`
12. Line number: 335, statement: `console.print(f'Applying the label {doc_only_label} the PR #{pr_number}')`
13. Line number: 342, statement: `console.print(f'Applying the label {changelog_skip_label} the PR #{pr_number}')`
14. Line number: 349, statement: `console.print(f'Skipping the PR #{pr_number}')`

File **find_newer_dependencies.py** contains 1 stall statement:

1. Line number: 115, statement: `progress.console.print(f'Package: {package}. Constraints version: {constraints_package_version}, Uploaded version: {package_version}, Upload date: {tz.convert(upload_date)} ({timezone})')`

File **check_files.py** contains 2 stall statements:

1. Line number: 91, statement: `print(p)`
2. Line number: 159, statement: `print(f' - [red]{file}[/red]')`

File **import_all_classes.py** contains 4 stall statements:

1. Line number: 109, statement: `error_console.print(f'Exception when importing: {package}\n\n')`
2. Line number: 110, statement: `error_console.print(trace)`
3. Line number: 111, statement: `error_console.print(f'[red]-----[/]')`
4. Line number: 137, statement: `print(f'[yellow]{w.filename}:{w.lineno}:`

{one_line_message}[/])

File **prepare_release_issue.py** contains 2 stall statements:

1. Line number: 305, statement: `progress.console.print(f'Retrieving Linked issue PR#{linked_issue_number}: https://github.com/apache/airflow/issue/{linked_issue_number}')`
2. Line number: 309, statement: `progress.console.print(f'Failed to retrieve linked issue #{linked_issue_number}: Unknown Issue')`

File **prepare_provider_packages.py** contains 11 stall statements:

1. Line number: 1522, statement: `console.print(f'[yellow]{message}[Y/N/Q]?[/] ', end="")`
2. Line number: 1831, statement: `console.print(provider)`
3. Line number: 653, statement: `console.print(f'{wrong_entity_type}: {message}')`
4. Line number: 1210, statement: `console.print(f'[red]The class {class_full_name} is wrongly named. The class name should be CamelCaseWithACRONYMS ![/]')`
5. Line number: 1213, statement: `console.print(f'[red]The class {class_full_name} is wrongly named. It is one of the {entity_type.value} so it should end with {class_suffix}[/]')`
6. Line number: 2044, statement: `console.print(f'{entity.value}: {TOTALS[entity]}')`
7. Line number: 2062, statement: `console.print(f'{w.filename}:{w.lineno}:[yellow]{one_line_message}[/]')`
8. Line number: 2541, statement: `console.print(f'Extracting PRs for provider {package_id}')`
9. Line number: 2544, statement: `console.print(f'Skipping extracting PRs for provider {package_id} as it is missing in dist')`
10. Line number: 2557, statement: `progress.console.print(f'Retrieving PR#{pr_number}: https://github.com/apache/airflow/pull/{pr_number}')`
11. Line number: 2564, statement: `console.print(f'[red]The PR #{pr_number} could not be found[/]')`

File **remove_old_releases.py** contains 1 stall statement:

1. Line number: 78, statement: `print(command)`

File **retag_docker_images.py** contains 1 stall statement:

1. Line number: 59, statement: `print(f'Copying image: {source_image} -> {target_image}')`

File **calculate_statistics_provider_testing_issues.py** contains 1 stall statement:

1. Line number: 194, statement: `print(stat)`

File **validate_version_added_fields_in_config.py** contains 1 stall statement:

1. Line number: 82, statement: `print('Processing version:', curr_version)`

File **test_docker_compose_quick_start.py** contains 3 stall statements:

1. Line number: 76, statement: `print(f'{container_name}: container_state={container_state}, health_status={health_status}')`
2. Line number: 80, statement: `print(f'{container_name}: container_state={container_state}')`

3. Line number: 108, statement: `print(f'Waiting for DAG Run: dag_state={dag_state}')`

File **build_docs.py** contains 6 stall statements:

1. Line number: 221, statement: `console.print(f'[blue]{package_name:60}:[/] Cleaning files')`
2. Line number: 336, statement: `console.print(f'[blue]{package_name:60}:[/] Scheduling documentation to build')`
3. Line number: 376, statement: `console.print(f'[blue]{package_name:60}:[/] Scheduling spellchecking')`
4. Line number: 440, statement: `console.print(f' - {pkg}')`
5. Line number: 460, statement: `console.print(f'{pkg_no}. {pkg}')`
6. Line number: 360, statement: `console.print(f'{result.package_name:60} {line}')`

File **airflow_intersphinx.py** contains 1 stall statement:

1. Line number: 155, statement: `print(f':{role_name}:`{name}:{entry}')`

File **errors.py** contains 7 stall statements:

1. Line number: 69, statement: `console.print('-' * 30, f'[red]Error {warning_no:3}[/]', '-' * 20)`
2. Line number: 70, statement: `console.print(error.message)`
3. Line number: 71, statement: `console.print()`
4. Line number: 73, statement: `console.print(f'File path: {os.path.relpath(error.file_path, start=DOCS_DIR)} ({error.line_no})')`
5. Line number: 74, statement: `console.print()`
6. Line number: 75, statement: `console.print(prepare_code_snippet(error.file_path, error.line_no))`
7. Line number: 77, statement: `console.print(f'File path: {error.file_path}')`

File **fetch_inventories.py** contains 1 stall statement:

1. Line number: 137, statement: `print(f'{pkg_no}. {pkg_name}')`

File **spelling_checks.py** contains 1 stall statement:

1. Line number: 150, statement: `console.print('-' * 30, f'Error {warning_no:3}', '-' * 30)`

File **publish_docs.py** contains 1 stall statement:

1. Line number: 98, statement: `print(f' - {pkg}')`

File **test_base.py** contains 6 stall statements:

1. Line number: 107, statement: `time.sleep(5)`
2. Line number: 110, statement: `print(f'Calling [monitor_task]#1 {get_string}')`
3. Line number: 113, statement: `check_call(['echo', 'api returned 404.'])`
4. Line number: 118, statement: `print(f'Received [monitor_task]#2: {result_json}')`
5. Line number: 120, statement: `print(f'Attempt {tries}: Current state of operator is {state}')`
6. Line number: 127, statement: `check_call(['echo', f'api call failed. trying again. error {e}'])`

File **pre_commit_chart_schema.py** contains 2 stall statements:

1. Line number: 50, statement: `print(f'{no_d}: {schema_path}')`
2. Line number: 51, statement: `print(json.dumps(schema_type, indent=2))`

File **pre_commit_check_extras_have_providers.py** contains 1 stall statement:

1. Line number: 83, statement: `print(message, file=sys.stderr)`

File **pre_commit_check_order_dockerfile_extras.py** contains 1 stall statement:

1. Line number: 102, statement: `print(error)`

File **pre_commit_check_order_setup.py** contains 3 stall statements:

1. Line number: 102, statement: `print(f'[blue]Checking alias-dependent group {dependent}[/']')`
2. Line number: 132, statement: `print(f'[blue]Checking setup.cfg group {key}[/']')`
3. Line number: 155, statement: `print(error)`

File **pre_commit_check_pre_commit_hook_names.py** contains 1 stall statement:

1. Line number: 56, statement: `print(f'" * '{hook_name}': length {len(hook_name)}')`

File **pre_commit_check_provider_yaml_files.py** contains 1 stall statement:

1. Line number: 402, statement: `console.print(f'[red]Error:[/] {error}')`

File **pre_commit_checkout_no_credentials.py** contains 3 stall statements:

1. Line number: 46, statement: `console.print(f'\n[red]The `with` clause is missing in step:[/]\n\n{pretty_step}')`
2. Line number: 51, statement: `console.print(f'\n[red]The `with` clause does not have persist-credentials in step:[/]\n\n{pretty_step}')`
3. Line number: 55, statement: `console.print(f'\n[red]The `with` clause have persist-credentials=True in step:[/]\n\n{pretty_step}')`

File **pre_commit_json_schema.py** contains 1 stall statement:

1. Line number: 130, statement: `print(error)`

File **summarize_junit_failures.py** contains 3 stall statements:

1. Line number: 112, statement: `print(f'{case_name}: {TEXT_YELLOW}{failure.get('message')}{TEXT_RESET}')`
2. Line number: 108, statement: `print(f'{case_name}: {TEXT_YELLOW}{err.get('message')}{TEXT_RESET}')`
3. Line number: 112, statement: `print(f'{case_name}: {TEXT_YELLOW}{failure.get('message')}{TEXT_RESET}')`

File **run_resource_check.py** contains 3 stall statements:

1. Line number: 83, statement: `console.print(f'[yellow]WARNING!!!: Not enough {resource} available for Docker.')`
2. Line number: 84, statement: `print(f'At least {capacity.minimumAllowed}{check} of {resource} required. You have {capacity.current}{check}\n')`
3. Line number: 87, statement: `console.print(f' * {resource} available`

{capacity.current}{check}. [green]OK.')

File **update_quarantined_test_status.py** contains 2 stall statements:

1. Line number: 186, statement: `print('Parsing: ' + test['classname'] + '::' + test['name'])`
2. Line number: 188, statement: `print(f'skipping {test['name']}')`

File **list-integrations.py** contains 2 stall statements:

1. Line number: 54, statement: `print(f'Module {full_module_name} can not be loaded.', file=sys.stderr)`
2. Line number: 119, statement: `print(clazz_to_print)`

Dependency Vulnerabilities

The following dependencies have vulnerabilities:

1. cryptography >= 0.9.3

- [CVE-2020-36242](#) **CVSS: 6.4**

- **Name:** Overflow Variables and Tags
- **Summary:** Blind SQL Injection results from an insufficient mitigation for SQL Injection. Although suppressing database error messages are considered best practice, the suppression alone is not sufficient to prevent SQL Injection. Blind SQL Injection is a form of SQL Injection that overcomes the lack of error messages. Without the error messages that facilitate SQL Injection, the adversary constructs input strings that probe the target through simple Boolean SQL expressions. The adversary can determine if the syntax and structure of the injection was successful based on whether the query was executed or not. Applied iteratively, the adversary determines how and where the target is vulnerable to SQL Injection.
- **Solution:** If possible, upgrade to the latest package release. Use a language or compiler that performs automatic bounds checking. Use an abstraction library to abstract away risky APIs. Not a complete solution. Compiler-based canary mechanisms such as StackGuard, ProPolice and the Microsoft Visual Studio /GS flag. Unless this provides automatic bounds checking, it is not a complete solution. Use OS-level preventative functionality. Not a complete solution. Do not trust input data from user. Validate all user input.

- [CVE-2016-9243](#) **CVSS: 5.0**

- **Name:** Blind SQL Injection
- **Summary:** This type of attack leverages the use of tags or variables from a formatted configuration data to cause buffer overflow. The attacker crafts a malicious HTML page or configuration file that includes oversized strings, thus causing an overflow.
- **Solution:** If possible, upgrade to the latest package release. Security by Obscurity is not a solution to preventing SQL Injection. Rather than suppress error messages and exceptions, the application must handle them gracefully, returning either a custom error page or


redirecting the user to a default page, without revealing any information about the database or the application internals. Strong input validation - All user-controllable input must be validated and filtered for illegal characters as well as SQL content. Keywords such as UNION, SELECT or INSERT must be filtered in addition to characters such as a single-quote(') or SQL-comments (--) based on the context in which they appear.

2. dill >= 0.2.2

- [CVE-2017-11424](#) **CVSS:** 5.0
 - **Name:** Unavailable
 - **Summary:** Unavailable
 - **Solution:** If possible, upgrade to the latest package release.

3. flask >= 1.1.0,

4. [CVE-2021-33026](#) **CVSS:** 7.5

- **Name:** Privilege Escalation
- **Summary:** This type of attack is a form of Cross-Site Scripting (XSS) where a malicious script is "reflected" off a vulnerable web application and then executed by a victim's browser. The process starts with an adversary delivering a malicious script to a victim and convincing the victim to send the script to the vulnerable web application. The most common method of this is through a phishing email where the adversary embeds the malicious script with a URL that the victim then clicks on. In processing the subsequent request, the vulnerable web application incorrectly considers the malicious script as valid input and uses it to create a response that is then sent back to the victim. To launch a successful Reflected XSS attack, an adversary looks for places where user-input is used directly in the generation of a response. This often involves elements that are not expected to host scripts such as image tags () or the addition of event attributes such as onload and onmouseover. These elements are often not subject to the same input validation, output encoding, and other content filtering and checking routines.
- **Solution:** If possible, upgrade to the latest package release.

5. [CVE-2021-41265](#) **CVSS:** 6.5

- **Name:** Man in the Middle Attack
- **Summary:** Unavailable
- **Solution:** If possible, upgrade to the latest package release.
Get your Public Key signed by a Certificate Authority Encrypt your communication using cryptography (SSL,...) Use Strong mutual authentication to always fully authenticate both ends of any communications channel. Exchange public keys using a secure channel

6. [CVE-2021-32805](#) **CVSS:** 5.8

- **Name:** Unavailable
- **Summary:** Unavailable
- **Solution:** If possible, upgrade to the latest package release.

[7. CVE-2021-32618](#) **CVSS:** 5.8

- **Name:** Unavailable
- **Summary:** Unavailable
- **Solution:** If possible, upgrade to the latest package release.

[8. CVE-2022-21659](#) **CVSS:** 5.0

- **Name:** Unavailable
- **Summary:** Unavailable
- **Solution:** If possible, upgrade to the latest package release.

[9. CVE-2021-29621](#) **CVSS:** 5.0

- **Name:** Unavailable
- **Summary:** An adversary exploits a weakness enabling them to elevate their privilege and perform an action that they are not supposed to be authorized to perform.
- **Solution:** If possible, upgrade to the latest package release.

[10. CVE-2020-25032](#) **CVSS:** 5.0

- **Name:** Unavailable
- **Summary:** This type of attack targets the communication between two components (typically client and server). The attacker places himself in the communication channel between the two components. Whenever one component attempts to communicate with the other (data flow, authentication challenges, etc.), the data first goes to the attacker, who has the opportunity to observe or alter it, and it is then passed on to the other component as if it was never observed. This interposition is transparent leaving the two compromised components unaware of the potential corruption or leakage of their communications. The potential for Man-in-the-Middle attacks yields an implicit lack of trust in communication or identify between two components. MITM attacks differ from sniffing attacks since they often modify the communications prior to delivering it to the intended recipient. These attacks also differ from interception attacks since they may forward the sender's original unmodified data, after copying it, instead of keeping it for themselves.
- **Solution:** If possible, upgrade to the latest package release.

[11. CVE-2018-16516](#) **CVSS:** 4.3

- **Name:** Reflected XSS
- **Summary:** Unavailable
- **Solution:** If possible, upgrade to the latest package release.
Use browser technologies that do not allow client-side scripting. Utilize strict

type, character, and encoding enforcement. Ensure that all user-supplied input is validated before use.

12. flask-caching >= 1.5.0,

13. [CVE-2021-33026](#) CVSS: 7.5


- **Name:** Privilege Escalation
- **Summary:** An adversary exploits a weakness enabling them to elevate their privilege and perform an action that they are not supposed to be authorized to perform.
- **Solution:** If possible, upgrade to the latest package release.

14. markdown >= 3.0

• [CVE-2021-23639](#) CVSS: 7.5

- **Name:** Unavailable
- **Summary:** An attacker manipulates inputs to the target software which the target software passes to file system calls in the OS. The goal is to gain access to, and perhaps modify, areas of the file system that the target software did not intend to be accessible.
- **Solution:** If possible, upgrade to the latest package release.

• [CVE-2022-21670](#) CVSS: 5.0

- **Name:** XML Entity Expansion
- **Summary:** This type of attack is a form of Cross-Site Scripting (XSS) where a malicious script is "reflected" off a vulnerable web application and then executed by a victim's browser. The process starts with an adversary delivering a malicious script to a victim and convincing the victim to send the script to the vulnerable web application. The most common method of this is through a phishing email where the adversary embeds the malicious script with a URL that the victim then clicks on. In processing the subsequent request, the vulnerable web application incorrectly considers the malicious script as valid input and uses it to create a response that is then sent back to the victim. To launch a successful Reflected XSS attack, an adversary looks for places where user-input is used directly in the generation of a response. This often involves elements that are not expected to host scripts such as image tags () , or the addition of event attributes such as onload and onmouseover. These elements are often not subject to the same input validation, output encoding, and other content filtering and checking routines.
- **Solution:** If possible, upgrade to the latest package release.
Design: Use libraries and templates that minimize unfiltered input. Use methods that limit entity expansion and throw exceptions on attempted entity expansion. Implementation: Disable altogether the use of inline DTD schemas in your XML parsing objects. If must use DTD, normalize, filter and white list and parse with methods and routines that will detect entity expansion from untrusted sources.

- [CVE-2021-21391](#) **CVSS: 4.3**
 - **Name:** XML Entity Expansion
 - **Summary:** An attacker submits an XML document to a target application where the XML document uses nested entity expansion to produce an excessively large output XML. XML allows the definition of macro-like structures that can be used to simplify the creation of complex structures. However, this capability can be abused to create excessive demands on a processor's CPU and memory. A small number of nested expansions can result in an exponential growth in demands on memory.
 - **Solution:** If possible, upgrade to the latest package release.
Design: Use libraries and templates that minimize unfiltered input. Use methods that limit entity expansion and throw exceptions on attempted entity expansion. Implementation: Disable altogether the use of inline DTD schemas in your XML parsing objects. If must use DTD, normalize, filter and white list and parse with methods and routines that will detect entity expansion from untrusted sources.

- [CVE-2020-7773](#) **CVSS: 4.3**
 - **Name:** Reflected XSS
 - **Summary:** Unavailable
 - **Solution:** If possible, upgrade to the latest package release.
Use browser technologies that do not allow client-side scripting. Utilize strict type, character, and encoding enforcement. Ensure that all user-supplied input is validated before use.

- [CVE-2018-3770](#) **CVSS: 2.1**
 - **Name:** Manipulating Web Input to File System Calls
 - **Summary:** An attacker submits an XML document to a target application where the XML document uses nested entity expansion to produce an excessively large output XML. XML allows the definition of macro-like structures that can be used to simplify the creation of complex structures. However, this capability can be abused to create excessive demands on a processor's CPU and memory. A small number of nested expansions can result in an exponential growth in demands on memory.
 - **Solution:** If possible, upgrade to the latest package release.
Design: Enforce principle of least privilege. Design: Ensure all input is validated, and does not contain file system commands Design: Run server interfaces with a non-root account and/or utilize chroot jails or other configuration techniques to constrain privileges even if attacker gains some limited access to commands. Design: For interactive user applications, consider if direct file system interface is necessary, instead consider having the application proxy communication. Implementation: Perform testing such as pen-testing and vulnerability scanning to identify directories, programs, and interfaces that grant direct access to executables.

15. psutil >= 4.2.0

- [CVE-2019-18874](#) **CVSS:** 5.0
 - **Name:** Unavailable
 - **Summary:** Unavailable
 - **Solution:** If possible, upgrade to the latest package release.

16. pygments >= 2.0.1

- [CVE-2015-8557](#) **CVSS:** 9.3
 - **Name:** OS Command Injection
 - **Summary:** In this type of an attack, an adversary injects operating system commands into existing application functions. An application that uses untrusted input to build command strings is vulnerable. An adversary can leverage OS command injection in an application to elevate privileges, execute arbitrary commands and compromise the underlying operating system.
 - **Solution:** If possible, upgrade to the latest package release.
Use language APIs rather than relying on passing data to the operating system shell or command line. Doing so ensures that the available protection mechanisms in the language are intact and applicable. Filter all incoming data to escape or remove characters or strings that can be potentially misinterpreted as operating system or shell commands All application processes should be run with the minimal privileges required. Also, processes must shed privileges as soon as they no longer require them.
- [CVE-2021-27291](#) **CVSS:** 5.0
 - **Name:** Unavailable
 - **Summary:** Unavailable
 - **Solution:** If possible, upgrade to the latest package release.
- [CVE-2021-20270](#) **CVSS:** 5.0
 - **Name:** Unavailable
 - **Summary:** Unavailable
 - **Solution:** If possible, upgrade to the latest package release.