Risk Level: MEDIUM

Executive Summary

After examining the `uuid` package (version 1.30) from PyPI, we believe the security risk associated with use of this module is **MEDIUM**. We make a few recommendations described below to work around potentially dangerous functionality, and standard Python distributions have included a UUID module since version 2.5 (September 2006), which happens to be derived from this module. **We recommend all users cease using this package and instead use the built-in module, which has a nearly equivalent API surface.**

Detailed Results

[High] LB-002: Time-based UUIDs are not guaranteed to be unique.

This package calls the `uuid_generate_time` functions located within `libuuid` or `libc`. These functions are not thread-safe, in that there is no guarantee that two threads calling the function at the same time won't receive the same time. This has been shown in real-world cases, though during our testing, we were unable to reproduce it. Nevertheless, this means the UUID generated cannot be guaranteed to be unique.

The UUID module that's part of the standard Python distribution solves this by attempting to call `uuid_generate_time_safe` and making available an `is_safe` property of the UUID object, which conveys whether the UUID was generated in a way that ensures each call returns a unique result.

Recommendation: Avoid using the `uuid.uuid1` in multi-threaded environments unless you explicitly account for the generation of duplicate UUIDs in your threat model. Alternatively, use the `uuid` module built into Python.

[High] LB-004: Module implements UUID versions that use weak hash algorithms.

This package implements the UUIDv3 and UUIDv5 algorithms, which use the deprecated MD5 and SHA-1 algorithms respectively. Since these algorithms are required by RFC 4122, their use should not be considered a code defect.

Note: This issue exists in many, if not all RFC-compliant UUID libraries; nothing about this issue is unique to this specific implementation.

Recommendation: Avoid using the `uuid.uuid3` and `uuid.uuid5` functions unless you explicitly account for cryptographic attacks (collision resistance) in your threat model.

[Medium] LB-001: Module implements UUID versions that reveal local hardware information (MAC address).

This package attempts to use the network adapter's MAC address as part of the `uuid.uuid1` function. Since the hardware address isn't protected (as per RFC 4122), receivers of the UUID generated will learn what one could argue is private information about the service.

Note: This issue exists in many, if not all RFC-compliant UUID libraries; nothing about this issue is unique to this specific implementation.

Recommendation: Avoid using the `uuid.uuid1` function unless your threat model accounts for MAC address disclosure.

[Medium] LB-003: External commands called from current path.

This package attempts to use the network adapter's MAC address as part of the `uuid.uuid1` function. If it's first attempt fails, it falls back to calling the `ifconfig` or `ipconfig` commands (depending on the operating system). Unfortunately, the package first looks in the current working directory for those commands. This means that if an attacker can place a file called `ifconfig` in a directory and then run a script that calls `uuid1` from that directory, the attacker's `ifconfig` file may be executed instead of the one intended.

This vulnerability has been remediated in the `uuid` module built into Python by only using the system path plus well known directories.

Recommendation: Ensure that attackers cannot cause `uuid.uuid1` to be called from a directory under their control.

[Medium] LB-005: Fallback to pseudo-random number generators.

The library makes attempts at using good sources of randomness (`getrandom`, `/dev/urandom`, `UuidCreateSequential`, or `UuidCreate`) but if those calls fail, the library will fall back to a pseudo-random number generator. Unfortunately, the library doesn't reveal to the caller which method was used.

Recommendation: Avoid using the `uuid.uuid4` function in cases where cryptographically secure UUIDs are needed (as dictated by your threat model). Instead, you can call the `UUID` constructor, passing in bytes you obtain from `os.urandom` or `secrets.token_bytes`.

Other Observations

[High] LB-006: The module is not compatible with Python 3.

The module uses a few constructs that are not permitted in Python 3, causing a syntax error during loading.

Recommendation: Since Python 2 has reached end-of-life, use the version that comes with the standard Python distribution.

[Medium] LB-007: The module is no longer maintained.

The module was last released in 2006, shortly before it was included as a standard Python module. The project's home page is simply a directory listing with source code.

Recommendation: Do not use this module; use the version that comes with the standard Python distribution.

[Medium] LB-008: The module is no longer loaded by default.

Even if you install the module using `pip install uuid==1.30`, Python will prefer the built-in uuid module. It's unclear whether any "normal" conditions would cause the module loader to choose this module over the built-in one.

```
# python -mvenv test-venv
# source test-venv/bin/activate
(test-venv) # pip install uuid
...
Successfully installed uuid-1.30
(test-venv) # python -c 'import uuid; print(uuid.__file__)'
/usr/lib/python3.10/uuid.py
```

Recommendation: Do not use this module; use the version that comes with the standard Python distribution.

Methodology & Metadata

Analysis Target:	https://pypi.org/project/uuid http://zesty.ca/python/uuid.py
Methodology	Static Analysis, Code Review, Web Search, Comparative Analysis
Language:	JavaScript
Code Review:	Yes (complete)
Timeline	Approximately four hours on 23 December 2022

Analyst Notes

- The source code available in the module uploaded to pypi.org is substantially identical to the source code on the zesty.ca website. The source code is clearly an ancestor of the standard Python module available in the CPython source tree.
- The module does not appear to be released under any license, but according to ClearlyDefined, the license is Python-2.0.1.
- To identify which `uuid` module is being used, you can use the `importlib` as described here.