MALDET · STATIC SECURITY REPORT

# xonsh

github.com/xonsh/xonsh

Scanned 24 September 2026 at 08:03 · Report generated 29 October 2026 · Python · 9,653 stars

OVERALL ASSESSMENT

**Critical**

Combined score 465 · 330 findings after filtering

## Summary

This repository has two critical security issues that are likely caused by ordinary insecure coding practices rather than malicious intent. The most serious problem is the use of shell=True when running subprocess commands, which allows attackers to inject dangerous code or steal secrets if the command is executed from an untrusted source. Additionally, the project uses a debugger and sandbox detection mechanism in its own code, which suggests it may be designed to evade security tools like antivirus software. While there are some medium-severity warnings about potential SQL injection risks and pickle vulnerabilities, these appear to be standard coding patterns rather than signs of active hacking attempts. The overall risk score of 360.0 indicates a high level of vulnerability that requires immediate attention before the code can be safely used in production environments.

Written by a language model running locally on the scanning machine, from the findings in this report. The risk levels above and below were decided by the scoring model, not by this summary.

## Risk, by type

MalDet scores two different questions separately, because they call for different responses: code can be badly written without being hostile, and hostile code is not always badly written.

VULNERABILITY

**Critical** (360)

Insecure coding an attacker could exploit. Usually mistakes, and usually fixable in place.

MALICIOUS PATTERN

**Critical** (105)

Signs the code may be trying to do harm: hidden payloads, credential access, persistence, exfiltration.

## What each detector found

| DETECTOR | LOOKS FOR | HIGH | MEDIUM | LOW | TOTAL |
|---|---|---|---|---|---|
| **bandit** | Insecure coding patterns in Python source | 4 | 35 | 109 | **148** |
| **semgrep** | Insecure coding patterns across many languages | 17 | 106 | 0 | **123** |
| **dep_checker** | Vulnerable, unpinned or typosquatted dependencies | 0 | 39 | 11 | **50** |
| **guarddog** | Malicious package and install-time behaviour | 3 | 2 | 0 | **5** |
| **yara** | Signatures of malicious behaviour in source files | 3 | 0 | 1 | **4** |

154 of 330 findings were rated more likely genuine than noise by the classifier.

## Evidence

Identical findings are grouped, so a single rule firing many times takes one row. Ordered worst first. “Seen” is how many times that finding occurred.

| SEVERITY | DETECTOR | FINDING | SEEN | EXAMPLE LOCATION |
|---|---|---|---|---|
| HIGH | guarddog | Detects suspicious autostart persistence mechanisms [threat-filesystem-autostart] | 3 | `/xonsh/history/main.py:84` |
| HIGH | semgrep | Using variable interpolation `${{...}}` with `github` context data in a `run:` step could allow an attacker to inject their own code into the runner. This would allow them to steal secrets and code. `github` context data can have arbitrary user input and should be treated as untrusted. Instead, use | 1 | `/.github/workflows/publish.yml:68` |
| HIGH | semgrep | Found 'subprocess' function 'check_output' with 'shell=True'. This is dangerous because this call will spawn the command using a shell process. Doing so propagates current shell settings and variables, which makes it much easier for a malicious actor to execute commands. Use 'shell=False' instead. | 1 | `/xonsh/tools.py:1190` |
| HIGH | semgrep | Avoiding SQL string concatenation: untrusted input concatenated with raw SQL query can result in SQL Injection. In order to execute raw query safely, prepared statement should be used. SQLAlchemy provides TextualSQL to easily used prepared statement with named parameters. For complex SQL composition | 15 | `/xonsh/history/sqlite.py:294` |

| SEVERITY | DETECTOR | FINDING | SEEN | EXAMPLE LOCATION |
|---|---|---|---|---|
| **HIGH** | bandit | Starting a process with a shell, possible injection detected, security issue. | 2 | `/xonsh/shells/ptk_shell/__init__.py:791` |
| **HIGH** | bandit | subprocess call with shell=True identified, security issue. | 2 | `/xonsh/pytest/tools.py:96` |
| **HIGH** | yara | YARA rule matched: detect_anti_analysis — Checks for a debugger, VM, or sandbox before continuing - a common malware evasion technique, essentially never present in ordinary application code | 1 | `/xonsh/xoreutils/umask.py` |
| **HIGH** | yara | YARA rule matched: detect_download_and_execute — Downloads a remote resource AND executes it - the actually dangerous combination, not just downloading alone | 1 | `/AI_POLICY.md` |
| **HIGH** | yara | YARA rule matched: detect_persistence_mechanism — Modifies a startup/scheduling location - legitimate application code essentially never does this | 1 | `/xonsh/history/main.py` |
| **MEDIUM** | semgrep | Avoid using `cPickle`, which is known to lead to code execution vulnerabilities. When unpickling, the serialized data could be manipulated to run arbitrary code. Instead, consider serializing the relevant data as JSON or a similar text-based serialization format. | 12 | `/xonsh/parsers/ply/yacc.py:2860` |
| **MEDIUM** | semgrep | The marshal module is not intended to be secure against erroneous or maliciously constructed data. Never unmarshal data received from an untrusted or unauthenticated source. See more details: https://docs.python.org/3/library/marshal.html?highlight=security | 3 | `/xonsh/codecache.py:217` |

| SEVERITY | DETECTOR | FINDING | SEEN | EXAMPLE LOCATION |
|---|---|---|---|---|
| MEDIUM | semgrep | This Dependabot configuration does not set a cooldown period. Newly published packages can be malicious or unstable. Add a `cooldown` block with `default-days: 7` to each `package-ecosystem` entry under `updates` to wait 7 days before proposing updates to newly published package versions. Reference: | 1 | `/.github/dependabot.yml:3` |
| MEDIUM | dep_checker | setup.py overrides the install process (class xsdist(sdist)) — review its run()/initialize_options() for unexpected behavior | 1 | `setup.py:163` |
| MEDIUM | dep_checker | setup.py overrides the install process (class xbuild_py(build_py)) — review its run()/initialize_options() for unexpected behavior | 1 | `setup.py:126` |
| MEDIUM | dep_checker | setup.py overrides the install process (class xdevelop(develop)) — review its run()/initialize_options() for unexpected behavior | 1 | `setup.py:217` |
| MEDIUM | dep_checker | setup.py overrides the install process (class xinstall(install)) — review its run()/initialize_options() for unexpected behavior | 1 | `setup.py:146` |
| MEDIUM | bandit | Using xml.dom.minidom.parseString to parse untrusted XML data is known to be vulnerable to XML attacks. Replace xml.dom.minidom.parseString with its defusedxml equivalent function or make sure defusedxml.defuse_stdlib() is called | 1 | `/xonsh/webconfig/tags.py:106` |
| MEDIUM | bandit | Deserialization with the marshal module is possibly dangerous. | 2 | `/xonsh/codecache.py:217` |
| MEDIUM | guarddog | Detects access to sensitive files (credentials, configs, keys) [threat-filesystem-read] | 2 | `/xonsh/parsers/base.py:3506` |
| MEDIUM | dep_checker | Unpinned dependency: 'gnureadline' has no version lock — supply chain risk | 2 | `pyproject.toml` |
| MEDIUM | dep_checker | Unpinned dependency: 'setproctitle' has no version lock — supply chain risk | 2 | `pyproject.toml` |
| MEDIUM | dep_checker | Unpinned dependency: 'distro' has no version lock — supply chain risk | 2 | `pyproject.toml` |

| SEVERITY | DETECTOR | FINDING | SEEN | EXAMPLE LOCATION |
|---|---|---|---|---|
| MEDIUM | dep_checker | Unpinned dependency: 'pytest-cov' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'matplotlib' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'myst-parser' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'numpydoc' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'pre-commit' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'psutil' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'pyperclip' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 're-ver' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'tornado' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'sphinx-prompt' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'xonsh' has no version lock — supply chain risk | 4 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'pytest-rerunfailures' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'restructuredtext_lint' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'runthis-sphinxext' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'sphinx-reredirects' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'tomli' has no version lock — supply chain risk | 1 | pyproject.toml |
| MEDIUM | dep_checker | Unpinned dependency: 'pyzmq' has no version lock — supply chain risk | 1 | pyproject.toml |

| SEVERITY | DETECTOR | FINDING | SEEN | EXAMPLE LOCATION |
|---|---|---|---|---|
| **MEDIUM** | dep_checker | Unpinned dependency: 'click' has no version lock — supply chain risk | 1 | `pyproject.toml` |
| **MEDIUM** | dep_checker | Unpinned dependency: 'furo' has no version lock — supply chain risk | 1 | `pyproject.toml` |
| **MEDIUM** | dep_checker | Unpinned dependency: 'ujson' has no version lock — supply chain risk | 1 | `pyproject.toml` |
| **MEDIUM** | dep_checker | Unpinned dependency: 'sphinx-autobuild' has no version lock — supply chain risk | 1 | `pyproject.toml` |
| **MEDIUM** | dep_checker | Unpinned dependency: 'sphinx-sitemap' has no version lock — supply chain risk | 1 | `pyproject.toml` |
| **MEDIUM** | semgrep | Detected possible formatted SQL query. Use parameterized queries instead. | 3 | `/xonsh/history/sqlite.py:53` |
| **MEDIUM** | semgrep | Detected a dynamic value being used with urllib. urllib supports 'file://' schemes, so a dynamic value controlled by a malicious actor may allow them to read arbitrary files. Audit uses of urllib calls to ensure user data cannot control the URLs, or consider using the 'requests' library instead. | 1 | `/xonsh/lib/openpy.py:117` |
| **MEDIUM** | semgrep | GitHub Actions step uses a mutable tag or branch reference. Tags and branch names can be silently repointed by the action owner, enabling supply-chain attacks — as seen in the trivy-action and kics-github-action compromises. Pin the reference to a full 40-character commit SHA instead, e.g. `uses: ac | 51 | `/.github/workflows/nightly-build.yml:73` |
| **MEDIUM** | dep_checker | Unpinned dependency: 'pytest-mock' has no version lock — supply chain risk | 1 | `pyproject.toml` |
| **MEDIUM** | semgrep | Detected the use of exec(). exec() can be dangerous if used to evaluate dynamic content. If this content can be input from outside the program, this may be a code injection vulnerability. Ensure evaluated content is not definable by external sources. | 6 | `/xonsh/parsers/ply/yacc.py:3262` |

| SEVERITY | DETECTOR | FINDING | SEEN | EXAMPLE LOCATION |
|---|---|---|---|---|
| **MEDIUM** | semgrep | Detected the use of eval(). eval() can be dangerous if used to evaluate dynamic content. If this content can be input from outside the program, this may be a code injection vulnerability. Ensure evaluated content is not definable by external sources. | 4 | `/xonsh/parsers/ply/yacc.py:1566` |
| **MEDIUM** | semgrep | Avoid using `pickle`, which is known to lead to code execution vulnerabilities. When unpickling, the serialized data could be manipulated to run arbitrary code. Instead, consider serializing the relevant data as JSON or a similar text-based serialization format. | 12 | `/xonsh/parsers/ply/yacc.py:2018` |
| **MEDIUM** | semgrep | Untrusted user input in `importlib.import_module()` function allows an attacker to load arbitrary code. Avoid dynamic values in `importlib.import_module()` or use a whitelist to prevent running untrusted code. | 11 | `/xonsh/pygments_cache.py:503` |
| **MEDIUM** | bandit | Audit url open for permitted schemes. Allowing use of file:/ or custom schemes is often unexpected. | 1 | `/xonsh/lib/openpy.py:117` |
| **MEDIUM** | bandit | Possible SQL injection vector through string-based query construction. | 13 | `/xonsh/history/sqlite.py:262` |
| **MEDIUM** | bandit | Function call with shell=True parameter identified, possible security issue. | 8 | `/xonsh/foreign_shells.py:653` |
| **MEDIUM** | semgrep | Found non static data as an index to 'globals()'. This is extremely dangerous because it allows an attacker to execute arbitrary code on the system. Refactor your code not to use 'globals()'. | 2 | `/xonsh/parsers/completion_context.py:268` |
| **MEDIUM** | dep_checker | Unpinned dependency: 'pytest-timeout' has no version lock — supply chain risk | 1 | `pyproject.toml` |
| **MEDIUM** | dep_checker | Unpinned dependency: 'pytest-subprocess' has no version lock — supply chain risk | 1 | `pyproject.toml` |
| **MEDIUM** | dep_checker | Unpinned dependency: 'requests' has no version lock — supply chain risk | 1 | `pyproject.toml` |

| SEVERITY | DETECTOR | FINDING | SEEN | EXAMPLE LOCATION |
|---|---|---|---|---|
| MEDIUM | bandit | Use of possibly insecure function - consider using safer ast.literal_eval. | 4 | `/xonsh/execer.py:199` |

Showing the 60 most significant of 88 distinct findings. The counts in the previous section cover all of them.

## How to read this report

- **This is a static scan.** The code was read, never executed. Behaviour that only appears at runtime is outside what any of these detectors can see.
- **Findings are not all equally real.** A classifier rates each finding on how likely it is to be genuine rather than noise, and that rating weights the scores. Measured against a hand-reviewed sample it reached precision 0.52 and recall 0.62 — useful for ranking, not a verdict on any single finding.
- **A high score is not proof of malice.** Security tools, penetration-testing frameworks and malware research repositories contain the same patterns as real malware by design. The two risk types above exist to keep that distinction visible.
- **Absence of findings is not proof of safety.** A detector that is not installed, or a language none of them parse, produces silence rather than a warning.
- **This scan is a snapshot.** The repository may have changed since 24 September 2026.