TCP Port Scanner + Evidence Logger

FREE CASEBOOK READER EDITION; NOT A COMMERCIAL BOOK / 66c12185-71d9-4558-8e1e-3b0a09aa4964

Complete written explanations, source, commands and references. Interactive actions have readable print equivalents.

Biothreat Corporation / sales@pythonprojects.org

Code listings wrap visually. Use the included exact .py source for execution and byte hashing; do not reconstruct it from wrapped PDF lines.

Contents

TCP Port Scanner + Evidence Logger 4

The case brief . . . 4

Computer & access checklist . . . 4

Choose your setup route . . . 4

1. The professional problem . . . 5

2. Your deliverable and success criteria . . . 5

3. Before you begin: files, terminals, and Python . . . 6

4. Windows setup: PowerShell and folders . . . 6

5. Linux setup: Bash and folders . . . 7

6. First result: the controlled loopback lab . . . 8

7. Checkpoint A: one ordinary TCP connection . . . 8

8. Checkpoint B: reuse the reviewed connection function . . . 9

9. Checkpoint C: arguments, reports, and the final program . . . 9

10. Timestamps, output bytes, and integrity . . . 10

11. Run the tests . . . 11

12. Troubleshooting clinic . . . 11

13. Exercises and worked solutions . . . 12

14. Professional handoff . . . 12

15. References and version record . . . 13

Reading the exact implementation . . . 13

Source walkthrough / module-imports-and-constants . . . 13

Source walkthrough / utc_now . . . 14

Source walkthrough / parse_ports . . . 14

Source walkthrough / parse_timeout . . . 16

Source walkthrough / check_tcp_port . . . 17

Source walkthrough / main . . . 18

Source walkthrough / entry-point 20
Complete source for independent reading 21
Run and verify the controlled lab 23
Read the evidence with its provenance 23
Write a defensible finding 23
Independent verification exercise 24
Verify your case deliverable 24
Professional fields and portfolio 24
Continue the investigation 24
Your complete project kit 24
Primary references 24
Primary references 25
Recorded test evidence 25

TCP Port Scanner + Evidence Logger

Build a bounded TCP scanner, record JSON observations, and verify report bytes with SHA-256.

License: MIT

Source document: 61970d26-f267-5cfa-9e62-db5b1868566b / schema 1

The case brief

Observe a known local TCP service, preserve its connection results as JSON, and hand off a report with a matching SHA-256 digest. Work on 127.0.0.1 for this investigation. A successful connection is an observation, not evidence of a vulnerability.

Computer & access checklist

Operating system

Windows 11 with PowerShell, or Linux with Bash. Windows-native and Ubuntu under WSL were tested. macOS, phones, tablets, and browser-only coding environments are not covered by this lesson. You can read the lesson on a smaller screen; the hands-on exercises require a computer with the listed setup.

Software

Python 3.14 is the lesson baseline; the release was tested with Python 3.14.4. Use uv or Python's built-in venv as shown in Setup. A plain-text editor, ZIP extractor, and browser are needed. The scanner and tests use only Python's standard library.

Computer permissions

Run the lab as a normal user. You need permission to run Python, open two terminal sessions, and create/read/write files in your extracted project folder. The local listener uses unprivileged TCP ports 8765/8766; tests use temporary high ports. Administrator/root access is not required to run the exercises. Installing Python or uv may require IT approval or administrator access, depending on your device.

Network access

Internet is needed for the initial downloads. The complete lab and tests run locally over 127.0.0.1. Device policy must allow Python's loopback TCP connections. Other hosts require the system owner's explicit authorization and agreed scope. Keep security controls enabled; ask IT if a managed device blocks the lab.

Recorded tests

15 tests passed on Windows 11 and Ubuntu Linux under WSL, with Python 3.14.4. WSL installation is optional and can require administrator approval. Fresh OS installation, all Linux distributions, and other Python versions were not verified.

Choose your setup route

Extract the complete project ZIP first and open a terminal in its folder. Install Python or uv from its official documentation if needed. The commands below create an isolated environment and run offline tests. They do not install packages; this project uses only the standard library. Installer procedures on a fresh machine have not been tested.

Windows PowerShell / uv

```
uv --version
uv venv --python 3.14
uv run --no-project python --version
uv run --no-project python -m unittest -v
```

Linux Bash / uv

```
uv --version
uv venv --python 3.14
uv run --no-project python --version
uv run --no-project python -m unittest -v
```

Windows PowerShell / standard Python

```
py -3.14 -m venv .venv
.\.venv\Scripts\python.exe --version
.\.venv\Scripts\python.exe -m unittest -v
```

Linux Bash / standard Python

```
python3.14 -m venv .venv
./.venv/bin/python --version
./.venv/bin/python -m unittest -v
```

1. The professional problem

A network-support technician may need to check whether a known service accepts a TCP connection from a particular machine. A defensive analyst may need a timestamped observation during incident triage. A forensic examiner may use the resulting report as contextual material, with appropriate authorization and preservation. These jobs need a small repeatable procedure and an explanation of what the procedure actually measured.

Our tool resolves one host to one IPv4 address, tries 1-64 ports sequentially, and writes a JSON report plus a SHA-256 sidecar. A TCP connection attempt may be logged by endpoint security and may consume service resources. Use loopback for this lesson. Use another person's systems only with permission and a defined scope.

A connection succeeds when the operating system completes the TCP handshake. It does not tell us which application accepted it, whether that application is vulnerable, or whether authentication would succeed. A refusal is an active rejection. A timeout means the deadline expired; several explanations are possible. We deliberately never label a timeout as proof of a firewall.

2. Your deliverable and success criteria

You will produce a JSON file with the requested host, resolved IPv4 address, start/end UTC timestamps, interpreter and platform details, timeout, and a list of connection observations. Each observation contains a

port, status, elapsed milliseconds, and an optional OS error number. A second file contains the hexadecimal SHA-256 digest of the exact JSON bytes.

Success means a local listening socket is reported as connected, the JSON parses, the independently calculated digest matches its sidecar, and you can explain the limitations. The tests create their own ephemeral local listener so they do not depend on a public host or on port 8765 being free.

This supports network support, network administration, security analysis, and forensic scripting practice. It is one portfolio artifact, not a professional qualification. See Careers & Pay for dated occupational references and broad-category proxy caveats.

3. Before you begin: files, terminals, and Python

A terminal is a window for entering commands. The shell interprets those commands: PowerShell on Windows, Bash in this Linux walkthrough. Python is a separate program called an interpreter; it reads and executes a source file ending in .py. Opening that file in an editor does not run it.

The current working directory is the folder a command starts from. Relative paths such as results/report.json are interpreted from that folder. A path with spaces must be quoted. Do not type a displayed shell prompt such as PS C:\> or $. If you see >>>, you are inside Python's interactive prompt; enter exit() to return to your shell.

PATH is the list of folders the shell searches for executable programs. Installing Python or uv may update PATH only for newly opened terminals. Reopening a terminal is a sensible first step when a just-installed command is not found.

Download the ZIP using this page's Download Project link. Extract the entire archive first. Do not run files from inside the ZIP viewer. Use VS Code, VSCodium, Notepad, or another plain-text editor to inspect network_triage.py. A word processor may insert formatting that breaks source files. No editor account is required.

4. Windows setup: PowerShell and folders

Open Start, search for PowerShell, and open it as an ordinary user. Administrator access is not needed for this loopback lab. In File Explorer, extract the download and locate the folder containing network_triage.py. Copy that folder's full path from the address bar.

```
Set-Location "C:\Users\YOUR_NAME\Downloads\PP-001-1.0.0"
Get-Location
Get-ChildItem
```

Replace YOUR_NAME and the folder path with your actual extracted location. Get-Location should show that folder; Get-ChildItem should include network_triage.py and test_network_triage.py. These commands only change or inspect your terminal's folder.

The recommended environment method is uv. Verify Windows Package Manager first; then install uv from its documented WinGet identifier.

```
winget --version
winget install --id=astral-sh.uv -e
```

Reopen PowerShell after installation, return to the project folder, then run:

```
uv --version
uv python install 3.14
uv run --python 3.14 network_triage.py --help
```

The expected help lists --host, --ports, --timeout, --output, and --authorized. Version strings vary. uv run selects the requested interpreter; it does not require a project account or external packages here. If WinGet is missing, use the official uv installation options. Follow your organization's installation policy.

The standard alternative is an installed CPython interpreter. Install Python from python.org using its official installer if needed. Verify the launcher before proceeding:

```
py --version
py -m venv .venv
.\.venv\Scripts\python.exe network_triage.py --help
```

The virtual environment is a disposable local interpreter environment. This project has no external dependencies, so it is optional; we introduce it because it is useful for later projects. Directly invoking its Python avoids PowerShell activation-policy changes. If py is missing but python --version works, substitute python for py in the first two commands. Check the reported version rather than assuming which interpreter a command selects.

5. Linux setup: Bash and folders

Open your distribution's terminal. Extract the ZIP using the file manager and locate the directory containing network_triage.py. Example commands below assume that directory is under Downloads; adjust the path to match your machine.

```
cd "$HOME/Downloads/PP-001-1.0.0"
pwd
ls
```

cd changes directory, pwd prints it, and ls lists files. Quoting a path preserves embedded spaces. Linux filenames are case-sensitive. The $HOME expression is expanded by Bash to your own home directory.

For uv, download the official installer to a file for inspection before executing it:

```
curl -LsSf https://astral.sh/uv/install.sh -o install-uv.sh
less install-uv.sh
sh install-uv.sh
```

Press q to leave less. Read the script and its source before deciding to run it. If curl or less is unavailable, use your distribution's documented package manager or another installer option from the official uv documentation. Do not run the installer with sudo to solve PATH problems. Reopen the terminal, return to the extracted folder, and verify:

```
uv --version
uv python install 3.14
uv run --python 3.14 network_triage.py --help
```

For the standard alternative:

```
python3 --version
python3 -m venv .venv
./.venv/bin/python network_triage.py --help
```

Some Debian/Ubuntu installations need the distribution's python3-venv package before the second command works. Consult the distro documentation for your actual version and permissions. Never use sudo pip to modify a system-managed interpreter. Do not copy a Windows .venv into Linux; recreate it there.

WSL is a Linux environment hosted on Windows, with distinct networking and filesystem behavior. It is useful additional evidence, but it is not equivalent to testing every standalone Linux distribution. The release matrix states which environment was actually used.

6. First result: the controlled loopback lab

An IP address identifies a network endpoint. The address 127.0.0.1 refers back to this machine in the current network environment. A TCP port identifies a destination within that endpoint. We will temporarily run Python's HTTP server as a known listener on port 8765, bound only to loopback.

Open two terminals in the project directory. Terminal A runs the fixture until you stop it. Terminal B runs the scanner. Select the command variants in the command panel above this article, or use these standard-interpreter commands.

Windows, terminal A:

```
.\.venv\Scripts\python.exe -m http.server 8765 --bind 127.0.0.1
```

Windows, terminal B:

```
.\.venv\Scripts\python.exe network_triage.py --ports 8765,8766 --output results/lab-before.json
Get-Content .\results\lab-before.json
```

Linux, terminal A:

```
./.venv/bin/python -m http.server 8765 --bind 127.0.0.1
```

Linux, terminal B:

```
./.venv/bin/python network_triage.py --ports 8765,8766 --output results/lab-before.json
cat results/lab-before.json
```

Port 8765 should show connected while that server is listening. If it cannot start because the address is in use, another program may already own that port. Stop your lab and choose an unused high port consistently in both terminals. Do not kill an unfamiliar system service.

Port 8766 may be refused, time out, or even connect if another service is there. Record what you observed instead of forcing the output to match a screenshot. This release's non-listener test produced a timeout on Windows and a refusal on Ubuntu WSL during development.

Press Ctrl+C in terminal A to stop the HTTP server. Repeat the scan in terminal B with --output results/lab-after.json. Compare the observations. Use a new filename because exclusive creation prevents overwriting previous evidence. The stopped listener should no longer provide the earlier successful connection, but another process or local policy can change the outcome.

7. Checkpoint A: one ordinary TCP connection

Create checkpoint_a.py with the following code. Run it while terminal A's local service is active. The code deliberately lets exceptions reach the terminal so you can see a real failure.

```
import socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as connection:
    connection.settimeout(0.5)
    connection.connect(("127.0.0.1", 8765))
print("connected")
```

The import binds the socket module to a name. The call socket.socket(...) returns a socket object. AF_INET chooses IPv4; SOCK_STREAM chooses TCP's stream interface. The as clause binds that object to connection. Four spaces indent the statements inside the with block; its context manager closes the socket when control leaves that block, including after an exception.

The dot accesses an attribute or method. Parentheses call a function, commas separate arguments, and the inner parentheses construct the address tuple. The string supplies the address and the integer supplies the port. A timeout is a float measured in seconds. connect returns None on success and raises an exception on failure.

Expected result with the fixture active: connected. Stop the service and rerun. The error details depend on the OS and local policy. Restart the fixture before continuing. The final scanner adds explicit classification and reporting around this same kind of operation.

8. Checkpoint B: reuse the reviewed connection function

Create checkpoint_b.py next to the final source:

```
from network_triage import check_tcp_port
for port in [8765, 8766]:
    observation = check_tcp_port("127.0.0.1", port, 0.5)
    print(observation)
```

The import guard in network_triage.py keeps importing from starting a scan. A list uses square brackets. The for statement binds port to each element in order, and indentation selects the repeated body. Each call returns a dictionary: a mapping from named keys to values. Strings carry labels; integers carry ports and error numbers; a float carries duration; None indicates that no OS error number was recorded.

Predict whether every returned dictionary will have the same keys. Answer: yes, because the function constructs one common return dictionary after its branches. Predict whether status and elapsed_ms will be identical. Answer: no; connection outcomes and timing are observations.

9. Checkpoint C: arguments, reports, and the final program

Run network_triage.py --help with your selected interpreter, then run the final lab command. Compare this progression: checkpoint A makes one raw connection; checkpoint B reuses classification for two ports; the final program validates options, resolves a target, gathers observations, records provenance, writes bytes, and hashes those exact bytes.

The source explorer below displays the exact downloaded source and its revision hash. Inline comments explain each important operation; this section supplies the language and operating-system context those comments refer to.

def introduces a function. Parameters name its inputs, return hands a value to the caller, and local variables exist within that call. main(argv=None) accepts an optional argument list for tests; ordinary CLI use reads the real process arguments. The __name__ guard runs main only when Python executes this file directly.

argparse stores named option values in an object. A type function such as parse_ports transforms and validates a string before collection begins. Defaults supplied as strings pass through argparse's conversion

too. action="store_true" creates a Boolean flag. Validation errors use exit status 2, successful collection uses 0, and operational failures use 1.

parse_ports splits on commas, strips surrounding whitespace, checks ASCII decimal notation, converts to integers, deduplicates with a set, and sorts the result. A set uses braces and contains unique values; a dictionary also uses braces but has key:value pairs. A generator expression supplies values to any without constructing another list. A comprehension expresses a compact loop. The input is limited to 64 tokens before deduplication.

and requires both conditions; or accepts either. not reverses truth. The comparisons <, >, and <= test ordering; == compares values. = assigns a name. A colon ends a compound statement header or separates dictionary keys from values. Quotes create strings; \n represents a newline inside a string. The multiplication by 1000 converts seconds to milliseconds; round(..., 3) limits decimal representation without increasing timing accuracy.

try encloses operations that may fail. except selects an exception type. raise propagates a failure; raise ... from exc preserves its cause. Connection timeouts, OS errors, DNS failures, and output failures mean different things and should not be silently collapsed into "closed". The Windows refusal code 10061 is handled alongside errno.ECONNREFUSED.

socket.gethostbyname resolves once, then the scanner uses that resolved address for all attempts. Hostnames can resolve to several addresses; this tool chooses one IPv4 result and records it. DNS resolution is outside the per-connection timeout. A non-loopback address requires --authorized, which is a reminder of scope, not a mechanism that grants permission.

The list comprehension in main calls check_tcp_port sequentially. With 64 ports and a 10-second timeout, connection waiting can approach 640 seconds; resolution and filesystem operations add overhead. There is no multi-host range, UDP, IPv6, banner retrieval, credential testing, stealth technique, or vulnerability detection.

10. Timestamps, output bytes, and integrity

UTC timestamps answer when the machine's wall clock said an event happened. They contain +00:00 to make the timezone explicit. They depend on the system clock and are not independently certified time. time.monotonic measures a duration; it is useful for elapsed time even when the wall clock changes.

Path builds filesystem paths independently of slash conventions. The output parent is created if needed. JSON is serialized with sorted keys and indentation, followed by a newline, then encoded once as UTF-8 bytes. Binary writing preserves those bytes across Windows and Linux. Changing indentation, line endings, or encoding changes the digest.

Exclusive modes xb and x fail if their destination already exists. This protects earlier outputs. The JSON and sidecar are separate writes: if the second write fails, a JSON file may remain. An interrupted or disk-full write may also leave an incomplete file. A failed command means you must inspect its artifacts and rerun to fresh paths. Do not treat a leftover file as a completed collection.

Verify independently on Windows:

```
Get-FileHash -Algorithm SHA256 .\results\lab-before.json
Get-Content .\results\lab-before.sha256
.\.venv\Scripts\python.exe -m json.tool .\results\lab-before.json
```

Verify independently on Linux:

```
sha256sum results/lab-before.json
cat results/lab-before.sha256
./.venv/bin/python -m json.tool results/lab-before.json
```

Compare the hexadecimal digests ignoring letter case. json.tool parses the file and prints a readable version; do not redirect it over the original evidence file. The package sidecar is a raw digest, not the filename-bearing format expected by sha256sum --check.

A matching digest demonstrates agreement with that particular baseline. Anyone who can replace both colocated files can generate a matching new pair. Independent authenticity, original acquisition, and chain of custody require additional controlled records and procedures. Record who collected the data, why, from which machine, with what scope, and where the baseline was preserved.

11. Run the tests

From the extracted project folder, use your chosen interpreter:

```
.\.venv\Scripts\python.exe -m unittest -v
./.venv/bin/python -m unittest -v
```

The suite exercises a real temporary loopback listener, a reserved non-listener, JSON parsing, independent hashing, tamper detection, invalid ports/timeouts, existing files, output path failures, help, and the authorization guard. DNS, timeouts, and some OS errors use mocks so those failures can be reproduced without scanning third parties. Mock success is not evidence that a real network produced that condition.

Tests use temporary directories and ephemeral ports assigned by the OS. Their expected result is an OK summary. Their execution time varies. The release matrix below reports actual runs, including the distinction between native Windows and Linux under WSL.

12. Troubleshooting clinic

python or py is not recognized: confirm the interpreter installation, reopen the terminal, and run its version command. On Linux, try python3. Do not type Python source into PowerShell.

uv: command not found: reopen the shell and inspect the installer's PATH guidance. Check the official installation documentation. Avoid adding untrusted directories to PATH.

Cannot open network_triage.py: run Get-Location/Get-ChildItem or pwd/ls. Enter the extracted folder. Check the exact filename; Windows may hide extensions, producing a mistaken .py.txt file.

Address already in use: the lab server could not bind the selected port. Choose another unused high port in both fixture and scanner commands. Confirm the bind address remains 127.0.0.1.

Invalid ports: use 1-64 comma-separated decimal integers from 1 to 65535. Ranges, blank tokens, signed numbers, and arbitrary strings are rejected.

Invalid timeout: use a finite number between 0.01 and 10. NaN and infinity are not usable deadlines.

Name resolution failure: confirm the hostname and network context. For this lab, use the literal 127.0.0.1 to avoid DNS. Do not interpret DNS failure as evidence about a port.

File exists or permission denied: choose a new output filename in a writable project folder. Inspect both JSON and sidecar destinations. Do not change system-wide permissions or run as administrator as a blanket fix.

Execution policy blocks activation: use the direct .venv interpreter path shown here. Activation is optional; this lab never requires disabling PowerShell protections.

JSONDecodeError or mismatched digest: preserve the suspect files, check whether collection failed or a text editor changed the file, and repeat with fresh paths. Do not edit a digest to make an altered report appear verified.

13. Exercises and worked solutions

Exercise 1 - duplicates: predict the parsed order for 8766,8765,8765. Solution: [8765, 8766]. The set removes repeats and sorted establishes numeric order. Add a unit test using assertEqual if you change the parser.

Exercise 2 - one byte: copy a lab report, append one space to the copy, and hash both files. Solution: digests should differ although both files may still parse as JSON. A hash covers bytes rather than a chosen interpretation of JSON.

Exercise 3 - report reasoning: after stopping the fixture, explain why timeout does not prove that the host is offline. Solution: local filtering, an unresponsive path, scheduling, and the short deadline are alternative explanations. The report should describe the observation and the conditions under which it was made.

Exercise 4 - timeout budget: estimate the connection-wait budget for 20 ports at 0.5 seconds. Solution: approximately 10 seconds in the worst case where every attempt waits to its deadline, plus other overhead; successful or refused attempts often finish sooner. DNS has no bound from this option.

Exercise 5 - refusal classification: explain why the suite mocks both errno.ECONNREFUSED and 10061. Solution: the operating-system interface may express the same kind of failure using different numeric values. The test validates classification independently of whether the host actually returns a refusal today.

Challenge - portfolio summary: collect before/after reports, verify their digests independently, and write a short comparison. Correctness requires scope, command, tool revision, environment, timestamps, observations, uncertainty, and limitations. Claiming a vulnerability merely because connected appears is incorrect.

14. Professional handoff

Include source revision, exact commands, original reports and sidecars, independent verification notes, and a findings summary. A sample template is included in the ZIP. If you share a real report, review hostnames, paths, and environment details for private information. The supplied fixture report is a development lab capture, not customer data.

Next, explore Network Engineering, Digital Forensics, the Network Snapshot Differ preview, and the Network Field Manual proposal. The other paid products remain planned.

15. References and version record

Release 1.0.0 introduces the scanner, bounded parser, loopback lab, tests, source annotations, and offline package. The release JSON records source SHA-256 and actual test environments.

References: Python socket, argparse, venv, hashlib, and official uv installation. Setup instructions were reviewed against these primary references on November 7, 2026; the installation commands themselves are not all claimed as executed on fresh machines.

Reading the exact implementation

The workbench below uses the immutable source snapshot in the free download. Each narrative step highlights a real source unit. Detailed commentary explains every meaningful source line; deeper notes distinguish language behavior, operating-system observations, and interpretation limits.

Source walkthrough / module-imports-and-constants

Imports load standard-library modules; constants name release policy.

Source network_triage.py / lines 2-18 / SHA-256
f7d735935961e2b11d996a1060092ef990f35dc88e4296a7a39db1b0040b4ada

```
import argparse  # Parse named command-line options and report usage errors.
import errno  # Translate operating-system error numbers without guessing vulnerabilities.
import hashlib  # Compute a SHA-256 digest of the exact report bytes.
import ipaddress  # Identify loopback addresses for the authorization guard.
import json  # Serialize dictionaries and lists into portable structured text.
import math  # Reject NaN and infinite timeout values.
import platform  # Record the environment that produced the observations.
import socket  # Ask the operating system to make ordinary TCP connections.
import sys  # Send operational failures to stderr and return an exit status.
import time  # Measure elapsed durations with a clock that cannot jump backwards.
from datetime import datetime, timezone  # Use explicit UTC wall-clock timestamps.
from pathlib import Path  # Build filesystem paths without shell-specific separators.
VERSION = "1.0.0"  # Identify the source and report format release.
DEFAULT_PORTS = "8765,8766"  # Use a small local lab rather than a public target.
```

Quick

Imports load standard-library modules; constants name release policy.

Detailed

Line 2: Parse named command-line options and report usage errors.

Line 3: Translate operating-system error numbers without guessing vulnerabilities.

Line 4: Compute a SHA-256 digest of the exact report bytes.

Line 5: Identify loopback addresses for the authorization guard.

Line 6: Serialize dictionaries and lists into portable structured text.

Line 7: Reject NaN and infinite timeout values.

Line 8: Record the environment that produced the observations.

Line 9: Ask the operating system to make ordinary TCP connections.

Line 10: Send operational failures to stderr and return an exit status.

Line 11: Measure elapsed durations with a clock that cannot jump backwards.

Line 12: Use explicit UTC wall-clock timestamps.

Line 13: Build filesystem paths without shell-specific separators.

Line 15: Identify the source and report format release.

Line 16: Use a small local lab rather than a public target.

Under the hood

Imports load standard-library modules; constants name release policy. Importing a module can execute its top-level statements. This module defers collection to main, so importing it for tests does not scan. No pip packages are needed.

Source walkthrough / utc_now

UTC makes a timestamp easier to compare, but the computer clock can be wrong.

Source network_triage.py / lines 19-20 / SHA-256
f7d735935961e2b11d996a1060092ef990f35dc88e4296a7a39db1b0040b4ada

```
def utc_now():
    return datetime.now(timezone.utc).isoformat()  # Return an aware UTC timestamp string.
```

Quick

UTC makes a timestamp easier to compare, but the computer clock can be wrong.

Detailed

Line 19: Define utc_now(). Parameters bind caller inputs; the indented body runs when called, not when defined.

Line 20: Return an aware UTC timestamp string.

Under the hood

UTC makes a timestamp easier to compare, but the computer clock can be wrong. An aware datetime records an offset; it does not establish an externally trusted time source. Durations elsewhere use a monotonic clock.

Source walkthrough / parse_ports

Validation bounds the amount of work before any sockets are created.

Source network_triage.py / lines 23-35 / SHA-256
f7d735935961e2b11d996a1060092ef990f35dc88e4296a7a39db1b0040b4ada

```
def parse_ports(text):
    parts = text.split(",")  # Split a comma-separated string into individual tokens.
    if len(parts) > 64:  # Bound the request before constructing connection attempts.
        raise argparse.ArgumentTypeError("provide at most 64 ports")
    try:
        if any(not part.strip().isascii() or not part.strip().isdecimal() for part in parts):  #
  Reject blanks, signs, ranges, and non-ASCII numerals.
            raise ValueError("use comma-separated decimal integers")
        ports = sorted({int(part.strip()) for part in parts})  # Convert, deduplicate with a set,
  and sort for stable order.
        if not ports or any(port < 1 or port > 65535 for port in ports):  # TCP port numbers must
  fit the valid nonzero range.
            raise ValueError("ports must be between 1 and 65535")
        return ports  # Hand a list of integers back to argparse and the caller.
    except ValueError as exc:  # Convert a conversion/range failure into a friendly CLI diagnostic.
        raise argparse.ArgumentTypeError(str(exc)) from exc  # Preserve the original cause for
  debugging.
```

Quick

Validation bounds the amount of work before any sockets are created.

Detailed

Line 23: Define parse_ports(text). Parameters bind caller inputs; the indented body runs when called, not when defined.

Line 24: Split a comma-separated string into individual tokens.

Line 25: Bound the request before constructing connection attempts.

Line 26: Convert validation failure to ArgumentTypeError so argparse can show help context and exit with status 2.

Line 27: Start a block whose failures are handled by the following except clauses. Successful statements continue in order.

Line 28: Reject blanks, signs, ranges, and non-ASCII numerals.

Line 29: Reject the invalid input with the specific message shown here. The surrounding except converts this ValueError to a CLI type error.

Line 30: Convert, deduplicate with a set, and sort for stable order.

Line 31: TCP port numbers must fit the valid nonzero range.

Line 32: Reject the invalid input with the specific message shown here. The surrounding except converts this ValueError to a CLI type error.

Line 33: Hand a list of integers back to argparse and the caller.

Line 34: Convert a conversion/range failure into a friendly CLI diagnostic.

Line 35: Preserve the original cause for debugging.

Under the hood

Validation bounds the amount of work before any sockets are created. A set removes duplicates and sorted gives deterministic ordering. ArgumentTypeError lets argparse report an input error separately from collection failures. No range expansion or parallelism is hidden here.

Source walkthrough / parse_timeout

A floating-point NaN has unusual comparison behavior.

Source network_triage.py / lines 38-45 / SHA-256 f7d735935961e2b11d996a1060092ef990f35dc88e4296a7a39db1b0040b4ada

```
def parse_timeout(text):
    try:
        value = float(text)  # Accept fractional seconds such as 0.5.
        if not math.isfinite(value) or not 0.01 <= value <= 10:  # NaN bypasses ordinary
  comparisons unless checked explicitly.
            raise ValueError("timeout must be finite and between 0.01 and 10 seconds")
        return value
    except ValueError as exc:
        raise argparse.ArgumentTypeError(str(exc)) from exc
```

Quick

A floating-point NaN has unusual comparison behavior.

Detailed

Line 38: Define parse_timeout(text). Parameters bind caller inputs; the indented body runs when called, not when defined.

Line 39: Start a block whose failures are handled by the following except clauses. Successful statements continue in order.

Line 40: Accept fractional seconds such as 0.5.

Line 41: NaN bypasses ordinary comparisons unless checked explicitly.

Line 42: Reject the invalid input with the specific message shown here. The surrounding except converts this ValueError to a CLI type error.

Line 43: Return the validated floating-point timeout to the caller; no further statements in this function run.

Line 44: Handle this exception class from the preceding try block. The as clause binds the exception so its message can become a useful CLI diagnostic.

Line 45: Convert validation failure to ArgumentTypeError so argparse can show help context and exit with status 2.

Under the hood

A floating-point NaN has unusual comparison behavior. math.isfinite explicitly rejects both NaN and infinity before checking the allowed interval. A bounded connect timeout does not bound DNS lookup or disk writes.

Source walkthrough / check_tcp_port

The operating system performs an ordinary TCP connection.

Source network_triage.py / lines 48-61 / SHA-256
f7d735935961e2b11d996a1060092ef990f35dc88e4296a7a39db1b0040b4ada

```
def check_tcp_port(address, port, timeout):
    started = time.monotonic()  # Duration is measured independently of wall-clock adjustments.
    error = None  # None becomes JSON null when no OS error was recorded.
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as connection:  # Create an IPv4 TCP
   socket; with closes it on every exit.
          connection.settimeout(timeout)  # Bound this connection attempt, not DNS or total
  filesystem work.
          connection.connect((address, port))  # A tuple identifies one destination; success
  returns None.
        status = "connected"  # A completed TCP handshake is an observation, not application
  identification.
    except (TimeoutError, socket.timeout):  # Timeout means the deadline expired; it does not prove
   filtering.
        status = "timeout"
    except OSError as exc:  # Preserve other failures, including refusals and unreachable hosts.
        error = exc.errno
        status = "refused" if exc.errno in (errno.ECONNREFUSED, 10061) else "error"  # Windows and
  Unix may use different numeric codes.
    return {"port": port, "status": status, "error_code": error, "elapsed_ms":
  round((time.monotonic() - started) * 1000, 3)}  # Values vary with the host and network.
```

Quick

The operating system performs an ordinary TCP connection.

Detailed

Line 48: Define check_tcp_port(address, port, timeout). Parameters bind caller inputs; the indented body runs when called, not when defined.

Line 49: Duration is measured independently of wall-clock adjustments.

Line 50: None becomes JSON null when no OS error was recorded.

Line 51: Start a block whose failures are handled by the following except clauses. Successful statements continue in order.

Line 52: Create an IPv4 TCP socket; with closes it on every exit.

Line 53: Bound this connection attempt, not DNS or total filesystem work.

Line 54: A tuple identifies one destination; success returns None.

Line 55: A completed TCP handshake is an observation, not application identification.

Line 56: Timeout means the deadline expired; it does not prove filtering.

Line 57: Classify only the expired deadline. This label does not infer a firewall or whether the host is offline.

Line 58: Preserve other failures, including refusals and unreachable hosts.

Line 59: Read the numeric operating-system error code from the caught exception. Preserve it in the returned observation.

Line 60: Windows and Unix may use different numeric codes.

Line 61: Values vary with the host and network.

Under the hood

The operating system performs an ordinary TCP connection. A connected result says a handshake completed from this machine at that time. Refused, timeout and other OS errors are different observations. None identifies a service, proves a vulnerability, or attributes a person. The context manager closes the socket even when an exception occurs.

Source walkthrough / main

The CLI validates input, resolves once, applies its scope guard, collects sequentially, encodes the JSON once, and writes without overwriting.

Source network_triage.py / lines 64-94 / SHA-256
f7d735935961e2b11d996a1060092ef990f35dc88e4296a7a39db1b0040b4ada

```
def main(argv=None):
    parser = argparse.ArgumentParser(description=__doc__)  # argv=None uses the real CLI; tests can
   supply a list.
    parser.add_argument("--host", default="127.0.0.1", help="one IPv4 address or hostname")
    parser.add_argument("--ports", type=parse_ports, default=DEFAULT_PORTS, help="1-64
  comma-separated ports")
    parser.add_argument("--timeout", type=parse_timeout, default=0.5, help="seconds per connection,
   0.01-10")
    parser.add_argument("--output", type=Path, help="new JSON path; existing files are never
  overwritten")
    parser.add_argument("--authorized", action="store_true", help="confirm authorization for a
  non-loopback destination")
    args = parser.parse_args(argv)  # Invalid arguments cause argparse to exit with status 2 before
   scanning.
    try:
        address = socket.gethostbyname(args.host)  # Resolve once to one IPv4 address; DNS time is
  not bounded by --timeout.
        if not ipaddress.IPv4Address(address).is_loopback and not args.authorized:  # The resolved
  address is also the address we connect to.
            parser.error("non-loopback targets require --authorized and permission from their
  owner")
        started = utc_now()
        observations = [check_tcp_port(address, port, args.timeout) for port in args.ports]  # Run
  sequentially, with no banner requests or concurrency.
        report = {"schema_version": 1, "tool_version": VERSION, "started_utc": started,
  "finished_utc": utc_now(), "requested_host": args.host, "resolved_ipv4": address,
  "timeout_seconds": args.timeout, "python": platform.python_version(), "platform":
  platform.platform(), "observations": observations}
        payload = (json.dumps(report, indent=2, sort_keys=True) + "\n").encode("utf-8")  # Encode
  once so the hash and saved bytes agree.
        output = args.output or Path("results") /
  (datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%S%fZ") + ".json")
        output.parent.mkdir(parents=True, exist_ok=True)  # Create missing parent directories;
  permission errors remain errors.
        with output.open("xb") as handle:  # Exclusive binary creation prevents silently
  overwriting an earlier report.
            handle.write(payload)
        sidecar = output.with_suffix(".sha256")  # Replace .json with .sha256, keeping the same
```

```
base name.
      with sidecar.open("x", encoding="utf-8", newline="\n") as handle:  # Exclusive text
creation also protects an existing sidecar.
          handle.write(hashlib.sha256(payload).hexdigest() + "\n")  # Store a digest only, not
proof of independent authenticity.
      print("Report:", output)
      print("SHA-256:", sidecar)
      for item in observations:
          print(str(item["port"]) + "/tcp", item["status"])  # Human summary; the JSON retains
the detailed observations.
      return 0  # Successful collection may include refused, timeout, or error observations.
  except (OSError, ValueError) as exc:  # DNS and file failures produce an operational error; a
JSON file may remain if sidecar creation fails.
      print("Collection failed:", exc, file=sys.stderr)
      return 1
```

Quick

The CLI validates input, resolves once, applies its scope guard, collects sequentially, encodes the JSON once, and writes without overwriting.

Detailed

Line 64: Define main(argv=None). Parameters bind caller inputs; the indented body runs when called, not when defined.

Line 65: argv=None uses the real CLI; tests can supply a list.

Line 66: Register this named CLI option, its default/conversion or Boolean action, and help text. parse_args later validates the supplied value.

Line 67: Register this named CLI option, its default/conversion or Boolean action, and help text. parse_args later validates the supplied value.

Line 68: Register this named CLI option, its default/conversion or Boolean action, and help text. parse_args later validates the supplied value.

Line 69: Register this named CLI option, its default/conversion or Boolean action, and help text. parse_args later validates the supplied value.

Line 70: Register this named CLI option, its default/conversion or Boolean action, and help text. parse_args later validates the supplied value.

Line 71: Invalid arguments cause argparse to exit with status 2 before scanning.

Line 72: Start a block whose failures are handled by the following except clauses. Successful statements continue in order.

Line 73: Resolve once to one IPv4 address; DNS time is not bounded by --timeout.

Line 74: The resolved address is also the address we connect to.

Line 75: Stop before scanning a resolved non-loopback destination without the explicit scope confirmation. argparse prints usage and exits with status 2.

Line 76: Capture the UTC start string before making connection attempts.

Line 77: Run sequentially, with no banner requests or concurrency.

Line 78: Assemble provenance and observations into a dictionary. Keys are strings; values include strings, numbers, and the observation list. No claim of clock authenticity is implied.

Line 79: Encode once so the hash and saved bytes agree.

Line 80: Use the requested Path when supplied; otherwise construct a UTC microsecond filename under results. Exclusive creation handles any filename collision.

Line 81: Create missing parent directories; permission errors remain errors.

Line 82: Exclusive binary creation prevents silently overwriting an earlier report.

Line 83: Write the exact encoded bytes to the exclusively created JSON file. A disk failure can leave a partial file.

Line 84: Replace .json with .sha256, keeping the same base name.

Line 85: Exclusive text creation also protects an existing sidecar.

Line 86: Store a digest only, not proof of independent authenticity.

Line 87: Print the path of the saved JSON for the operator. This does not independently verify its contents.

Line 88: Print the separate digest path after its write succeeds.

Line 89: Iterate over the collected dictionaries in port order, binding item to each observation.

Line 90: Human summary; the JSON retains the detailed observations.

Line 91: Successful collection may include refused, timeout, or error observations.

Line 92: DNS and file failures produce an operational error; a JSON file may remain if sidecar creation fails.

Line 93: Write the operational failure and original exception text to stderr so collection failure is visible to callers.

Line 94: Return an operational-failure exit code. Inspect stderr and any partial files before rerunning.

Under the hood

The CLI validates input, resolves once, applies its scope guard, collects sequentially, encodes the JSON once, and writes without overwriting. The digest describes those exact bytes. JSON and sidecar writes are not one atomic filesystem transaction: a failure can leave a partial pair. Preserve the diagnostic and inspect both files before a retry.

Source walkthrough / entry-point

Python sets __name__ differently for a directly executed script and an imported module.

Source network_triage.py / lines 97-98 / SHA-256
f7d735935961e2b11d996a1060092ef990f35dc88e4296a7a39db1b0040b4ada

```
if __name__ == "__main__":  # Importing the file for tests does not execute a scan.
    raise SystemExit(main())  # Pass the main function's status to the invoking shell.
```

Quick

Python sets __name__ differently for a directly executed script and an imported module.

Detailed

Line 97: Importing the file for tests does not execute a scan.

Line 98: Pass the main function's status to the invoking shell.

Under the hood

Python sets __name__ differently for a directly executed script and an imported module. This guard keeps tests from triggering collection on import. SystemExit passes the return value to the shell, where automation can distinguish successful collection from an operational failure.

Complete source for independent reading

This source is shown in full and remains readable without JavaScript.

Source SHA-256: f7d735935961e2b11d996a1060092ef990f35dc88e4296a7a39db1b0040b4ada

```
"""PP-001: single-host IPv4 TCP observations, JSON, and SHA-256."""
import argparse  # Parse named command-line options and report usage errors.
import errno  # Translate operating-system error numbers without guessing vulnerabilities.
import hashlib  # Compute a SHA-256 digest of the exact report bytes.
import ipaddress  # Identify loopback addresses for the authorization guard.
import json  # Serialize dictionaries and lists into portable structured text.
import math  # Reject NaN and infinite timeout values.
import platform  # Record the environment that produced the observations.
import socket  # Ask the operating system to make ordinary TCP connections.
import sys  # Send operational failures to stderr and return an exit status.
import time  # Measure elapsed durations with a clock that cannot jump backwards.
from datetime import datetime, timezone  # Use explicit UTC wall-clock timestamps.
from pathlib import Path  # Build filesystem paths without shell-specific separators.
VERSION = "1.0.0"  # Identify the source and report format release.
DEFAULT_PORTS = "8765,8766"  # Use a small local lab rather than a public target.
def utc_now():
    return datetime.now(timezone.utc).isoformat()  # Return an aware UTC timestamp string.
def parse_ports(text):
    parts = text.split(",")  # Split a comma-separated string into individual tokens.
    if len(parts) > 64:  # Bound the request before constructing connection attempts.
        raise argparse.ArgumentTypeError("provide at most 64 ports")
    try:
        if any(not part.strip().isascii() or not part.strip().isdecimal() for part in parts):  #
  Reject blanks, signs, ranges, and non-ASCII numerals.
            raise ValueError("use comma-separated decimal integers")
        ports = sorted({int(part.strip()) for part in parts})  # Convert, deduplicate with a set,
  and sort for stable order.
        if not ports or any(port < 1 or port > 65535 for port in ports):  # TCP port numbers must
  fit the valid nonzero range.
            raise ValueError("ports must be between 1 and 65535")
        return ports  # Hand a list of integers back to argparse and the caller.
    except ValueError as exc:  # Convert a conversion/range failure into a friendly CLI diagnostic.
        raise argparse.ArgumentTypeError(str(exc)) from exc  # Preserve the original cause for
```

```
  debugging.
def parse_timeout(text):
    try:
        value = float(text)  # Accept fractional seconds such as 0.5.
        if not math.isfinite(value) or not 0.01 <= value <= 10:  # NaN bypasses ordinary
  comparisons unless checked explicitly.
            raise ValueError("timeout must be finite and between 0.01 and 10 seconds")
        return value
    except ValueError as exc:
        raise argparse.ArgumentTypeError(str(exc)) from exc
def check_tcp_port(address, port, timeout):
    started = time.monotonic()  # Duration is measured independently of wall-clock adjustments.
    error = None  # None becomes JSON null when no OS error was recorded.
    try:
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as connection:  # Create an IPv4 TCP
   socket; with closes it on every exit.
            connection.settimeout(timeout)  # Bound this connection attempt, not DNS or total
  filesystem work.
            connection.connect((address, port))  # A tuple identifies one destination; success
  returns None.
        status = "connected"  # A completed TCP handshake is an observation, not application
  identification.
    except (TimeoutError, socket.timeout):  # Timeout means the deadline expired; it does not prove
   filtering.
        status = "timeout"
    except OSError as exc:  # Preserve other failures, including refusals and unreachable hosts.
        error = exc.errno
        status = "refused" if exc.errno in (errno.ECONNREFUSED, 10061) else "error"  # Windows and
  Unix may use different numeric codes.
    return {"port": port, "status": status, "error_code": error, "elapsed_ms":
  round((time.monotonic() - started) * 1000, 3)}  # Values vary with the host and network.
def main(argv=None):
    parser = argparse.ArgumentParser(description=__doc__)  # argv=None uses the real CLI; tests can
   supply a list.
    parser.add_argument("--host", default="127.0.0.1", help="one IPv4 address or hostname")
    parser.add_argument("--ports", type=parse_ports, default=DEFAULT_PORTS, help="1-64
  comma-separated ports")
    parser.add_argument("--timeout", type=parse_timeout, default=0.5, help="seconds per connection,
   0.01-10")
    parser.add_argument("--output", type=Path, help="new JSON path; existing files are never
  overwritten")
    parser.add_argument("--authorized", action="store_true", help="confirm authorization for a
  non-loopback destination")
    args = parser.parse_args(argv)  # Invalid arguments cause argparse to exit with status 2 before
   scanning.
    try:
        address = socket.gethostbyname(args.host)  # Resolve once to one IPv4 address; DNS time is
  not bounded by --timeout.
        if not ipaddress.IPv4Address(address).is_loopback and not args.authorized:  # The resolved
  address is also the address we connect to.
            parser.error("non-loopback targets require --authorized and permission from their
  owner")
        started = utc_now()
        observations = [check_tcp_port(address, port, args.timeout) for port in args.ports]  # Run
  sequentially, with no banner requests or concurrency.
        report = {"schema_version": 1, "tool_version": VERSION, "started_utc": started,
  "finished_utc": utc_now(), "requested_host": args.host, "resolved_ipv4": address,
  "timeout_seconds": args.timeout, "python": platform.python_version(), "platform":
  platform.platform(), "observations": observations}
        payload = (json.dumps(report, indent=2, sort_keys=True) + "\n").encode("utf-8")  # Encode
  once so the hash and saved bytes agree.
        output = args.output or Path("results") /
  (datetime.now(timezone.utc).strftime("%Y%m%dT%H%M%S%fZ") + ".json")
        output.parent.mkdir(parents=True, exist_ok=True)  # Create missing parent directories;
  permission errors remain errors.
        with output.open("xb") as handle:  # Exclusive binary creation prevents silently
  overwriting an earlier report.
            handle.write(payload)
```

```
        sidecar = output.with_suffix(".sha256")  # Replace .json with .sha256, keeping the same
  base name.
        with sidecar.open("x", encoding="utf-8", newline="\n") as handle:  # Exclusive text
  creation also protects an existing sidecar.
            handle.write(hashlib.sha256(payload).hexdigest() + "\n")  # Store a digest only, not
  proof of independent authenticity.
        print("Report:", output)
        print("SHA-256:", sidecar)
        for item in observations:
            print(str(item["port"]) + "/tcp", item["status"])  # Human summary; the JSON retains
  the detailed observations.
        return 0  # Successful collection may include refused, timeout, or error observations.
    except (OSError, ValueError) as exc:  # DNS and file failures produce an operational error; a
  JSON file may remain if sidecar creation fails.
        print("Collection failed:", exc, file=sys.stderr)
        return 1
if __name__ == "__main__":  # Importing the file for tests does not execute a scan.
    raise SystemExit(main())  # Pass the main function's status to the invoking shell.
```

Run and verify the controlled lab

Use two terminals. Start a loopback-only Python HTTP server in one; run the scanner in the other. Stop the listener with Ctrl+C and compare a second fresh observation. Values vary; the offline tests create isolated ephemeral ports.

Execution mode: local-native

```
python -m http.server 8765 --bind 127.0.0.1
# In a second terminal, in the same extracted folder:
python network_triage.py --host 127.0.0.1 --ports 8765,8766 --output results/casebook.json
python -m unittest -v
Report: results/casebook.json
SHA-256: results/casebook.sha256
8765/tcp connected
8766/tcp refused
# Illustrative shape only; exact states depend on your local listeners.
```

Read the evidence with its provenance

The included fixture is a recorded local development capture, not a scan of a public system. Keep requested host, resolved address, environment, UTC observations, durations and OS error numbers together. Download the kit for fixtures and independent hash verification commands.

/downloads/PP-001-1.0.0.zip

Write a defensible finding

Observed fact: A completed TCP connection was reported by this program.

Reasonable inference: A TCP endpoint accepted the handshake from this host at that moment.

Hypothesis: A particular service may be listening, but this tool does not identify it.

Unknown: The application, its authentication state, vulnerability status and clock accuracy.

Limitations: One source host, one resolved IPv4 address, a bounded port list, and environment-dependent timing. A matching SHA-256 establishes byte equality, not independent authenticity or chain of custody.

Independent verification exercise

Change a copy of one JSON report by one byte without changing the original. Recompute its SHA-256 and explain what the mismatch establishes.

Success criteria: The preserved original still matches its sidecar. The changed copy has a different digest. Explain why neither digest proves who created the report.

Worked explanation: Read and hash each file as bytes with hashlib.sha256. Equal digests support byte equality for this comparison. A changed copy should produce a different digest; keep the original unmodified. Origin and custody need independent records and procedures.

Verify your case deliverable

Confirm your JSON parses, independent hashing matches the sidecar, and the test suite passes. Write a short report distinguishing observations from inferences.

Professional fields and portfolio

Network support, network administration, defensive analysis and forensic scripting use reproducible observations and careful reporting. Include source version, test scope, inputs, outputs and limitations in your portfolio. Completing this lesson does not qualify someone for a job. Sourced occupation wage context is linked separately, with broad-proxy caveats.

/careers/

Continue the investigation

No books are available at the moment. The free lesson is complete; future books will add broader systems work only when their finished deliverables exist.

/books/

Your complete project kit

Source, detailed tutorial, tests, local fixtures, README, MIT license and integrity manifest. The source revision demonstrated here matches this release.

/downloads/PP-001-1.0.0.zip

SHA-256: 724904191911217267ee889adebe7b139a10ee0bb7f4504e054d7e332845f017

Primary references

Consult the official references for API contracts and environment-specific setup.

Primary references

- Python socket documentation: https://docs.python.org/3/library/socket.html

- Python hashlib documentation: https://docs.python.org/3/library/hashlib.html

- uv documentation: https://docs.astral.sh/uv/

- Python venv documentation: https://docs.python.org/3/library/venv.html

Recorded test evidence


```
[
  {
    "environment": "linux-wsl",
    "python": "3.14.4",
    "platform": "Linux-6.6.114.1-microsoft-standard-WSL2-x86_64-with-glibc2.43",
    "verified_utc": "2026-11-08T00:32:47.519742+00:00",
    "tests_run": 15,
    "failures": 0,
    "errors": 0,
    "source_sha256": "f7d735935961e2b11d996a1060092ef990f35dc88e4296a7a39db1b0040b4ada",
    "scope": "unittest suite; fresh-machine installers and browser interactions not tested"
  },
  {
    "environment": "windows-native",
    "python": "3.14.4",
    "platform": "Windows-11-10.0.26220-SP0",
    "verified_utc": "2026-11-08T00:32:43.187342+00:00",
    "tests_run": 15,
    "failures": 0,
    "errors": 0,
    "source_sha256": "f7d735935961e2b11d996a1060092ef990f35dc88e4296a7a39db1b0040b4ada",
    "scope": "unittest suite; fresh-machine installers and browser interactions not tested"
  }
]
```