

A Libre Architecture for Verifiable Data Collection and Proof-of-Check Timestamping

A Libre Architecture for Verifiable Data Collection and
Proof-of-Check Timestamping

Cédric Bonhomme, *Computer Incident Response Center*

Luxembourg, cedric.bonhomme@circl.lu

2026

Contents

1 Abstract	2
2 Introduction: The Imperative for Verifiable System Audits	2
3 System Architecture and Core Components	3
3.1 Architectural Overview and Data Flow	3
3.2 Analysis of Core Components	3
3.3 The SPADE Agent-Based Framework	4
4 Implementation Details and Key Mechanisms	6
4.1 Data Integrity via RFC 3161 Timestamping	6
4.2 API Services and Real-time Communication	6
4.3 Standardized Data Formats	7
5 Applications, Use Cases, and Extensibility	8
5.1 Core Use Cases	8
5.2 Extensibility and Integration	9
6 Conclusion	9
7 License	9
References	10

1 Abstract

Establishing trusted, time-stamped records of system states in distributed environments presents a significant challenge for maintaining accountability and security. Organizations often struggle to produce non-repudiable proof that a specific check was performed or that a system was in a particular state at a precise moment in time. SCANDALE is a libre software solution designed to address this challenge by providing a robust backend architecture for collecting data from distributed probes and storing immutable proofs of those checks. Its core components include a high-performance HTTP API with real-time capabilities, an agent-based backend built on the Smart Python Agent Development Environment (SPADE) for scalable probe management, and a dedicated service for cryptographic timestamping in compliance with RFC 3161. The platform's primary value is its capacity to transform abstract operational data into concrete, non-repudiable evidence, providing a verifiable and cryptographically secured audit trail.

2 Introduction: The Imperative for Verifiable System Audits

In modern cybersecurity and IT operations, the ability to produce verifiable data is of strategic importance. Proving that a specific action was taken or that a system was in a particular state at a precise time is critical, especially during incident response, compliance audits, or contractual disputes. Without a mechanism for creating tamper-evident records, organizations are left vulnerable to misrepresentation and lack the concrete evidence needed to enforce accountability.

SCANDALE is a novel, open-source platform engineered to create a verifiable and non-repudiable audit trail. It provides a comprehensive architecture for collecting data from a network of distributed probes and cryptographically timestamping the results to generate an authoritative “proof of check.” This ensures that the existence of a given piece of data at a specific time can be proven without question.

The platform is composed of three primary components:

- A documented HTTP API featuring a Pub/Sub mechanism for real-time data dissemination.
- A backend based on the Smart Python Agent Development Environment (SPADE(Palanca et al. 2020)) for deploying and monitoring a network of probes.
- A service to timestamp collected data with a third party according to RFC 3161 standards(Zuccherato et al. 2001), providing cryptographic proof.

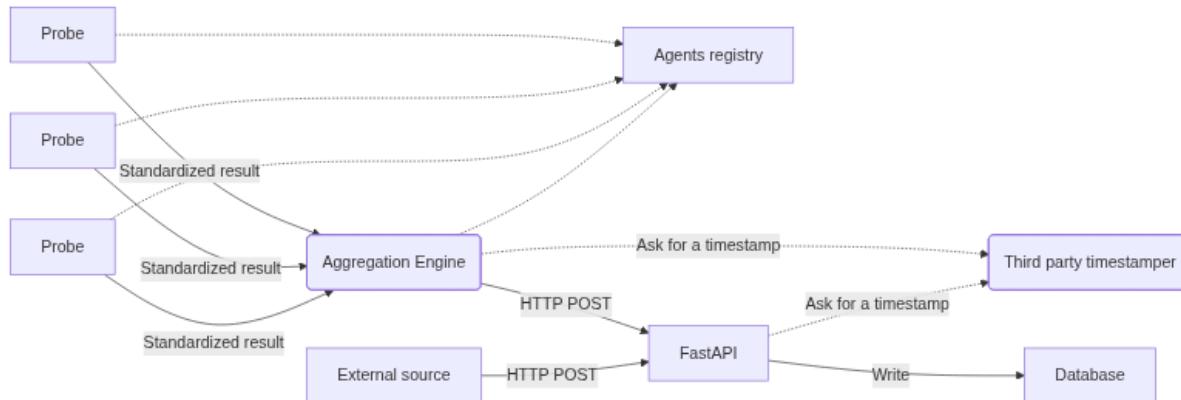
This document provides a detailed examination of the system’s architecture, its core mechanisms, and its practical applications in establishing digital trust.

3 System Architecture and Core Components

The SCANDALE architecture is designed for scalability and reliability, orchestrating data collection, aggregation, and verification through a set of specialized, interoperable components. This modular design ensures that each part of the system can perform its function efficiently while contributing to the overall goal of creating a trusted audit log. This section deconstructs the architecture and clarifies the role of each component.

3.1 Architectural Overview and Data Flow

Reference implementation: <https://github.com/scandale-project/scandale>



The system's data flow provides multiple ingress paths, ensuring flexibility for both agent-based and external data sources. The primary operational flows are as follows:

1. Data Collection: A distributed network of Probes conducts localized scans. These agents execute their tasks and transmit normalized, standardized results to the Aggregation Engine.
2. Aggregation & Timestamping: The Aggregation Engine consolidates data from the probe network. It can then request a cryptographic timestamp from a third-party RFC 3161 service for the consolidated data before forwarding it for storage.
3. Storage & Retrieval: A high-performance FastAPI-based API serves as the primary data interface. It receives data via HTTP POST from the Aggregation Engine or directly from an External source, writes the information to a database, and provides services for retrieval. This API component can also independently request timestamps from the third-party service for data it receives.

3.2 Analysis of Core Components

Probe Agents These agents are the primary data collectors distributed across the monitored environment. They are responsible for executing scans and feeding the results back into the system. Their

two main responsibilities are embedding various scanning tools (probes) and normalizing the output from these tools into a standardized format before transferring the data. Probe agents operate in two distinct modes:

- One-shot: Designed for punctual tasks that are often triggered by a user action. For large-scale jobs, the system can parallelize multiple one-shot agents to handle an extensive list of tasks efficiently.
- Periodic: Configured to execute a specific task at scheduled intervals, enabling continuous and automated monitoring of system states.

Aggregation Engine The Aggregation Engine acts as the central consolidator of data from the entire probe network. Its main responsibility is to collect and centralize data from the various scanning tools. As an additional critical function, this agent is also responsible for managing the Time-Stamp Protocol (TSP) process as defined by RFC 3161. By interacting with a trusted third-party provider (e.g., freetsa.org), it can obtain a cryptographic timestamp for the collected data.

HTTP API The API is the central hub for all data interaction within the SCANDALE platform. It is built on the high-performance FastAPI framework, a choice made to support high-throughput data ingestion and provide asynchronous capabilities essential for real-time services like the Pub/Sub mechanism. The API's key functions include collecting data, verifying the integrity and format of incoming data using Pydantic models, and providing robust services for the storage and retrieval of all checks and their corresponding proofs.

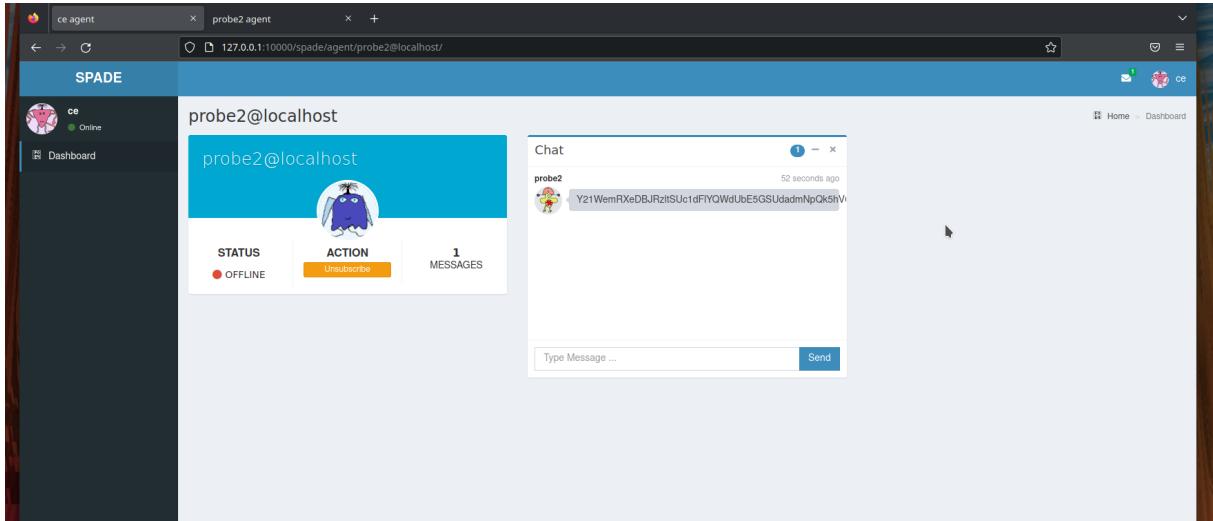
3.3 The SPADE Agent-Based Framework

The entire backend architecture for deploying and managing the probe network relies on the Smart Python Agent Development Environment (SPADE). The strategic decision to build on a mature, agent-based framework offloads the complexity of agent lifecycle management, allowing SCANDALE to focus on its core mission of data collection and verification rather than reinventing foundational distributed systems infrastructure. Within this framework, each agent is an independent entity whose lifecycle is managed by the platform; each agent is authenticated, registered, and declares its availability via a presence notification system. This model provides a flexible and scalable foundation for orchestrating complex data collection tasks across a distributed environment.

Having detailed the architectural components, the following section will explore the specific mechanisms and data formats that enable the system's core functionality.

The image displays three separate browser windows showing the SPADE web interface:

- Top Window:** Shows the 'ce agent' dashboard for 'CyclicBehaviour/CollectingBehav'. It features four cards: 'Mailbox' (2), 'Template' (None), 'Is killed?' (False), and 'Exit Code' (0). Below the cards is a 'Chat' section with messages from 'probe1' and 'probe2'.
- Middle Window:** Shows the 'ce agent' dashboard for 'Messages'. It lists two messages from 'probe1' and 'probe2' with their details (Thread_Id, performative, ontology, language).
- Bottom Window:** Shows the 'ce agent' dashboard for 'probe2 agent'. It displays the 'Behaviours' section (with one entry: 'CyclicBehaviour/CollectingBehav') and the 'Contacts' section (listing 'probe2@localhost' as OFFLINE).



4 Implementation Details and Key Mechanisms

The robustness and utility of the SCANDALE platform are rooted in its specific technical implementations. These mechanisms ensure data integrity, facilitate real-time communication, and maintain interoperability through standardized data formats. This section explores these core technical features.

4.1 Data Integrity via RFC 3161 Timestamping

The cornerstone of SCANDALE's integrity model is its use of a third-party, RFC 3161 compliant timestamping service. The critical benefit of this approach is that the resulting proof is verifiable independently of the SCANDALE system itself. By sending a hash of the collected data to a trusted Time-Stamping Authority (TSA), the system receives a signed token that cryptographically binds the data's hash to a specific point in time. This provides non-repudiable proof of the check's existence, as any third party can validate the signed timestamp against the data without needing to trust the SCANDALE database or its operators.

4.2 API Services and Real-time Communication

The platform's API is built on the FastAPI framework, ensuring high performance and adherence to modern web standards. It is fully compliant with the OpenAPI Specification v3.1.0, making it well-documented and easy to integrate with other tools and services.

A key feature of the API is its Pub/Sub (Publish/Subscribe) mechanism, which allows clients to receive real-time notifications about system events. The following Python code snippet demonstrates a simple client that subscribes to the scan and tst topics to receive event data as it is generated.

```
import asyncio
import os
import sys
from fastapi_websocket_pubsub import PubSubClient

PORT = int(os.environ.get("PORT") or "8000")

async def on_events(data, topic):
    print(f"running callback for {topic}!")
    print(data)

async def main():
    # Create a client and subscribe to topics 'scan' and 'tst'.
    client = PubSubClient(["scan", "tst"], callback=on_events)
    client.start_client(f"ws://localhost:{PORT}/pubsub")
    await client.wait_until_done()

asyncio.run(main())
```

4.3 Standardized Data Formats

To ensure seamless interoperability between components, SCANDALE employs standardized JSON formats, with validation enforced by Pydantic.

The structure for data originating from scans is designed for clarity and completeness. The meta object contains essential context like a UUID, source, and timestamp, while the payload contains the base64-encoded raw output from the scanning tool.

Data from the scans:

```
{
  "version": "1",
  "format": "scanning",
  "meta": {
    "uuid": "<UUID>",
    "source": "<source>",
    "ts": "date",
    "type": "nmap-scan"
  },
}
```

```
"payload": {  
    "raw": "<base64-encoded-string>"  
}  
}
```

The configuration for each probe agent is also defined in a standardized format. This allows administrators to define an agent's behavior, including its execution period, target, and the command it should run. The result_parser field specifies the logic for normalizing scan output, while the up_agent field designates the destination for the processed data, typically the Aggregation Engine.

Agent configuration:

```
{  
    "uuid": "",  
    "period": 3600,  
    "target": "",  
    "command": "",  
    "args": [],  
    "expected_value": "",  
    "result_parser": "",  
    "up_agent": "",  
    "jid": "",  
    "passwd": ""  
}
```

These technical features provide the foundation for SCANDALE's practical application in real-world scenarios requiring verifiable data.

5 Applications, Use Cases, and Extensibility

The architectural and technical features of SCANDALE translate directly into powerful capabilities for solving real-world trust and verification problems across various operational domains.

5.1 Core Use Cases

The platform is designed to address critical needs for accountability and certified evidence. Two primary use cases highlight its value:

- Vendor Accountability: Consider a scenario where a Small and Medium-sized Enterprise (SME) instructs its IT provider to patch a critical system. The provider claims the work was completed at a specific date but has actually failed to do so. By using the certified scan logs generated by

SCANDALE, the SME can produce irrefutable, time-stamped proof of the system's vulnerable state after the claimed patch date, providing concrete grounds to break the contract and hold the provider accountable.

- Incident Timeline Certification: During a security incident, establishing a precise and verifiable timeline of events is crucial for forensic analysis, reporting, and future prevention. SCANDALE can be used to take snapshots of system states at critical moments, creating a cryptographically certified timeline. This timeline is invaluable for investigators and provides a trusted record for regulatory or legal purposes.

5.2 Extensibility and Integration

SCANDALE is not a closed system; it is designed for extensibility at both the agent and platform level. At a high level, an Ad hoc module facilitates integration with external systems, such as sharing data with MISP, a popular open-source threat intelligence platform. This allows certified logs to enrich and be enriched by broader threat intelligence ecosystems. At the component level, each agent has the capability to provide its own HTML views and discrete services, enabling custom functionality and reporting directly from the data collection points.

These applications demonstrate SCANDALE's capacity to serve as a foundational tool for building trust in digital operations.

6 Conclusion

SCANDALE offers a robust, libre software architecture for addressing the critical need for creating immutable, time-stamped proofs of system checks. By leveraging a distributed agent network, it provides a scalable and flexible solution for data collection in complex IT environments. Its core strengths—the use of the SPADE framework for agent management, a high-performance FastAPI interface for data interaction, and the fundamental guarantee of data integrity through RFC 3161 timestamping—combine to create a powerful platform for digital verification. Ultimately, SCANDALE provides a critical capability for any organization needing to enforce accountability and maintain a verifiable audit trail of its digital operations, transforming abstract operational data into concrete, non-repudiable evidence.

7 License

The SCANDALE project is licensed under the GNU Affero General Public License version 3.

- Copyright (C) 2022-2025 Cédric Bonhomme
- Copyright (C) 2022-2025 CIRCL - Computer Incident Response Center Luxembourg

This paper is licensed under the Attribution-ShareAlike 4.0 International license.

References

- Palanca, Javier, Andrés Terrasa, Vicente Julian, and Carlos Carrascosa. 2020. “SPADE 3: Supporting the New Generation of Multi-Agent Systems.” *IEEE Access* 8: 182537–49. <https://doi.org/10.1109/ACCESS.2020.3027357>.
- Zuccherato, Robert, Patrick Cain, Dr. Carlisle Adams, and Denis Pinkas. 2001. “Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP).” Request for Comments. RFC 3161; RFC Editor. <https://doi.org/10.17487/RFC3161>.