RETRACE

Security Whitepaper

Threat model and security architecture for recording, storage,and replay of production execution traces.

Version 1.0

February 2026

Retrace Software Limited

CONFIDENTIAL

Contents

1. Executive Summary

Retrace records live Python program executions in production and replays them deterministically for debugging and analysis. A recording contains **everything**: source code behaviour, user data, secrets in memory, API request and response payloads. This makes the recording the most sensitive artifact in the system.

This document describes the security architecture that protects recordings throughout their lifecycle: from the moment they are captured in production, through encrypted storage, to isolated replay in sandboxed containers. Every design decision follows from one principle:

> *Encrypt early, decrypt late, isolate completely, minimise what leaves the sandbox.*

Key properties of the security model:

- **Encrypted at the boundary:** recordings are encrypted with a per-run public key before leaving the production environment. The production host cannot read its own recordings back.
- **Fully sandboxed replay:** customer code runs in a container with no network, no writable filesystem, no credentials, and no Linux capabilities.
- **Per-session ephemeral keys:** each debugging session receives a scoped decryption key that exists only in agent memory. Compromise is bounded to one recording.
- **Layered escalation:** an attacker must break out of the container, compromise the agent, reach the Retrace Server, and access KMS — each hop requires a separate exploit against a minimal attack surface.
- **Strictly safer than production:** replaying a recording in Retrace gives the code far fewer privileges than it already has in the customer's production environment.

2. Design Principle

The recording contains everything. Source code behaviour, user data, secrets in memory, API responses. Every design decision follows from treating the recording as the most sensitive artifact in the system.

The architecture is built around four constraints:

1. **Encrypt early.** The proxy encrypts with a per-run public key before anything leaves the production environment. Each recording gets its own key pair.
2. **Decrypt late.** Decryption happens only on an ephemeral agent host, only for the duration of a debugging session, only on tmpfs.
3. **Isolate completely.** The replay container has no network, no writable filesystem, no credentials, and no Linux capabilities. The only output is a fixed-size binary pipe.
4. **Minimise what leaves the sandbox.** The pipe carries hashes and instruction counters. No strings. No repr() output. No raw object contents.

3. Trust Boundaries

The system has four layers, each with decreasing trust and access. No layer can communicate with anything other than the layer directly above it.

Layer	Trust Level	Has Access To	Network Access	Lifetime
Retrace Server	Highest	KMS/HSM, S3 URIs, all agents	Full (API, KMS, S3, frontends)	Always running
Retrace Agent	Medium	One decryption key + one recording	Retrace Server + S3 only	One session
Replay Container	None	Decrypted traces (read-only), client code	None	One query

3.1 What Crosses Each Boundary

Boundary	What Crosses It	Direction
Production → S3	Encrypted recording bytes	One way (write only)
Server → Agent	Session assignment (S3 URI + decryption key)	Push
Agent → S3	Fetch encrypted recording	Pull
Agent → Container	Decrypted traces (mounted read-only)	One way
Container → Agent	Fixed-size binary structs (hashes, counters)	Result pipe

4. Encryption

4.1 At Rest

Recordings are encrypted at the proxy with a per-run public key before leaving the production environment. The proxy never has the private key. The production environment cannot read its own recordings back.

Even if the production host is compromised, historical recordings on S3 remain encrypted. The attacker would need the private key from KMS/HSM — a completely separate system with its own access controls.

Encryption uses **asymmetric key encapsulation (RSA/ECIES)** for the per-run key exchange and **AES-256-GCM** for data encryption.

4.2 Per-Run Key Rotation

> *Each recording session generates a fresh public/private key pair. If a private key is ever compromised, the blast radius is exactly one recording — not the entire corpus.*

This is a critical design choice for enterprise deployments. Traditional approaches use a single encryption key for all recordings, which means a key compromise exposes everything. Per-run rotation bounds the damage to the smallest possible unit: one execution, one recording, one key.

4.3 In Transit

- **Production → S3:** TLS (standard S3 upload)
- **S3 → Retrace Agent:** TLS (standard S3 download)
- **Agent → Container:** Local mount (no network, no transit)
- **Container → Agent:** Pipe (local IPC, no network)

No recording data ever traverses a network in plaintext.

4.4 Key Management

Key	Location	Access
Public key (per-run)	Production proxy, customer infrastructure	Widely distributed, not secret
Private key (per-run)	KMS/HSM, accessed by Retrace Server	Never on disk in extractable form
Per-session key	Retrace Agent memory (ephemeral)	One session, then discarded

5. Container Isolation

The replay container is the lowest-trust component in the system. Customer application code runs inside it during replay, so it must be treated as potentially hostile.

5.1 Podman Configuration

All containers run with rootless Podman and the following flags:

Flag	Effect
`--network=none`	No network interfaces (not even loopback DNS)
`--read-only`	Root filesystem is read-only
`--cap-drop=ALL`	No Linux capabilities
`--security-opt=no-new-privileges`	Cannot gain privileges via setuid/setgid

All mounts are read-only. The container cannot write to any filesystem.

5.2 Output Channel

The only path from container to host is the result pipe. The pipe carries a fixed binary protocol with exactly two message types, each 28 bytes:

- **CheckInMsg:** thread_id, global_counter, provenance_hash, creation_map_size
- **HitMsg:** thread_id, creation_hash, access_hash, global_counter

No variable-length data. No strings. No repr() output. No raw object contents. The pipe protocol is defined by the provenance engine (C code) — the client's code has no control over what is written.

5.3 What Is and Is Not Inside the Container

Inside Container	NOT Inside Container	Why Not
Client application code	Private key	Agent decrypts before mounting
Decrypted recording (read-only)	Network access	--network=none
Provenance engine (our code)	Other recordings	Only target recording mounted
Retrace open-source libraries	Agent/S3 credentials	Never injected

6. Replay vs Production: Attack Surface Comparison

Retrace replays customer code — but in a far more restricted environment than the code already runs in production. This is the core security argument: replaying a recording in Retrace is strictly safer than running the code in production, which the customer is already doing.

Capability	Production	Retrace Replay
Network access	Full (serves traffic, calls APIs)	**None**
Filesystem writes	Yes (logs, temp, uploads)	**None (read-only)**
AWS credentials	Yes (IAM role for the app)	**None**
Secrets in environment	Yes (API keys, DB passwords)	**None (only pipe FDs)**
Database / cache access	Yes (live connections)	**None (replayed from recording)**
Can generate new side effects	Yes	**No — all I/O is replay of past events**
Data exfiltration requires	One HTTP request (trivial)	**Kernel exploit to escape container**

> *In production, malicious code has the full privileges of the running application. Exfiltration is trivial: one outbound HTTP request. In Retrace, the only viable attack is a container runtime escape, which requires a kernel vulnerability that bypasses rootless Podman's namespaces, cgroups, and seccomp filters.*

7. Threat Model

7.1 Malicious Client Code

The client's application code runs inside the container during replay. If the code is malicious (or compromised), it could attempt the following attacks:

Attack Vector	Mitigation
Exfiltrate data over network	--network=none — no network interfaces exist
Write data to disk for later retrieval	--read-only + :ro mounts — all writes fail
Escalate privileges	--cap-drop=ALL + --security-opt=no-new-privileges
Read the private key	Key is not in the container — agent decrypts before mounting
Inject data into result pipe	Pipe FD held by provenance engine (C code), not exposed to Python. Fixed-size binary protocol. Agent validates message types
Container escape (kernel exploit)	Podman is rootless by default — container runs as unprivileged user in a user namespace. Requires kernel vulnerability bypassing namespaces, cgroups, and seccomp filters

7.2 Malicious Recording

A crafted recording could attempt to trigger vulnerabilities in the replay layer or provenance engine:

- **Buffer overflow in replay:** The replay layer processes I/O through Python (memory-safe). The C++ provenance engine operates on CPython internal structures, not raw recording bytes.
- **Excessive resource usage:** The Retrace Agent sets container resource limits (memory, CPU, time). CRIU checkpoints provide bounded execution windows.
- **Misleading results:** Provenance hashes are computed deterministically from execution. A recording can only produce the results its execution generates.

7.3 Compromised Retrace Agent

Each agent is session-scoped — one recording, one ephemeral key, for the duration of one debugging session. If compromised, the attacker gains access to:

- One decryption key for one recording (session-scoped, in memory)
- One decrypted recording on tmpfs
- Result pipe contents (hashes and counters)

The attacker cannot: reach any other host (network restricted to Retrace Server + S3), decrypt other recordings (no master key), access the Retrace Server's KMS credentials, or access other customers' data.

7.4 Compromised Retrace Server

The Retrace Server is the highest-trust component. If compromised, the attacker can request decryption keys from KMS for any recording and direct agents to decrypt and replay.

Mitigations:

- KMS audit logs — all key access is logged and alerted on
- Rate limiting on key requests — anomalous decryption volume triggers alerts
- Minimal attack surface — no client code, no recordings on disk
- Per-customer key pairs — limits KMS compromise to one customer
- Multi-party access controls for KMS key policy changes

7.5 S3 Compromise

If the S3 bucket is compromised, the attacker gets encrypted recordings. Without the private key, these are useless.

Additional mitigations: IAM-enforced access matrix (see below), S3 server-side encryption as defence-in-depth, bucket versioning with MFA delete to prevent tampering, and no component can delete — retention managed by S3 lifecycle rules only.

8. S3 Access Matrix

Each component has the minimum S3 permissions required for its role. Enforced via IAM policies per component.

Component	PutObject	GetObject	ListBucket	DeleteObject
Production proxy	**Yes**	No	No	No
Retrace Server	No	**Yes**	**Yes**	No
Retrace Agent	No	**Yes**	No	No
Replay Container	—	—	—	—

Nobody can delete. Recordings are immutable once written. Deletion is handled exclusively by S3 lifecycle rules (time-based expiry). This supports GDPR right-to-deletion through retention policy configuration.

9. Blast Radius Summary

Every compromise scenario is bounded. The system is designed so that each layer's compromise exposes the minimum possible data, and escalation to the next layer requires a separate exploit.

Compromised	What's Exposed	Scope	Can Escalate To
Production host	Future recordings (proxy tampered). No decrypt key.	1 customer	Nothing — no key
S3 bucket	Nothing (encrypted)	None	Nothing
Replay container	One decrypted recording	1 recording	Agent host (sandboxed)
Retrace Agent	One ephemeral key + one recording	1 recording	Retrace Server only
Per-run private key	One recording decryptable	1 recording	Nothing else
Retrace Server	KMS access, all session routing	All recordings	KMS/HSM

Escalation is layered — each compromise only reaches the next layer:

> *Container escape → Agent host (no master key, network-sandboxed) → Retrace Server (the only reachable host) → KMS (requires Server credentials). Each hop requires a separate exploit.*

10. Data Lifecycle

Recording data passes through ten stages. Plaintext exists in only two locations, neither accessible to client code.

1. **Production:** I/O intercepted → encrypted → S3 (plaintext in proxy memory only)
2. **Storage:** Encrypted on S3 (at rest, encrypted)
3. **Session request:** Frontend → Retrace Server (no recording data)
4. **Session assignment:** Retrace Server → Agent (S3 URI + ephemeral key)
5. **Retrieval:** Agent pulls from S3 (in transit, TLS)
6. **Decryption:** Agent decrypts to tmpfs (plaintext on tmpfs)
7. **Replay:** Mounted read-only into isolated container (plaintext inside sandbox)
8. **Results:** Hashes + counters via pipe → Agent (no sensitive data leaves container)
9. **Return:** Agent → Server → Frontend (query results only)
10. **Cleanup:** Agent wipes tmpfs on session end (plaintext destroyed)

> **Plaintext recording exists in only two places:**
> 1. Production proxy memory buffer (milliseconds, before encryption)
> 2. Retrace Agent tmpfs (duration of session, wiped after)

Neither location is accessible to the client's code. The Retrace Server never sees plaintext recordings — it only routes sessions and collects results.

11. Compliance Considerations

11.1 GDPR Right to Deletion

Recordings cannot be surgically edited to remove individual requests from a sequential execution trace. Instead, recordings expire via S3 lifecycle rules (time-based retention policies). Organisations configure retention periods appropriate to their compliance requirements.

11.2 Data Residency

Recordings are stored in S3 and can be configured to use region-specific buckets to meet data residency requirements. The Retrace Server and Agent hosts can similarly be deployed in specific regions.

11.3 Audit Trail

All key distribution from the Retrace Server is logged. KMS/HSM provides its own audit trail for key access. Combined, these logs provide a complete record of who accessed which recording and when.

11.4 Self-Hosted Option

For organisations with strict data sovereignty requirements, Retrace can be deployed entirely within the customer's own infrastructure. In this configuration, the customer controls all components: the recording proxy, S3 storage, KMS, the Retrace Server, and agents. No data leaves the customer's network.

12. Summary

Property	How
Recordings encrypted at rest	Per-run public-key encryption at proxy, before S3
Recordings encrypted in transit	TLS everywhere
Master key isolated	KMS/HSM only, accessed by Retrace Server, never on disk
Per-session keys ephemeral	Scoped key per session, agent discards on session end
Per-run key rotation	Each recording gets its own key pair; compromise = one recording
Agent network-sandboxed	Can only reach Retrace Server + S3
Replay fully sandboxed	No network, no writes, no capabilities, rootless Podman
Minimal output channel	Fixed-size binary pipe protocol — hashes and counters only
Compromise bounded	Each layer's compromise exposes one recording at most
Layered escalation	Container → Agent → Server → KMS — each hop requires separate exploit

Intellectual Property

Retrace's value-level provenance mechanism is protected by granted patents in the UK (GB2593858B), US (US11880279B2), and EU (EP4100831B1).

Retrace Software Limited

For security questions or to request a detailed threat model review, contact:
security@retracesoftware.com