# **Attestium:** A Framework for Verifiable Runtime Integrity

The only 100% open-source framework for verifiable, TPM-backed runtime integrity

Nicholas Baugh, Founder
Forward Email LLC
nick@forwardemail.net
https://forwardemail.net
https://github.com/attestium/attestium

First Published: March 3, 2026
Last Revised: March 3, 2026

## Foreword

Recent high-profile supply chain attacks, from SolarWinds to the xz-utils backdoor, have exposed a fundamental flaw in modern software security: the gap between build-time security and runtime integrity. We have made significant strides in securing the software supply chain with frameworks like SLSA and tools like Sigstore. These are essential, but they only verify the integrity of software *before* it is deployed.

This whitepaper introduces Attestium, an open-source framework we built to bridge this gap. Attestium provides continuous, verifiable runtime integrity for server-side applications by leveraging Trusted Platform Modules (TPMs) and a unique layered architecture. It offers a practical, scalable solution for achieving end-to-end trust, ensuring the code you run is the code you trust.

*Dedicated to the open-source community and the pursuit of transparency.*

## Contents

## 1 Introduction

We built Attestium because we believe trust in software should be earned, not assumed. It is a **runtime code verification and integrity monitoring library** that provides cryptographic proof of your application's code integrity. Like an element in the periodic table, Attestium represents the fundamental building block of **attestation**—the ability to prove that your code is running exactly as intended, without tampering or modification.

The SolarWinds[1] and xz-utils[2] attacks were not isolated failures; they were symptoms of a systemic problem. Our industry has focused heavily on build-time security—verifying code *before* it's deployed. This is essential, but it's only half the story.

What happens after deployment? How can you be certain the code running in your production environment is the same code you so carefully vetted and signed? This is the gap where trust breaks down, and it's the gap we designed Attestium to fill.

For years, we've relied on point-in-time audits. But in a world of continuous deployment, a snapshot in time is not enough. As we've argued before, true security requires continuous assurance[3]. This isn't just a theoretical problem. A 2023 Sonatype report found over 245,000 malicious packages in open-source repositories[4]. The threat is real, and it's growing.

Projects like Mullvad's System Transparency have shown a path forward, proving that verifiable systems are possible[5]. Attestium builds on this foundation, but shifts the focus from the boot process to the runtime environment. We provide a practical, developer-friendly framework for continuously verifying the integrity of running applications.

The urgency of this problem is recognized at the highest levels. Executive Order 14028[6] and the NIST Secure Software Development Framework[7] both call for stronger software supply chain security. Attestium is our answer: a layered, open-source architecture for verifiable runtime integrity. In the following sections, we detail the Attestium stack, demonstrate its use of TPM-based attestation, and outline our roadmap for the future.

[1]CISA Alert (AA20-352A): Advanced Persistent Threat Compromise of Government Agencies: https://www.cisa.gov/news-events/cybersecurity-advisories/aa20-352a
[2]Andres Freund, "Backdoor in xz-utils": https://www.openwall.com/lists/oss-security/2024/03/29/4
[3]Forward Email, "Best Security Audit Companies": https://forwardemail.net/en/blog/docs/best-security-audit-companies
[4]Sonatype, "2023 State of the Software Supply Chain": https://www.sonatype.com/state-of-the-software-supply-chain/introduction
[5]Mullvad VPN, "Introducing System Transparency for our VPN servers": https://mullvad.net/en/blog/diskless-infrastructure-beta-system-transparency-stboot
[6]The White House, "Executive Order 14028: Improving the Nation's Cybersecurity": https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity
[7]NIST SP 800-218, "Secure Software Development Framework (SSDF) Version 1.1": https://csrc.nist.gov/publications/detail/sp/800-218/final

# 2 The Problem

The software supply chain is broken. We have focused intensely on securing software before it's deployed, but we've largely ignored what happens after. This is the critical gap where trust evaporates.

## 2.1 Build-Time Security is Not Enough

We have made great strides with initiatives like SLSA[8] and Sigstore[9], creating a chain of trust from source code to binary. But that chain breaks at runtime. As Google's engineers noted, build-time enforcement alone cannot protect infrastructure from compromised code[10]. A runtime mechanism is not just a nice-to-have; it is a necessity.

## 2.2 Where Existing Solutions Fall Short

Before developing Attestium, we conducted extensive research into existing verification, attestation, and integrity monitoring solutions. Our analysis revealed seven critical gaps that existing solutions couldn't address for our specific requirements:

1. **Runtime Application Verification Gap**: Most solutions focus on build-time, deployment-time, or infrastructure-level verification. They cannot detect runtime tampering or code injection attacks.
2. **Third-Party Verification API Gap**: Existing solutions lack standardized APIs for external verification, making it difficult for auditors to independently verify system integrity.
3. **Developer Experience Gap**: Hardware-based solutions require specialized knowledge and infrastructure, creating a high barrier to adoption for typical web applications.
4. **Node.js Ecosystem Gap**: Most solutions are language-agnostic or focused on other platforms, with poor integration into Node.js applications and workflows.
5. **Cost and Complexity Gap**: Many solutions are expensive or too complex for many applications and organizations.
6. **Granular Monitoring Gap**: File integrity tools monitor files, and application tools monitor performance, but no solution provides granular application code verification.
7. **Continuous Verification Gap**: Most solutions provide point-in-time verification, which cannot detect tampering between verification intervals.

We need a solution that is holistic, layered, and developer-first. This is why we built Attestium.

[8]SLSA, "Supply-chain Levels for Software Artifacts": https://slsa.dev/
[9]Sigstore, "A new standard for signing, verifying, and protecting software": https://www.sigstore.dev/
[10]Google Cloud, "Binary Authorization for Borg": https://cloud.google.com/docs/security/binary-authorization-for-borg

## 3 The Solution

We designed Attestium as a layered, practical, and open-source solution for verifiable runtime integrity. It is not a monolithic system but a modular stack that can be adapted to different environments and needs. Attestium addresses the gaps in existing solutions by providing:

- **Runtime Code Verification**: Continuous monitoring of running application code, real-time detection of unauthorized modifications, and in-memory integrity checking capabilities.
- **Third-Party Verification APIs**: RESTful APIs for external verification, nonce-based challenge-response protocols, and cryptographically signed verification reports.
- **Developer-Friendly Design**: A simple npm package installation, minimal configuration requirements, and seamless integration with existing Node.js applications.
- **Granular File Categorization**: Intelligent categorization of source code, tests, configuration, and dependencies, with customizable include/exclude patterns and Git integration for baseline establishment.
- **Cryptographic Proof Generation**: SHA-256 checksums for all monitored files, signed verification reports, and tamper-evident audit trails.
- **Modern Workflow Integration**: Git commit hash tracking, CI/CD pipeline integration, and Cosmiconfig-based configuration management.

### 3.1 The Attestium Stack

Our architecture consists of three layers:

1. **Attestium (Core Engine)**: A Node.js library that provides the low-level primitives for verification. It interacts with the TPM for hardware-backed attestation and provides APIs for measuring running processes and file integrity.
2. **Audit Status (Monitoring Tool)**: A command-line tool that uses Attestium to perform periodic server checks. It's a single, self-contained binary that can be easily deployed and configured via a simple YAML file.
3. **Upptime (Uptime Monitor)**: We extended this popular open-source uptime monitor with a new `ssh-audit` check. It connects to a remote server, runs Audit Status, and parses the results, enabling continuous, third-party verifiable attestation.

### 3.2 Why GitHub Actions

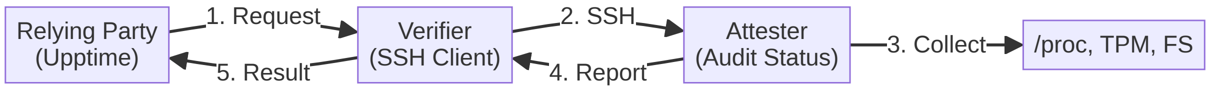
A key design decision was how to orchestrate these checks without introducing a new processor or subprocessor into our data pipeline. We already trust GitHub — our source code lives there, our CI runs there, and our team authenticates through it every day. Adding another third-party service just to run periodic integrity checks would mean onboarding a new vendor, negotiating a new DPA, and expanding our attack surface for no good reason.

That's why we built the orchestration layer on top of GitHub Actions via Upptime. Upptime runs as a scheduled GitHub Actions workflow — no additional infrastructure, no new credentials to manage, and no new trust relationships to establish. The checks run in GitHub's environment, and the results are committed directly to the repository as structured data.

But this approach isn't limited to Upptime or even GitHub. The same pattern works with any task runner or CI/CD system. You could wire up the same `auditstatus check --json` command in a plain GitHub Actions workflow file, a GitLab CI pipeline, a Jenkins job, or even a simple cron job on a bastion host. The Audit Status binary is self-contained and stateless — it takes CLI flags, runs its checks, and outputs JSON. That makes it trivially composable with whatever orchestration you already have in place.

### 3.3 The Verification Flow

Our verification flow follows the IETF RATS architecture (RFC 9334)[11]. Upptime acts as the Relying Party, initiating an attestation request. The ssh-audit helper acts as the Verifier, connecting to the remote server and executing Audit Status, the Attester. Audit Status collects evidence from the system (TPM, /proc, filesystem), generates a report, and sends it back up the chain.



This architecture provides a clean separation of concerns and a secure, flexible flow for remote verification. In the following sections, we will explore each layer in greater detail.

## 4 Systems Architecture

Here, we detail the architecture of each layer in the Attestium stack, from the core library to the uptime monitor integration.

### 4.1 Layer 1: Attestium (Core Verification Engine)

Attestium is the heart of our stack. It's a Node.js library providing the essential tools for runtime integrity verification. Its key functions include:

- **`generateVerificationReport()`**: The main entry point, orchestrating all checks and compiling

[11] H. Birkholz, et al., "Remote Attestation Procedures Architecture," IETF RFC 9334, 2022: https://datatracker.ietf.org/doc/rfc9334/

the final JSON report.

- **`checkCodeIntegrity()`**: Verifies application code against a git commit hash or a recursive checksum of all project files.
- **`checkRunningProcesses()`**: On Linux, this function iterates through `/proc` to identify running processes and hashes their executables from `/proc/<pid>/exe`, ensuring the running code hasn't been tampered with.
- **`checkBinarySignatures()`**: Verifies the integrity of critical system binaries like `node`, `npm`, and `git` by comparing their checksums to known-good values.
- **`checkTpmAttestation()`**: Interacts with the system's TPM using `tpm2-tools` to create a hardware-signed quote, verifying the integrity of the boot process and kernel.

## 4.2 Layer 2: Audit Status (Monitoring Tool)

Audit Status is our command-line interface for Attestium. It's designed for periodic server audits and is distributed as a single executable binary created with Node.js's SEA feature. Its main features are:

- **YAML Configuration**: A simple `auditstatus.config.yml` file allows you to specify which checks to run and their expected outcomes.
- **JSON Output**: It outputs a structured JSON report with a top-level `passed` flag and detailed results for each check, making it easy to integrate with other tools.
- **Simple CLI**: The `auditstatus check` command runs a full system audit.

## 4.3 Layer 3: Upptime (Uptime Monitoring System)

We extended Upptime, an open-source uptime monitor, with a new `ssh-audit` check type for remote runtime integrity verification. The process is straightforward:

1. **Download Binary**: Upptime fetches the specified version of the Audit Status binary from GitHub Releases.
2. **SCP to Server**: It copies the binary to `/dev/shm` on the remote server, an in-memory filesystem that ensures automatic cleanup.
3. **Execute Audit**: It runs the binary via SSH, passing all configuration as command-line flags.
4. **Parse Output**: It captures and parses the JSON output to determine the check's result.
5. **Cleanup**: Finally, it removes the binary from the remote server.

This approach requires no pre-installed software on the remote server besides an SSH server, and all configuration is managed centrally in the `.upptimerc.yml` file.

## 5 Security Architecture

Our security strategy is built on a defense-in-depth approach, combining standard cryptographic primitives, a hardware root of trust, and a clear threat model.

### 5.1 Cryptographic Primitives

We use standard, well-vetted cryptographic tools:

- **SHA-256**: For all file and process hashing.
- **HMAC-SHA256**: For keyed-hash message authentication.
- **AES-256-GCM**: For authenticated encryption of sensitive data at rest.

### 5.2 TPM 2.0 Hardware-Backed Security

The security of any attestation system rests on the trustworthiness of its hardware. We designed Attestium to leverage a Trusted Platform Module (TPM) as its hardware root of trust. The TPM provides a secure environment for cryptographic operations and key storage.

#### 5.2.1 Why TPM 2.0 is Critical

Attestium **requires TPM 2.0** for production deployments where maximum security is needed. While Attestium can operate in software-only mode for development and testing, **TPM 2.0 integration is essential** for addressing the fundamental limitations of software-only verification systems.

When running in software-only mode, Attestium is vulnerable to several sophisticated attack vectors:

- **Runtime Patching**: An attacker with root access can modify the Node.js runtime or kernel to bypass Attestium's checks.
- **Memory Manipulation**: Direct memory access can alter verification logic or cryptographic keys.
- **Filesystem Tampering**: An attacker can modify files on disk and then intercept filesystem calls to return the original content to Attestium.
- **Verification Bypass**: The entire verification process can be mocked or disabled by a sufficiently privileged attacker.

TPM 2.0 provides a hardware root of trust that mitigates these attacks:

- **Hardware-Protected Keys**: Cryptographic keys are stored in the TPM chip and cannot be extracted.
- **Measured Boot**: The TPM measures the entire boot process, creating a cryptographic record of the system state.

- **Sealed Storage**: Data can be encrypted and "sealed" to a specific system state. It can only be unsealed if the system is in the exact same state.
- **Remote Attestation**: The TPM can provide a signed quote of its internal state, allowing a remote party to verify the system's integrity.

### 5.2.2 Attestium's TPM 2.0 Integration

Attestium leverages these TPM 2.0 features to provide a secure verification environment:

- **Key Management**: Attestium's cryptographic keys are generated and stored in the TPM.
- **Integrity Verification**: The TPM is used to verify the integrity of the boot process and the running system.
- **Sealed Data**: Verification baselines are sealed to the TPM, preventing tampering.
- **Hardware Random**: The TPM's hardware random number generator is used for cryptographic operations.

## 5.3 Threat Model

We consider three primary adversaries:

1. **External Attacker**: An attacker who has gained unauthorized access to the server (e.g., through a remote code execution vulnerability). They may be able to tamper with files but cannot compromise the TPM or Attestium itself.
2. **Insider Threat**: A user with legitimate server access who may tamper with files or modify the Attestium configuration, but cannot compromise the TPM.
3. **Supply Chain Compromise**: An attacker who has compromised an upstream dependency (e.g., a malicious npm package). They may be able to inject malicious code into the application but cannot compromise the underlying OS or TPM.

Attestium is designed to detect and mitigate all three threats. File and process integrity checks detect tampering by external or internal actors. TPM-based attestation detects a compromised OS or boot process. Our layered architecture, with its clear separation of concerns, helps mitigate the risk of a supply chain compromise.

## 6 Our Security Background

Before we built Attestium, we spent years hardening our own infrastructure at Forward Email. Security isn't a feature we bolted on — it's the foundation we built everything on top of. Our production environment is managed entirely through open-source Ansible playbooks that anyone can inspect. This section documents what we've done, why we built Attestium instead of hiring a third-party auditor, and why we think continuous automated verification beats a point-in-time audit.

## 6.1 Ansible-Managed Infrastructure

Every server we operate is provisioned and hardened through Ansible. There are no manual SSH sessions to configure things. No snowflake servers. Every security measure is codified, version-controlled, and reproducible. Here's what our playbooks enforce.

### 6.1.1 Kernel and System Hardening

We disable core dumps entirely — hard and soft limits set to zero, `fs.suid_dumpable=0`, `ProcessSizeMax=0` in systemd, and `kernel.core_pattern` piped to `/bin/false`. Transparent Huge Pages are disabled via a systemd service. Swap is turned off on all non-database servers (database servers keep it with `vm.swappiness=1`). We run over 50 sysctl kernel parameters including ASLR (`kernel.randomize_va_space=2`), TCP SYN cookies for flood protection, RFC 1337 TIME-WAIT assassination protection, and dynamically scaled TCP buffers based on available RAM.

Our filesystem mounts use `tmpfs` on `/dev/shm` with `noexec,nosuid,nodev` flags — code cannot execute from shared memory. Data partitions use `noatime` and `nodiratime`. I/O schedulers are tuned per drive type: `none` for NVMe, `deadline` for SSDs, with optimized read-ahead values. Web-facing servers use TCP BBR congestion control with `fq` queueing; internal servers use CUBIC with `fq_codel`.

### 6.1.2 USB Device Whitelisting

The `usb-storage` kernel module is disabled via `modprobe.d` and the initramfs is rebuilt to persist this across reboots. We maintain a whitelist of authorized USB devices by `vendor:product` ID in `/etc/security-monitor/authorized-usb-devices.conf`. Any unrecognized USB device triggers an immediate email alert to the team. Udev rules (`99-usb-monitor.rules`) provide real-time detection on top of the 5-minute polling cycle. A datacenter technician plugging in a USB drive gets flagged instantly.

### 6.1.3 SSH and Access Control

Root login is disabled (`PermitRootLogin no`). Password authentication is disabled — only key-based auth is allowed. The root password is locked. We maintain two users: `devops` (with sudo, no password prompt) and `deploy` (with a 4096-bit SSH key, limited privileges). Fail2Ban is configured aggressively: 2 failed attempts triggers a permanent ban (`bantime=-1`) with a 365-day find window. We replaced Postfix with `msmtp` — a lightweight send-only SMTP client with no listening daemon and no open ports.

### 6.1.4 Eight Security Monitoring Systems

We run eight independent monitoring systems, all implemented as systemd timers with rate-limited email alerts:

1. **System Resource Monitor** — CPU, memory, and disk usage with five threshold levels (75%, 80%, 90%, 95%, 100%), checked every 5 minutes.
2. **SSH Security Monitor** — Failed logins, successful logins, root access, unknown IPs, and after-hours logins, checked every 10 minutes.
3. **USB Device Monitor** — Unknown device detection with vendor:product ID whitelisting, checked every 5 minutes plus real-time udev rules.
4. **Root Access Monitor** — Direct root login, sudo usage, `su` to root, and privilege escalation attempts, checked every 5 minutes.
5. **Lynis Audit Monitor** — Automated Lynis security audits run on a schedule.
6. **Package Monitor** — Tracks every package installation and removal.
7. **Open Ports Monitor** — Detects unexpected listening services.
8. **SSL Certificate Monitor** — Monitors TLS certificate expiry dates.

Each monitor uses whitelist files in `/etc/security-monitor/` for authorized IPs, users, USB devices, root users, and sudo users. Rate limiting prevents alert flooding — resource alerts have a 1-hour cooldown per threshold, SSH root access alerts have no cooldown (always alert), and USB alerts have a 1-hour cooldown per device.

### 6.1.5 Command Logging and Auditing

Every command executed on our servers is logged through multiple layers: `auditd` with custom audit rules, enhanced bash logging in both `/etc/profile.d/` (login shells) and `/etc/bash.bashrc` (all interactive shells), zsh logging in `/etc/zsh/zshrc.d/`, and `rsyslog` capturing everything to `/var/log/bash-commands.log` with 30-day logrotate retention. If someone runs a command on our servers, we have a record of it.

### 6.1.6 Automatic Security Updates

Unattended upgrades are enabled for security patches with automatic reboots at 02:00 (except database servers). We deploy a custom port scan protection script (our own maintained fork). DNS resolves through Cloudflare and Google with a local Unbound caching resolver. MongoDB is installed from the official repository, Valkey is compiled from source — we have zero Ansible Galaxy dependencies. No external roles, no third-party playbook code.

## 6.2 Why Not a Third-Party Audit?

A one-time security audit from a reputable firm costs $5,000–$10,000 USD or more. That buys you a snapshot — a report that says "on this date, we checked these things and they looked fine." The moment the audit ends, the report starts going stale. New code gets deployed, packages get updated, configurations change. The audit doesn't tell you what happened last Tuesday at 3 AM.

But the cost isn't the real problem. The real problem is trust.

We manage email. Email is the most sensitive communication channel most people have — password resets, financial statements, legal correspondence, medical records. Giving a third-party auditor SSH access to our production servers means trusting them with access to all of that. Even with the best intentions, an audit can take weeks or months. During that entire window, we'd need to monitor their access, verify they aren't exfiltrating data, and hope that their own systems haven't been compromised. We'd essentially need to audit the auditor.

We do plan to eventually undergo a third-party audit from one of our recommended providers. But we believe continuous, automated, cryptographic verification is strictly better than periodic human inspection. Attestium runs every few minutes. It doesn't get tired, it doesn't have a bad day, and it doesn't need SSH access to your production email servers.

## 6.3 From Hardening to Verification

All of the hardening described above — the sysctl parameters, the USB whitelisting, the eight monitoring systems, the command logging — these are preventive controls. They make it harder for an attacker to do damage. But they don't answer the fundamental question: is the code running on this server right now the same code that's in our public repository?

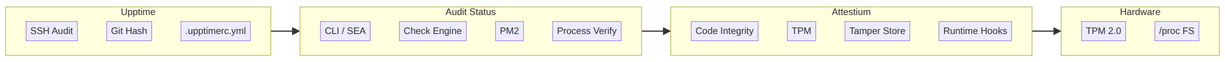
That's the gap Attestium fills. We built it because we needed it ourselves. Every measure in our Ansible playbooks is designed to prevent unauthorized changes. Attestium is designed to detect them — continuously, cryptographically, and in a way that anyone can verify.

# 7 The Attestium Stack in Context

We didn't build Attestium in a vacuum. It stands on the shoulders of giants in system transparency, runtime attestation, and supply chain security. Here, we discuss how Attestium complements and builds upon this existing work.

## 7.1 How We Compare

| Project | Focus | How We Complement It |
|---|---|---|
| **System Transparency** | Boot-time integrity | We extend ST's principles to the runtime environment, providing continuous verification of running applications. |
| **Keylime** | Cloud infrastructure attestation | We offer a more application-focused, developer-friendly approach, particularly for Node.js environments. |
| **Sigstore** | Software artifact signing | We provide the post-deployment verification that ensures the code running in production is the same code Sigstore signed at build time. |
| **SLSA** | Supply chain security framework | We provide a mechanism to enforce SLSA principles at runtime, ensuring only authorized code is running. |
| **Reproducible Builds** | Verifiable software builds | We can verify that the code running in production matches a specific reproducible build, closing the loop between build and runtime. |



## 7.2 Building on Existing Work

**System Transparency (ST)**[12] pioneered verifiable infrastructure for VPN servers. We take their boot-time integrity model and apply it to the application runtime, extending the chain of trust to the code you actually run.

**Keylime**[13] offers powerful TPM-based attestation for cloud infrastructure. We provide a more lightweight, application-centric alternative, making TPM-based attestation more accessible for Node.js developers.

**Sigstore**[14] and **SLSA**[15] are essential for securing the software supply chain at build time. We

[12] Mullvad VPN, "Introducing System Transparency for our VPN servers": https://mullvad.net/en/blog/diskless-infrastructure-beta-system-transparency-stboot
[13] Keylime, "A CNCF project for TPM-based cloud attestation": https://keylime.dev/
[14] Sigstore, "A new standard for signing, verifying, and protecting software": https://www.sigstore.dev/
[15] SLSA, "Supply-chain Levels for Software Artifacts": https://slsa.dev/

complete the picture by providing the post-deployment verification they lack, creating an end-to-end chain of trust from developer to production.

**Reproducible Builds**[16] ensure that a given set of source code always produces the same binary. This is a vital step, but as Lamb and Zacchiroli note, it is a necessary but not sufficient condition for a trustworthy supply chain[17]. Attestium closes the loop by verifying that the code running in production matches a specific reproducible build.

## 8 Future Roadmap

We are just getting started. Here is a look at what we have planned for the future of Attestium.

### 8.1 Enhanced Runtime Analysis

We plan to deepen our runtime analysis capabilities:

- **Memory Analysis**: We will add the ability to perform in-memory analysis of running processes to detect fileless malware and code injection attacks.
- **Behavioral Analysis**: We will add support for behavioral analysis to detect anomalous activity, such as unexpected network connections or file access patterns.
- **eBPF Integration**: We are exploring eBPF to provide a more efficient and powerful mechanism for runtime monitoring with minimal overhead.

### 8.2 Broader Platform Support

Our goal is to make Attestium a cross-platform solution. We are currently focused on Linux and Node.js, but we plan to add support for other operating systems and programming languages like Python, Go, and Rust.

### 8.3 Cloud Provider Integration

We are exploring integrations with cloud providers to enhance security in cloud environments:

- **AWS Nitro Enclaves**: We plan to use Nitro Enclaves to create a secure and isolated environment for the Attestium verifier, enabling remote attestation of EC2 instances without trusting the underlying hypervisor.
- **Google Cloud and Azure Confidential Computing**: We are also exploring similar integrations with Google Cloud and Azure's confidential computing offerings.

[16]Reproducible Builds, "Increasing the integrity of software supply chains": https://reproducible-builds.org/
[17]D. E. Lamb and S. Zacchiroli, "The Trustworthy Software Supply Chain," IEEE Software, 2021: https://ieeexplore.ieee.org/document/9423215

### 8.4 Community and Collaboration

Attestium is an open-source project, and we are committed to building a strong community around it. We welcome contributions and are eager to collaborate with other projects in the software supply chain security space, including Sigstore, SLSA, and Keylime. We believe that by working together, we can build a more secure and trustworthy digital ecosystem.

## 9 Adoption

We built Attestium not just for ourselves, but for the entire open-source community. We believe that verifiable runtime integrity is a fundamental building block for a more secure and trustworthy internet. This section outlines our own adoption, our vision for community adoption, and the critical security problems this framework solves for open-source companies.

### 9.1 Live Transparency at Forward Email

We practice what we preach. At Forward Email, we use the full Attestium, Audit Status, and Upptime stack to continuously monitor the integrity of our production servers. We have made our real-time audit results public for anyone to inspect at any time:

**https://status.forwardemail.net**

This status page is not just a simple uptime monitor; it is a live feed of our server integrity checks. When Upptime runs an `ssh-audit` check, it executes the Audit Status binary on our servers, which in turn uses Attestium to perform TPM-backed cryptographic verification of our entire runtime environment. The results are pushed to our public status page, providing a transparent, third-party verifiable record of our production state.

If a check fails—whether due to a file mismatch, an unexpected process, or a TPM attestation failure—our team is immediately alerted via text messages and other notifications, allowing us to investigate and respond within minutes. This is not a theoretical exercise; it is a live, production-grade security system that we rely on every day.

### 9.2 A Call to the Open-Source Community

We strongly encourage other open-source companies to adopt this framework. The modern software landscape is built on trust, but that trust is increasingly under attack. High-profile supply chain attacks have demonstrated that build-time security is not enough. We need to be able to verify the integrity of our software *as it runs.*

This is especially critical for companies that are building the next generation of open-source infrastructure and services. We have identified several companies that we believe would be ideal candidates for adopting Attestium, as their business models are built on providing trustworthy, transparent, and secure services. For each, we have included their website, primary GitHub repository, and the main programming languages they use:

- **Supabase** (supabase.com) - github.com/supabase/supabase (TypeScript, Go)
- **Cal.com** (cal.com) - github.com/calcom/cal.com (TypeScript)
- **Documenso** (documenso.com) - github.com/documenso/documenso (TypeScript)
- **Bitwarden** (bitwarden.com) - github.com/bitwarden/clients (TypeScript)
- **Infisical** (infisical.com) - github.com/Infisical/infisical (TypeScript)
- **Jitsi** (jitsi.org) - github.com/jitsi/jitsi-meet (TypeScript, JavaScript)
- **Element** (element.io) - github.com/element-hq/element-web (TypeScript, CSS)
- **Mattermost** (mattermost.com) - github.com/mattermost/mattermost (Go, TypeScript)
- **Ghost** (ghost.org) - github.com/TryGhost/Ghost (JavaScript, TypeScript)
- **Plausible Analytics** (plausible.io) - github.com/plausible/analytics (Elixir, React)
- **PostHog** (posthog.com) - github.com/PostHog/posthog (Python, TypeScript)
- **Chatwoot** (chatwoot.com) - github.com/chatwoot/chatwoot (Ruby, Vue, JavaScript)
- **Twenty** (twenty.com) - github.com/twentyhq/twenty (TypeScript)
- **Gitea** (gitea.io) - github.com/go-gitea/gitea (Go, TypeScript)
- **Rocket.Chat** (rocket.chat) - github.com/RocketChat/Rocket.Chat (TypeScript)
- **Plane** (plane.so) - github.com/makeplane/plane (TypeScript, Python)

By adopting a framework like Attestium, these companies can provide a new level of assurance to their users, customers, and enterprise clients. They can prove, with cryptographic certainty, that the code running on their servers is the exact same code that is in their public repositories—unmodified and uncompromised.

## 9.3 Preventing the Insider Threat

While external threats get the most attention, the insider threat remains one of the most difficult to mitigate. A rogue employee, a compromised third-party vendor, or even a datacenter technician with physical access to a server (an "evil-maid" attack) can bypass traditional security measures and inject malicious code directly into a running application.

This is not a theoretical risk. With SSH access, a malicious actor can easily modify application files, install backdoors, or alter dependencies. Attestium is designed to defeat this entire class of attacks. Because it hashes the entire project directory and verifies the integrity of running processes against their on-disk binaries, any unauthorized modification will be immediately detected and flagged.

This provides a powerful layer of defense, not just for end-users, but for the entire team. It ensures that even with privileged access, no single individual can compromise the integrity of the production environment without being detected. It builds a culture of trust and accountability, backed by cryptographic proof.

## 10 Conclusion

The software supply chain is at a crossroads. We can no longer afford to trust blindly. We must move to a new paradigm of continuous verification and zero-trust.

Attestium is our contribution to this new paradigm. It is a practical, scalable, and open-source solution for verifiable runtime integrity. It bridges the critical gap between build-time security and runtime integrity, providing a crucial missing piece in the software supply chain security puzzle.

We believe the future of software security is layered and holistic, combining build-time checks, runtime verification, and a hardware root of trust. Attestium is a significant step in this direction, but it is only the beginning. We are committed to working with the broader community to build a more secure and trustworthy digital ecosystem for everyone.

The road ahead is long, but we are confident that by working together, we can build a future where software is more secure, transparent, and trustworthy. We invite you to join us.