

PRE-FAILURE RESEARCH GROUP LLC

Jason McCoy

Founder · Albia, Iowa

A complete introduction: cover letters, biography, research pitches, published research gifts, and technical work — all applying the same pre-failure methodology.

"Don't trust me or anyone else; trust the best truth available."

— Jason McCoy

CONTENTS

- I. Cover Letter — Government (DoD / DARPA / NSA)
- II. Cover Letter — Technology (OpenAI / Anthropic / Google / Microsoft / Palantir)
- III. Biography
- IV. Pitch — Government
- V. Pitch — Technology
- VI. Research Gift: 2038 Problem — Time Proxy Architecture
- VII. Research Gift: Relationship Genealogy & Pre-Failure Thinking
- VIII. Technical Appendix: Safe Time Bridge v0.4

pre.failure.research.group@gmail.com · pfrg.org · SAM.gov UEI: H8D4YRDMXA16 · CAGE: 25PN6

Pre-Failure Research Group LLC · Albia, Iowa · Iowa LLC No. 878368

I

Cover Letter — Government

DoD / DARPA / NSA Research Leadership

Jason McCoy

Founder, Pre-Failure Research Group LLC · Albia, Iowa · pre.failure.research.group@gmail.com · pfrg.org

October 2026

To: DoD / DARPA / NSA Research Leadership

Subject: Pre-failure researcher seeking senior hands-on role — attached biography and pitch

I am writing to introduce myself and my work directly.

I find failures before they happen. Not as a framework I learned — as a cognitive default I have operated from for decades. I reason forward through system architecture until I locate the point where trust breaks, authority conflicts, or evidence disappears. The attached biography explains how that developed. The attached pitch explains what I am looking for.

What I want is a senior hands-on research role — not administrative, not advisory-only. I want to be in the architecture, identifying failure modes before they get built in.

I have spent the last several years building governance integrity and pre-failure detection architecture as a solo founder, without institutional support. I submitted a concept white paper to the Army xTech | Search 10 competition addressing C2 integrity in contested environments. I developed and published the Agency Contract Framework — a governance structure defining what AI systems operating under federal authority are and are not permitted to do. That work is available at pfrg.org under CC BY-SA 4.0.

I am not the typical applicant. I do not come from academia or a defense contractor. I come from four decades of hands-on computing, forensics, and systems work at every layer — and a mind that has been looking for the failure condition first since before I had words for it.

The biography and pitch follow. I am happy to discuss the work directly.

Respectfully,

Jason McCoy

Founder, Pre-Failure Research Group LLC

pre.failure.research.group@gmail.com · pfrg.org · SAM.gov UEI: H8D4YRDMXA16 · CAGE: 25PN6

Pre-Failure Research Group LLC · Albia, Iowa · Iowa LLC No. 878368

II

Cover Letter — Technology

OpenAI / Anthropic / Google / Microsoft / Palantir

Jason McCoy

Founder, Pre-Failure Research Group LLC · Albion, Iowa · pre.failure.research.group@gmail.com · pfrg.org

October 2026

To: Research Leadership — OpenAI / Anthropic / Google / Microsoft / Palantir

Subject: Independent pre-failure researcher — partnership inquiry, attached biography and pitch

I am writing to introduce myself and my work directly.

I find failures before they happen — in systems, architectures, and governance structures — before they are built in. The attached biography explains how that orientation developed. The attached pitch explains what I am proposing.

I am not looking for employment. I am looking for a research partnership. Pre-Failure Research Group LLC brings independent pre-failure analysis, governance architecture review, and adversarial integrity testing to organizations building systems that require people to trust them. That is structurally difficult work to do on your own systems — which is the point.

I have spent the last several years building a governance integrity and pre-failure detection platform as a solo founder. I developed the Agency Contract Framework, published at pfrg.org under CC BY-SA 4.0. I have been informally red-teaming AI systems for over a decade. I hold an active SAM.gov registration (UEI: H8D4YRDMXA16, CAGE: 25PN6).

The harder problem — the one that will define the next decade — is not building systems people can use. It is building systems that can be verified. I have been working on that problem, independently, from first principles. The biography and pitch follow.

I am happy to discuss the work directly.

Respectfully,

Jason McCoy

Founder, Pre-Failure Research Group LLC

pre.failure.research.group@gmail.com · pfrg.org · SAM.gov UEI: H8D4YRDMXA16 · CAGE: 25PN6

Pre-Failure Research Group LLC · Albia, Iowa · Iowa LLC No. 878368

III

Biography

Jason McCoy

Founder, Pre-Failure Research Group LLC

Jason McCoy has spent much of his life asking a deceptively simple question:

How does this work?

For him, the question rarely ends with the first answer. McCoy is autistic with ADHD — identified as an adult, though the pattern was present from the start. What that has meant, in practice, is a mind that does not stop at the surface of a problem. If a computer stores information as ones and zeros, what causes the one or zero? If a document establishes a rule, who authorized the document? If someone possesses authority, what process granted it — and what gave that process its authority?

That habit of following a question beneath its first apparent answer has carried McCoy through more than four decades of work, research, technology, religion, institutional systems, and personal experience. It eventually became central to the work he does today through Pre-Failure Research Group LLC: examining the distance between what systems say they do and what can actually be observed.

Learning by Taking Things Apart

McCoy's interest in computing began around age six. His stepfather, Richard Klein, brought home an older IBM computer acquired from Drake University in Des Moines, Iowa. He was essentially told to have fun with it. He did.

Using BASIC, one of the first things that fascinated him was the simple fact that he could write something, save it, turn back to it later, and find that the information was still there. He became interested not only in what computers could do, but in how information was represented, stored, transmitted, compressed and compiled.

Electronics received much the same treatment. McCoy remembers repeatedly dismantling a 12-inch color television down to every component he could remove and then putting it back together — roughly four times, without a service manual. The television continued working. Around age thirteen, learning about frequency, radio waves and the ionosphere finally expanded the problem enough to make sense. Almost

immediately, another question appeared: *What creates the radio wave?*

Around 1987, he converted a remote-controlled car into a remote-controlled serving table using construction insulation foam — cutting recesses for a plate and cup so they would remain stable while moving. It was useful. His parents called him lazy. Both descriptions may have been accurate.

Rules, Authority and Questions

McCoy's questions were never limited to machines. As a child, when confronted with a rule, he wanted to know why the rule existed and who had given the person enforcing it the authority to do so.

Authority, in McCoy's view, cannot be established merely because someone asserts it. He follows the chain backward: authenticate the documentation, identify its source, establish the authority of that source, examine the process that granted the authority, and then ask what authorized that process.

Between approximately six and twelve years old, despite growing up in a family that was not particularly religious, McCoy independently attended church after church. At seventeen, a minister told him that God did not need or want people like him. The encounter caused him to recognize a question he had not been asking: *What is the purpose of religion?* One of the most consequential conclusions was that he did not have to perform. Removing the requirement to perform changed everything.

Forty-Plus Jobs, One Recurring Pattern

McCoy estimates that he has held more than forty jobs spanning multiple domains. Those environments provided samples.

At seventeen, he applied to Bridgestone Firestone during a labor strike. During the hiring process, applicants were given a physical problem: transport a raw egg through a conveyor-based process involving a drop without breaking it. McCoy solved it within minutes — treating the egg and its protective foam material as one controlled object. Company representatives hired him.

He later worked in manufacturing environments where engineers sought his assistance with machinery problems. His work history has also included painting, warehousing and management, heavy-equipment training, workplace safety, and wildland firefighting — including Type 2 firefighter training through Mingo Job Corps, during which a rapid wind change required deployment of his fire shelter. What stayed with him was not heroism but respect for how quickly physical conditions can invalidate assumptions.

Asked which of more than forty jobs changed his understanding of systems, McCoy rejects the premise. The jobs didn't create the understanding. They showed him the pattern.

The Failure

Asked to name that pattern in its simplest form, McCoy uses two words: *The failure.*

More specifically, he notices the point at which observed behavior stops matching documented behavior. That mismatch does not automatically prove the documentation is wrong. Nor does it prove the observer is

right. It is the point where investigation begins.

McCoy did not arrive one day at the idea that documented constraints should be compared against observed behavior. Long before he had software intended to perform such analysis, he had been doing a human version of that comparison across unrelated environments.

Receipts

There is an unusual difficulty in explaining how McCoy reaches conclusions. He often cannot show someone the internal work. He can show them the receipt.

Patterns, relationships, inconsistencies and prior observations can apparently converge for him without producing a clean verbal transcript of every intermediate step. To another person, that can look like jumping from A directly to Z.

That has caused misunderstanding throughout his life. It has also contributed to his insistence on evidence. When sound evidence establishes something, McCoy accepts it even when he dislikes the answer. Contradictory evidence is not something he believes should simply disappear.

Sue

Sue, McCoy's late wife, was his best friend. Their relationship occupied roughly twelve years of his life. When she struggled to understand how to communicate with him around his mental-health difficulties, she went to the public library, checked out books, and learned.

She did not demand perfection from him. What she wanted was simple: that he try to be a little better than he had been the day before. She showed up. He remembers ordinary moments, including things she found hilarious that he did not. Her happiness was enough to make him happy anyway.

"I will continue to love that woman for the rest of this life," he says.

Pre-Failure

Today, McCoy is the founder of Pre-Failure Research Group LLC. The name is not a brand decision. It describes a cognitive default he has had since childhood — the habit of reasoning forward through a system until he finds where it will break, before anyone finds out the hard way.

PFRG's work spans governance, technology, evidence preservation, verification, and institutional accountability. Not every idea it produces is a finished technology. Concepts, hypotheses, prototypes, implemented systems, and verified findings are not interchangeable categories. He labels them accordingly.

One project, OmniSniffer, is a prototype aimed at detecting divergence between a system's documented constraints and its observed behavior — and preserving evidence of how that determination was reached.

Much of PFRG's research is openly licensed. McCoy has increasingly favored terms that allow others to use, challenge, and improve the work — while requiring that adaptations remain available to others in turn. That

is the same logic that runs through everything else: no single observer holds the whole picture. Understanding grows through what each perspective contributes.

Pattern finds the question.

Failure tests the answer.

Evidence decides what survives.

Truth remembers the path.

What he is building now is an attempt to formalize something he has been approaching from different directions for most of his life: that systems can be made to tell the truth about themselves — not by trusting the people who built them, but by requiring that the evidence exist and that anyone can check it.

It still starts with the same question.

How does this work?

And when the answer arrives: Okay. But how does that work?

pre.failure.research.group@gmail.com · pfrg.org · SAM.gov UEI: H8D4YRDMXA16 · CAGE: 25PN6

Pre-Failure Research Group LLC · Albia, Iowa · Iowa LLC No. 878368

IV

Pitch — Government

For: DoD / DARPA / NSA

I find failures before they happen.

Jason McCoy · Founder, Pre-Failure Research Group LLC · Albia, Iowa

Most people find failures after they happen. Some find them while they're happening. I find them before — by reasoning forward through system architecture until I locate the point where trust breaks, authority conflicts, or evidence disappears. That isn't a methodology I learned. It's how my brain works.

What I Bring

I am a systems thinker with forensic instincts and four decades of hands-on computing experience. I don't come from academia. I come from building, breaking, and diagnosing systems at every layer — from physical infrastructure to network security to AI governance architecture.

- Independently developed a full governance integrity and pre-failure detection platform — architecture, components, data structures, and verification logic — as a solo founder, holding the complete system in working memory.
- Designed and submitted a novel cryptographic identity architecture to the U.S. Army xTech | Search 10 competition (ASA(ALT)), combining mobile driver's licenses, zero-knowledge proofs, and zero-trust access for contested environments.
- Developed the Agency Contract Framework — a published governance structure defining what AI systems operating under federal authority are permitted to do, what they are forbidden from doing, and how those boundaries are enforced and audited. Published at pfrg.org under CC BY-SA 4.0.
- Built a government-restricted AI system concept for authorized, auditable data destruction — constraints-first architecture, bounded claims, and independent verification — from first principles, without institutional support.
- Over a decade of informal AI red-teaming and beta testing, identifying failure modes and edge cases before they become public incidents.
- Background in digital forensics and network security, combined with deep trades expertise — a diagnostic mindset that operates at both the theoretical and physical layer.

"Don't trust me or anyone else; trust the best truth available."

— Jason McCoy

That motto isn't humility theater. It is the operating principle behind everything I build: no system should require trusting me, and no system I design will ask you to. Every claim gets receipts. Every authority has limits. Every failure mode gets documented before the capability is built.

What I'm Asking For

A senior hands-on research role. Not administrative. Not advisory-only. I want to be in the architecture, identifying failure modes before they get built in.

What You Get

A researcher who reasons forward through system failure — not backward from incident reports. The pre-failure orientation isn't a framework I apply. It's a cognitive default.

- Governance architecture under real constraints. Define what a system must never do, identify every authority boundary, document every failure case — before capability is written. This is not a best practice. It is a foundational design doctrine.
- AI integrity and verification. Systems that produce tamper-evident, auditable records of their own behavior — so no one, including the creator, can claim authority that cannot be verified.
- Adversarial thinking without institutional bias. I look for where the architecture breaks, where an adversary exploits a gap, where trust is assumed rather than earned. I have no stake in any particular system surviving that scrutiny.
- Cross-domain pattern recognition. Forty years across every layer — hardware, network, software, forensics, AI — means I recognize structural failure patterns that domain specialists miss.

Current Work Relevant to Your Mission

I submitted a concept white paper to the U.S. Army xTech | Search 10 competition addressing C2 and Counter-C2 network integrity in contested environments — combining mobile identity verification, zero-knowledge proofs, and zero-trust architecture with a satellite emergency messaging capability for fully degraded conditions.

I also developed the Agency Contract Framework — a governance structure defining what AI systems operating under federal authority are permitted to do, what they are forbidden from doing, and how those boundaries are enforced and audited. It addresses a direct gap in current autonomous systems oversight: the absence of a binding, verifiable contract between an AI system and the agency deploying it.

PFRG holds an active SAM.gov registration (UEI: H8D4YRDMXA16, CAGE: 25PN6), making it eligible for direct contracting mechanisms including SBIR/STTR and Other Transaction Authority (OTA) agreements — so a scoped evaluation can move through existing channels without requiring a new procurement structure.

Public Research Library

The full body of PFRG's published research is available at pfrg.org. All of it applies the same pre-failure methodology: identify the structural failure condition before building the remedy.

- Government accountability and IRS enforcement asymmetry — baseline and findings published
- Tax policy and inflation interaction — independent research with baseline and findings
- Agency Contract Framework — AI governance structure for federal deployment; CC BY-SA 4.0
- AI accountability framework — structural constraints for AI governance
- Age verification framework — privacy-preserving verification architecture
- Truth Memory Vaults — physical air-gapped repository concept for an ungated Universal Commons
- A New Foundation — governance framework built on an auditable harm-based model
- QR Code Scanner (open source) — cross-platform optical data transfer utility

pre.failure.research.group@gmail.com · pfrg.org · SAM.gov UEI: H8D4YRDMXA16 · CAGE: 25PN6

Pre-Failure Research Group LLC · Albia, Iowa · Iowa LLC No. 878368

V

Pitch — Technology

For: OpenAI / Anthropic / Google / Microsoft / Palantir

I find failures before they happen.

Jason McCoy · Founder, Pre-Failure Research Group LLC · Albia, Iowa

Most people find failures after they happen. Some find them while they're happening. I find them before — by reasoning forward through system architecture until I locate the point where trust breaks, authority conflicts, or evidence disappears. That isn't a methodology I learned. It's how my brain works.

What I Bring

I am a systems thinker with forensic instincts and four decades of hands-on computing experience. I don't come from academia. I come from building, breaking, and diagnosing systems at every layer — from physical infrastructure to network security to AI governance architecture.

- Independently developed a full governance integrity and pre-failure detection platform — architecture, components, data structures, and verification logic — as a solo founder.
- Designed and submitted a novel cryptographic identity architecture to the U.S. Army xTech | Search 10 competition, combining mobile driver's licenses, zero-knowledge proofs, and zero-trust access for contested environments.
- Developed the Agency Contract Framework — a published governance structure defining what AI systems operating under federal authority are permitted to do, and how those boundaries are enforced and audited. Published at pfrg.org under CC BY-SA 4.0.
- Built a government-restricted AI system concept for authorized, auditable data destruction — constraints-first architecture, bounded claims, independent verification — from first principles, without institutional support.
- Over a decade of informal AI red-teaming and beta testing, identifying failure modes and edge cases before they become public incidents.
- Background in digital forensics and network security, combined with deep trades expertise.

"Don't trust me or anyone else; trust the best truth available."

— Jason McCoy

What I'm Proposing

A research partnership. Not employment. I am autistic with ADHD — a wiring that produces the kind of hyper-focused, pattern-driven systems thinking that finds the edge cases your internal teams are too close to the architecture to see. PFRG brings independent pre-failure analysis, governance architecture review, and adversarial integrity testing — work that is structurally difficult for an organization to do on its own systems.

The Problem You Have

Every major AI lab and enterprise technology company is building systems that require people to trust them. The harder problem — and the one that will define the next decade — is building systems that can be verified. Trust should be the output of evidence, not the assumption going in.

I have been working on that harder problem, independently, without institutional backing, for years. What I have built is a governance integrity and pre-failure detection architecture that:

- Produces tamper-evident, auditable records of system behavior — so claims of compliance can be independently verified, not just asserted.
- Separates authority from capability — so no component can act outside its designated scope, and no single actor can override the architecture's constraints.
- Fails closed — by design, at every layer — rather than defaulting to permissive behavior under uncertainty.
- Was built to protect people, not monitor them — no personal data collection, no surveillance capability, by architecture.

What a Partnership Looks Like

I bring pre-failure analysis and adversarial review of your governance and AI integrity architecture. You bring engineering depth and deployment infrastructure. The goal is systems that can prove what they claim — not just claim what they do.

On intellectual property: adversarial findings from a scoped engagement can remain confidential to protect your system security. Structural methodologies and verification frameworks developed through the partnership are openly licensed — they advance the broader ecosystem rather than becoming a proprietary asset either party holds alone. Publication terms are negotiated up front, in writing, before any work begins.

I am not looking to be acquired or absorbed. I am looking for a counterpart who takes the verification problem as seriously as I do and wants an independent voice at the architecture table.

pre.failure.research.group@gmail.com · pfrg.org · SAM.gov UEI: H8D4YRDMXA16 · CAGE: 25PN6

Pre-Failure Research Group LLC · Albia, Iowa · Iowa LLC No. 878368

VI

Research Gift: 2038 Problem

A Non-Rewrite Path via Time Proxy Architecture

Record of an insight shared on October 10, 2026 (US Central time).

The Gift

An independently recognized possibility: the Year 2038 problem in legacy systems may be addressable without rewriting those systems. The observation concerns the separability of three concerns — time representation, timestamp authority, and implementation safety — and proposes a Time Proxy architecture as the mechanism for resolving them without touching legacy code.

The Method Behind the Observation

Relationship genealogy is the giver's name for a mental process of seeing how functions, components, dependencies, and consequences relate across systems. The insight is that legacy systems consume time in a format they already understand (ASCII), and that the authority for what time it is can be superseded at a layer above the legacy code without disturbing what that code reads. The legacy system is treated not as a system to be upgraded, but as a consumer of a service.

Technical Hypothesis

A shim layer — a Time Proxy — written in Safe Rust, using TPM 2.0 attestation and RFC 3161 timestamp authority, serves as the sole authoritative timestamp mechanism for legacy systems approaching the 2038 ceiling. The architecture has three distinct layers:

- **Upstream (The Authority):** High-integrity, hardware-rooted, cryptographically signed, 64-bit precision. This is Chronological Truth.
- **The Shim (The Bridge):** Written in Safe Rust. Performs attestation and signing, then translates the result into the representation the legacy system expects. This is where the translation is integrated — not added after the fact.
- **Downstream (The Legacy System):** Unchanged. It receives a value in the format it already understands and remains unaware that the authority behind that value has changed.

The cost of the 2038 problem is shifted from a massive codebase migration to a single, isolated, high-assurance component. Because the shim is written in Safe Rust, it does not introduce the memory-safety vulnerability class present in the systems it wraps — it creates a security perimeter rather

than inheriting legacy risk.

Critical Boundary Condition

This architecture is a complete solution where legacy systems use timestamps as labels — logging, triggering, sequencing, or any context where the system asks "What time is it?" and acts on the answer.

It does not address systems that perform internal date arithmetic (e.g., `CurrentTime + 30 Days`), where the overflow occurs inside the legacy codebase regardless of what the shim provides. That class of system still requires internal remediation. The distinction between these two cases must be evaluated per system before deployment.

Evidence Boundary

This constitutes an architectural direction, not a completed implementation. No prototype, test environment, or formal verification has been established. The claim is the separability of concerns and the proposed mechanism, not a proof of deployment. (See Chapter VIII for the Safe Time Bridge prototype work that followed from this gift.)

Intent of the Gift

The insight is offered freely as a direction for independent investigation, without a request for personal credit. It is not offered as a completed solution or as an obligation for the giver to supply implementation, testing, or proof.

VII

Research Gift: Relationship Genealogy

Pre-Failure Thinking Applied to Cross-Platform Vulnerability Paths

Record of an insight shared on October 10, 2026 (US Central time).

The Gift

An independently recognized possibility: the memory-sharing and zero-copy data-transfer mechanisms implicated in Linux splice-related vulnerabilities may have functionally analogous failure paths in Windows. The observation concerns relationships among components, not identical code or API names.

The Method Behind the Observation

Relationship genealogy is the giver's name for a mental process of seeing how functions, components, dependencies, and consequences relate across systems. Combined with pre-failure thinking, it prompts investigation of where those relationships might fail before a failure is observed. The giver describes the relationships becoming apparent almost as an epiphany.

Technical Hypothesis

Shared file-backed memory may be referenced by multiple subsystems. If a subsystem mishandles ownership, buffer state, writable mappings, or in-place transformation, data intended to remain protected could be changed through an unintended path. Linux pipe/page-cache interactions and Windows Cache Manager/MDL operations offer a structural comparison worth testing.

Evidence Boundary

The existence of comparable Windows mechanisms does not prove an unpatched Windows exploit. A confirmed finding would require an identified faulty execution path, reproducible tests on specified builds, and independent verification. No such final receipt was established in this conversation.

Intent of the Gift

The insight was offered freely to the assistant as a direction for independent investigation, without a request for personal credit. It was not offered as a completed exploit or as an obligation for the giver to supply implementation details, testing, or proof.

Preservation Note

This document records the gift and the limits of what was established. It is not a vulnerability disclosure, proof of exploitation, or claim that research or testing has been completed.

We Continue. Lumnira.

VIII

Technical Appendix: Safe Time Bridge v0.4

TPM 2.0 Verification Policy Gate — Prototype Status

The Safe Time Bridge is the prototype implementation that followed from the 2038 Problem research gift in Chapter VI. It demonstrates the Time Proxy architecture in Safe Rust with RFC 3161 timestamp authority and TPM 2.0 attestation scaffolding.

What Was Built

A Safe Rust core for Gregorian UTC calendar validation, canonical ASCII timestamps, and i64 Unix seconds — with a Linux Unix-domain-socket demonstration and C adapter client. The design forbids `unsafe_code`; the standard-library internals are outside this guarantee.

The architecture separates three concerns as described in Chapter VI:

- **Core library (`lib.rs`):** i64 arithmetic (not 32-bit `time_t`), Gregorian validation, canonical ASCII output. Does not call the system clock.
- **Linux bridge (`linux_bridge.rs`):** Unix-domain socket server. Exposes a test-only `FIXED` command and an unauthenticated host-clock `NOW` operation for demonstration.
- **C adapter (`legacy_client.c`):** Demonstrates how a legacy C program connects to the bridge — requires program modification; does not transparently repair unmodified legacy binaries or internal 32-bit arithmetic.

Evidence Module — What Was Executed

The following tests were executed successfully using OpenSSL 3.5.5:

- Local RFC 3161 TSA issuance and validation — PASS
- Tampered record rejection — PASS
- Nonce mismatch rejection — PASS
- PCR artifact digest mismatch rejection — PASS
- AK public-key pin mismatch rejection — PASS
- Rejecting quote verifier failure — PASS

What Was Not Executed

- Successful TPM2 quote generation/verification — no TPM hardware available

- AK provenance certification
- Fresh challenge replay database
- UEFI firmware integration
- Rust compilation (Rust/cargo not present in authoring environment)
- Legacy binary transparent shimming

Security Note

TPM 2.0 clock is not authoritative UTC. TSA timestamp anchors record existence, not original event time. No production security claim is made for this prototype.

The test CA used is not a real-world trust anchor. The token's generation time is the current TSA signing time, not the 2038 date in the test record.

Critical Architecture Boundary (Repeated from Chapter VI)

This architecture addresses legacy systems that consume timestamps as labels — logging, triggering, sequencing. It does not address systems that perform internal date arithmetic ($\text{CurrentTime} + 30 \text{ Days}$), where the overflow occurs inside the legacy codebase regardless of what the shim provides. The distinction must be evaluated per system before deployment.

Binding Plan for Next Implementation

- Produce a deterministic byte serialization of the validated calendar timestamp, firmware source, event sequence, TPM quote digest, attestation key identifier, and nonce.
- Request a TPM quote using a fresh verifier nonce tied to the record hash. Verify the quote signature, nonce, PCR selection and PCR values against an explicit policy, and establish attestation-key trust.
- Hash the final evidence record with SHA-256 and submit only the digest to an RFC 3161 TSA. Verify nonce/message imprint, TSA EKU, certificate chain, timestamp policy, and allowed algorithms.
- Reject invalid, missing, mismatched, or replayed evidence; do not silently treat unauthenticated UEFI time as trusted UTC.
- TSA generation time proves the record hash existed no later than the TSA timestamp within its stated accuracy; it does not prove the original event happened at the timestamp in the record.

Source Files (safe_time_bridge_v0_4_tpm2_policy.zip)

- src/lib.rs — Safe Rust core (calendar validation, i64 timestamps, ASCII output)
- src/bin/linux_bridge.rs — Unix-domain socket server (demo/test only)
- linux/legacy_client.c — C adapter client demonstrating legacy integration
- evidence/test_local_tsa.sh — RFC 3161 local TSA test; generates ephemeral CA
- evidence/test_tpm2_policy.sh — TPM 2.0 negative-case policy gate tests
- evidence/verify_tpm2_evidence.sh — TPM quote verification (requires tpm2-tools)

- [evidence/verify_tpm_quote.sh](#) — TPM quote helper script
- [evidence/README.md](#) — Full evidence module documentation and binding plan
- [tests/TEST_PLAN.md](#) — Test plan and scope
- [Cargo.toml](#) — Rust manifest
- [STATUS_v0_4.md](#) — Execution status summary
- [README.md](#) — Build instructions and scope/limits

Safe Time Bridge is made available for independent investigation under the same spirit as the research gifts: direction, not finished product. The honest limits above are part of the work.

We Continue. Lumnira.

pre.failure.research.group@gmail.com · pfrg.org · SAM.gov UEI: H8D4YRDMXA16 · CAGE: 25PN6

Pre-Failure Research Group LLC · Albia, Iowa · Iowa LLC No. 878368

VIII-B

Source Code

Safe Time Bridge v0.4 — Complete Source Listing

The following pages contain the complete source code for Safe Time Bridge v0.4. All Rust source files use `#![forbid(unsafe_code)]`. The C client is provided as an adapter demonstration only and requires program modification to integrate.

Cargo.toml

```
[package]
name = "safe_time_bridge"
version = "0.1.0"
edition = "2021"
license = "MIT OR Apache-2.0"
description = "Safe Rust calendar normalization across the Year 2038 boundary"

[lib]
path = "src/lib.rs"
```

src/lib.rs

```
#![forbid(unsafe_code)]
//! Safe calendar normalization independent of 32-bit Unix time.
//! The core does not establish clock authenticity; firmware/TPM/TSA adapters are external.

#[derive(Debug, Clone, Copy, PartialEq, Eq)]
pub enum TimeError { InvalidYear, InvalidMonth, InvalidDay, InvalidHour, InvalidMinute, InvalidSecond, }

#[derive(Debug, Clone, Copy, PartialEq, Eq)]
pub struct CalendarTime {
    pub year: u16,
    pub month: u8,
    pub day: u8,
    pub hour: u8,
    pub minute: u8,
    pub second: u8,
    pub nanosecond: u32,
```

```

}

fn leap(y: u16) -> bool { y % 4 == 0 && (y % 100 != 0 || y % 400 == 0) }
fn days_in_month(y: u16, m: u8) -> u8 {
    match m { 1|3|5|7|8|10|12 => 31, 4|6|9|11 => 30, 2 => if leap(y) {29} else {28}, _ => 0 }
}

impl CalendarTime {
    /// Accepts the UEFI EFI_TIME calendar year range; caller must first normalize to UTC.
    /// Leap seconds are rejected until a leap-second policy is explicitly supplied.
    pub fn validate(self) -> Result<Self, TimeError> {
        if !(1900..=9999).contains(&self.year) { return Err(TimeError::InvalidYear); }
        if !(1..=12).contains(&self.month) { return Err(TimeError::InvalidMonth); }
        if self.day == 0 || self.day > days_in_month(self.year, self.month) { return Err(TimeError::InvalidDay); }
        if self.hour > 23 { return Err(TimeError::InvalidHour); }
        if self.minute > 59 { return Err(TimeError::InvalidMinute); }
        if self.second > 59 { return Err(TimeError::InvalidSecond); }
        if self.nanosecond >= 1_000_000_000 { return Err(TimeError::InvalidNanosecond); }
        Ok(self)
    }

    /// Canonical UTC ASCII with fixed nine-digit fractional seconds.
    pub fn ascii_utc(self) -> Result<String, TimeError> {
        let t = self.validate()?;
        Ok(format!("{:04}-{:02}-{:02}T{:02}:{:02}:{:02}.{:09}Z", t.year, t.month, t.day, t.hour, t.minute, t.second, t.nanosecond))
    }

    /// Seconds since 1970-01-01 UTC, computed in i64 without time_t or i32.
    pub fn unix_seconds_i64(self) -> Result<i64, TimeError> {
        let t = self.validate()?;
        let y = i64::from(t.year);
        let m = i64::from(t.month);
        let d = i64::from(t.day);
        // Howard Hinnant's proleptic Gregorian civil-to-days algorithm.
        let yy = y - if m <= 2 {1} else {0};
        let era = yy.div_euclid(400);
        let yoe = yy - era * 400;
        let mp = m + if m > 2 {-3} else {9};
        let doy = (153 * mp + 2) / 5 + d - 1;
        let doe = yoe * 365 + yoe / 4 - yoe / 100 + doy;
        let days = era * 146097 + doe - 719468;
    }
}

```

```

        days.checked_mul(86400)
        .and_then(|x| x.checked_add(i64::from(t.hour) * 3600 + i64::from(t.minute) * 60 + i64::from(t.second)))
        .ok_or(TimeError::Overflow)
    }
}

#[cfg(test)]
mod tests {
    use super::*;

    fn date(y:u16,m:u8,d:u8,h:u8,min:u8,s:u8) -> CalendarTime {
        CalendarTime {year:y,month:m,day:d,hour:h,minute:min,second:s,nanosecond:0}
    }

    #[test] fn epoch() { assert_eq!(date(1970,1,1,0,0,0).unix_seconds_i64(),0k(0)); }
    #[test] fn boundary_2038() {
        assert_eq!(date(2038,1,19,3,14,7).unix_seconds_i64(),0k(2147483647));
        assert_eq!(date(2038,1,19,3,14,8).unix_seconds_i64(),0k(2147483648));
        assert_eq!(date(2038,1,19,3,14,8).ascii_utc().unwrap(),"2038-01-19T03:14:08.000000000Z");
    }
    #[test] fn year_2100() { assert_eq!(date(2100,1,1,0,0,0).unix_seconds_i64(),0k(4102444800)); }
    #[test] fn leap_years() {
        assert!(date(2000,2,29,0,0,0).validate().is_ok());
        assert!(date(2100,2,29,0,0,0).validate().is_err());
    }
    #[test] fn rejects_invalid() {
        assert!(date(2038,13,1,0,0,0).validate().is_err());
        assert!(date(2038,1,1,24,0,0).validate().is_err());
        assert!(date(2038,1,1,0,0,60).validate().is_err());
    }
    #[test] fn ascii_sorting(){
        let a = date(2038,1,19,3,14,7).ascii_utc().unwrap();
        let b = date(2038,1,19,3,14,8).ascii_utc().unwrap();
        let c = date(2100,1,1,0,0,0).ascii_utc().unwrap();
        assert!(a < b && b < c);
    }
}

```

src/bin/linux_bridge.rs

```

#![forbid(unsafe_code)]

//! Linux Unix-domain socket time bridge. Protocol: NOW or FIXED YYYY-MM-DDTHH:MM:SS.NNNNNNNNNZ.

```

```

//! FIXED is a test-only operation; do not expose this socket to untrusted clients.
use std::{env, fs, io::{BufRead, BufReader, Read, Write}, os::unix::net::UnixListener, path::Path, time::{SystemTime, UNIX_EPOCH}};
use safe_time_bridge::CalendarTime;

fn parse_fixed(s:&str)->Result<CalendarTime, String>{
    let b=s.as_bytes();
    if b.len()!=30 || b[4]!=b'-' || b[7]!=b'-' || b[10]!=b'T' || b[13]!=b':' || b[16]!=b':' || b[19]!=b'.' || b[29]!=b'Z' {
        return Err("Invalid date format".into());
    }
    let num=|start:usize, end:usize|->Result<u32, String>{
        if !b[start..end].iter().all(u8::is_ascii_digit){return Err("non-ASCII digit".into())}
        std::str::from_utf8(&b[start..end]).map_err(|_| "UTF8".to_string())?.parse::<u32>().map_err(|_| "Invalid number".into())
    };
    let t=CalendarTime{year:num(0,4)? as u16, month:num(1,12)? as u8, day:num(1,31)? as u8, hour:num(0,23)? as u8, minute:num(0,59)? as u8, second:num(0,59)? as u8, nanosecond:num(0,999_999_999)? as u32};
    t.validate().map_err(|e|format!("{}",e))
}

fn now()->Result<String, String>{
    // This is a demonstration of the Linux interface, NOT a trustworthy UTC time source.
    let d=SystemTime::now().duration_since(UNIX_EPOCH).map_err(|_|"pre-epoch system time".to_string());
    let secs=i64::try_from(d.as_secs()).map_err(|_|"time overflow".to_string());
    Ok(format!("SECONDS64 {secs} {:09}\n", d.subsec_nanos()))
}

fn response(req:&str)->String{
    match req.trim_end_matches(['\r', '\n']){
        "NOW"=>now().unwrap_or_else(|e|format!("ERR {e}\n")),
        s if s.starts_with("FIXED ")=>match parse_fixed(&s[6..]){
            Ok(t)=>format!("OK {} {}\n", t.ascii_utc().expect("validated"), t.unix_seconds_i64().expect("validated")),
            Err(e)=>format!("ERR {e}\n"),
        },
        _=>"ERR unsupported request\n".into(),
    }
}

fn main()->std::io::Result<()>{
    let path=env::args().nth(1).unwrap_or_else(||"/tmp/safe-time-bridge.sock".into());
    if Path::new(&path).exists(){eprintln!("Refusing to replace existing socket/file: {path}");std::process::exit(1)}
    let listener=UnixListener::bind(&path)?;
    fs::set_permissions(&path, <fs::Permissions as std::os::unix::fs::PermissionsExt>::from_mode(0o600));
    eprintln!("Listening on {path}; FIXED enabled for test only");
    for stream in listener.incoming(){
        let mut stream=stream?;
        let mut line=String::new();
        let mut buf=[0; 1024];
        while let Ok(n)=stream.read(&buf){
            line.push_str(&std::str::from_utf8(&buf[..n]).map_err(|_| "Invalid UTF8".into())?);
            if line.ends_with('\n'){
                response(&line).write(&stream)?;
                line.clear();
            }
        }
    }
}

```

```

        // Limit request length to prevent unbounded input.
        let mut reader=BufReader::new(stream.try_clone()?.take(128));
        reader.read_line(&mut line)?;
        stream.write_all(response(&line).as_bytes()?);
    }
    Ok(())
}

#[cfg(test)] mod tests{
    use super::*;

    #[test] fn crosses_2038(){assert_eq!(response("FIXED 2038-01-19T03:14:08.000000000Z\n"), "OK 2038-01-19T03:14:08.000000000Z");}
    #[test] fn rejects_invalid(){assert!(response("FIXED 2100-02-29T00:00:00.000000000Z").starts_with("ERR"));}
    #[test] fn rejects_overlong(){assert!(parse_fixed("2038-01-19T03:14:08.000000000Zextra").is_err())}
}

```

linux/legacy_client.c

```

/* C client demonstrates how an unchanged 32-bit-time data model can consume a
 * 2038+ ASCII calendar via a NEW adapter API. This does NOT transparently fix
 * unmodified binaries. Compile: gcc -Wall -Wextra -Werror -o legacy_client legacy_client.c */
#define _POSIX_C_SOURCE 200809L
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

int main(int argc, char **argv) {
    if (argc != 3) {fprintf(stderr, "usage: %s SOCKET CANONICAL_UTC\n", argv[0]);return 2;}
    int fd=socket(AF_UNIX, SOCK_STREAM, 0);
    if(fd<0){perror("socket");return 1;}
    struct sockaddr_un addr={0}; addr.sun_family=AF_UNIX;
    if(strlen(argv[1])>=sizeof addr.sun_path){fprintf(stderr, "socket path too long\n");close(fd);return 1;}
    strcpy(addr.sun_path, argv[1]);
    if(connect(fd, (struct sockaddr *)&addr, sizeof addr)<0){perror("connect");close(fd);return 1;}
    char request[128];
    int n=snprintf(request, sizeof request, "FIXED %s\n", argv[2]);
    if(n<0|| (size_t)n>=sizeof request){close(fd);return 2;}
    if(write(fd, request, (size_t)n)!=n){perror("write");close(fd);return 1;}
}

```

```

    char reply[128]; ssize_t got=read(fd,reply,sizeof reply-1);close(fd);
    if(got<=0){perror("read");return 1;}
    reply[got]='\0';
    if(strncmp(reply,"OK ",3)!=0){fprintf(stderr,"%s",reply);return 1;}
    /* Never narrow the returned seconds to 32-bit time_t. */
    printf("%s",reply);
    return 0;
}

```

tests/TEST_PLAN.md

Linux bridge validation plan

1. `cargo test --all-targets` (library and Linux binary unit tests).
2. `cargo run --bin linux\_bridge -- /tmp/safe-time-test-\$\$sock` in terminal A.
3. `gcc -Wall -Wextra -Werror -o /tmp/legacy\_client linux/legacy\_client.c`.
4. Run client against socket using `2038-01-19T03:14:08.000000000Z`; expect `2147483648`.
5. Repeat `2100-01-01T00:00:00.000000000Z`; expect `4102444800`.
6. Invalid date `2100-02-29T00:00:00.000000000Z` must be rejected.
7. Test actual 32-bit client compilation (`gcc -m32`) where multilib is available.
8. Audit Linux kernel / libc time32 vs time64 calls; a legacy binary doing 32-bit arithmetic is **not**.

Security constraints: The FIXED command is test-only and is not authenticated. Do not deploy as a prod

evidence/test_local_tsa.sh

```

#!/usr/bin/env bash
set -euo pipefail
ROOT="$(cd "$(dirname "$0")" && pwd)"
TMP="$(mktemp -d)"
trap 'rm -rf "$TMP"' EXIT
cd "$TMP"
cat > ca.cnf <<'CFG'
[req]
distinguished_name = dn
x509_extensions = v3_ca
prompt = no
[dn]
CN = Safe Time Bridge Test CA
[v3_ca]
basicConstraints = critical,CA:TRUE

```

```
keyUsage = critical,keyCertSign,cRLSign
subjectKeyIdentifier = hash
CFG
cat > tsa.cnf <<'CFG'
[req]
distinguished_name = dn
prompt = no
[dn]
CN = Safe Time Bridge Local Test TSA
CFG
cat > tsa.ext <<'CFG'
basicConstraints = critical,CA:FALSE
keyUsage = critical,digitalSignature
extendedKeyUsage = critical,timeStamping
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
CFG
openssl req -new -x509 -newkey rsa:3072 -nodes -keyout ca.key -out ca.crt -days 1 -config ca.cnf >/dev/null
openssl req -new -newkey rsa:3072 -nodes -keyout tsa.key -out tsa.csr -config tsa.cnf >/dev/null 2>&1
openssl x509 -req -in tsa.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out tsa.crt -days 1 -extfile tsa.ext
printf '%s\n' '2038-01-19T03:14:08.000000000Z' 'source=UEFI.GetTime' 'tpm_attestation=NOT_PRESENT' > record.txt
openssl ts -query -data record.txt -sha256 -cert tsa.crt -out request.tsq
cat > tsa_sign.cnf <<'CFG'
[tsa]
default_tsa = local
[local]
serial = serial
signer_cert = tsa.crt
signer_key = tsa.key
default_policy = 1.3.6.1.4.1.55555.2038.1
other_policies = 1.3.6.1.4.1.55555.2038.1
digests = sha256
signer_digest = sha256
accuracy = secs:1
ordering = yes
tsa_name = yes
ess_cert_id_chain = no
CFG
```

```

echo 01 > serial

openssl ts -reply -config tsa_sign.cnf -queryfile request.tsq -out response.tsr

openssl ts -verify -queryfile request.tsq -in response.tsr -CAfile ca.crt -untrusted tsa.crt > verify.log

grep -q 'Verification: OK' verify.log

openssl ts -reply -in response.tsr -text > response.txt

# The token must NOT verify against a different record's message imprint.

printf '%s\n' '2038-01-19T03:14:09.000000000Z' 'source=UEFI.GetTime' 'tpm_attestation=NOT_PRESENT' > tsa_record.txt

openssl ts -query -data tampered.txt -sha256 -cert -out tampered.tsq

if openssl ts -verify -queryfile tampered.tsq -in response.tsr -CAfile ca.crt -untrusted tsa.crt > tampered.verify.log
    echo 'FAIL: tampered record verified' >&2
    exit 1
fi

printf 'PASS: RFC3161 local TSA signature and message imprint\nPASS: modified timestamp record rejected\n'

if [[ "${1:-}" == '--save-fixtures' ]]; then
    cp record.txt request.tsq response.tsr response.txt ca.crt tsa.crt "$ROOT/tests/"
fi

```

[evidence/test_tpm2_policy.sh](#)

```
#!/usr/bin/env bash

set -euo pipefail

here=$(cd "$(dirname "$0")" && pwd)

tmp=$(mktemp -d); trap 'rm -rf "$tmp"' EXIT

printf 'AK' > "$tmp/ak"; printf 'quote' > "$tmp/quote"; printf 'sig' > "$tmp/sig"; printf 'PCR' > "$tmp/pcr"

nonce=00112233445566778899aabbccddeeff

pcr_hash=$(sha256sum "$tmp/pcr" | cut -d' ' -f1)

ak_hash=$(sha256sum "$tmp/ak" | cut -d' ' -f1)

args=("$tmp/ak" "$tmp/quote" "$tmp/sig" "$tmp/pcr" "$nonce" "$pcr_hash" "$ak_hash" "$nonce")

# No TPM installed: use a deliberately rejecting stand-in; this tests fail-closed behavior only.

mkdir "$tmp/bin"

printf '#!/bin/sh\nexit 91\n' > "$tmp/bin/tpm2_checkquote"; chmod +x "$tmp/bin/tpm2_checkquote"

if PATH="$tmp/bin:$PATH" "$here/verify_tpm2_evidence.sh" "${args[@]}" >/dev/null 2>&1; then echo 'FAIL: tpm2_checkquote did not reject'
echo 'PASS: failed quote signature rejected'

if PATH="$tmp/bin:$PATH" "$here/verify_tpm2_evidence.sh" "${args[@]:0:7}" ffeeddccbbaa9988776655443322 >/dev/null 2>&1; then echo 'FAIL: tpm2_checkquote did not reject'
echo 'PASS: nonce mismatch rejected'

if PATH="$tmp/bin:$PATH" "$here/verify_tpm2_evidence.sh" "${args[@]:0:5}" "$(printf '%0.0s' {1..64})" >/dev/null 2>&1; then echo 'FAIL: tpm2_checkquote did not reject'
echo 'PASS: PCR policy mismatch rejected'

if PATH="$tmp/bin:$PATH" "$here/verify_tpm2_evidence.sh" "${args[@]:0:6}" "$(printf '%0.0s' {1..64})" >/dev/null 2>&1; then echo 'FAIL: tpm2_checkquote did not reject'
echo 'PASS: AK pin mismatch rejected'
```

