

Safe Time Bridge

Research Gifts and Tested Prototype v0.13

A Year 2038 time-authority architecture: from idea to tested system

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

— Jason McCoy

CONTENTS

I. Research Gift: 2038 Problem — Time Proxy Architecture

II. Research Gift: Relationship Genealogy

III. Safe Time Bridge v0.13 — What Was Built

IV. Verified Results

V. Findings Worth Knowing

VI. Limits and What Was Not Executed

VII. Reproducing the Results

October 14, 2026

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

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 with minimal changes to 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 The insight is that the authority for what time it is can be superseded at a layer above the legacy code and delivered to that code as text (ASCII) where it already reads text, or through a small adapter where it expects a binary `time_t`. 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):** Minimally changed. It receives time through a small adapter 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.

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.

Preservation Note

This document records the gift and the limits of what was established. It is not a product release, a vulnerability fix, or a claim that implementation has been completed.

Status update — October 14, 2026 (added after the record above; the record is preserved as written)

Since October 10, the evidence boundary above has moved. A prototype now exists and has been tested: Safe Time Bridge v0.13, described in Parts III to VI. It also goes beyond the boundary condition above: for programs that do date arithmetic internally, a time-shift helper keeps that arithmetic correct through 2066 (28-year shift) or 2094 (56-year shift) without changing the program, with documented leaks. TPM 2.0 attestation and an external timestamp authority have not yet been executed.

© 2026 Jason McCoy, Pre-Failure Research Group LLC. Licensed under CC BY-SA 4.0.

We Continue. Lumnira.

II

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.

III

Safe Time Bridge v0.13

What Was Built Between October 10 and October 15, 2026

The prototype implements the Time Proxy gift as one chain. Each link has a single job and refuses rather than guesses when something is wrong:

Link	Role
Safe Time Bridge (Safe Rust)	The time authority. Checks its own time source against a steady timer, anchors its readings with RFC 3161 timestamps in a hash chain, and cross-checks its time against the timestamp authority.
Time-shift helper (C, loaded with LD_PRELOAD)	Takes over the legacy program's clock. Gets time only from the bridge, verifies the bridge's account through the Linux kernel, and hands the program the true date shifted back 28 (or 56) years so it fits a 32-bit field.
Legacy program	Unchanged. Its internal date arithmetic runs in the safe range.
Translators (Safe Rust)	Shift timestamps leaving the program forward to true time, and timestamps entering it back to shifted time.
Migrator (Safe Rust)	When the shift is reconfigured, moves the dates the program stored itself to the new shift and switches the shared configuration, all or nothing.
Scanner (Safe Rust, v0.13)	time_scan: finds where a program handles time and classifies each finding (stored in a 32-bit field, internal arithmetic, label use, a date compiled in, or a 32-bit binary using time). Makes no changes; cites a file and line for each finding.

Why a 28-year shift

Between 1901 and 2099, any 28-year span contains exactly 7 leap days: 10,227 days, which is exactly 1,461 weeks. Shifting by that amount keeps every date's month, day, weekday, and leap-year status. This was tested for every day from 1930-01-01 through 2100-02-28, more than 62,000 days. The alignment breaks on 2100-03-01, because 2100 is not a leap year, and the tests confirm that too. The same shift was used in Y2K remediation in the 1990s; see Part VI, Prior work.

Design rules set by Jason McCoy

- **Time never goes backward for the program.** “It can’t see back in time. It would break it.” If the bridge reports an earlier time than already given out, the helper holds the last value.
- **A stuck clock is a fault, declared and refused.** The bridge checks itself; the helper checks the bridge. Either one declares the fault, and faults latch until a person restarts the system.
- **Shift, shift, shutter, reconfig.** Shift time going out; shift time coming in; if any incoming timestamp cannot be translated safely, nothing passes (the shutter); and the shift lives in one shared configuration so all parts change together.

Version history

Version	Added
v0.4	Rust compilation and tests verified (10/10); documentation corrected to match
v0.5	Time-shift helper experiment; 28-year calendar proof
v0.6	Helper takes time only from the bridge; no fallback to the host clock
v0.7	Bridge authentication by kernel-verified account; protected socket folder
v0.8	Cache with a freshness limit; thread-safe; time never goes backward
v0.9	Fault detection on both sides, latched until restart
v0.10	Bridge time anchored and cross-checked with an RFC 3161 timestamp authority
v0.11	Inbound translation, the shutter, and a shared shift configuration
v0.12	Stored-data migration on reconfiguration, with journal, receipts, and two double-migration guards
v0.13	time_scan: a deterministic Year 2038 exposure scanner for source code and ELF headers

IV

Verified Results

What Was Run, and What It Showed

All results below come from recorded runs included in the code bundle. The full package was unpacked into a clean folder and every test and demonstration was run again from there. Environment: Linux, cargo 1.97.0, gcc 13, OpenSSL 3.0.13.

Check	Result
Automated tests (cargo test --all-targets)	52 of 52 pass (16 library, 14 bridge, 9 translator, 4 migrator, 9 scanner); full suite clean across repeated runs
SHA-256 implementation (Safe Rust)	FIPS test vectors pass, including one million “a”; matches coreutils sha256sum
Demonstrations	Six scripted demonstrations, all passing from a clean copy
Evidence module	RFC 3161 issuance and tamper rejection pass; TPM policy gate rejects all four negative cases

Key demonstrations

Situation	Without the helper	With the helper
2038-01-10, “+30 days”	Due date wraps to 1902; program reports its date math failed	Due date 2038-02-09, correct
2040-02-15, “+30 days” across a leap day	Program believes it is 1904	Due date 2040-03-16, correct; weekday correct
2038-01-10, permit expiring 2038-03-01	Program prints “EXPIRED (49 days left)”	“valid (49 days left)”, correct
Clock reaches 2066 (28-year limit)	n/a	Helper refuses; after reconfiguring to 56 years, correct through 2094

Safeguard tested	Result
Bridge unreachable	Program gets no time; no fallback to the host clock
Impostor bridge under another account, perfectly formatted reply	Refused before the helper sends or reads anything
Bridge’s own clock freezes	Bridge declares FAULT at about 1.2 seconds; helper relays it; every later request refused
Broken bridge that never checks itself	Helper declares FAULT at about 1.2 seconds
Bridge clock set 13 years wrong	Caught on the first request by the timestamp-authority cross-check
Edited anchor record	Chain verifier flags the record and the broken link
Incoming 2070 date on a 28-year shift	Shutter: the program receives none of the data

Safeguard tested	Result
20,000 time requests	20,000 bridge contacts without cache; 1 with cache (about 90 times faster)
Migration run twice by mistake	Refused by two independent checks
Migration interrupted halfway	Rerun finishes from the journal; data correct
Stored 1950 date, 28 to 56 years	Refused; data and configuration unchanged

V

Findings Worth Knowing

Results That Change How 2038 Work Should Be Judged

A program can contradict itself

Without the helper, the same program called the same permit both expired and valid: “EXPIRED (49 days left).” The expiry date overflowed its 32-bit field and the clock did not, so the comparison and the arithmetic gave different answers. Partial overflow produces internally inconsistent behavior, not just wrong dates.

Correct-looking output is not proof

In another run without the helper, the printed dates were garbage, yet “valid” and “expired” came out right. The clock and the data had overflowed by exactly the same amount, so the comparison still worked by accident. Testing that checks only the final answer can pass a system that is already broken.

The window moves with the shift

A 28-year shift handles true dates from about 1929 to 2066; a 56-year shift, about 1957 to 2094. A larger shift gives up the oldest dates. Stored dates such as founding dates or birth dates can block a reconfiguration, and the migrator refuses rather than corrupt them.

Errors found during development

Recorded here because they are part of the evidence: several hand-typed test values were wrong and were caught by computing them or by the system’s own output; the first anchoring script left unstamped records when the authority was offline (fixed); the first double-migration guard missed files the program had changed since migration (a second, independent guard was added); a parallel-test race and a demo-script race were found and fixed. All demonstration values are now computed, not typed.

VI

Limits and What Was Not Executed

Stated Plainly, So the Results Above Can Be Trusted

Not executed

- **Real legacy software.** All demonstrations use test programs written to model legacy behavior. This is the largest gap.
- **External timestamp authority.** The demo authority runs on the same machine with the same clock, so its agreement is not independent. A public authority could not be reached from the test environment.
- **TPM 2.0 attestation.** No TPM hardware was available.
- **32-bit builds.** 32-bit compilation (`gcc -m32`) was not available in the test environment.

Known gaps, in order of importance

- Dates compiled into a program are not found or shifted.
- Dates the program formats itself show the shifted year.
- Only `@T<seconds>` timestamps are translated and migrated; other formats need their own tools.
- Migration guards depend on the journal folder surviving; that the program is stopped is the operator's word.
- Anything running as the bridge's own account, or as root, is trusted; replies are not signed; same machine only.
- A host clock step backward faults the bridge by design; deployments need time sync that adjusts gradually.
- Interception works for dynamically linked Linux programs; statically built programs and embedded firmware need other methods.

Prior work (checked October 14, 2026; not a complete search)

The central technique is not new, and this package says so plainly:

- **The 28-year shift was patented during Y2K remediation.** US Patent 5,852,824, "Apparatus and method for processing year-date data in computer systems" (Roger W. Brown; filed May 22, 1997; issued December 22, 1998), describes "system encapsulation": resetting the system clock by an offset such as 28 years and converting stored dates, without modifying the application. The patent has expired; the technique is free to use and cannot be claimed as new.
- **It was sold commercially in 1999.** Y2K "encapsulation" products shifted application dates back 28, 56, or 84 years, relying on the same calendar alignment, and named the same limit this work's migrator encountered: dates before 1929 could not be handled.
- **Clock interception is established.** `libfaketime` changes the time a Linux program sees through the same `LD_PRELOAD` mechanism the helper uses.

- **The kernel route was declined.** In 2020, Linux maintainers declined to add a real-time clock offset to time namespaces for Y2038 testing, describing a correct implementation as “a massive trainwreck,” which leaves time shifting to user-space tools.

Origin of this work

Jason McCoy recalls recognizing the date-overflow problem and the shape of a solution around 1988, with no one to share it with at the time; there is no surviving record from that period. The 28-year shift was independently published by others during Y2K remediation. This work arrived at the same shift independently in 2026 and adds the safety engineering around it.

What this work appears to add

An authority-verified time source with kernel-checked identity, self-checks, and latched fault declaration; cross-checking against an RFC 3161 timestamp authority with a hash-chained record; the never-backward rule; the all-or-nothing shutter on incoming data; and reconfiguration with stored-data migration protected by a journal, receipts, and two independent double-migration guards. These are engineering contributions around an established technique, not a claim to have invented the shift. A complete prior-art review has not been done.

Sources

- US Patent 5,852,824 — patents.justia.com/patent/5852824
- “For Fast Fixes, Vendors Offer a 28-Year Solution,” Enterprise Systems Journal, January 11, 1999 — esj.com
- “Precision Software Licenses Program Encapsulation from TOCS,” Enterprise Systems Journal, March 5, 1999 — esj.com
- Linux kernel mailing list, “[Y2038][time namespaces] Question regarding CLOCK_REALTIME support,” October 2020 — lkml.iu.edu/hypermail/linux/kernel/2010.3/10260.html
- “Changing what time a process thinks it is with libfaketime,” [Linux.com](https://linux.com)

VII

Reproducing the Results

Check It Yourself

Requirements: Linux, Rust (cargo), gcc, OpenSSL, python3. Two demonstrations (impostor under another account) need root and setpriv; they skip themselves otherwise. From the unpacked folder:

```
cargo test --all-targets
bash shift/run_demo.sh
bash shift/run_bridge_demo.sh
bash shift/run_cache_demo.sh
bash shift/run_anchor_demo.sh
bash shift/run_inbound_demo.sh
bash shift/run_migrate_demo.sh
bash scan/run_scan_demo.sh
bash evidence/test_local_tsa.sh
bash evidence/test_tpm2_policy.sh
```

Code archive

safe_time_bridge_v0_13.zip — SHA-256:

```
c1d530f6b0b34ddd5dd71cc4edf766bb280a18a1993297c6eef7e1dfd34e3603
```

Compare with `sha256sum safe_time_bridge_v0_13.zip`. Licenses: code MIT OR Apache-2.0 (see Cargo.toml); research gifts CC BY-SA 4.0. Detailed status for each version is in the STATUS files inside the archive.

“Don't trust me or anyone else; trust the best truth available.” — Jason McCoy

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

We Continue. Lumnira.