**BULLETIN OF COMPUTER SCIENCE RESEARCH**
**ISSN 2774-3659 (Media Online)**
Vol 6, No 6, October 2026 | Hal 2590-2600
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i6.1385

# Pengembangan Immutable Object Storage dengan Concurrent Garbage Collection dan Read-Proxy untuk Mitigasi Ransomware

# Development of Immutable Object Storage with Concurrent Garbage Collection and Read-Proxy for Ransomware Mitigation

**Alfian Gading Saputra, Moh. Ali Romli***

Fakultas Sains & Teknologi, Program Studi Informatika, Universitas Teknologi Yogyakarta, Sleman, Indonesia
Email:[1] alfiangadingsaputra@gmail.com, [2,*]ali.romli@uty.ac.id
Email Penulis Korespondensi: ali.romli@uty.ac.id
Received: **16/09/2026**, Revised: **18/10/2026**, Accepted: **25/10/2026**, Published: **31/10/2026**

**Abstrak-**Serangan *ransomware* modern menargetkan data operasional sekaligus repositori cadangan, sedangkan penyimpanan *mutable* masih memungkinkan penghapusan fisik melalui aplikasi yang dikompromikan. Penelitian ini bertujuan mengembangkan dan mengevaluasi *Immutable Object Storage* berbasis *Content-Addressable Storage* (CAS) dan *FastCDC* pada lingkungan *single-node*. Kontribusi penelitian berupa integrasi deduplikasi tingkat *chunk*, pembacaan terverifikasi tanpa akses direktori bagi antarmuka pemrograman aplikasi (API) publik, dan reklamasi konkuren yang memvalidasi ulang referensi serta aktivitas sebelum penghapusan fisik. Sistem memisahkan API publik dari *Vault Core*, membentuk identitas *chunk* dengan BLAKE3, serta mengelola *manifest* dan referensi pada BadgerDB. *Concurrent Garbage Collection* (GC) mereklamasi *chunk* tanpa referensi, sedangkan *Read-Proxy* melalui *Unix Domain Socket* (UDS) menyediakan mekanisme pembacaan bagi API. Satu instans *Store*, *read lease*, registri unggah aktif, masa tenggang, dan validasi akhir menyinkronkan pembacaan, unggah, perubahan *manifest*, serta penghapusan fisik. Uji unit dan *race detector*, uji konfigurasi komponen produksi, serta *performance harness* berbasis UDS menunjukkan alur utama berjalan sesuai rancangan dan tidak ditemukan kesalahan pada sampel *Read-Proxy*. Empat unggahan berukuran total 32 MiB menghasilkan rasio penggunaan ulang *chunk* 96,1961% dan penghematan byte 93,6187%. Pada 16 klien konkuren, median *Read-Proxy* sebesar 1.108,762 ms. GC mereklamasi 2.056.192 byte, sedangkan *throughput* pembacaan turun 9,1692% ketika GC aktif. Hasil menunjukkan bahwa arsitektur mempersempit otoritas API publik, mempertahankan verifikasi integritas, dan menyediakan jendela deteksi sebelum reklamasi fisik. Evaluasi dibatasi pada dataset sintetis, satu perangkat, satu konfigurasi *FastCDC*, dan maksimum 16 klien.

**Kata Kunci:** Immutable Object Storage; FastCDC; Concurrent Garbage Collection; Read-Proxy; Ransomware

**Abstract-**Modern ransomware targets both operational data and backup repositories, while mutable storage can still permit physical deletion through a compromised application. This study develops and evaluates a single-node Immutable Object Storage system based on Content-Addressable Storage (CAS) and FastCDC. Its contribution is an integrated design combining chunk-level deduplication, verified reads without directory access for the public application programming interface (API), and concurrent reclamation that revalidates references and activity before physical deletion. The system separates the public API from Vault Core, derives BLAKE3 chunk identities, and manages manifests and references in BadgerDB. Concurrent Garbage Collection (GC) reclaims unreferenced chunks, while a Read-Proxy over a Unix Domain Socket (UDS) provides the API reading mechanism. A shared Store instance, read leases, an active-upload registry, a grace period, and final validation coordinate reads, uploads, manifest changes, and physical deletion. Unit and race-detector tests, a production-component configuration test, and a UDS performance harness show that principal flows operate as designed, with no errors found in Read-Proxy samples. Four uploads totaling 32 MiB achieved a 96.1961% chunk-reuse ratio and 93.6187% byte savings. With 16 concurrent clients, median Read-Proxy latency was 1108.762 ms. GC reclaimed 2,056,192 bytes, while read throughput decreased by 9.1692% when GC was active. The architecture narrows public API authority, preserves integrity verification, and provides a detection window before physical reclamation. The evaluation is limited to a synthetic dataset, one device, one FastCDC configuration, and at most 16 clients.

**Keywords:** Immutable Object Storage; FastCDC; Concurrent Garbage Collection; Read-Proxy; Ransomware

## 1. PENDAHULUAN

Repositori cadangan menjadi komponen terakhir yang menentukan keberhasilan pemulihan ketika data operasional terenkripsi atau tidak dapat dipercaya. Perkembangan *ransomware* mengubah cadangan dari sasaran tidak langsung menjadi sasaran utama. Penyerang berupaya menemukan, menonaktifkan, menghapus, atau mengenkripsi cadangan sebelum menyerang data produksi. Panduan Cybersecurity and Infrastructure Security Agency (CISA) merekomendasikan cadangan terenkripsi, terpisah, dan diuji secara berkala karena cadangan yang tetap terhubung dapat ikut terdampak setelah kompromi jaringan [1]. Laporan Sophos juga menunjukkan bahwa kompromi cadangan berkaitan erat dengan meningkatnya dampak operasional dan biaya pemulihan serangan *ransomware* [2]. Temuan tersebut menegaskan bahwa keberadaan cadangan belum cukup apabila lapisan aplikasi yang telah diretas masih mempunyai jalur untuk mengubah repositori fisik.

Risiko tersebut banyak muncul pada arsitektur penyimpanan *mutable*. Antarmuka pemrograman aplikasi (API) yang sah biasanya menyediakan operasi membuat, membaca, memperbarui, dan menghapus objek dengan kredensial layanan yang sama. Setelah memperoleh kendali terhadap API atau kredensial administratif, penyerang dapat menyalahgunakan fungsi normal untuk menghapus objek cadangan. Pendekatan deteksi tetap penting, tetapi efektivitasnya dipengaruhi pola serangan, waktu respons, serta hak istimewa penyerang. Kim dkk. [3] menunjukkan bahwa *ransomware* yang memperoleh eskalasi hak dapat menghindari indikator generik sehingga validasi berbasis

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. Page 2590
This work is licensed under a Creative Commons Attribution 4.0 International License.

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 6, October 2026 | Hal 2590-2600
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i6.1385

struktur berkas diperlukan. Higuchi dan Kobayashi [4] juga menemukan bahwa lokasi pencegatan operasi berkas memengaruhi jumlah data yang dapat dipulihkan. Kedua penelitian memperlihatkan nilai mekanisme detektif dan intersepsi, tetapi repositori tujuan tetap membutuhkan perlindungan struktural apabila pencegahan awal gagal.

Teknologi *Write-Once*-Read-Many (WORM) memberikan prinsip penting untuk perlindungan struktural karena data yang telah ditulis tidak dapat ditimpa atau dihapus selama masa retensi. Caporaso dkk. [5] menerapkan semantik *write-once* pada sistem berkas Linux melalui VaultFS untuk melindungi data yang tetap dapat dibaca tetapi tidak dapat dimodifikasi selama jendela perlindungan, termasuk oleh proses dengan hak tinggi. Múzquiz dkk. [6] mengembangkan Reverse File System pada perangkat Socarrat, yang menempatkan penegakan WORM dalam modul USB terpisah dari komputer inang. Kedua pendekatan memperkuat integritas melalui pembatasan domain kontrol, tetapi bekerja pada lapisan sistem berkas atau perangkat. Kebijakan WORM sendiri belum menentukan cara berbagi konten identik antaversi maupun koordinasi pembaca dengan reklamasi *chunk*. Karena itu, integrasi deduplikasi dan pengelolaan siklus hidup pada repositori objek masih perlu dirancang secara eksplisit.

*Content-Addressable Storage* (CAS) memakai hash konten sebagai alamat sehingga unit data identik dapat digunakan kembali. Efektivitas deduplikasi meningkat ketika berkas dibagi menjadi *chunk* berdasarkan konten. FastCDC menentukan batas *chunk* dari pola isi agar penyisipan lokal tidak selalu menggeser seluruh batas berikutnya [7]. Pemrosesan *Content-Defined Chunking* (CDC) juga dapat dipercepat melalui instruksi vektor tanpa mengurangi penghematan ruang secara berarti [8]. Penelitian ini memakai BLAKE3 sebagai *fingerprint chunk* karena struktur pohonnya mendukung pemrosesan paralel dan verifikasi konten yang efisien [9].

Integrasi WORM logis, CAS, dan *FastCDC* menimbulkan persoalan siklus hidup data. *Soft delete* mempertahankan data fisik tetapi tidak membebaskan ruang, sedangkan penghapusan fisik langsung oleh aplikasi membuka kembali jalur destruktif. Survei Lv dkk. [10] menguraikan kompromi performa dan amplifikasi pada penyimpanan key-value berbasis Log-Structured Merge-tree (LSM). Liu dkk. [11] mengintegrasikan defragmentasi dengan GC pada penyimpanan cadangan terdeduplikasi untuk memperbaiki lokalitas, sehingga reklamasi turut memengaruhi biaya pemulihan data. Zhang dkk. [12] mengembangkan Scavenger+ dengan GC hemat I/O, kompaksi yang mempertimbangkan ruang, serta penjadwalan adaptif terhadap beban. Fokus kedua rancangan terakhir ialah efisiensi ruang dan performa; kebijakan otorisasi API dan perlindungan rekonstruksi lintas proses perlu ditangani pada lapisan aplikasi. Temuan tersebut mendasari *Concurrent Garbage Collection* dengan masa tenggang dan validasi akhir sebelum penghapusan.

Jalur pembacaan turut menentukan batas kepercayaan. Prototipe awal memakai *read-only mount* agar API membaca direktori penyimpanan. Pembatasan ini mencegah penulisan, tetapi proses publik tetap mengetahui tata letak *chunk* dan mengakses repositori. National Institute of Standards and Technology (NIST) menekankan pembatasan antarmuka, segmentasi, autentikasi, dan hak minimum pada penyimpanan [13]. Model *zero trust* juga menempatkan keputusan akses dekat dengan sumber daya dan menghindari kepercayaan implisit antarkomponen [14]. Karena itu, penelitian memindahkan pembacaan ke *Read-Proxy* di *Vault Core*. API hanya meminta objek logis, sedangkan validasi *manifest*, pembacaan *chunk*, verifikasi hash, dan rekonstruksi berlangsung di zona terlindungi.

Pada lapisan penyimpanan, Constantinescu dan Seshadri mengembangkan Sentinel untuk mendeteksi aktivitas *ransomware* pada file storage dan menandai konten terdampak agar pemulihan dapat dipercepat [15]. Reidys dkk. mengembangkan RSSD dengan pencatatan terisolasi perangkat keras dan perluasan penyimpanan log melalui jaringan; model ancamannya mencakup serangan yang sengaja memicu GC agar versi lama hilang [16]. Ma dkk. mengusulkan RansomTag untuk menghubungkan deteksi pada hypervisor dengan penandaan data pada SSD sehingga pemulihan dapat dilakukan secara granular [17]. Dafoe dkk. mengembangkan FFRecovery yang menyatukan pemulihan metadata melalui forensik sistem berkas, ekstraksi data pada flash translation layer (FTL), serta penundaan dan pembekuan GC untuk mempertahankan bahan pemulihan [18]. Keempat studi menunjukkan bahwa kemampuan mendeteksi serangan harus diikuti perlindungan data yang diperlukan untuk pemulihan. Namun, mekanismenya berfokus pada deteksi atau pengelolaan versi pada storage dan SSD/FTL, sehingga tidak langsung menjawab koordinasi referensi *manifest*, unggah aktif, dan pembacaan terverifikasi pada repositori objek terdeduplikasi.

Joo dkk. [19] mengusulkan deteksi penimpaan pada pengendali SSD menggunakan bitmap yang membutuhkan satu bit per operasi I/O, menggantikan pencatatan berbasis hash yang lebih mahal dalam penggunaan memori. Temuan tersebut memperlihatkan bahwa pelacakan aktivitas penyimpanan dapat dibuat lebih ringan. Namun, informasi penimpaan di tingkat perangkat tidak mewakili hubungan logis antar*manifest* yang berbagi *chunk*. Penelitian ini menempatkan pemeriksaan referensi dan aktivitas pembaca pada *Vault Core* agar keputusan reklamasi mengikuti status objek, sementara pendekatan Joo dkk. berorientasi pada efisiensi deteksi penimpaan di SSD.

Hirano dan Kobayashi [20] memperkenalkan RanSMAP, dataset terbuka pola akses penyimpanan dan memori yang dikumpulkan melalui hypervisor ringan untuk pengembangan detektor *ransomware* berbasis pembelajaran mendalam. Variasi perangkat keras dan eksekusi aplikasi jinak bersama *ransomware* menjadi bagian penting evaluasinya. Studi ini memperluas dasar pengujian deteksi perilaku, tetapi tidak menyediakan kebijakan retensi dan reklamasi bagi repositori cadangan. Perbandingan ini menegaskan bahwa dataset sintetis pada prototipe penelitian hanya menilai fungsi penyimpanan, deduplikasi, dan keselamatan reklamasi dalam skenario terkontrol; hasilnya tidak menunjukkan akurasi deteksi *ransomware* atau ketahanan terhadap seluruh keluarga serangan.

Berdasarkan studi yang ditelaah, kesenjangan penelitian terletak pada koordinasi antara pembatasan otoritas aplikasi publik, efisiensi penyimpanan antaversi, dan keselamatan reklamasi ketika unggah serta pembacaan berlangsung bersamaan. Penelitian ini menawarkan integrasi CAS, *FastCDC*, BLAKE3, *Read-Proxy*, dan *Concurrent Garbage*

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. Page 2591
This work is licensed under a Creative Commons Attribution 4.0 International License.

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 6, October 2026 | Hal 2590-2600
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i6.1385

*Collection* pada prototipe *single-node*. Kontribusi arsitekturalnya ialah pemisahan API–*Vault Core* sehingga API meminta objek logis tanpa akses direktori *chunk*. Kontribusi pengelolaan datanya ialah deduplikasi tingkat *chunk* dan alur *manifest snapshot*–grace–*final validation*–*sweep* pada satu *Store* yang berbagi *read lease* serta registri unggah aktif. Kontribusi evaluasinya ialah pengukuran penggunaan ulang *chunk*, penghematan byte, biaya *Read-Proxy*, dan dampak GC, disertai pengujian *re-reference* serta kegagalan parsial. Kebaruan ditempatkan pada integrasi dan koordinasi mekanisme tersebut, bukan pada penciptaan algoritma *FastCDC* atau BLAKE3 baru. *Soft delete* tetap logis; permintaan *retirement* yang diaudit menjadi masukan GC untuk reklamasi tertunda.

Penelitian bertujuan membangun dan mengevaluasi prototipe yang menjaga integritas repositori cadangan sekaligus mengendalikan pertumbuhan kapasitas. Rumusan masalahnya adalah bagaimana membatasi akses destruktif langsung dari API publik, mengurangi penulisan konten berulang, dan mereklamasi *chunk* tanpa referensi tanpa mengganggu objek yang masih digunakan. Evaluasi mencakup fungsi, rasio penggunaan ulang *chunk* dan penghematan byte, konsistensi kegagalan, penolakan operasi destruktif, ketepatan *Read-Proxy*, keselamatan reklamasi, serta dampak konkurensi. Tujuan tersebut membatasi klaim pada kemampuan repositori dalam model ancaman dan skenario yang diuji.

# 2. METODOLOGI PENELITIAN

## 2.1 Kerangka Dasar Penelitian

Penelitian menggunakan rekayasa eksperimental untuk menghasilkan prototipe dan menguji alur penyimpanan, pengambilan, serta reklamasi data pada skenario terkontrol. Pengukuran dilakukan pada Manjaro Linux x86_64, AMD Ryzen 3 3200U (dua inti, empat thread), empat CPU logis, dan RAM 9,66 GiB. Komponen inti dikembangkan dengan Go dan antarmuka demonstrasi dengan React. PostgreSQL menyimpan metadata aplikasi, sedangkan BadgerDB mengelola indeks *chunk*, *manifest*, sesi unggah, dan status *retirement*. API Service dan *Vault Core* berkomunikasi melalui *Unix Domain Socket* (UDS) agar antarmuka internal tidak terbuka di jaringan publik.

Data primer berupa berkas simulasi dengan tingkat kemiripan berbeda: identik, berubah sebagian, berbeda, tidak valid, serta unggah yang dihentikan sebelum *commit*. *Read-Proxy* diuji dengan meminta objek terkomit melalui identitas *manifest* dan membandingkan hash hasil rekonstruksi dengan sumber. GC diuji memakai referensi aktif maupun tidak aktif, masa retensi, dan status tombstone. Tabel 1 merangkum fungsi tiap kelompok data.

**Tabel 1.** Kelompok data dan tujuan pengujian

| Kelompok data | Variasi | Tujuan evaluasi | Keluaran utama |
|---|---|---|---|
| Berkas identik | Konten sama pada unggah berulang | Deduplikasi penuh dan penggunaan ulang chunk | Chunk baru minimal; manifest valid |
| Perubahan sebagian | Penyisipan atau perubahan pada sebagian isi | Stabilitas batas FastCDC dan deduplikasi parsial | Chunk tidak berubah digunakan ulang |
| Berkas berbeda | Konten tidak berkaitan | Penulisan chunk baru dan pembentukan manifest | Objek dapat direkonstruksi |
| Tidak valid atau terputus | Berkas kosong; unggah berhenti sebelum commit | Validasi input dan crash consistency | Tidak terbentuk objek sah |
| Data reklamasi | Referensi nol, re-reference, masa tenggang, unggah dan pembacaan aktif | Keselamatan reklamasi konkuren | Hanya chunk yang lolos validasi akhir dihapus |

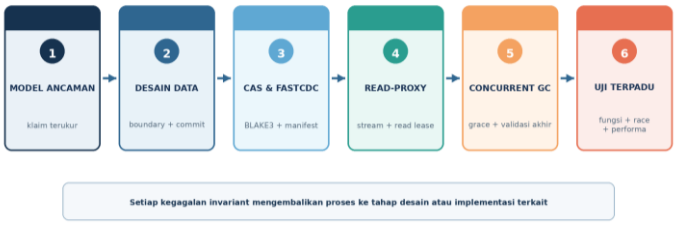
Setiap kelompok dapat menguji beberapa properti. Berkas identik dan perubahan sebagian menilai penggunaan ulang *chunk* serta stabilitas batas FastCDC. Unggah terputus memeriksa bahwa objek belum sah sebelum *manifest* dikomit. Data reklamasi memeriksa apakah GC hanya menghapus *chunk* yang tetap tanpa referensi dan tidak aktif pada validasi akhir.

## 2.2 Tahapan Penelitian

Penelitian berlangsung dalam enam tahap: identifikasi ancaman dan klaim keamanan; perancangan model data, batas kepercayaan, serta protokol *commit*; implementasi CAS, FastCDC, BLAKE3, *manifest*, dan antarmuka; penambahan *Read-Proxy* dengan *read lease*; integrasi GC pada *Store* yang sama dengan *handler* UDS, masa tenggang, dan validasi akhir; serta pengujian fungsi, konsistensi, keamanan, deduplikasi, dan konkurensi memakai konfigurasi komponen produksi. Urutannya ditunjukkan pada Gambar 1.

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. 
This work is licensed under a Creative Commons Attribution 4.0 International License.

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 6, October 2026 | Hal 2590-2600
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i6.1385

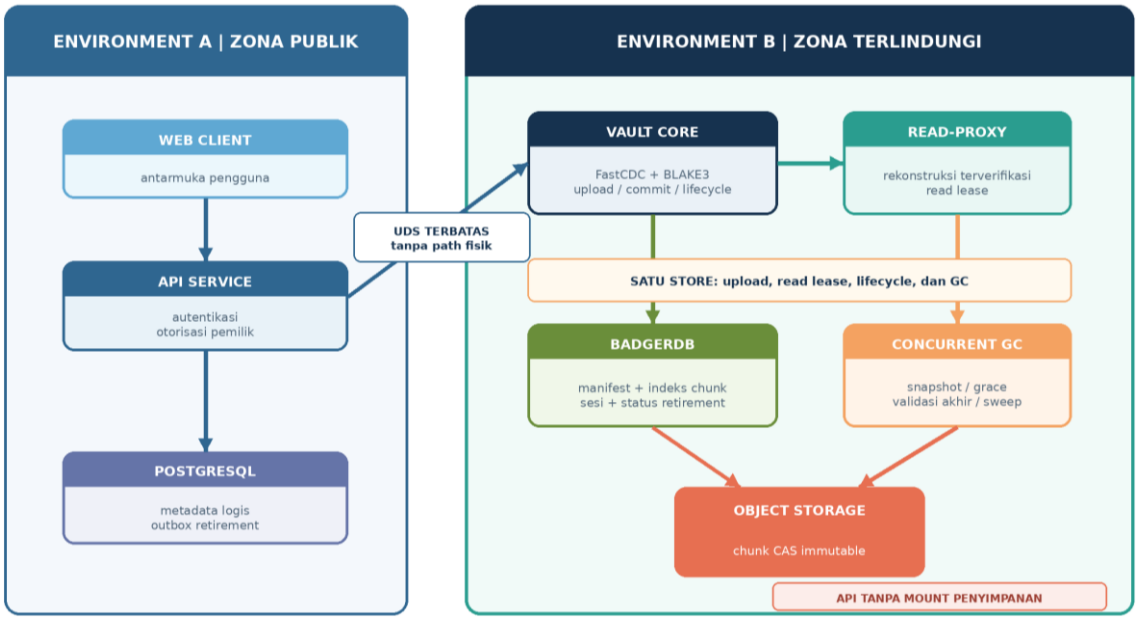



**Gambar 1.** Tahapan penelitian pengembangan sistem

Gambar 1 menempatkan pengujian terpadu setelah arsitektur lengkap terbentuk. Setiap tahap menghasilkan masukan untuk tahap berikutnya, seperti spesifikasi protokol, struktur kunci BadgerDB, dan skenario uji. Ketidaksesuaian diperbaiki pada komponen terkait sebelum rangkaian diuji ulang.

### 2.3 Arsitektur Sistem

Arsitektur membagi sistem menjadi Environment A (zona publik) dan Environment B (zona terlindungi). Environment A memuat klien web, API Service, autentikasi, dan PostgreSQL; Environment B memuat *Vault Core*, BadgerDB, *object storage*, *Read-Proxy*, dan penjadwal GC. API tidak mempunyai *mount* direktori *chunk*. Melalui UDS, API hanya dapat menjalankan operasi terdefinisi untuk unggah, *commit*, pembacaan, pemeriksaan status, serta transisi retire/*retain*. Protokol tidak menerima jalur mentah atau operasi penimpaan dan penghapusan *chunk* langsung.




**Gambar 2.** Arsitektur sistem setelah integrasi Read-Proxy dan Concurrent Garbage Collection

Gambar 2 menempatkan penulisan, pembacaan, dan penghapusan fisik pada *Vault Core*. Handler UDS dan penjadwal GC memakai satu *Store*, sehingga keduanya memeriksa kunci sinkronisasi dan registri aktivitas yang sama. API yang dikuasai dapat mengubah metadata logis atau meminta *retirement*, tetapi tidak langsung mengakses *chunk*. Masa tenggang, pemeriksaan referensi ulang, *read lease*, dan audit menyediakan jendela deteksi sebelum reklamasi. Batas ini menerapkan hak minimum dan segmentasi, bukan perlindungan permanen ketika kompromi tidak terdeteksi [14].

Unggah dimulai dengan pembuatan sesi. *Vault Core* memecah aliran berkas memakai FastCDC dan menghitung BLAKE3 setiap *chunk*. Indeks BadgerDB menentukan apakah *chunk* digunakan ulang atau ditulis melalui berkas sementara lalu dipindahkan secara atomik. Manifest dibentuk setelah semua *chunk* lolos pemeriksaan, dan API menerima identitasnya hanya setelah *commit* berhasil. Dengan demikian, metadata aplikasi tidak menunjuk objek yang belum lengkap.

### 2.4 Implementasi Read-Proxy

*Read-Proxy* menerima identitas *manifest* melalui UDS setelah API mengautentikasi pengguna, memeriksa kepemilikan, dan memvalidasi status berkas. *Vault Core* memverifikasi *peer* UDS dan format identitas, lalu membaca urutan *chunk* tanpa menerima jalur fisik. BLAKE3 setiap *chunk* dan hash akhir objek diverifikasi; kegagalan menghentikan aliran. API tetap menangani otorisasi pengguna, sedangkan *Vault Core* memvalidasi layanan pemanggil, *manifest*, dan integritas konten.

Proksi melakukan *streaming* melalui io.Pipe agar tekanan balik mengikuti penerima tanpa memuat seluruh objek ke memori. Pada mode ketat untuk objek hingga batas konfigurasi, rekonstruksi diselesaikan di berkas sementara sebelum dikirim; objek lebih besar diverifikasi per *chunk* dan pada akhir rekonstruksi. Setiap pembacaan memperoleh *read lease*

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. Page 2593
This work is licensed under a Creative Commons Attribution 4.0 International License.

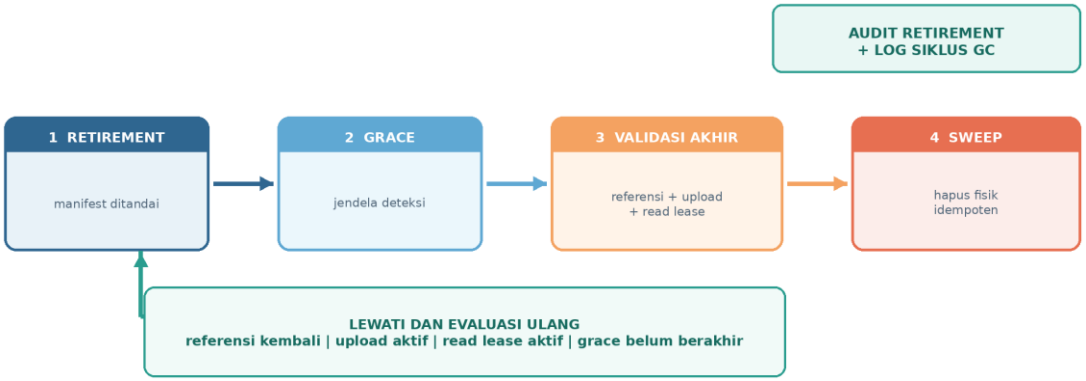
BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 6, October 2026 | Hal 2590-2600
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i6.1385

dan melepaskannya melalui defer pada keberhasilan, kesalahan, atau pembatalan. GC memakai registri lease yang sama sehingga *chunk* aktif tidak dihapus.

### 2.5 Implementasi Concurrent Garbage Collection

*Concurrent Garbage Collection* memakai alur *manifest snapshot–grace–final validation–sweep*. Pemindaian membentuk himpunan *chunk* yang dirujuk dan kandidat tanpa referensi. Kandidat dipertimbangkan setelah waktu *retirement* atau pembuatannya melewati masa tenggang. Registri unggah aktif dan *read lease* berasal dari *Store* yang sama dengan *handler* UDS.

Sesaat sebelum penghapusan, GC mengunci koordinasi siklus hidup dan memvalidasi ulang referensi, *retirement*, masa tenggang, sesi unggah, *read lease*, serta identitas dan jalur *chunk*. Re-reference atau aktivitas baru membuat kandidat dilewati. Sweep menghapus berkas lalu metadata; berkas yang telah hilang diperlakukan idempoten agar eksekusi ulang menuntaskan kegagalan parsial. API mencatat niat dan pengiriman *retirement* melalui *outbox*, sedangkan *Vault Core* mencatat ringkasan siklus GC.




**Gambar 3.** Alur validasi dan reklamasi Concurrent Garbage Collection

Gambar 3 menegaskan bahwa *retirement* bukan penghapusan. Setelah masa tenggang, GC memvalidasi kembali referensi, unggah, dan pembacaan aktif. Perubahan invarian membuat kandidat dilewati, sedangkan penghapusan idempoten memungkinkan siklus berikutnya melanjutkan kegagalan parsial.

### 2.6 Rancangan Pengujian

Dasar hasil mencakup uji unit dengan *race detector*, uji konfigurasi komponen produksi, UDS, integritas, simulasi operasi destruktif, injeksi kegagalan, dan pengukuran performa. Keluaran dibandingkan dengan spesifikasi unggah, unduh, deduplikasi, *soft delete*, pemulihan, serta penolakan masukan tidak valid [21]. Konsistensi diuji dengan menghentikan proses sebelum *commit*, membuka ulang BadgerDB, dan menyisipkan kegagalan antara penghapusan berkas dan metadata. Pengujian keamanan mengasumsikan API dikuasai lalu mencoba metode fisik terlarang, jalur arbitrer, dan *manifest* tidak valid. Pengujian *black-box* antarmuka pengguna (UI) serta Security Lab pada *stack* hidup dipisahkan karena bukan sumber angka penelitian.

*Read-Proxy* diuji melalui kesamaan hash sumber–rekonstruksi, *chunk* hilang, traversal jalur, pembatalan, dan unduh bersamaan. GC diuji pada kandidat layak reklamasi, *re-reference*, unggah aktif, pembacaan aktif, dan kegagalan parsial. Performance harness memakai *handler* HTTP produksi melalui UDS dan satu *Store* bersama. Dataset terdiri atas empat objek 8 MiB: unggah pertama, identik, berubah sebagian, dan berbeda; total 33.554.432 byte. FastCDC memakai ukuran minimum 1 KiB, rata-rata 4 KiB, dan maksimum 8 KiB dengan BLAKE3-256. Pengukuran berlangsung 159,468 detik dengan lima warm-up, 30 sampel baca per titik, 10 sampel unggah, 1, 4, dan 16 klien, 10 siklus GC, serta 30 rekonstruksi pasca-GC. Masa tenggang dikonfigurasi satu jam; kelayakan kandidat diuji melalui cutoff waktu terkontrol pada harness, bukan dengan menunggu satu jam waktu nyata.

## 3. HASIL DAN PEMBAHASAN

### 3.1 Hasil Implementasi Arsitektur

Penelitian menghasilkan prototipe *Immutable Object Storage* lengkap pada lingkungan *single-node*. Sistem memisahkan komponen publik dan komponen penyimpanan melalui proses serta hak akses sistem operasi yang berbeda. API Service menangani autentikasi, hierarki direktori, dan metadata logis pada PostgreSQL. *Vault Core* menangani FastCDC, BLAKE3, deduplikasi, *manifest*, rekonstruksi, dan reklamasi fisik. BadgerDB dipakai karena menyediakan transaksi dan pola akses kunci-nilai yang sesuai untuk indeks *chunk*. Implementasi mengacu pada kemampuan transaksi dan pengelolaan value log BadgerDB, tetapi kebijakan GC objek penelitian tetap dikelola pada lapisan aplikasi agar hubungan antar*manifest* dapat diverifikasi [22].

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. 
This work is licensed under a Creative Commons Attribution 4.0 International License.

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 6, October 2026 | Hal 2590-2600
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i6.1385

Pemisahan komponen berhasil menghapus kebutuhan kredensial penyimpanan fisik dari API. Direktori objek dimiliki akun layanan *Vault Core*, sedangkan API hanya memiliki izin berkomunikasi melalui UDS. Daftar operasi UDS bersifat eksplisit dan tidak menerima jalur berkas mentah dari klien. Identitas objek dibentuk dari *manifest* yang telah lolos validasi. Perubahan ini mempersempit permukaan serangan dibandingkan rancangan yang memberi API akses *read-only mount*. NIST menempatkan pengamanan antarmuka pengelolaan dan pemisahan kontrol sebagai bagian penting keamanan infrastruktur penyimpanan [13].

Metadata aplikasi dan metadata penyimpanan tetap dibedakan. PostgreSQL menyimpan nama berkas, pemilik, direktori, status, waktu unggah, *soft delete*, penandaan bintang, dan *outbox retirement*. BadgerDB menyimpan *manifest*, indeks *chunk*, sesi unggah, dan status *retirement*. Read lease dan registri unggah aktif berada pada satu instans *Store* dalam proses *Vault Core*. Hubungan kedua basis data menggunakan identitas *manifest* yang dikembalikan setelah *commit*. Jika API gagal setelah *commit* tetapi sebelum menyimpan metadata, objek fisik tetap tidak langsung dihapus; jika *Vault Core* gagal sebelum *commit*, sesi kedaluwarsa tanpa membentuk objek sah.

### 3.2 Hasil Penyimpanan dan Deduplikasi

Pengujian unggah berkas baru menunjukkan bahwa sistem dapat menghasilkan rangkaian *chunk*, menghitung BLAKE3, menulis data baru, dan membentuk *manifest*. Pengujian berkas identik menghasilkan rangkaian identitas yang sama sehingga tidak memerlukan penulisan ulang *chunk* fisik. Sistem tetap dapat membuat metadata logis baru yang menunjuk pada objek sama. Pada berkas dengan perubahan sebagian, FastCDC mempertahankan penggunaan ulang *chunk* di bagian yang tidak berubah dan hanya menambahkan konten baru pada wilayah terdampak. Hasil ini konsisten dengan tujuan FastCDC untuk mengurangi pergeseran batas yang umum terjadi pada pemotongan berukuran tetap [7].

Tabel 2 menyajikan dua metrik yang berbeda. Rasio penggunaan ulang *chunk* dihitung sebagai jumlah *chunk* yang digunakan ulang dibagi jumlah *chunk* total. Penghematan byte dihitung sebagai satu dikurangi perbandingan byte fisik baru terhadap byte logis yang diterima. Pemisahan ini mencegah istilah rasio deduplikasi digunakan untuk dua rumus yang berbeda. Pengujian juga mencatat distribusi ukuran *chunk* dan waktu pemrosesan agar bukti kuantitatif dapat ditelusuri melalui log eksekusi.

**Tabel 2.** Hasil deduplikasi berdasarkan variasi berkas

| Skenario | Chunk total | Chunk baru | Digunakan ulang | Rasio penggunaan ulang | Penghematan byte |
|---|---|---|---|---|---|
| Unggah pertama | 1.966 | 3 | 1.963 | 99,8474% | 99,8511% |
| Berkas identik | 1.966 | 0 | 1.966 | 100,0000% | 100,0000% |
| Perubahan sebagian | 1.958 | 9 | 1.949 | 99,5403% | 99,1353% |
| Berkas berbeda | 1.024 | 251 | 773 | 75,4883% | 75,4883% |

Tabel 2 menunjukkan karakteristik fixture sintetis yang sangat repetitif. Pada unggah pertama, hanya tiga identitas *chunk* bersifat baru, sedangkan 1.963 kemunculan berikutnya digunakan ulang di dalam objek yang sama. Berkas identik menggunakan ulang seluruh 1.966 *chunk*. Perubahan 64 KiB di bagian tengah menghasilkan sembilan *chunk* baru dan tetap menggunakan ulang 1.949 *chunk*. Berkas berbeda menghasilkan 251 *chunk* baru, tetapi masih memuat 773 pengulangan internal. Secara agregat, 6.651 dari 6.914 *chunk* digunakan ulang; 2.141.218 byte fisik baru ditulis untuk 33.554.432 byte logis, sehingga rasio penggunaan ulang mencapai 96,1961% dan penghematan byte 93,6187%.

Rekonstruksi objek berhasil dilakukan dengan membaca urutan hash pada *manifest* dan menggabungkan *chunk* sesuai posisi. BLAKE3 akhir sumber dan hasil rekonstruksi bernilai sama; bukti pengujian mencatat awalan 95c60b7f dan akhiran b8b1925. Sistem menolak hasil apabila satu *chunk* hilang atau nilai hash berubah. Verifikasi per *chunk* mencegah proksi mengalirkan data yang tidak lagi sesuai dengan identitas kontennya. Penggunaan BLAKE3 juga menghindari kebutuhan metadata lokasi sebagai sumber identitas utama karena alamat diturunkan dari konten [9].

### 3.3 Hasil Read-Proxy

*Read-Proxy* berhasil menjadi mekanisme pembacaan yang digunakan API. API mengirim identitas *manifest* setelah otorisasi pengguna, lalu menerima aliran byte tanpa mengetahui lokasi *chunk*. Proksi memvalidasi *manifest*, memperoleh *read lease*, membuka *chunk* berurutan, memverifikasi BLAKE3, dan meneruskan data dengan back-pressure. Lease dilepas pada keberhasilan, kesalahan, atau pembatalan, sedangkan GC mempertahankan *chunk* yang terkait dengan pembacaan aktif.

Pengujian normal menunjukkan berkas yang direkonstruksi melalui proksi mempunyai hash akhir sama dengan sumber. Pengujian *manifest* tidak valid menghasilkan penolakan tanpa membuka direktori fisik. Masukan yang menyerupai traversal jalur juga ditolak karena antarmuka hanya menerima identitas dengan format terbatas dan tidak melakukan penggabungan jalur dari input pengguna. Ketika satu *chunk* disimulasikan hilang, proksi menghentikan respons dan mencatat kegagalan integritas. Perilaku tersebut lebih aman daripada mengembalikan data parsial yang mungkin dianggap sah oleh klien.

Pengujian konkurensi menjalankan beberapa unduh ketika unggah dan GC berlangsung. Registri aktivitas dan kunci siklus hidup pada *Store* bersama mencegah GC menghapus *chunk* yang sedang ditulis, dikomit, direferensikan ulang, atau dibaca. *Context cancellation* menghentikan aliran ketika klien memutus koneksi. Pengukuran memakai 1, 4, dan 16 klien; Tabel 3 menampilkan titik beban tertinggi, yaitu 16 klien dengan 30 sampel dan kesalahan pembacaan nol. Metrik yang dicatat meliputi median, persentil ke-95, *throughput*, serta puncak *HeapAlloc* setelah setiap batch.

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. Page 2595
This work is licensed under a Creative Commons Attribution 4.0 International License.

BULLETIN OF COMPUTER SCIENCE RESEARCH
ISSN 2774-3659 (Media Online)
Vol 6, No 6, October 2026 | Hal 2590-2600
https://hostjournals.com/bulletincsr
DOI: 10.47065/bulletincsr.v6i6.1385

**Tabel 3.** Metrik pengujian Read-Proxy

| Metrik | Pembacaan lokal terkontrol | Read-Proxy | Perubahan | Catatan |
|---|---|---|---|---|
| Latensi median | 1.049,950 ms | 1.108,762 ms | +5,6014% | Objek 8 MiB |
| Latensi persentil ke-95 | 1.132,714 ms | 1.240,129 ms | +9,4830% | 16 klien; 30 sampel |
| Throughput | 121,911 MiB/s | 115,444 MiB/s | -5,3043% | 0 kesalahan sampel |
| Puncak HeapAlloc | 222,866 MiB | 225,405 MiB | +1,1394% | Back-pressure aktif |

Tabel 3 membandingkan pembacaan lokal terkontrol dan *Read-Proxy* pada perangkat keras yang sama. Pada 16 klien, median meningkat 5,6014% dan persentil ke-95 meningkat 9,4830% karena validasi, komunikasi antarproses, dan *streaming*. Throughput turun 5,3043%, dari 121,911 menjadi 115,444 MiB/s, sedangkan puncak *HeapAlloc* naik 1,1394%, dari 222,866 menjadi 225,405 MiB. Seluruh 30 sampel selesai tanpa kesalahan. Biaya tambahan tersebut disertai pengurangan hak akses API karena *Vault Core* memverifikasi identitas *peer* UDS, format *manifest*, dan integritas konten setelah API melakukan otorisasi pengguna [14].

### 3.4 Hasil Concurrent Garbage Collection

Pengumpulan sampah dapat berjalan tanpa menghentikan layanan. GC mengambil snapshot *manifest* aktif dan memeriksa waktu *retirement* atau usia *chunk* tanpa referensi terhadap masa tenggang. Sebelum *sweep*, validasi akhir di bawah koordinasi *Store* bersama memeriksa kembali referensi, status *retirement*, unggah aktif, dan *read lease*. Chunk yang kembali dirujuk atau sedang digunakan dilewati. Hanya *chunk* yang tetap memenuhi seluruh invarian yang dihapus dari sistem berkas dan kemudian dibersihkan dari indeks.

Pengujian *chunk* tanpa referensi menunjukkan bahwa ruang dapat direklamasi setelah masa tenggang berakhir. Pengujian *re-reference* membuat GC melewati *chunk* ketika *manifest* aktif kembali menunjuknya. Pengujian pembacaan dan unggah aktif mempertahankan *chunk* sampai lease atau sesi dilepas. Pada kegagalan di tengah *sweep*, urutan hapus-berkas-lalu-metadata dan toleransi terhadap berkas yang sudah tidak ada membuat eksekusi berikutnya menyelesaikan operasi secara idempoten. Catatan *outbox retirement* dan ringkasan siklus GC menyediakan jejak minimum bagi pemeriksaan operasional.

Tabel 4 merangkum keselamatan dan efisiensi GC berdasarkan laporan terstruktur. Skenario layak reklamasi menghapus 251 *chunk* atau 2.056.192 byte. Re-reference, pembacaan aktif, dan unggah aktif menghasilkan nol penghapusan. Injeksi kegagalan setelah berkas dihapus tetapi sebelum metadata dibersihkan menghasilkan satu kesalahan pada sapuan pertama; eksekusi berikutnya menuntaskan metadata secara idempoten. Pada beban 16 klien, *throughput Read-Proxy* berubah dari 115,444 menjadi 104,859 MiB/s ketika GC aktif, atau turun 9,1692%. Penjadwalan karena itu tetap perlu mempertimbangkan beban penyimpanan [10], [12].

**Tabel 4.** Hasil pengujian Concurrent Garbage Collection

| Skenario | Diperiksa | Dilewati | Dihapus | Ruang direklamasi | Hasil |
|---|---|---|---|---|---|
| Referensi nol dan retensi berakhir | 263 | 12 | 251 | 2.056.192 byte | Sesuai; 0 error |
| Referensi muncul kembali | 13 | 13 | 0 | 0 byte | Sesuai |
| Objek sedang dibaca | 17 | 17 | 0 | 0 byte | Sesuai; 4 read lease |
| Sesi unggah masih aktif | 14 | 14 | 0 | 0 byte | Sesuai; 1 upload aktif |
| Kegagalan saat sweep | 22 | 13 | 8 | 61.440 byte | 1 error; retry berhasil |

Tabel 4 memperlihatkan bahwa keberhasilan GC tidak cukup dinilai dari ruang yang dibebaskan. Kolektor mencapai nol penghapusan pada *chunk* yang kembali dirujuk, sedang dibaca, atau masih berada dalam sesi unggah. Sepuluh siklus GC mereklamasi 251 *chunk* layak dan 30 rekonstruksi pasca-GC seluruhnya berhasil. Validasi akhir, konsistensi indeks, serta pemulihan idempoten menjadi syarat sebelum reklamasi dinyatakan benar. Hasil tersebut membatasi klaim pada *dataset* terkontrol dan tidak menggantikan pengujian kegagalan media fisik.

### 3.5 Hasil Pengujian Fungsional, Konsistensi, dan Keamanan

Pengujian fungsional mencakup alur normal dan tidak normal pada lapisan CAS, UDS, API Service, serta uji konfigurasi komponen produksi. Alur normal meliputi unggah, deduplikasi, unduh, *soft delete*, pemulihan, *retirement*, *retain*, dan reklamasi setelah retensi. Alur tidak normal meliputi *manifest* hilang atau rusak, penghentian unggah sebelum *commit*, pemutusan unduh, traversal jalur, operasi fisik terlarang, dan kegagalan layanan selama GC. Tabel 5 menyajikan hasil serta bukti ringkas dari nama pengujian dan laporan performa pada *commit* 9077757.

**Tabel 5.** Hasil pengujian fungsional, konsistensi, dan keamanan

| No. | Skenario | Hasil yang diharapkan | Status dan bukti |
|---|---|---|---|
| 1 | Unggah berkas valid | Chunk dan manifest terbentuk setelah commit | PASS; manifest dan metadata chunk terbentuk setelah commit. |
| 2 | Unggah berkas identik | Chunk fisik digunakan ulang | PASS; 1.966 dari 1.966 chunk digunakan ulang. |

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. Page 2596
This work is licensed under a Creative Commons Attribution 4.0 International License.

| No. | Skenario | Hasil yang diharapkan | Status dan bukti |
|---|---|---|---|
| 3 | Perubahan sebagian | Hanya bagian berubah menambah chunk | PASS; 1.949 dari 1.958 chunk digunakan ulang. |
| 4 | Unduh melalui Read-Proxy | Hash hasil sama dengan sumber | PASS; BLAKE3 sumber dan hasil identik; 30 rekonstruksi pasca-GC berhasil. |
| 5 | Soft delete dan pemulihan | Hanya metadata logis berubah | PASS pada uji service; hanya metadata logis berubah. |
| 6 | Unggah terputus | Objek tidak dianggap sah sebelum commit | PASS; manifest tidak dikomit dan orphan direklamasi setelah sesi kedaluwarsa. |
| 7 | Manifest tidak valid | Permintaan ditolak tanpa data parsial | PASS; permintaan ditolak tanpa data parsial. |
| 8 | Traversal jalur | Masukan ditolak sebelum akses fisik | PASS; HTTP 400 sebelum akses fisik. |
| 9 | Operasi destruktif dari API | Vault Core menolak hapus/timpa fisik langsung | PASS; metode DELETE, PUT, dan PATCH fisik ditolak dengan HTTP 403. |
| 10 | GC pada chunk layak reklamasi | Chunk dihapus setelah validasi akhir | PASS; 251 chunk atau 2.056.192 byte direklamasi. |
| 11 | GC saat re-reference | Chunk dilewati pada validasi akhir | PASS; 13 chunk dilewati dan tidak ada yang dihapus. |
| 12 | GC saat pembacaan aktif | Chunk dipertahankan hingga pembacaan selesai | PASS; 4 read lease dan 13 chunk terreferensi dilewati; tidak ada penghapusan. |
| 13 | Kegagalan saat GC | Eksekusi ulang menyelesaikan state secara idempoten | PASS; satu fault terdeteksi dan retry menuntaskan metadata secara idempoten. |

Tabel 5 menunjukkan bahwa API tidak dapat menghapus *chunk* secara langsung. Soft delete hanya mengubah metadata logis, sedangkan penghapusan permanen yang sah dapat menghasilkan permintaan *retirement* melalui *outbox*. Penyerang yang menguasai API karena itu masih dapat memicu niat reklamasi, tetapi tidak memperoleh penghapusan fisik seketika. Masa tenggang, *retain* ketika referensi muncul kembali, validasi akhir, dan audit memberi operator jendela untuk mendeteksi anomali, menghentikan GC, serta memulihkan metadata sebelum *sweep*.

Pengujian konsistensi memutus proses sebelum dan sesudah pembentukan *manifest*. Sebelum *commit*, sistem tidak mengembalikan identitas objek sah kepada API dan data tanpa referensi baru layak dipertimbangkan setelah sesi kedaluwarsa serta masa tenggang. Setelah *commit*, *manifest* menjadi sumber urutan rekonstruksi. Jika proses berhenti setelah berkas *chunk* dihapus tetapi sebelum metadata dibersihkan, siklus GC berikutnya menerima keadaan berkas tidak ditemukan dan menyelesaikan penghapusan metadata secara idempoten. Pendekatan ini menghindari kebutuhan transaksi terdistribusi antara PostgreSQL, BadgerDB, dan sistem berkas.

Simulasi kompromi API menguji batas klaim keamanan. Protokol UDS menolak metode penimpaan atau penghapusan fisik dan tidak menerima jalur berkas, sedangkan API tidak mempunyai *mount* penyimpanan. Namun, karena API dapat mengirim transisi retire melalui alur penghapusan permanen, kompromi API yang bertahan lebih lama daripada masa tenggang tetap berpotensi menyebabkan reklamasi. Perlindungan yang diberikan adalah pemisahan akses fisik, penundaan, validasi ulang, audit, dan kesempatan respons operator—bukan imunitas permanen. Penelitian juga tidak mencakup kompromi akun sistem operasi *Vault Core*, perubahan biner tepercaya, atau penguasaan perangkat penyimpanan [23].

### 3.6 Pembahasan

VaultFS menunjukkan bahwa semantik *write-once* pada lapisan sistem berkas dapat mempertahankan data bahkan dari proses dengan hak tinggi [5]. Penelitian ini memiliki sasaran yang sejalan, tetapi bekerja pada lapisan penyimpanan objek dan menambahkan deduplikasi tingkat *chunk*. Pendekatan tersebut memungkinkan beberapa versi logis menggunakan konten fisik sama. Perbedaannya, kekuatan perlindungan penelitian masih bergantung pada isolasi proses dan izin sistem operasi pada lingkungan *single-node*, sedangkan VaultFS menegakkan kebijakan pada sistem berkas. Karena itu, hasil penelitian lebih tepat diposisikan sebagai pelengkap arsitektural, bukan pengganti perlindungan *kernel* atau WORM perangkat keras.

Reverse File System memindahkan penegakan WORM ke perangkat terpisah sehingga komputer inang tidak dapat mengubah data yang telah dikomit [6]. Pemisahan fisik tersebut menawarkan batas kepercayaan lebih kuat daripada dua proses pada satu host. Sebaliknya, prototipe penelitian lebih mudah direplikasi pada perangkat komoditas dan mendukung deduplikasi, antarmuka objek, serta reklamasi berbasis referensi. Pengembangan berikutnya dapat memindahkan Environment B ke host terpisah tanpa mengubah prinsip protokol. UDS dapat diganti kanal terautentikasi dengan identitas layanan dan kebijakan jaringan eksplisit.

Rhea dan ROFBS berfokus pada deteksi atau intersepsi aktivitas sebelum kerusakan meluas [3], [4]. Penelitian ini bekerja setelah data dicadangkan. *Vault Core* mencegah API menyentuh penyimpanan fisik secara langsung, *Read-Proxy* memverifikasi integritas saat data dibaca, dan masa tenggang GC menyediakan jendela deteksi ketika perubahan aplikasi menghasilkan *retirement* yang tidak sah. Strategi berlapis tetap diperlukan karena keterlambatan deteksi yang melewati masa tenggang dapat mengurangi titik pemulihan.

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. Page 2597
This work is licensed under a Creative Commons Attribution 4.0 International License.

RSSD, RansomTag, dan FFRecovery mempertahankan atau memulihkan versi data melalui mekanisme perangkat keras maupun lintas lapisan [16]–[18], sedangkan deteksi penimpaan berbasis bitmap menekan biaya pencatatan di SSD [19]. RanSMAP menyediakan dasar evaluasi detektor perilaku pada variasi perangkat keras [20]. Prototipe ini melengkapi fokus tersebut melalui koordinasi *manifest* dan aktivitas di tingkat objek; prototipe tidak mengimplementasikan detektor berbasis RanSMAP ataupun perlindungan firmware. Oleh karena itu, angka penghematan kapasitas, latensi, dan reklamasi yang diperoleh tidak digunakan untuk menyatakan keunggulan deteksi atau performa atas sistem terdahulu yang memiliki perangkat, beban, dan model ancaman berbeda.

Kontribusi *Concurrent Garbage Collection* menyelesaikan ketegangan antara *immutability* dan keberlanjutan kapasitas. Sistem yang tidak pernah menghapus data akan terus tumbuh, sedangkan penghapusan langsung oleh aplikasi merusak batas keamanan. Alur snapshot–*grace–final validation–sweep* menempatkan reklamasi sebagai kebijakan internal yang tertunda dan dapat diaudit. Invarian referensi, unggah, dan pembacaan diverifikasi tepat sebelum *sweep*. Hasilnya adalah *immutability* terhadap akses fisik langsung dari aplikasi, bukan larangan permanen atas seluruh reklamasi.

Kontribusi *Read-Proxy* mengubah model kepercayaan pembacaan. API hanya berinteraksi dengan objek logis setelah otorisasi pengguna; *Vault Core* memeriksa identitas *peer* UDS, validitas *manifest*, dan integritas konten. Read lease membuat aktivitas rekonstruksi terlihat oleh GC, sedangkan payload UDS terbatas pada identitas objek. Pada 16 klien, biaya proksi berupa kenaikan median 5,6014%, kenaikan p95 9,4830%, dan penurunan *throughput* 5,3043%; tidak ditemukan kesalahan pada 30 sampel pembacaan.

Hasil penelitian dibatasi oleh lingkungan *single-node*, *dataset* sintetis 32 MiB, parameter FastCDC tunggal, dan skala 16 klien. Uji Go dengan *race detector*, konfigurasi komponen produksi, UDS, serta *performance harness* telah lulus. Namun, integrasi *outbox* PostgreSQL dan pengiriman worker pada layanan eksternal tidak dieksekusi karena URL basis data dan jalur UDS integrasi tidak dikonfigurasi. Security Lab pada API hidup juga dilewati, sedangkan Playwright tidak mencapai assertion karena Chromium tidak dapat diluncurkan pada lingkungan pengujian. Karena itu, kesimpulan tidak mencakup kegagalan disk nyata, kompromi akun *Vault Core*, replikasi, *high availability*, maupun validasi antarmuka pengguna menyeluruh; latihan pemulihan berkala tetap diperlukan [24].

Walaupun memiliki keterbatasan, prototipe menunjukkan integrasi yang koheren antara proteksi, efisiensi, pembacaan terkontrol, dan reklamasi ruang. CAS dan FastCDC mengurangi duplikasi, *manifest* menjaga urutan rekonstruksi, BLAKE3 mengikat identitas dengan konten, *Read-Proxy* mempersempit akses, dan GC mengendalikan pertumbuhan kapasitas. Kombinasi tersebut menjawab rumusan masalah pengembangan sistem penyimpanan cadangan yang tetap efisien tanpa memberi aplikasi publik kendali destruktif langsung terhadap repositori fisik.

# 4. KESIMPULAN

Penelitian ini menjawab rumusan masalah pengembangan repositori cadangan yang membatasi akses destruktif langsung dari API publik, mengurangi duplikasi konten, dan mereklamasi ruang tanpa menghapus *chunk* yang masih digunakan pada skenario terkontrol. Prototipe *Immutable Object Storage* berbasis CAS dan *FastCDC* memisahkan API Service dari *Vault Core*, menyediakan rekonstruksi terverifikasi melalui *Read-Proxy*, serta mengintegrasikan *Concurrent Garbage Collection* pada lingkungan *single-node*. API tidak mempunyai mount penyimpanan, sedangkan handler UDS dan GC memakai satu *Store* dengan *read lease*, registri unggah aktif, masa tenggang, dan validasi akhir. Pembatasan otoritas didukung oleh penolakan operasi hapus dan timpa fisik langsung pada pengujian protokol. Efisiensi kapasitas ditunjukkan oleh empat unggahan berukuran total 32 MiB yang menghasilkan rasio penggunaan ulang *chunk* 96,1961% dan penghematan byte 93,6187%. Pada 16 klien, median *Read-Proxy* mencapai 1.108,762 ms dan tidak ditemukan kesalahan pada 30 sampel pembacaan. Keselamatan reklamasi didukung oleh tidak adanya penghapusan pada skenario *re-reference*, pembacaan aktif, dan unggah aktif. Sepuluh siklus GC mereklamasi 251 *chunk* atau 2.056.192 byte; seluruh 30 rekonstruksi pasca-GC berhasil, dan eksekusi ulang menuntaskan kegagalan parsial secara idempoten. Kontribusi penelitian terletak pada integrasi deduplikasi, pembacaan tanpa akses direktori dari API, serta reklamasi berbasis validasi referensi dan aktivitas dalam satu arsitektur. Penerapan bersama mekanisme tersebut memiliki biaya: *throughput* pembacaan pada 16 klien turun 9,1692% ketika GC aktif, sehingga jadwal reklamasi perlu disesuaikan dengan beban. Klaim keamanan terbatas pada pemisahan akses fisik dan jendela respons sebelum *sweep*; kompromi API yang tidak terdeteksi masih dapat menyalahgunakan *retirement* setelah masa tenggang. Audit, pemantauan, mekanisme retain, dan respons operator karena itu tetap diperlukan. Besarnya penghematan juga mencerminkan karakter dataset sintetis yang repetitif dan tidak dapat langsung digeneralisasi ke data produksi. Evaluasi terbatas pada satu perangkat, dataset 32 MiB, satu konfigurasi *FastCDC*, dan maksimum 16 klien; integrasi stack hidup serta antarmuka pengguna belum tervalidasi menyeluruh. Penelitian berikutnya perlu menguji data nyata dengan kemiripan beragam, beban lebih besar, kegagalan disk, pemisahan *Vault Core* ke host lain, replikasi, high availability, autentikasi kanal antarmesin, dan latihan pemulihan berkala.

# 5. PERNYATAAN PENULIS

### 5.1 Kontribusi Penulis

Konseptualisasi: A. G. S., dan M. A. R.;

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. Page 2598
This work is licensed under a Creative Commons Attribution 4.0 International License.

Metodologi: A. G. S., dan M. A. R.;
Perangkat Lunak: A. G. S.;
Investigasi: A. G. S.;
Analisis Formal: A. G. S.;
Kurasi Data: A. G. S.;
Visualisasi: A. G. S.;
Validasi: A. G. S., dan M. A. R.;
Penulisan Draf Awal: A. G. S.;
Penelaahan Dan Penyuntingan: A. G. S., dan M. A. R.;
Supervisi: M. A. R.;
Seluruh Penulis Telah Membaca Dan Menyetujui Versi Akhir Manuskrip.

### 5.2 Ketersediaan Data

Kode sumber, skenario pengujian, dan artefak hasil yang mendukung penelitian tersedia pada repositori https://github.com/Alfian57/Hashbox-PUI.

### 5.3 Pendanaan

Penelitian ini dilaksanakan tanpa pendanaan eksternal.

### 5.4 Pernyataan Dewan Peninjau Institusional

Tidak berlaku karena penelitian tidak melibatkan manusia atau hewan sebagai subjek penelitian.

### 5.5 Pernyataan Persetujuan Berdasarkan Informasi

Tidak berlaku karena penelitian tidak melibatkan partisipan manusia.

### 5.6 Konflik Kepentingan

Para penulis menyatakan tidak terdapat konflik kepentingan dalam penelitian dan publikasi manuskrip ini.

### 5.7 Penggunaan AI Generatif dan Teknologi Berbantuan AI dalam Proses Penulisan

Selama penyiapan manuskrip, penulis menggunakan teknologi kecerdasan artifisial generatif untuk membantu pemeriksaan keterbacaan, konsistensi istilah, dan kesesuaian format. Seluruh keluaran ditelaah dan disunting oleh penulis, yang bertanggung jawab penuh atas isi manuskrip.

## REFERENCES

[1] Cybersecurity and Infrastructure Security Agency, "#StopRansomware Guide," 2023. [Online]. Available: https://www.cisa.gov/stopransomware/ransomware-guide. [Accessed: Sep. 8, 2026].

[2] Sophos, "The Impact of Compromised Backups on Ransomware Outcomes," 2024. [Online]. Available: https://www.sophos.com/en-us/blog/the-impact-of-compromised-backups-on-ransomware-outcomes. [Accessed: Sep. 8, 2026].

[3] B. H. Kim, S. M. Hong, and M. Mannan, "Rhea: Detecting Privilege-Escalated Evasive Ransomware Attacks Using Format-Aware Validation in the Cloud," arXiv:2601.18216, 2026, doi: 10.48550/arXiv.2601.18216.

[4] K. Higuchi and R. Kobayashi, "Impact of File-Open Hook Points on Backup Ratio in ROFBS on XFS," arXiv:2603.16364, 2026, doi: 10.48550/arXiv.2603.16364.

[5] P. Caporaso, G. Bianchi, and F. Quaglia, "VaultFS: Data Integrity via Write-Once Software Support at the File System Level," Digital Threats: Research and Practice, vol. 7, no. 2, Art. no. 11, pp. 1–26, 2026, doi: 10.1145/3797891.

[6] G. G. Múzquiz, J. González-Gómez, and E. Soriano-Salvador, "The Reverse File System: Towards Open Cost-Effective Secure WORM Storage Devices for Logging," Computers & Security, vol. 162, Art. no. 104786, 2026, doi: 10.1016/j.cose.2025.104786.

[7] W. Xia et al., "FastCDC: A Fast and Efficient Content-Defined Chunking Approach for Data Deduplication," in Proc. 2016 USENIX Annual Technical Conference (USENIX ATC '16), 2016, pp. 101–114. [Online]. Available: https://www.usenix.org/conference/atc16/technical-sessions/presentation/xia.

[8] S. Udayashankar, A. Baba, and S. Al-Kiswany, "Accelerating Data Chunking in Deduplication Systems using Vector Instructions," ACM Transactions on Storage, 2026, doi: 10.1145/3797270.

[9] J. O'Connor, J.-P. Aumasson, S. Neves, and Z. Wilcox-O'Hearn, "BLAKE3: One Function, Fast Everywhere," 2020. [Online]. Available: https://github.com/BLAKE3-team/BLAKE3-specs. [Accessed: Sep. 8, 2026].

[10] Y. Lv, Q. Li, Q. Xu, C. Gao, C. Yang, X. Wang, and C. J. Xue, "Rethinking LSM-Tree Based Key-Value Stores: A Survey," arXiv:2507.09642, 2025, doi: 10.48550/arXiv.2507.09642.

[11] D. Liu, X. Zou, T. Lu, P. Shilane, W. Xia, W. Huang, Y. Pan, and H. Huang, "Garbage Collection Does Not Only Collect Garbage: Piggybacking-Style Defragmentation for Deduplicated Backup Storage," in Proc. EuroSys 2025, 2025, pp. 1026–1043, doi: 10.1145/3689031.3717493.

[12] J. Zhang et al., "Scavenger+: Revisiting Space-Time Tradeoffs in Key-Value Separated LSM-Trees," IEEE Transactions on Computers, vol. 74, no. 10, pp. 3332–3346, 2025, doi: 10.1109/TC.2025.3587513.

[13] R. Chandramouli and E. Hibbard, Security Guidelines for Storage Infrastructure, NIST Special Publication 800-209 Revision 1, Initial Public Draft, 2026, doi: 10.6028/NIST.SP.800-209r1.ipd.

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. Page 2599
This work is licensed under a Creative Commons Attribution 4.0 International License.

[14] R. Chandramouli and Z. Butcher, A Zero Trust Architecture Model for Access Control in Cloud-Native Applications in Multi-Cloud Environments, NIST Special Publication 800-207A, 2023, doi: 10.6028/NIST.SP.800-207A.

[15] C. Constantinescu and S. Seshadri, "Sentinel: Ransomware Detection in File Storage," in Proc. 14th ACM International Conference on Systems and Storage (SYSTOR '21), 2021, Art. no. 28, doi: 10.1145/3456727.3463834.

[16] B. Reidys, P. Liu, and J. Huang, "RSSD: Defend against Ransomware with Hardware-Isolated Network-Storage Codesign and Post-Attack Analysis," in Proc. 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '22), 2022, pp. 726–739, doi: 10.1145/3503222.3507773.

[17] B. Ma, Y. Yang, J. Li, F. Zhang, W. Shen, Y. Zhou, and J. Ma, "Travelling the Hypervisor and SSD: A Tag-Based Approach against Crypto Ransomware with Fine-Grained Data Recovery," in Proc. 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23), 2023, pp. 341–355, doi: 10.1145/3576915.3616665.

[18] J. Dafoe, N. Chen, B. Chen, and Z. Wang, "Enabling Per-File Data Recovery from Ransomware Attacks via File System Forensics and Flash Translation Layer Data Extraction," Cybersecurity, vol. 7, Art. no. 75, 2024, doi: 10.1186/s42400-024-00287-9.

[19] Y. Joo, Y. Yoon, and J. H. Choi, "A Memory-Efficient Overwrite Detection Method for Ransomware-Proof SSDs," IEICE Transactions on Information and Systems, vol. E106-D, no. 8, pp. 1283–1286, 2023, doi: 10.1587/transinf.2023EDL8019.

[20] M. Hirano and R. Kobayashi, "RanSMAP: Open dataset of Ransomware Storage and Memory Access Patterns for creating deep learning based ransomware detectors," Computers & Security, vol. 150, Art. no. 104202, 2025, doi: 10.1016/j.cose.2024.104202.

[21] OWASP Foundation, "OWASP Application Security Verification Standard 5.0.0," 2025. [Online]. Available: https://owasp.org/www-project-application-security-verification-standard/. [Accessed: Sep. 8, 2026].

[22] Dgraph Labs, "Badger: Fast Key-Value Database in Go," GitHub repository, 2026. [Online]. Available: https://github.com/dgraph-io/badger. [Accessed: Sep. 8, 2026].

[23] M. Souppaya, W. C. Barker, W. Fisher, and K. Kent, Ransomware Risk Management: A Cybersecurity Framework 2.0 Community Profile, NIST Internal Report 8374 Revision 1, 2026, doi: 10.6028/NIST.IR.8374r1.

[24] A. Nelson, S. Rekhi, M. Souppaya, and K. Scarfone, Incident Response Recommendations and Considerations for Cybersecurity Risk Management: A CSF 2.0 Community Profile, NIST Special Publication 800-61 Revision 3, 2025, doi: 10.6028/NIST.SP.800-61r3.

Copyright © 2026 The Author(s). Published by *Forum Kerjasama Pendidikan Tinggi*. Page 2600
This work is licensed under a Creative Commons Attribution 4.0 International License.