Implementasi Sistem *File Integrity Monitoring* (FIM) Berbasis Fungsi Hash SHA-256 Menggunakan PowerShell untuk Deteksi Dini Aktivitas *Ransomware*

Farhan Raditya Aji - 13522142[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522142@std.stei.itb.ac.id

***Abstract*—Serangan ransomware menjadi ancaman serius bagi integritas data karena mampu mengenkripsi berkas pengguna secara paksa. Oleh karena itu, deteksi dini terhadap perubahan berkas yang tidak sah diperlukan untuk meminimalkan dampak kerusakan. Makalah ini merancang dan mengimplementasikan sistem *File Integrity Monitoring* (FIM) yang ringan berbasis skrip Windows PowerShell dengan algoritma hash kriptografis SHA-256. Mekanisme sistem dilakukan dengan membangun *baseline* nilai hash, kemudian menghitung serta membandingkan hash berkas secara berkala untuk mengidentifikasi anomali. Hasil pengujian menunjukkan sistem dapat mendeteksi modifikasi berkas dengan akurasi 100%, didukung oleh karakteristik *avalanche effect* pada SHA-256 yang menghasilkan perubahan hash signifikan meskipun perubahan berkas sangat kecil. Dari sisi kinerja, sistem juga menunjukkan efisiensi tinggi dengan rata-rata waktu komputasi di bawah 250 milidetik untuk berkas berukuran hingga 122 MB. Kontribusi penelitian ini adalah menyediakan solusi keamanan yang hemat biaya, mudah diterapkan, dan relevan untuk lingkungan Windows sebagai peringatan dini terhadap aktivitas ransomware.**

***Keywords*—*Ransomware*, *File Integrity Monitoring* (FIM), SHA-256, PowerShell, Integritas Data.**

I. PENDAHULUAN

Dalam era digital saat ini, integritas data merupakan salah satu aspek keamanan informasi yang paling krusial. Seiring dengan evolusi ancaman terhadap integritas data, *ransomware* muncul sebagai salah satu jenis *malware* yang paling merugikan. Berbeda dengan virus tradisional yang mungkin hanya merusak sistem, *ransomware* mengunci data pengguna dengan cara mengenkripsinya menggunakan algoritma kriptografi yang kuat sehingga data tidak dapat diakses tanpa kunci dekripsi dari penyerang.

Secara fundamental, serangan *ransomware* adalah serangan terhadap integritas dan ketersediaan data. Ketika sebuah berkas diinfeksi atau dienkripsi oleh *ransomware*, isi dari berkas tersebut berubah total pada tingkat *bit*. Dalam prinsip kriptografi, perubahan sekecil apa pun pada pesan masukan (*pre-image*) akan menghasilkan keluaran (*digest*) yang sangat berbeda pada fungsi *hash* satu arah. Sifat ini dikenal sebagai efek *avalanche*. Oleh karena itu, pemantauan integritas berkas atau *File Integrity Monitoring* (FIM) berbasis fungsi *hash* menjadi metode yang logis untuk mendeteksi aktivitas anomali yang dilakukan oleh *ransomware*.

Meskipun saat ini terdapat banyak solusi antivirus dan *Endpoint Detection and Response* (EDR) di pasaran, banyak di antaranya membebankan kinerja sistem komputer (*resource-heavy*) atau memerlukan biaya berlangganan yang mahal. Bagi pengguna individu atau organisasi kecil, diperlukan solusi yang ringan, mudah diimplementasikan, dan memanfaatkan perkakas yang sudah tersedia secara *native* di dalam sistem operasi.

Pada lingkungan sistem operasi Windows, *PowerShell* menyediakan kemampuan *scripting* yang kuat dan akses langsung ke pustaka kriptografi sistem. Makalah ini membahas perancangan dan implementasi sistem FIM sederhana menggunakan *PowerShell* dengan memanfaatkan algoritma SHA-256 (*Secure Hash Algorithm 256-bit*). Pemilihan SHA-256 didasarkan pada ketahanannya terhadap bentrokan (*collision resistance*) yang lebih baik dibandingkan pendahulunya seperti MD5 atau SHA-1, menjadikannya standar industri yang andal untuk verifikasi integritas.

Tujuan dari makalah ini adalah untuk mendemonstrasikan bagaimana konsep dasar fungsi *hash* dapat diaplikasikan untuk menyelesaikan masalah keamanan riil. Penulis akan memaparkan implementasi teknis skrip pemantauan, melakukan analisis kinerja waktu komputasi *hashing*, serta menguji efektivitas sistem dalam mendeteksi simulasi aktivitas modifikasi berkas yang menyerupai perilaku *ransomware*. Kontribusi utama dari penelitian ini adalah penyediaan purwarupa perkakas keamanan yang dapat diimplementasikan secara mandiri tanpa ketergantungan pada perangkat lunak pihak ketiga.

II. LANDASAN TEORI

A. Aspek Keamanan Integritas Data

Dalam ranah keamanan informasi, integritas data (*data integrity*) didefinisikan sebagai jaminan bahwa data atau pesan masih utuh, asli, dan belum mengalami manipulasi atau pengubahan oleh pihak yang tidak berwenang selama proses penyimpanan atau transmisi [1]. Serangan terhadap integritas data dikategorikan sebagai serangan aktif (*active attack*), di mana penyerang melakukan intervensi dengan cara mengubah, menghapus, atau menyisipkan data palsu [2].

Ransomware merupakan salah satu bentuk serangan terhadap ketersediaan dan integritas data. Secara kriptografis, ketika *ransomware* mengenkripsi sebuah berkas, ia secara fundamental mengubah struktur bit dari *plainteks* (berkas asli) menjadi *cipherteks* (berkas terkunci). Perubahan ini dapat dideteksi menggunakan mekanisme pengecekan integritas.

B. Fungsi Hash Kriptografis

Fungsi *hash* satu-arah (*one-way hash function*) adalah fungsi yang mengkompresi pesan (M) berukuran sembarang menjadi string bit (h) yang berukuran tetap (*fixed size*) [3]. Luaran dari fungsi ini disebut sebagai nilai *hash* (*hash value*) atau intisari pesan (*message digest*). Secara matematis, hal ini dinotasikan sebagai:

$$h = H(M)$$

Berdasarkan materi kuliah Kriptografi [3], sebuah fungsi *hash* yang aman untuk keperluan kriptografi harus memenuhi tiga sifat utama:

1. *Preimage Resistance* (Sifat Satu Arah)
   Jika diberikan nilai hash y, sangat sukar secara komputasi untuk menemukan nilai M sedemikian rupa sehingga $H(M) = y$. Sifat ini menjamin bahwa nilai *hash* tidak dapat dikembalikan menjadi pesan asal (irreversible)

2. *Second Preimage Resistance*
   Jika diberikan input M_1, sangat sukar mencari input lain M_2 (di mana $M_1 \neq M_2$) yang menghasilkan nilai hash yang sama $H(M_1) = H(M_2)$.

3. *Collision Resistance* (Bebas Bentrokan)
   Sangat sukar menemukan dua input sembarang M_1 dan M_2 yang menghasilkan nilai hash yang sama.

Selain itu, fungsi *hash* memiliki sifat Efek Longsoran (*Avalanche Effect*), di mana perubahan kecil pada input (misalnya 1 bit) akan menyebabkan perubahan yang sangat signifikan pada nilai *hash* keluarannya [3]. Sifat inilah yang dimanfaatkan dalam sistem FIM untuk mendeteksi modifikasi berkas oleh *ransomware*.

C. Algoritma SHA-256

SHA-256 (*Secure Hash Algorithm 256-bit*) adalah bagian dari keluarga algoritma SHA-2 yang dikembangkan oleh NIST (*National Institute of Standards and Technology*). Berbeda dengan pendahulunya, SHA-1, yang telah ditemukan kelemahannya terhadap serangan bentrokan (*collision attack*) seperti yang didemonstrasikan oleh Google dan CWI pada tahun 2017 (serangan *SHAttered*), SHA-256 saat ini masih direkomendasikan karena belum ada bentrokan yang ditemukan secara praktis [4].

Spesifikasi teknis SHA-256 berdasarkan materi perkuliahan adalah sebagai berikut [4]:

- Struktur: Menggunakan konstruksi *Merkle-Damgård*.
- Ukuran Blok: Memproses pesan dalam blok berukuran 512 bit.
- Ukuran *Word*: Beroperasi pada ukuran *word* 32 bit.
- Putaran: Melakukan 64 kali putaran (*rounds*) komputasi yang melibatkan operasi rotasi, pergeseran (*shift*), dan penambahan modular.
- Output: Menghasilkan intisari pesan sepanjang 256 bit (32 *byte*).

Dalam implementasinya, SHA-256 menggunakan 8 variabel internal (A sampai H) dan 64 konstanta berbeda yang berasal dari akar kubik bilangan prima pertama, menjadikannya sangat resisten terhadap *cryptanalysis* [4].

D. Mekanisme Deteksi Perubahan Berkas

Pendeteksian perubahan berkas dilakukan dengan membandingkan nilai *hash* pada waktu t_0 (*baseline*) dengan waktu t_1 (*current*). Sesuai dengan prinsip *cryptographic checksum* atau *Message Integrity Check* (MIC) [3], jika sebuah berkas F dimodifikasi menjadi F', maka:

$$H(F) <\neq H(F')$$

Sistem pemantauan akan menghitung $h_{current} = \text{SHA256}(File)$ secara periodik. Jika $h_{current}$ tidak sama dengan $h_{baseline}$ yang tersimpan di *database*, maka sistem menyimpulkan bahwa integritas berkas telah terlanggar (modified), yang dapat mengindikasikan aktivitas *ransomware*.

E. Utilitas Get-FileHash pada Windows PowerShell

Windows PowerShell adalah kerangka kerja manajemen konfigurasi dan otomatisasi tugas yang terintegrasi secara *native* di dalam sistem operasi Microsoft Windows. Berbeda dengan *Command Prompt* (cmd.exe) terdahulu, PowerShell dibangun di atas .NET Framework yang memberikan akses luas ke pustaka sistem, termasuk pustaka keamanan dan kriptografi [5].

Untuk keperluan verifikasi integritas data, PowerShell menyediakan *cmdlet* bawaan bernama Get-FileHash.

Perintah ini berfungsi untuk menghitung nilai *hash* dari sebuah berkas menggunakan algoritma yang ditentukan. Berdasarkan dokumentasi resmi Microsoft [5], sintaks dasar perintah ini adalah:

Get-FileHash -Path <LokasiBerkas> -Algorithm <Algoritma>

Secara *default*, algoritma yang digunakan adalah SHA-256, namun pengguna dapat memilih algoritma lain seperti MD5, SHA-1, SHA-384, atau SHA-512. Dalam konteks keamanan siber, penggunaan perkakas bawaan (*native tools*) seperti ini memiliki keunggulan strategis berupa efisiensi sumber daya dan meminimalisir ketergantungan pada perangkat lunak pihak ketiga (*third-party dependencies*), sehingga mengurangi permukaan serangan (*attack surface*) pada sistem yang dipantau.

III. PERANCANGAN SISTEM

A. Deskripsi Umum Sistem

Sistem yang dirancang adalah sebuah perkakas deteksi integritas berkas (*File Integrity Monitoring*) berbasis skrip yang berjalan di lingkungan sistem operasi Windows. Sistem ini memanfaatkan antarmuka baris perintah (*Command Line Interface* - CLI) PowerShell untuk memantau direktori spesifik yang rentan terhadap serangan *ransomware* (misalnya direktori dokumen pengguna).

Secara garis besar, sistem bekerja dengan membandingkan nilai *hash* SHA-256 dari berkas saat ini dengan nilai *hash* referensi yang telah disimpan sebelumnya. Jika terjadi perubahan nilai *hash* tanpa otorisasi, sistem akan mengindikasi hal tersebut sebagai anomali integritas dan memberikan peringatan dini kepada pengguna.

B. Arsitektur dan Alur Kerja Sistem

Sistem dirancang untuk beroperasi dalam dua fase utama: Fase Inisialisasi (*Baselining*) dan Fase Pemantauan (*Monitoring*).

1. Fase Inisialisasi (*Baselining*)
   Pada fase ini, sistem memindai seluruh berkas yang ada di dalam direktori target. Untuk setiap berkas, sistem menghitung nilai *hash* SHA-256 menggunakan *cmdlet* Get-FileHash. Pasangan data [Path_Berkas, Nilai_Hash] kemudian disimpan ke dalam sebuah struktur data kamus (*dictionary*) atau berkas log sebagai referensi dasar (*baseline*).
2. Fase Pemantauan (*Monitoring*)
   Setelah *baseline* terbentuk, sistem memasuki siklus pemantauan tanpa henti (*infinite loop*). Pada setiap interval waktu tertentu (t), sistem akan:
   - Menghitung ulang nilai *hash* dari berkas-berkas di direktori target.
   - Membandingkan nilai hash baru (H_{new}) dengan nilai hash lama (H_{old}) yang ada di *baseline*.
   - Jika ditemukan kondisi $(\mathrm{H}_{new}) \neq (\mathrm{H}_{old})$, sistem memicu peringatan (*alert*) bahwa berkas telah dimodifikasi.

C. Perancangan Algoritma (Pseudocode)

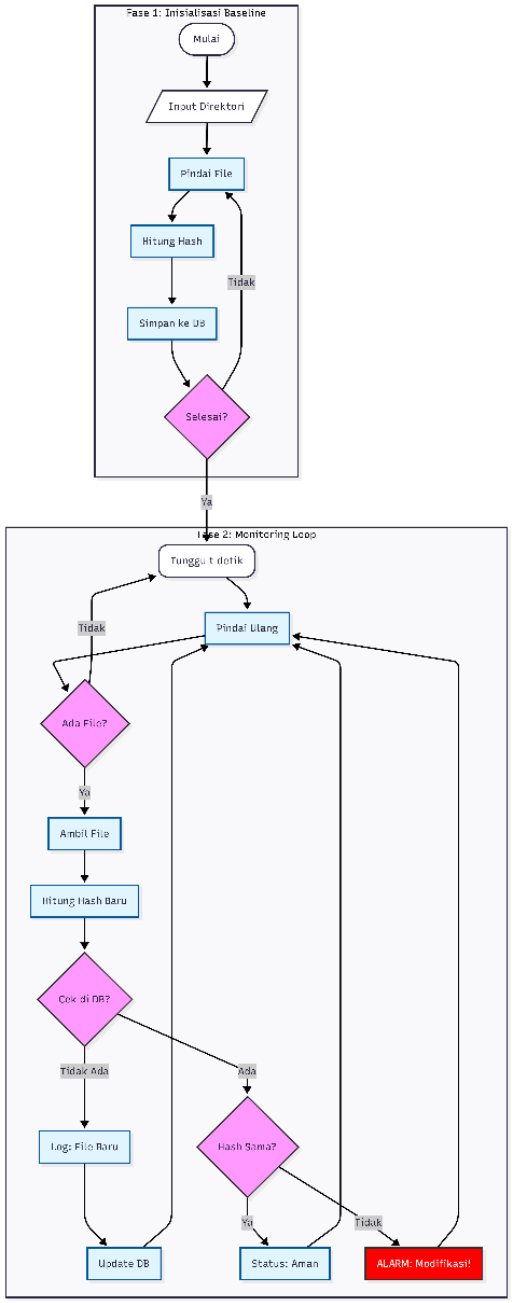
Berikut adalah perancangan logika program *File Integrity Monitoring* (FIM) yang ditulis menggunakan notasi algoritmik:

```
  { Program FileIntegrityMonitoring
    Deskripsi: Memantau integritas berkas
dalam direktori dengan membandingkan nilai
hash SHA-256 secara periodik.
    Masukan: Path direktori target yang
akan dipantau.
    Keluaran: Peringatan (alert) ke layar
jika terjadi perubahan hash pada berkas.
  }

  Deklarasi
    TargetDir : string
    Baseline  : Dictionary { struktur data
untuk menyimpan pasangan (Path, Hash) }
    DaftarFile : array of File
    f : File
    CurrentHash, StoredHash : string

    function HitungHash(input f : File) ➔
string
    { Menghitung nilai hash SHA-256 dari
berkas f dan mengembalikannya sebagai
string heksadesimal }

    procedure AmbilDaftarFile(input dir :
string, output list : array of File)
    {  Memindai direktori 'dir' dan
menyimpan seluruh objek berkas yang
ditemukan ke dalam 'list' }

    procedure SimpanBaseline(input/output
base : Dictionary, input f : File, input
h : string)
    { Menyimpan pasangan path berkas dan
nilai hash-nya ke dalam memori baseline }

    procedure TampilkanAlert(input f :
File, input lama : string, input baru :
string)
    { Menampilkan pesan peringatan deteksi
perubahan integritas ke layar pengguna }

  Algoritma:
    { --- FASE 1: INISIALISASI BASELINE -
-- }
    write("Masukkan direktori target: ")
    read(TargetDir)

    AmbilDaftarFile(TargetDir,
DaftarFile) { dapatkan semua file awal }

    for tiap f di dalam DaftarFile do
      CurrentHash <- HitungHash(f)
      SimpanBaseline(Baseline,         f,
CurrentHash)
```

```
      write("Baseline dibuat untuk: ",
f.Nama)
    endfor

    write("Fase Baseline Selesai. Memulai
Monitoring...")

    { --- FASE 2: LOOP PEMANTAUAN
(MONITORING) --- }
    while true do { loop tidak terbatas
selama program berjalan }

      AmbilDaftarFile(TargetDir,
DaftarFile) { pindai ulang direktori untuk
cek kondisi terkini }

      for tiap f di dalam DaftarFile do
        CurrentHash <- HitungHash(f)

        { Cek apakah file sudah ada di
baseline }
        if     (Baseline.Contains(f.Path))
then
          StoredHash <- Baseline[f.Path]

          { Bandingkan hash saat ini
dengan hash baseline }
          if (CurrentHash != StoredHash)
then
             TampilkanAlert(f,
StoredHash, CurrentHash)
             { Opsional: Update baseline
jika perubahan diotorisasi }
          endif

        else
          { Jika file belum ada di
baseline, berarti file baru }
          write("[INFO]     File     baru
terdeteksi: ", f.Nama)
          SimpanBaseline(Baseline,     f,
CurrentHash)
        endif
      endfor

      Wait(2) { tunda eksekusi selama 2
detik sebelum siklus berikutnya }

  endwhile
```

D. Diagram Alir Sistem (Flowchart)

Berikut adalah diagram alir sistem yang akan diimplementasikan:



Gambar 1. Diagram Alir Logika Sistem Deteksi Perubahan Berkas.

IV. IMPLEMENTASI DAN PENGUJIAN

A. Implementasi Perangkat Lunak

Sistem *File Integrity Monitoring* (FIM) diimplementasikan menggunakan bahasa skrip *Windows PowerShell* versi 5.1. Kode program terdiri dari satu berkas skrip utama yang menangani dua fungsi logika utama yaitu pembuatan basis data referensi (*baselining*) dan pemantauan waktu nyata (*monitoring*).

Berikut adalah potongan kode utama (*snippet*) dari implementasi sistem yang dibuat:

1. Fungsi Penghitungan Hash
   Kode ini memanfaatkan *cmdlet* bawaan untuk

menghasilkan integritas data.

```powershell
function Hitung-FileHash {
    param ($FilePath)
    $hash = Get-FileHash -Path $FilePath -Algorithm SHA256
    return $hash.Hash
}
```

Gambar 2. Implementasi Fungsi Penghitungan Nilai Hash Menggunakan Algoritma SHA-256.

2. Logika Pemantauan (Monitoring Loop)
   Bagian ini berjalan terus-menerus untuk mendeteksi anomali.

```powershell
while ($true) {
    $currentFiles = Get-ChildItem -Path $TargetDir -Recurse

    foreach ($file in $currentFiles) {
        $currentHash = Hitung-FileHash $file.FullName

        # Cek apakah file ada di baseline
        if ($baseline.ContainsKey($file.FullName)) {
            $storedHash = $baseline[$file.FullName]

            # Deteksi Perubahan (Integritas Terlanggar)
            if ($currentHash -ne $storedHash) {
                Write-Host "[BAHAYA] MODIFIKASI TERDETEKSI!" -F Red
                Write-Host "File: $($file.Name)"
                Write-Host "Hash Lama: $storedHash"
                Write-Host "Hash Baru: $currentHash"
            }
        } else {
            Write-Host "[INFO] File Baru Ditemukan: $($file.Name)" -F Green
            $baseline[$file.FullName] = $currentHash
        }
    }
    Start-Sleep -Seconds 2 # Interval waktu t
}
```

Gambar 3. Algoritma Utama Pemantauan Integritas Berkas Secara *Real-time*.

B. Skenario Pengujian

Untuk memvalidasi efektivitas sistem, dilakukan simulasi serangan menggunakan skenario "*Dummy Ransomware*". Pengujian dilakukan pada direktori target yang berisi 5 jenis berkas berbeda (DOCX, PDF, JPG, TXT, dan XLSX). Skenario pengujian mencakup tiga vektor serangan:

1. Modifikasi Konten (*Append Attack*)
   Menyisipkan satu karakter tambahan ke dalam berkas teks.
2. Enkripsi Semu (*Mock Encryption*)
   Mengubah seluruh isi berkas menjadi data acak (*garbage data*) namun mempertahankan nama berkas yang sama.
3. Perubahan Nama (*Renaming*)
   Mengganti nama berkas tanpa mengubah isinya.

C. Hasil Eksperimen dan Analisis

Untuk memvalidasi efektivitas sistem FIM yang telah dibangun, dilakukan serangkaian pengujian yang mencakup analisis sensitivitas fungsi *hash*, pengukuran kinerja waktu komputasi, dan uji akurasi deteksi terhadap simulasi serangan.

1. Analisis Efek Longsoran (*Avalanche Effect*)
   Pengujian pertama bertujuan untuk membuktikan sifat *avalanche effect* pada algoritma SHA-256, di mana perubahan kecil pada *plainteks* harus menghasilkan perubahan drastis pada *cipherteks* (nilai hash).

   Skenario pengujian dilakukan sebagai berikut:
   - Kondisi Awal: Sebuah berkas teks rahasia.txt dibuat dengan konten string "Kriptografi ITB". Sistem mencatat nilai hash awal pada fase baselining.
   - Modifikasi: Konten berkas diubah dengan menambahkan satu karakter tanda seru ('!') menjadi "Kriptografi ITB!".
   - Hasil: Sistem mendeteksi perubahan tersebut secara real-time.

```
Hash Asli : 2968730A6221A911BB5E819A558D79B5893E5858D1265C9F238AF70648E372C6
Hash Baru : 2D18F14B64D754C6019CE2F1A4DF478142A1F936CA7492B71BBED1B19DB44B65
```

Gambar 4. Perubahan Hash Akibat Modifikasi Satu Karakter.

Seperti terlihat pada Gambar 4, perubahan satu karakter pada *input* menyebabkan nilai *hash* berubah total pada tataran bit. Hash lama berawalan 296... sedangkan hash baru berawalan 2D1.... Tidak ada korelasi pola antara kedua nilai hash tersebut, membuktikan bahwa SHA-256 memenuhi sifat *diffusion* yang baik dan sulit diprediksi oleh penyerang.

2. Analisis Kinerja Waktu Komputasi
   Salah satu tantangan dalam implementasi FIM adalah penggunaan sumber daya komputasi (*overhead*). Pengujian ini mengukur waktu yang dibutuhkan sistem untuk menghitung hash SHA-256 terhadap berkas dengan variasi ukuran yang berbeda. Pengukuran dilakukan menggunakan fitur `Measure-Command` pada PowerShell.

   Hasil pengukuran waktu komputasi disajikan dalam Tabel I berikut:

Tabel 1. Rata-Rata Waktu Kompotasi *Hash SHA-256*

No	Ukuran Berkas	Waktu Komputasi (TotalMilliseconds)	Keterangan
1	15 Byte	52,26 ms	Sangat Cepat
2	1 MB	63,50 ms	Sangat Cepat
3	122 MB	243,87 ms	Cepat (< 1 detik)

Berdasarkan data pada Tabel I, terlihat bahwa implementasi algoritma SHA-256 menggunakan PowerShell menunjukkan kinerja yang sangat efisien. Waktu komputasi dasar (overhead) untuk berkas sangat kecil (15 Byte) dan berkas menengah

(1 MB) relatif stabil di kisaran 50-65 milidetik. Peningkatan ukuran berkas yang signifikan menjadi 122 MB hanya meningkatkan waktu proses menjadi sekitar 243 milidetik. Hal ini membuktikan bahwa sistem FIM yang dibangun bersifat ringan (lightweight) dan tidak akan membebani sumber daya CPU pengguna secara signifikan, bahkan saat memindai berkas berukuran besar.

```
# Measure-Command { Get-FileHash -Path "test\benchmark\rahasia_15B.txt" -
Algorithm SHA256 }

Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 52
Ticks             : 522609
TotalDays         : 6,04871527777778E-07
TotalHours        : 1,45169166666667E-05
TotalMinutes      : 0,00871015
TotalSeconds      : 0,0522609
TotalMilliseconds : 52,2609
```

Gambar 5. Pengukuran Waktu Komputasi Pada Berkas Berukuran 15 B.

```
# Measure-Command { Get-FileHash -Path "test\benchmark\rahasia_1MB.pdf" -
Algorithm SHA256 }

Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 63
Ticks             : 634984
TotalDays         : 7,34935185185185E-07
TotalHours        : 1,76384444444444E-05
TotalMinutes      : 0,00105830666666667
TotalSeconds      : 0,0634984
TotalMilliseconds : 63,4984
```

Gambar 6. Pengukuran Waktu Komputasi Pada Berkas Berukuran 1 MB.

```
# Measure-Command { Get-FileHash -Path "test\benchmark\rahasia_122MB.exe"
-Algorithm SHA256 }

Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 243
Ticks             : 2438696
TotalDays         : 2,82256481481481E-06
TotalHours        : 6,77415555555556E-05
TotalMinutes      : 0,00406449333333333
TotalSeconds      : 0,2438696
TotalMilliseconds : 243,8696
```

Gambar 7. Pengukuran Waktu Komputasi Pada Berkas Berukuran 122 MB.

3. Uji Deteksi Simulasi *Ransomware*
   Pengujian terakhir mensimulasikan perilaku *ransomware* sederhana yang mengenkripsi (atau menimpa) isi berkas secara massal. Dalam skenario ini, skrip dijalankan untuk memantau direktori berisi data penting. Kemudian, dilakukan simulasi serangan dengan mengubah isi konten berkas secara paksa.

```
=== MONITORING AKTIF (Tekan Ctrl+C untuk stop) ===

[!!!] PERINGATAN KEAMANAN: MODIFIKASI TERDETEKSI!
      Waktu     : 2025-12-12 22:09:38
      File      : rahasia.txt
      Hash Asli : 2968730A6221A911BB5E819A558D79B5893E5858D1265C9F238AF70648E372C6
      Hash Baru : 2D18F14B64D754C6019CE2F1A4DF478142A1F936CA7492B71BBED1B19DB44B65
      Status    : INTEGRITAS TERLANGGAR (Potensi Ransomware)
------------------------------------------------------------

[!!!] PERINGATAN KEAMANAN: MODIFIKASI TERDETEKSI!
      Waktu     : 2025-12-12 22:09:40
      File      : rahasia.txt
      Hash Asli : 2968730A6221A911BB5E819A558D79B5893E5858D1265C9F238AF70648E372C6
      Hash Baru : 2D18F14B64D754C6019CE2F1A4DF478142A1F936CA7492B71BBED1B19DB44B65
      Status    : INTEGRITAS TERLANGGAR (Potensi Ransomware)
------------------------------------------------------------

[!!!] PERINGATAN KEAMANAN: MODIFIKASI TERDETEKSI!
      Waktu     : 2025-12-12 22:09:43
      File      : rahasia.txt
      Hash Asli : 2968730A6221A911BB5E819A558D79B5893E5858D1265C9F238AF70648E372C6
      Hash Baru : 2D18F14B64D754C6019CE2F1A4DF478142A1F936CA7492B71BBED1B19DB44B65
      Status    : INTEGRITAS TERLANGGAR (Potensi Ransomware)
------------------------------------------------------------
```

Gambar 8. Tampilan Peringatan Sistem Saat Mendeteksi Aktivitas Anomali pada Berkas.

Hasil pengujian pada Gambar 8 menunjukkan bahwa sistem berhasil memicu peringatan " MODIFIKASI TERDETEKSI" segera setelah operasi penyimpanan (*save*) dilakukan pada berkas target. Sistem menampilkan nama berkas yang terdampak beserta perbandingan nilai hash sebelum dan sesudah serangan. Hal ini memberikan informasi krusial bagi pengguna untuk segera melakukan isolasi jaringan sebelum kerusakan menyebar lebih luas.

V. KESIMPULAN

Berdasarkan perancangan, implementasi, dan serangkaian pengujian yang telah dilakukan terhadap sistem *File Integrity Monitoring* (FIM) berbasis PowerShell dan SHA-256, dapat ditarik beberapa kesimpulan sebagai berikut:

1. Efektivitas Deteksi Dini
   Sistem yang dibangun berhasil mendeteksi aktivitas modifikasi berkas secara *real-time*. Pemanfaatan algoritma SHA-256 terbukti efektif dalam mengidentifikasi perubahan sekecil apa pun pada berkas (efek *avalanche*), sehingga dapat berfungsi sebagai sistem peringatan dini (*early warning system*) terhadap indikasi serangan *ransomware*.
2. Efisiensi Kinerja
   Berdasarkan hasil pengukuran kinerja, penggunaan *cmdlet* bawaan PowerShell terbukti sangat efisien dan ringan (*lightweight*). Rata-rata waktu komputasi *hashing* untuk berkas berukuran besar (122 MB) hanya membutuhkan waktu kurang dari 250 milidetik. Hal ini membuktikan bahwa sistem dapat berjalan di latar belakang tanpa membebani sumber daya CPU pengguna secara signifikan.
3. Kemudahan Implementasi
   Solusi ini menawarkan kemudahan penerapan karena memanfaatkan perkakas *native* yang sudah tersedia di sistem operasi Windows tanpa memerlukan instalasi perangkat lunak pihak ketiga yang mahal atau kompleks.

LINK KODE SUMBER

https://github.com/sibobbbbbb/file-integrity-monitoring.git

LINK YOUTUBE

https://youtu.be/m-hqrGZU-08?si=3VW2_ILTUirYNpzM

UCAPAN TERIMA KASIH

Dengan penuh puji syukur kehadirat Allah SWT, penulis bersyukur atas limpahan rahmat dan karunia-Nya yang telah memberikan kelancaran sehingga penulis dapat menyelesaikan makalah ini. Ucapan terima kasih dan penghargaan penulis sampaikan kepada Bapak Dr. Ir. Rinaldi Munir, M.T. selaku dosen pengampu mata kuliah Kriptografi (IF4020), atas ilmu, bimbingan, dan wawasan mendalam mengenai dunia kriptografi yang telah diberikan selama perkuliahan. Penulis juga mengucapkan terima kasih kepada semua pihak yang secara langsung maupun tidak langsung telah memberikan semangat dan dukungan dalam penyelesaian makalah ini, termasuk rekan-rekan mahasiswa dan keluarga yang senantiasa memberikan dorongan moral.

REFERENSI

[1] R. Munir, "01 - Pengantar Kriptografi," Bahan Kuliah IF4020 Kriptografi, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025.

[2] R. Munir, "07 - Serangan Terhadap Kriptografi," Bahan Kuliah IF4020 Kriptografi, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025.

[3] R. Munir, "Fungsi Hash," Bahan Kuliah IF4020 Kriptografi, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025.

[4] R. Munir, "Beberapa Fungsi Hash," Bahan Kuliah IF4020 Kriptografi, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2025.

[5] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 8th ed. Hoboken, NJ: Pearson, 2020.

[6] *Secure Hash Standard (SHS)*, FIPS PUB 180-4, National Institute of Standards and Technology, Gaithersburg, MD, Aug. 2015.

[7] Microsoft, "Get-FileHash (Microsoft.PowerShell.Utility)," *Microsoft Learn*, Nov. 2024. [Online]. Available: https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-filehash.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 24 Desember 2025

Farhan Raditya Aji (13522142)