Vol. 10, No. 2, Oktober 2025, Page. 507-517
https://journal.uin-alauddin.ac.id/index.php/instek
p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

Analisis Reverse Engineering Malware WinRAR SFX Menggunakan Ghidra untuk Deteksi Teknik Obfuscation dan UAC Bypass

Rosmiati[*1], Muh. Ikhsan Amar[2] , Muhammad Arham Arsyad[3]

[1,3] Sistem Informasi, Institut Teknologi Bacharuddin Jusuf Habibie, Indonesia
[2]Teknik Robotika dan Kecerdasan Buatan, Institut Teknologi Bacharuddin Jusuf Habibie, Indonesia
Email: [1]rosmiati@ith.ac.id, [2]ikhsan.amar93@ith.ac.id, [3]mrham1908@gmail.com

Abstrak

Ancaman malware yang semakin kompleks dan sulit dideteksi menuntut pendekatan analisis yang lebih mendalam dalam keamanan siber. Penelitian ini bertujuan untuk menganalisis efektivitas penerapan reverse engineering berbasis analisis statis dalam mendeteksi dan mengklasifikasikan malware. Parameter keberhasilan analisis diukur melalui kemampuan mengidentifikasi struktur PE abnormal, deteksi API mencurigakan, indikasi teknik evasion, serta klasifikasi tingkat ancaman (threat level). Sampel malware diperoleh dari platform MalwareBazaar dan dianalisis menggunakan PE Studio, Detect It Easy, Strings, dan Ghidra. Tahapan penelitian meliputi ekstraksi metadata, identifikasi API call, disassembly, serta deteksi teknik packing dan obfuscation. Hasil analisis menunjukkan bahwa malware menyamarkan diri sebagai aplikasi sah (WinRAR SFX), menggunakan command-line injection, bypass UAC, dan teknik obfuscation lanjutan untuk menghindari deteksi. Ditemukan pula indikasi teknik evasif seperti anti-debugging dan komunikasi antar-proses melalui shared memory. Berdasarkan parameter keberhasilan tersebut, pendekatan analisis statis dengan Ghidra terbukti mampu mengungkap pola perilaku malware tingkat menengah secara efektif. Temuan ini menegaskan pentingnya reverse engineering dalam mendeteksi ancaman malware sejak tahap awal tanpa perlu menjalankan kode berbahaya.

Kata kunci: Malware, Reverse Engineering, Analisis Statis, Ghidra, Keamanan Siber

Abstract

The increasing complexity and stealth of malware pose a significant challenge in the field of cybersecurity. This study aims to explore the application of static analysis-based reverse engineering to identify the internal structure and behavior of malware. Malware samples were collected from the MalwareBazaar platform and analyzed using tools such as PE Studio, Detect It Easy, Strings, and Ghidra. The research stages included metadata extraction, API call identification, disassembly, and detection of packing and obfuscation techniques. The results revealed that the analyzed malware disguised itself as a legitimate application (WinRAR SFX), employing command-line injection, UAC bypass, and advanced obfuscation methods to evade detection. Furthermore, strong indicators of evasive techniques were found, such as anti-debugging and inter-process communication through shared memory. Based on technical indicators and observed behaviors, the malware was classified as a medium-level threat with potential escalation if used as a dropper. These findings highlight the importance of reverse engineering as a comprehensive approach to detecting and understanding malware threats, especially at the initial stage without the need to execute the sample.

Keywords: *Malware, Reverse Engineering, Static Analysis, Ghidra, Cybersecurity*

1. PENDAHULUAN

Ancaman siber di Indonesia pertama kali muncul sekitar tahun 2000, ditandai dengan merebaknya kasus *carding* [1]. Ancaman siber terus mengalami peningkatan yang siginifikan, baik dari segi intensitas maupun tingkat kerumitannya. Salah satu ancaman terbesar dalam dunia keamanan siber adalah penyebaran *malware* (malicious software) yang semakin canggih dan sulit dideteksi [2], [3], [4]. Malware dapat menyusup ke sistem operasi dan membuat sistem komputer menggunakan sumber daya tanpa sepengetahuan pemilik perangkat, bahkan mengumpulkan informasi pribadi untuk dibagikan ke pihak ketiga tanpa persetujuan pengguna [5]. Beberapa insiden serangan *malware* yang sempat populer di Indonesia adalah serangan *ransomware* pada Pusat Data Nasional (PDN) oleh Kelompok Cipher

p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

terjadi pada Juni 2024. Varian *ransomware* Lock Bit 3.0 digunakan oleh kelompok peretas Brain Cipher untuk mengunci data 282 lembaga pemerintah, termasuk Kementerian dan Lembaga Daerah. Mereka menuntut tebusan 8 juta dolar AS (sekitar Rp. 131,8 miliar) untuk membuka akses ke data tersebut [6]. Selanjutnya kebocoran data Pegawai Negeri Sipil (PNS) yang terjadi pada Agustus 2024. Peretas dengan nama TopiAx membocorkan data pada Badan Kepegawaian Negara (BKN) di forum *dark web*. Sejumlah 4.759.218 ata PNS dibocorkan. Termasuk nama, tanggal lahir, nomor identitas, alamat, *email* dan informasi jabatan. Data ini menunjukkan kerentanan data di Lembaga pemerintah dan dijual dengan harga 10 ribu dolar AS atau sekitar Rp. 160 juta [7].

Kedua kasus tersebut menunjukkan bahwa serangan siber berbasis *malware* semakin canggih dengan menerapkan teknik yang dirancang untuk menghindari deteksi oleh sistem konvensional. Oleh karena itu diperlukan pendekatan komprehensif dalam deteksi *malware*, salah satunya adalah pendekatan analisis statis dalam *reverse engineering* [8] [9]. *Reverse Engineering* adalah proses menganalisis struktur, fungsi, dan perilaku suatu perangkat lunak dengan tujuan memahami cara kerjanya tanpa memerlukan kode sumber asli. Ini adalah cara yang efektif untuk memahami dan mengatasi ancaman *malware* [10].

Seiring dengan meningkatnya kompleksitas serangan siber, penelitian mengenai *reverse engineering* pada *malware* untuk mendeteksi ancaman siber telah berkembang pesat dalam beberapa tahun terakhir. *Static Analysis* adalah salah satu pendekatan yang digunakan dalam analisis *malware* [11]. *Static Analysis* adalah metode yang melibatkan pemeriksaan kode biner *malware* tanpa menjalankannya. Metode ini melibatkan pemeriksaan kode sumber atau biner aplikasi untuk mencari tanda-tanda aktivitas yang mencurigakan atau berbahaya. Beberapa teknik digunakan dalam *Static Analysis*, seperti dekompilasi, analisis izin, dan analisis fitur [10]. Sebuah penelitian oleh Jusoh et al. menunjukkan bahwa *Static Analysis* dapat dengan sukses mendeteksi *malware* yang berbasis pola, seperti penggunaan *obfuscation* atau pemanggilan fungsi API yang mencurigakan. Namun, ketika *malware* menggunakan teknik *packing* dan enkripsi, metode ini memiliki keterbatasan, yang membuat kode sulit untuk dianalisis secara langsung [12].

2. METODE PENELITIAN

Penelitian ini menggunakan pendekatan analisis statis untuk melakukan *reverse engineering* terhadap *dataset malware* yang telah dikumpulkan. Pendekatan analisis statis dipilih dalam penelitian ini karena memberikan kemampuan untuk mengamati struktur internal *malware* secara menyeluruh tanpa risiko mengeksekusi kode berbahaya secara langsung [13]. Secara sederhana, *reverse engineering malware* dapat didefinisikan sebagai proses merekayasa *malware* untuk memperoleh informasi tentang perilaku *malware* tersebut. Beberapa alat yang dibutuhkan untuk melakukan *reverse engineering malware* antara lain disassembler, debugger, PE Viewer, dan network analyzer. Metode ini cocok digunakan pada tahap awal analisis, terutama untuk mengidentifikasi indikasi awal adanya aktivitas mencurigakan dari segi struktur file, pemanggilan fungsi API, serta teknik pengaburan (obfuscation) yang umum digunakan oleh malware modern [14].

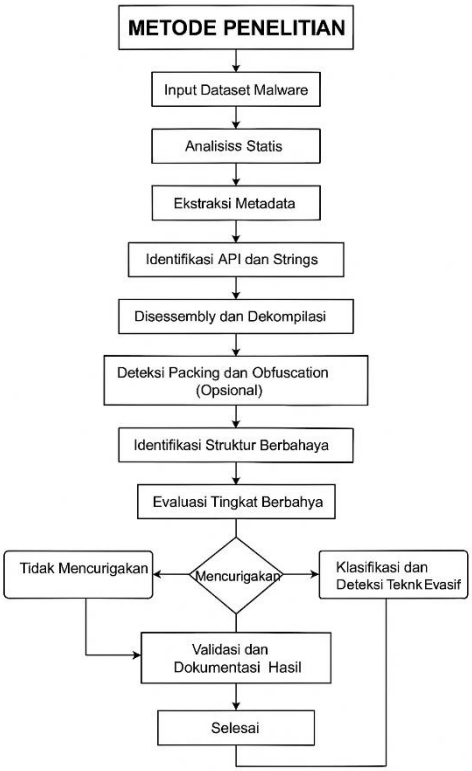
Dalam penelitian ini, pemilihan perangkat analisis dilakukan secara selektif berdasarkan kemampuan teknis dan relevansinya terhadap tujuan penelitian. PE Studio digunakan karena mampu menampilkan import/export table, entropy, dan compiler signature yang relevan untuk mendeteksi adanya packer atau obfuscator . Sementara itu, Detect It Easy dipilih karena memiliki heuristik deteksi packer dan kompiler yang tinggi serta akurat dalam mengenali struktur PE32 yang telah dimodifikasi. Adapun Ghidra digunakan sebagai decompiler open-source dengan kemampuan control flow analysis dan symbol recovery yang kuat, sehingga memungkinkan identifikasi teknik injeksi, anti-debugging, serta obfuscation secara statis dan mendalam.

Penggunaan Strings dan PE View memungkinkan identifikasi terhadap string tersembunyi dan referensi API, yang merupakan indikator penting untuk mendeteksi kemampuan malware dalam mengakses jaringan, memanipulasi sistem file, atau menjalankan perintah tertentu. Jika terindikasi adanya teknik evasif seperti *anti-debugging* atau *self-modifying code*, dilakukan langkah klasifikasi lanjutan untuk mengetahui jenis *malware* dan potensinya dalam melakukan serangan siber lanjutan.

Dengan mengintegrasikan berbagai *tools* analisis, pendekatan ini tidak hanya memberikan gambaran struktural terhadap malware, tetapi juga mampu mengungkap strategi dan teknik yang

p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

digunakan untuk menyusupi serta mempertahankan eksistensinya di dalam sistem target. Tahapan metode penelitian dapat dilihat pada Gambar 1.



Gambar 1. Metode Penelitian

3. HASIL DAN PEMBAHASAN

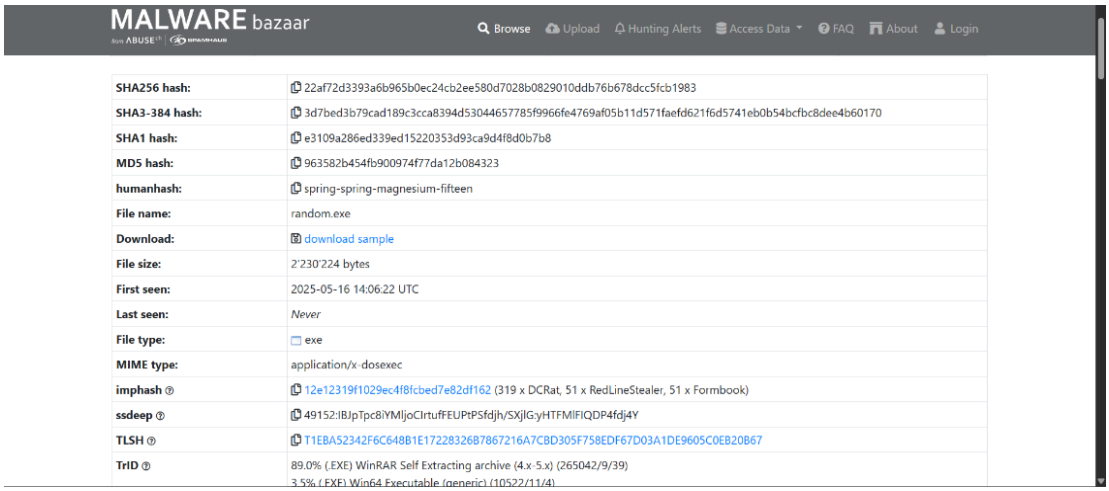
Static Analysis merupakan langkah awal dalam proses *reverse engineering* yang dilakukan tanpa mengeksekusi *file malware* [15], [16]. Pada tahap ini, peneliti berfokus pada pemeriksaan struktur internal berkas biner menggunakan *tools disassembler* dan *decompiler* untuk memahami perilaku program. Dalam penelitian ini, analisis statis dilakukan menggunakan Ghidra guna mengidentifikasi fungsi-fungsi penting, string tersembunyi serta pola instruksi yang mencurigakan yang dapat memberikan indikasi awal terhadap potensi ancaman atau teknik penyamaran yang digunakan oleh *malware*. Adapun langkah-langkah *static analysis* dengan menggunakan Ghidra adalah sebagai berikut:

3.1. Input Dataset Malware

Proses *static analysis* diawali dengan pengambilan sampel *malware* dari *dataset* yang tersedia. Pada penelitian ini, sampel dipilih dari platform MalwareBazaar yaitu sebuah repositori publik yang menyediakan berbagai *file malware* untuk keperluan penelitian dan edukasi.

Sampel yang dianalisis memiliki kode *hash* SHA256: 22af72d3393a6b965b0ec24cb2ee580d7028b0829010ddb76b678dcc5fcb1983 dengan nama berkas **random.exe**. Berdasarkan hasil identifikasi awal, *file* tersebut terdeteksi sebagai WinRAR *self-extracting archive* yang menunjukkan bahwa *malware* dikemas dalam format arsip yang dapat mengekstraksi dirinya sendiri saat dijalankan.

p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

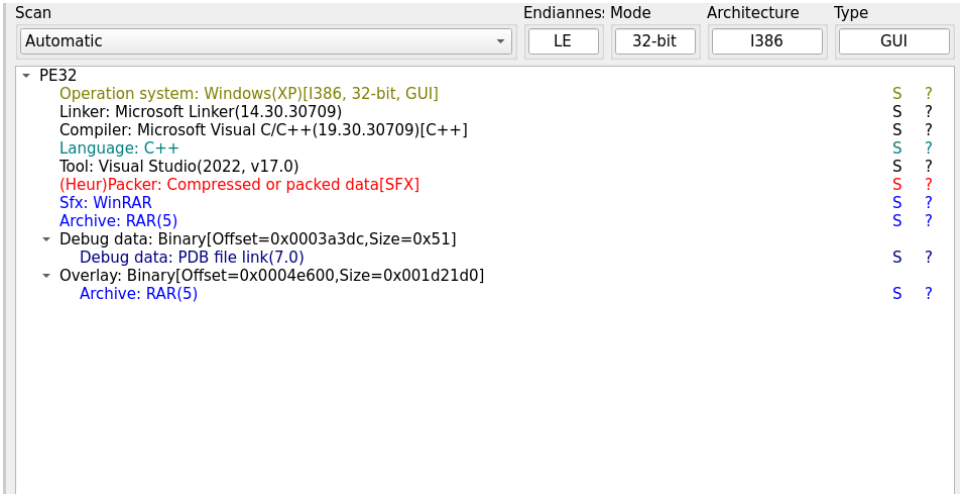


Gambar 2. Informasi detail sampel malware random.exe.

3.2. Identifikasi Jenis dan Struktur

a. Analisis menggunakan Detect it Easy

Deteksi jenis *file*, *compiler*, dan kemungkinan *packer* dilakukan menggunakan aplikasi Detect it Easy. Hasil pemindaian menunjukkan bahwa *file* tersebut merupakan *file* PE32 (Windows) dengan arsitektur i386 atau 32-bit dan menggunakan mode *Graphical User Interface* (GUI). *File* ini dikompilasi menggunakan Microsoft Visual C/C++ versi 19 yang merupakan bagian dari *tools* Visual Studio 2022. Pada analisis *heuristic packer*, ditemukan indikasi bahwa data dalam *file* tersebut dikompresi atau dikodekan ulang (compressed or packed data). Selain itu, file ini juga dideteksi sebagai *Self-Extracting Archive* (SFX) yang berarti *file* tersebut berfungsi seperti arsip yang dapat mengekstrak dan menjalankan dirinya sendiri secara otomatis.



Gambar 3. Analisis File dengan Detect it Easy

Implikasi dari temuan ini bagi analisis *malware* adalah bahwa penggunaan *packer* merupakan teknik umum yang sering dimanfaatkan *malware* untuk menyembunyikan isi atau perilaku berbahayanya dari deteksi antivirus.

b. Analisis menggunakan PE Studio

Vol. 10, No. 2, Oktober 2025, Page. 507-517
https://journal.uin-alauddin.ac.id/index.php/instek
p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

Analisis metadata awal menggunakan PE Studio memperkuat temuan sebelumnya, yaitu *file* yang dianalisis teridentifikasi sebagai *file Portable Executable* (PE) 32-bit untuk sistem operasi Windows dengan indikasi keberadaan *packer*.

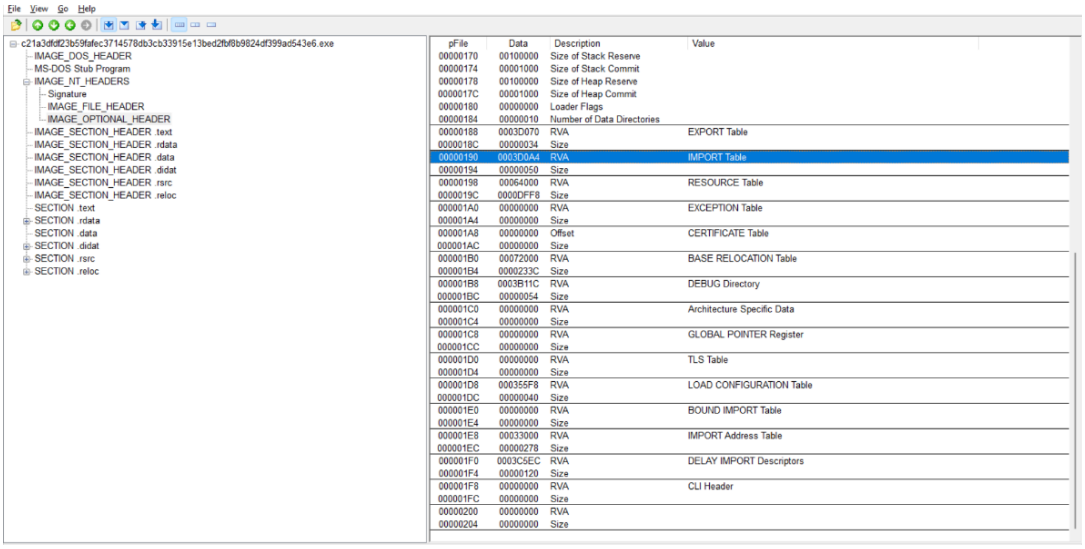
indicator (17)	detail
file > name	[redacted]\sampel\c21a3dfdf23b59fafec...
file > signature	Microsoft Linker 14.30 | Microsoft Visual C++ 6.0 - 8.0 | Visual Studio 2013
file > info	size: 2230224 bytes, entropy: 7.936
file > type	executable, 32-bit, GUI
stamp > compiler	Thu Mar 03 13:15:57 2022
languages > names	English-US
resources > info	count: 27, size: 55642 bytes, file-ratio: 2.49%
manifest > general	name: WinRAR SFX, description: WinRAR SFX module, severity: asInvoker
file > version	n/a
entry-point > location	0x0001F530 (section: .text)
certificate	n/a
imphash > md5	F247D9D88A177486653FC8143140D2C0
exports	n/a
overlay > info	signature: unknown, offset: 0x0004E600, size: 1909200 bytes, entropy: 8.0...
overlay > first 1909200 bytes (hex)	52 61 72 21 1A 07 01 00 E2 B1 B3 5A 0C 01 05 08 00 07 01 01 C3 C0 F4 80 0...
overlay > first 1909200 bytes (text)	Rar!.....Z.....i.....D.....CMT;The comment ...
overlay > entropy	8.000

Gambar 4. Analisis File dengan PE Studio

Berdasarkan hasil analisis dari kedua *tools* tersebut, *file* yang diperiksa merupakan *executable* windows-32 yang menggunakan teknik *packing*. Hal ini mengindikasikan bahwa *file* tersebut kemungkinan dirancang untuk mengelabui sistem keamanan dengan menyembunyikan kode asli, sehingga mempersulit proses analisis lebih lanjut.

3.3. Ekstraksi metadata file

Proses ekstraksi metadata dilakukan dengan menggunakan PEview, sebuah *tools* yang umum digunakan dalam analisis struktur internal file PE. Informasi metadata dasar seperti *timestamp*, ukuran file, dan karakteristik kompilasi juga tersedia pada *tools* lain yang digunakan sebelumnya, seperti PE Studio dengan hasil yang relatif serupa.



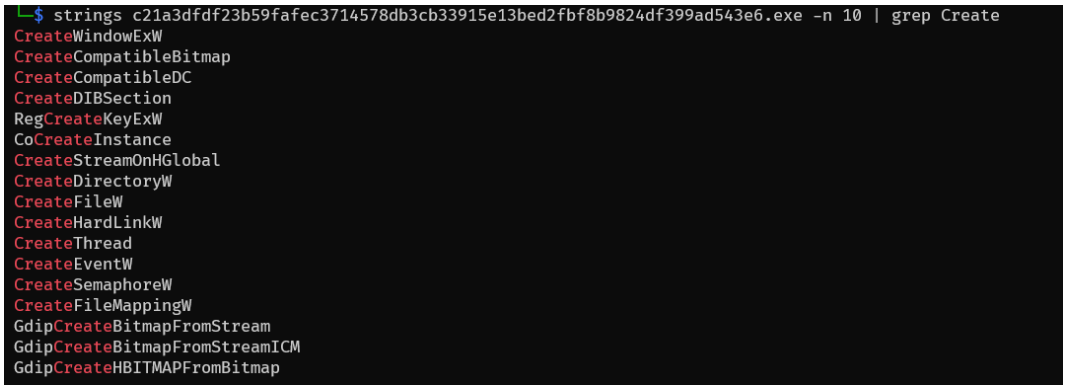
Gambar 5. Ekstraksi metadata file

Namun demikian, pada analisis ini PEview tidak menampilkan informasi terkait *import* dan *export table*. Ketidakhadiran informasi ini kemungkinan disebabkan oleh teknik *packing* atau *obfuscation* yang diterapkan pada *malware* sehingga struktur PE tidak dapat dikenali sepenuhnya oleh *tool*

Vol. 10, No. 2, Oktober 2025, Page. 507-517
https://journal.uin-alauddin.ac.id/index.php/instek
p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

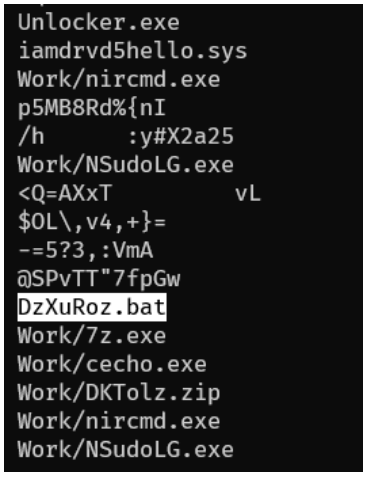
3.4. Ekstraksi String dan Resource Tersembunyi

Proses ekstraksi string dilakukan untuk mengidentifikasi teks tersembunyi, nama fungsi *Application Programming Interface* (API) dan indikasi fungsionalitasnya dari sampel *malware*. Tujuan dari tahapan ini adalah untuk memahami potensi komunikasi jaringan, manipulasi *file*, serta teknik evasi yang mungkin digunakan oleh *malware*.



Gambar 6. Hasil Ekstraksi String API

Hasil ekstraksi menunjukkan sejumlah *string* penting yang terkandung dalam sampel, termasuk referensi terhadap API Windows yang umum digunakan untuk melakukan modifikasi file, membuat *thread* baru, hinggal menghentikan proses lain secara paksa. Hal ini mengindikasikan adanya mekanisme kontrol proses secara aktif dari dalam *malware*.



Gambar 7. Ekstraksi File terkait Eksekusi

Dugaan awal mengarah pada kemungkinan bahwa sampel malware ini merupakan *self-extracting archive* berbasis RAR dengan *payload* disimpan di bagian dalam file dan diekstrak saat *runtime*. Selain itu, ditemukan pula string yang menunjukkan keberadaan *file* berekstensi .exe dan .bat yang diduga merupakan bagian dari payload yang digunakan untuk mengeksekusi perintah secara jarak jauh / *Remote Code Execution* (RCE).

Berdasarkan hasil ekstraksi string dari sampel *malware* yang dianalisis, ditemukan sejumlah fungsi API yang memberikan indikasi terhadap perilaku dan kapabilitas *malware* tersebut. Temuan ini dapat dikelompokkan ke dalam beberapa kategori berikut:

a. Fungsi antarmuka Grafis (GUI)

Fungsi seperti *ShowWindow*, *GetDlgItem*, *SetWindowTextW*, *DialogBoxParamW*, *MessageBoxW*, *CreateWindowExW*, *DestroyWindow* menunjukkan bahwa *malware*

p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

kemungkinan memiliki antarmuka pengguna grafis atau menggunakan elemen GUI untuk menyamarkan aktivitas jahatnya sebagai aplikasi yang sah.

b. Manipulasi File Sistem
Fungsi *CreateFileW, DeleteFileW, SetFileAttributesW, GetFileAttributesW, MoveFileExW, SetFileTime, FlushFileBuffers, FindFirstFileW, FindNextFileW* menandakan bahwa *malware* memiliki kemampuan untuk membaca, menulis, memodifikasi, dan menghapus *file* maupun direktori dalam sistem target.

c. Manajemen Proses dan Thread
Fungsi seperti *CreateThread, WaitForSingleObject, GetCurrentProcessId, TerminateProcess, SetThreadPriority,* dan *ExitProcess* mengindikasikan kemampuan *malware* dalam mengelola eksekusi, membuat *thread* baru atau menghentikan proses yang berjalan.

d. Anti-Debugging dan Teknik Evasion
Adanya fungsi seperti *IsDebuggerPresent, GetStartupInfoW, UnhandledExceptionFilter, SetUnhandledExceptionFilter, RaiseException, VirtualProtect, EncodePointer* dan *DecodePointer* memperlihatkan bahwa *malware* menerapkan teknik untuk menghindari deteksi atau analisis melalui pengacakan pointer dan pengelolaan *handler exception*.

e. Modifikasi Registry Windows
Fungsi *RegOpenKeyExW, RegSetValueExW, RegQueryValueExW, RegCreateKeyExW* dan *RegCloseKey* menunjukkan bahwa *malware* memanfaatkan *windows registry* untuk menyimpan konfigurasi, mengatur persistensi atau memodifikasi pengaturan sistem.

f. Eksekusi Perintah dan Proses Baru
Fungsi seperti *ShellExecuteExW, CreateProcess, GetCommandLineW,* dan *GetEnvironmentStringsW* menunjukkan bahwa *malware* mampu menjalankan perintah sistem atau memicu eksekusi *payload* tambahan.

g. Pemrosesan Gambar dan Grafik
Fungsi seperti *GdipCreateBitmapFromStream, GdipDisposeImage, CreateCompatibleBitmap,* dan *SelectObject, StretchBlt* mungkin digunakan *malware* untuk menyamarkan aktivitas dengan grafis atau sebagai bagian dari teknik manipulasi visual pada sistem korban.

h. Manajemen Meori dan Objek COM
Fungsi seperti *CoCreateInstance, CreateStreamOnHGlobal, VirtualAlloc, MapViewOfFile,* dan *HeapReAlloc* mengindikasikan pemanfaatan alokasi memori dinamis serta objek COM yang dapat digunakan untuk komunikasi interna; atau penyimpanan data terenkripsi.

i. Pengumpulan Informasi Sistem
Fungsi *GetSystemInfo, GetSystemMetrics, GetVersionExW,* dan *GetProcessHeap* memperlihatkan bahwa *malware* berusaha mengumpulkan data sistem untuk menyesuaikan perilaku atau menentukan vector serangan yang optimal terhadap target.

3.5. Disassembly / Dekompilasi menggunakan Ghidra

Setelah melakukan dekompilasi menggunakan Ghidra, ditemukan kode yang mencurigakan pada fungsi yang memanggil *shellExecuteExW* dengan parameter *runas*, yang tersimpan pada *offset unaf_EBP + (-0x58)*. Penggunaan parameter ini mengindikasikan adanya upaya eksplisit untuk melakukan elevasi hak akses ke tingkat administratif.

Vol. 10, No. 2, Oktober 2025, Page. 507-517
https://journal.uin-alauddin.ac.id/index.php/instek
p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

```
265             FID_conflict:_memcpy(_Dst,&DAT_00457b80,0x7104);
266           }
267           BVar5 = ShellExecuteExW((SHELLEXECUTEINFOW *)(unaff_EBP + -0x58));
268           FUN_0040f445(unaff_EBP + -0x574,0x80);
269           FUN_0040f445(unaff_EBP + -0xfe8c,0x430c);
270           if (BVar5 == 0) {
271             lpBaseAddress = *(char **)(unaff_EBP + -0x1c);
272             *(undefined *)(unaff_EBP + -0xd) = 1;
273           }
274           else {
275             WaitForInputIdle(*(HANDLE *)(unaff_EBP + -0x20),10000);
276             *(undefined4 *)(unaff_EBP + -0x18) = 0;
277             lpBaseAddress = *(char **)(unaff_EBP + -0x1c);
```

Gambar 8. Eksekusi shellExecuteEXW untuk Elevasi Akses

Selanjutnya, ditemukan pemanggilan fungsi *FUN_0040a0b1* yang berfungsi untuk memverifikasi akses terhadap direktori tertentu. Jika akses ditolak dan fungsi GetLastError() mengembalikan nilai kesalahan 5 ERROR_ACCESS_DENIED, maka hak akses sebagai respons terhadap kegagalan akses tersebut.

```
199         *(undefined *)(unaff_EBP + -0xd) = 0;
200         cVar1 = FUN_0040a0b1(unaff_EBP + -0x2474,0,0);
201         if ((cVar1 == '\0') && ((DVar7 = GetLastError(), DVar7 == 5 || (DVar7 == 3)))) {
202           *(undefined *)(unaff_EBP + -0xd) = 1;
203         }
204         cVar1 = FUN_0041ac04(unaff_EBP + -0x2474);
```

Gambar 9. Deteksi Akses ditolak

Selain itu, terdapat argumen *command line* yang mencurigakan, seperti *-el -s2 "-d%s" "-sp%s"*, yang kemungkinan besar merupakan *switch* atau parameter khusus yang ditujukan untuk proses yang dijalankan setelah berhasil dielevasi. Ini menandakan bahwa *malware* telah diprogram untuk melakukan eksekusi tahap lanjut setelah memperoleh hak administratif.

```
232           GetModuleFileNameW((HMODULE)0x0,(LPWSTR)(unaff_EBP + -0x3474),0x800);
233           FUN_0040f28c(unaff_EBP + -0x574,0x80);
234           FUN_00404092(unaff_EBP + -0xfe8c,0x430c,L"-el -s2 \"-d%s\" \"-sp%s\"",
235                        unaff_EBP + -0x2474,&DAT_0044b472);
236           *(undefined4 *)(unaff_EBP + -0x58) = 0x3c;
237           *(undefined4 *)(unaff_EBP + -0x54) = 0x40;
```

Gambar 10. Command Line untuk Elevasi Proses

Untuk menutupi aktivitas berbahaya tersebut, *malware* juga menerapkan teknik *social engineering* dengan menampilkan dialog bertuliskan "LICENSEDLG" yang dimaksudkan untuk mengelabui pengguna agar mengira bahwa proses yang sedang berjalan adalah bagian dari prosedur lisensi resmi. Teknik ini digunakan untuk menyamarkan tindakan eksekusi atau instalasi yang tidak sah di latar belakang.

```
  }
  if (DAT_0045fca0 != 0) {
    FUN_0041c73f(hWnd,DAT_0045fc90,0);
    if ((DAT_0045fc94 != (void *)0x0) && (DAT_0044a46c == 0)) {
      FUN_00419ed5(DAT_0044102c,*(undefined4 *)(unaff_EBP + -0x14),DAT_0045fc94,0,0);
      FID_conflict:_free(DAT_0045fc94);
    }
    if ((((DAT_0044a46c != 1) && (SetForegroundWindow(hWnd), DAT_0044a46c != 1)) &&
         (DAT_0044a471 == '\0')) &&
        ((FUN_0041c73f(hWnd,DAT_0045fc90,3), DAT_0045fc98 != 0 &&
          (IVar14 = DialogBoxParamW(DAT_0044102c,L"LICENSEDLG",(HWND)0x0,FUN_0041b5c0,0),
          IVar14 == 0)))) {
B_0041b958:
      DAT_00448454 = '\x01';
B_0041b95e:
      IVar14 = 1;
```

Gambar 11. Pengelabuan Pengguna dengan Dialog “LICENSEDLG”

Vol. 10, No. 2, Oktober 2025, Page. 507-517
https://journal.uin-alauddin.ac.id/index.php/instek
p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

3.6. Analisis API Claas dan Library

Analisis hasil disassembly menunjukkan bahwa *malware* memanfaatkan fungsi Windows API *CreateFileMappingW* untuk membuat sebuah *file mapping* ini mengindikasikan bahwa *malware* berusaha menciptakan ruang *shared memory* yang dapat diakses oleh proses lain. Teknik ini umum digunakan dalam skenario komunikasi antar proses / *inter-process communication* (IPC), yang memungkinkan dua atau lebih proses untuk berbagi data secara langsung di memori tanpa perlu melalui file atau jaringan.

```
*(undefined **)(unaff_EBP + -0x40) = &DAT_00448468;
hFileMappingObject =
     CreateFileMappingW((HANDLE)0xffffffff,(LPSECURITY_ATTRIBUTES)0x0,0x8000004,0,0x...
     04
                        ,L"winrarsfxmappingfile.tmp");
*(HANDLE *)(unaff_EBP + -0x14) = hFileMappingObject;
if (hFileMappingObject == (HANDLE)0x0) {
  *(undefined4 *)(unaff_EBP + -0x1c) = *(undefined4 *)(unaff_EBP + -0x14);
}
else {
  DAT_00457b80 = 0;
  pWVar8 = GetCommandLineW();
  if (pWVar8 != (LPWSTR)0x0) {
    FUN_00410602(&DAT_00457b82,pWVar8,0x2000);
  }
```

Gambar 12. CreateFileMappingQ pada Malware

Selain menciptakan *file mapping*, *malware* juga menggunakan *OpenFileMappingW* untuk membuka kembali *file mapping* yang telah dibuat sebelumnya. Kombinasi antara *CreateFileMappingW* dan *OpenFileMappingW* merupakan indikator teknik persistence atau koordinasi antara proses *malware*. Dengan memanfaatkan pendekatan ini, *malware* dapat memastikan kelangsungan eksekusinya, bahkan setelah proses utama dihentikan atau dapat digunakan untuk menyampaikan perintah dan data antar modul. Teknik ini sering ditemukan dalam *malware* yang dirancang untuk bertahan lama dalam sistem target dan menghindari deteksi oleh mekanisme keamanan tradisional.

```
}
else {
  hFileMappingObject = OpenFileMappingW(0xf001f,0,L"winrarsfxmappingfile.tmp");
  if (hFileMappingObject != (HANDLE)0x0) {
    _Src = (undefined *)MapViewOfFile(hFileMappingObject,0xf001f,0,0,0x7104);
    if (_Src != (undefined *)0x0) {
      FID_conflict:_memcpy(&DAT_00457b80,_Src,0x7104);
      *_Src = 1;
      FUN_0041dbde(&DAT_00457b82);
      puVar3 = (undefined4 *)&DAT_0045eb82;
      puVar4 = (undefined4 *)&DAT_0044c472;
      for (iVar2 = 0x40; iVar2 != 0; iVar2 = iVar2 + -1) {
        *puVar4 = *puVar3;
        puVar3 = puVar3 + 1;
        puVar4 = puVar4 + 1;
      }
      *(undefined2 *)puVar4 = *(undefined2 *)puVar3;
    }
    UnmapViewOfFile(_Src);
  }
  CloseHandle(hFileMappingObject);
}
```

Gambar 13. Akses File Mapping oleh Malware

Berdasarkan hasil analisis Disassembly / Dekompilasi menggunakan Ghidra dan Analisis API Calls dan Library, dapat disimpulkan bahwa *malware* ini berkamuflase sebagai aplikasi yang terlihat sah, yaitu *WinRAR SFX*. Teknik ini digunakan untuk mengelabui pengguna maupun perangkat lunak anti virus. *Malware* ini menunjukkan indikator jelas dari aktivitas berbahaya seperti *commad-line injection* dengan parameter mencurigakan, upaya *User Account Control* (UAC) *bypass* untuk mendapatkan hak administratif, serta penyalahgunaan antarmuka pengguna berupa *misdirection* untuk mengecoh pengguna. Selain itu, penerapan teknik obfuscation yang kompleks menandakan bahwa ini merupakan

p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

malware tingkat lanjut, kemungkinan termasuk dalam kategori *Trojan* atau *Dropper* yang berfungsi mengunduh atau menjalankan komponen tambahan setelah mendapatkan akses yang lebih tinggi.
Hasil analisis teknis:

1. Ditemukan penggunaan sejumlah API dan string yang mencurigakan;
2. Struktur program tidak lazim dan tidak mengikuti pola aplikasi sah pada umumnya;
3. Terdapat indikasi kuat penggunaan teknik evasif, seperti anti-debugging dan obfuskasi kode untuk menghindari deteksi dan analisis

Berdasarkan perilaku dan teknik yang digunakan, *malware* ini diklasifikasikan dalam kategori bahaya sedang (Medium Threat Level). Meskipun belum menunjukkan aktivitas destruktif secara eksplisit, kemampuan untuk melakukan eskalasi hak akses, menyembunyikan aktivitas, serta potensi untuk menurunkan muatan tambahan menjadikannya ancaman serius dalam sistem yang tidak memiliki perlindungan memadai.

4. KESIMPULAN

Penelitian ini menunjukkan bahwa *reverse engineering* berbasis analisis statis dengan Ghidra efektif mengidentifikasi teknik penyamaran, injeksi, dan eskalasi hak akses yang digunakan *malware* tingkat menengah. Implikasi praktis bagi praktisi keamanan siber adalah bahwa pendekatan ini dapat dijadikan acuan untuk deteksi awal *malware* tanpa risiko eksekusi langsung. Keterbatasan penelitian ini terletak pada belum mencakup pengamatan perilaku dinamis atau *fileless malware*. Saran penelitian selanjutnya adalah pengembangan analisis *hybrid* yang menggabungkan pendekatan statis dan dinamis, serta penerapan *machine learning* untuk klasifikasi otomatis berdasarkan pola API dan struktur kode.

ACKNOWLEDGEMENT

Penelitian ini didukung oleh Hibah Internal Institut Teknologi Bachruddin Jusuf Habibie (ITH) Tahun 2025. Penulis berterima kasih atas dukungan dan fasilitas yang diberikan sehingga penelitian ini dapat diselesaikan dengan baik.

DAFTAR PUSTAKA

[1] Patria Nezar *et al.*, “Pembangunan Digital Indonesia,” Indonesia, Aug. 2024.

[2] M. Hazri, “Analisis Malware PlasmaRAT dengan Metode Reverse Engineering,” *J. Rekayasa Teknol. Inf. JURTI*, vol. 4, no. 2, p. 192, Nov. 2020, doi: 10.30872/jurti.v4i2.4131.

[3] M. R. T. Hidayat, N. Widiyasono, and R. Gunawan, “OPTIMASI DETEKSI MALWARE PADA SIEM WAZUH MELALUI INTEGRASI CYBER THREAT INTELLIGENCE DENGAN MISP DAN DFIR-IRIS,” *J. Inform. Dan Tek. Elektro Terap.*, vol. 13, no. 1, Jan. 2025, doi: 10.23960/jitet.v13i1.5686.

[4] H. N. Aditya, N. Widiyasono, and A. Rahmatulloh, “Analisis Malware Aquvaprn.exe Untuk Investigasi Sistem Operasi Dengan Metode Memory Forensics,” *J. Tek. Inform. Dan Sist. Inf.*, vol. 10, no. 2, pp. 161–172, Aug. 2024, doi: 10.28932/jutisi.v10i2.6562.

[5] A. R. Damanik, H. B. Seta, and T. Theresiawati, “ANALISIS TROJAN DAN SPYWARE MENGGUNAKAN METODE HYBRID ANALYSIS,” *J. Ilm. Matrik*, vol. 25, no. 1, pp. 89–97, May 2023, doi: 10.33557/jurnalmatrik.v25i1.2327.

[6] SATRIO PANGARSO WISANGGENI, “Siapa Itu Brain Cipher, Operator Serangan ”Ransomware” PDN?,” *https://www.kompas.id/baca/investigasi/2024/07/03/siapa-itu-brain-cipher-operator-serangan-ransomware-pdn*, July 03, 2024.

[7] N. H. IQBAL BASYARI, “BKN Diduga Diretas, Peretas Tawarkan Data ASN Rp 160 Juta,” *https://www.kompas.id/baca/polhuk/2024/08/11/bkn-diduga-diretas-peretas-tawarkan-data-asn-rp-160-juta*, Aug. 11, 2024.

[8] A. Damodaran, F. Di Troia, V. A. Corrado, T. H. Austin, and M. Stamp, “A Comparison of Static, Dynamic, and Hybrid Analysis for Malware Detection,” Mar. 2022, doi: 10.1007/s11416-015-0261-z.

p-ISSN: 2541-1179
e-ISSN: 2581-1711
JURNAL INSTEK
INFORMATIKA SAINS DAN TEKNOLOGI

[9] Y. Kamalrul Bin Mohamed Yunus and S. Bin Ngah, "Review of Hybrid Analysis Technique for Malware Detection," in *IOP Conference Series: Materials Science and Engineering*, Institute of Physics Publishing, June 2020. doi: 10.1088/1757-899X/769/1/012075.

[10] I. Gede Adnyana, P. Gede, S. Cipta Nugraha, B. Rahmat, and A. Nugroho, "Reverse Engineering for Static Analysis of Android Malware in Instant Messaging Apps," *Archit. High Perform. Comput.*, vol. 6, no. 3, 2024, doi: 10.47709/cnapc.v6i3.4417.

[11] K. Shaukat, S. Luo, and V. Varadharajan, "A novel deep learning-based approach for malware detection," *Eng. Appl. Artif. Intell.*, vol. 122, June 2023, doi: 10.1016/j.engappai.2023.106030.

[12] R. Jusoh, A. Firdaus, S. Anwar, M. Z. Osman, M. F. Darmawan, and M. F. A. Razak, "Malware Detection Using Static Analysis in Android: a review of FeCO (Features, Classification, and Obfuscation)," *PeerJ Comput. Sci.*, vol. 7, pp. 1–54, 2021, doi: 10.7717/peerj-cs.522.

[13] Muhammad Taseer Suleman, "Malware Detection and Analysis Using Reverse Engineering," *Int. J. Electron. Crime Investig.*, vol. 8, no. 1, pp. 109–123, Mar. 2024, doi: 10.54692/ijeci.2024.0801191.

[14] A. Amiruddin, P. N. H. Suryani, S. D. Santoso, and M. Y. B. Setiadji, "Utilizing Reverse Engineering Technique for A Malware Analysis Model," *Sci. J. Inform.*, vol. 8, no. 2, pp. 222–229, Nov. 2021, doi: 10.15294/sji.v8i2.24755.

[15] E. Tansen and D. W. Nurdiarto, "Analisis dan Deteksi Malware dengan Metode Hybrid Analysis Menggunakan Framework MOBSF," *J. Teknol. Inf.*, vol. 4, no. 2, pp. 191–201, Dec. 2020, doi: 10.36294/jurti.v4i2.1338.

[16] Nur Widiyasono, Siti Rahayu Selamat, Angga Sinjaya, Rianto, Randi Rizal, and Mugi Praseptiawan, "Investigation of Malware Redline Stealer Using Static and Dynamic Analysis Method Forensic," *J. Adv. Res. Appl. Sci. Eng. Technol.*, vol. 48, no. 2, pp. 49–62, July 2024, doi: 10.37934/araset.48.2.4962.