ANALISIS DAN DEKRIPSI RC4 PADA MALWARE (STUDI KASUS: KEYLOGGER)

Kornelis Febriano Kapa Api, I Gede Putu Krisna Juliharta, Made Adi Paramartha Putra
Informatika, Universitas Primakara
Jl. Tukad Badung No.135, Renon, Denpasar Selatan, Kota Denpasar, Bali 80226, Indonesia
rynnoapi.02@gmail.com

ABSTRAK

Penelitian ini bertujuan untuk menganalisis dan mendekripsi algoritma RC4 yang digunakan dalam *malware*, dengan studi kasus pada *keylogger*. RC4 merupakan algoritma enkripsi *stream cipher* yang sering dimanfaatkan oleh *malware* untuk menyembunyikan aktivitas berbahaya. Proses dekripsi dilakukan menggunakan metode *reverse engineering* dan emulasi dengan Unicorn Engine, sebuah CPU *emulator* berbasis QEMU yang mendukung multi-arsitektur dan memiliki API yang ringan serta fleksibel. Penelitian ini dimulai dengan analisis statis terhadap sampel malware untuk mengidentifikasi lokasi algoritma RC4, diikuti dengan *disassembling*, *decompiling*, dan identifikasi *byte code* yang digunakan untuk enkripsi. Selanjutnya, *byte code* tersebut dijalankan dalam lingkungan emulasi menggunakan Unicorn Engine dengan konfigurasi memori dan register CPU yang sesuai. Proses dekripsi dilakukan dengan mengeksekusi kode RC4 secara parsial dan terkontrol, tanpa menjalankan keseluruhan *malware* secara langsung di sistem. Hasil penelitian menunjukkan bahwa pendekatan ini berhasil mendekripsi data hasil *keylogging* dengan rata-rata tingkat keberhasilan sebesar 94,46% dari total 278.983 *byte* data yang terenkripsi. Selain meningkatkan efisiensi analisis, metode ini juga mengurangi risiko infeksi sistem dan memungkinkan ekstraksi informasi penting dari *malware* secara aman. Dengan demikian, penggunaan *emulator* Unicorn terbukti efektif dalam mendukung proses dekripsi algoritma RC4 pada malware.

Kata kunci : *Malware, RC4, Reverse Engineering, Emulasi, Unicorn Engine, Dekripsi, Keylogger*

1. PENDAHULUAN

Dalam dunia keamanan informasi, *malware* menjadi ancaman yang terus berkembang dan menantang [1] [2] [3]. *Malware* adalah perangkat lunak yang dirancang untuk merusak, mengganggu, atau mengambil kontrol atas sistem komputer tanpa izin pengguna. Salah satu aspek yang membuat *malware* menjadi sulit untuk diatasi adalah penggunaan teknik enkripsi untuk menyembunyikan perilaku jahatnya. Salah satu algoritma enkripsi yang sering digunakan oleh *malware* adalah RC4 (Rivest Cipher 4) [4] [5] [6] [7]. RC4 adalah algoritma symmetric key yang ditemukan oleh Ron Rivest pada tahun 1987. Algoritma ini telah digunakan secara luas dalam berbagai aplikasi, termasuk protokol keamanan seperti SSL/TLS. Namun, kelemahan dalam implementasi RC4 telah terungkap, membuatnya rentan terhadap serangan dan penguraian [8]. Meskipun demikian, banyak pembuat *malware* masih menggunakan RC4 sebagai metode enkripsi untuk menyembunyikan *payload* dan perilaku jahatnya dari deteksi dan analisis.

Peneliti keamanan informasi memainkan peran kunci dalam memerangi ancaman *malware*. Salah satu tantangan utama yang dihadapi oleh para peneliti adalah kemampuan untuk menganalisis dan mengatasi *malware* yang menggunakan enkripsi RC4. Proses dekripsi *malware* dapat menjadi sangat sulit apabila algoritma RC4 sudah dimodifikasi dan sering kali memerlukan upaya *reverse engineering* yang intensif [9] [10] [11]. Metode tradisional untuk menganalisis *malware* RC4 melibatkan proses manual *reverse engineering* yang melibatkan pemahaman struktur kode dan algoritma enkripsi yang terlibat. Namun, pendekatan ini membutuhkan waktu dan tenaga yang signifikan, terutama karena kompleksitas yang mungkin terdapat dalam malware modern. Selain itu, proses ini dapat menjadi sulit jika malware telah dirancang untuk menghambat analisis dan deteksi.

Untuk mengatasi tantangan ini, diperlukan metode *reverse engineering* dan alat bantu yang lebih efisien. Salah satu pendekatan yang menjanjikan adalah penggunaan emulasi untuk menjalankan *malware* dalam lingkungan terkendali dan terisolasi. Emulasi memungkinkan peneliti untuk memeriksa perilaku *malware* tanpa harus menjalankan secara langsung pada sistem *host*. Unicorn adalah *framework* emulasi ringan dan fleksibel yang dapat digunakan untuk menjalankan kode biner pada berbagai arsitektur dan platform [12]. Dengan menggunakan Unicorn, peneliti keamanan dapat membuat lingkungan yang aman di mana mereka dapat menjalankan dan menganalisis *malware* tanpa risiko merusak sistem *host* [13]. Dengan memadukan metode *reverse engineering* dan emulasi menggunakan Unicorn, maka dapat dikembangkan pendekatan baru untuk menganalisis dan mengatasi *malware* RC4. Tujuan utamanya adalah untuk memfasilitasi proses dekripsi *malware*, memungkinkan peneliti keamanan untuk mengidentifikasi dan merespons ancaman dengan lebih cepat dan efisien.

2. TINJAUAN PUSTAKA

Berikut merupakan landasan teori dalam penelitian ini.

2.1. Kriptografi

Kriptografi adalah ilmu dan seni mengamankan informasi dengan mengubahnya menjadi bentuk yang tidak dapat dibaca oleh pihak yang tidak berwenang. Ini melibatkan teknik matematika dan algoritma yang dirancang untuk melindungi data dari akses tidak sah, modifikasi, atau gangguan [14]. Kriptografi mencakup berbagai metode dan konsep seperti enkripsi, dekripsi, *hash*, tanda tangan digital, dan autentikasi.

a. Konsep dasar kriptografi
   Berikut adalah beberapa konsep dasar kriptografi [15]:
   - Enkripsi: Proses mengubah plaintext menjadi ciphertext menggunakan algoritma dan kunci enkripsi. Ciphertext adalah teks yang tidak dapat dibaca tanpa kunci dekripsi yang benar.
   - Dekripsi: Proses mengubah ciphertext kembali menjadi plaintext menggunakan algoritma dan kunci dekripsi yang sesuai.
   - Kunci Kriptografi / Key:
     Kunci simetris: menggunakan kunci yang sama untuk enkripsi dan dekripsi.
     Contoh: RC4, Advanced Encryption Standard (AES), Data Encryption Standard (DES).
     Kunci asimetris: menggunakan pasangan kunci publik dan kunci privat.
     Kunci publik digunakan untuk enkripsi, sedangkan kunci privat digunakan untuk dekripsi.
     Contoh: Rivest–Shamir–Adleman (RSA), Elliptic Curve Cryptography (ECC).

b. Aplikasi kriptografi
   Berikut adalah beberapa aplikasi kriptografi [17][18]:
   - Internet dan jaringan: Secure Socket Layer / Transport Layer Security (SSL/TLS), Virtual Private Network (VPN).
   - Perbankan dan keuangan: ATM, tanda tangan digital untuk autentikasi.
   - Perangkat mobile: enkripsi end-to-end untuk aplikasi pesan dan keamanan data.
   - Blockchain dan Cryptocurrency: digunakan untuk mengamankan transaksi dan membentuk konsensus.
   - Tanda tangan digital (digital signatures): memverifikasi keaslian dan integritas data menggunakan algoritma asimetris.
   - Fungsi hash: algoritma yang menghasilkan nilai tetap (hash value) untuk memastikan integritas data.
     Contoh: SHA-256, MD5.

2.2. Algoritma RC4

Algoritma RC4 adalah sebuah algoritma kunci simetris yang dirancang oleh Ronald Rivest pada tahun 1987. Algoritma ini dikenal karena kesederhanaannya dan kecepatan operasinya sehingga banyak digunakan dalam berbagai aplikasi kriptografi [16].

a. Komponen utama RC4
   RC4 terdiri dari dua komponen utama:
   - Key Scheduling Algorithm (KSA):
     KSA adalah langkah pertama RC4 yang berfungsi untuk menginisialisasi state array S menggunakan kunci enkripsi K.
     State array S adalah array berukuran 256 byte yang berisi nilai-nilai dari 0 hingga 255:
     S = {0, 1, 2, ..., 255}
     j = 0
     for i from 0 to 255
     j = (j + S[i] + K[i mod length(K)]) mod 256
     Swap S[i] and S[j]
   - Pseudo-Random Generation Algorithm (PRGA):
     Langkah kedua untuk menghasilkan keystream (K).
     i = 0
     j = 0
     while generating output:
     i = (i + 1) mod 256
     j = (j + S[i]) mod 256
     Swap S[i] and S[j]
     K = S[(S[i] + S[j]) mod 256]
     output K
   - Proses enkripsi dan dekripsi
     RC4 menggunakan operasi XOR (exclusive OR) untuk menggabungkan keystream dengan plaintext atau ciphertext:
     enkripsi: ciphertext = plaintext XOR keystream
     dekripsi: plaintext = ciphertext XOR keystream

2.3. Emulasi

Emulasi adalah proses di mana suatu sistem komputer meniru (emulates) fungsi perangkat keras atau perangkat lunak sistem lain, memungkinkan program atau aplikasi yang dirancang untuk sistem tersebut untuk berjalan di lingkungan yang berbeda. Dalam konteks penelitian keamanan informasi dan analisis malware, emulasi sering digunakan untuk menjalankan dan menganalisis perangkat lunak berbahaya dalam lingkungan yang terkendali tanpa risiko merusak sistem host yang sebenarnya.

a. Penggunaan emulasi dalam analisis malware
   Berikut adalah penggunaan emulasi:
   - Analisis dinamis: emulasi memungkinkan peneliti menjalankan malware dalam lingkungan terkendali dan mengamati perilakunya secara langsung, tanpa risiko infeksi ke sistem host.
   - Pemahaman algoritma enkripsi: dengan meniru eksekusi malware, peneliti dapat mengamati bagaimana algoritma enkripsi

dijalankan dan bagaimana kunci enkripsi dihasilkan dan digunakan.
- Debugging dan reverse engineering: emulator memungkinkan peneliti untuk menjalankan malware langkah demi langkah, memeriksa register, memori, dan aliran instruksi, yang sangat berguna dalam proses reverse engineering.
- Pengujian hipotesis: peneliti dapat mengubah parameter eksekusi, menginjeksikan kode, atau memodifikasi memori untuk menguji hipotesis tentang bagaimana malware beroperasi atau bagaimana mekanisme dekripsi.

b. Emulator Unicorn
Unicorn adalah framework emulasi Central Processing Unit (CPU) yang fleksibel dan ringan, yang dirancang untuk mendukung banyak arsitektur prosesor. Unicorn menggunakan QEMU (Quick Emulator) sebagai mesin emulasi dasar dan menyediakan API yang sederhana untuk mengontrol eksekusi instruksi. Berikut adalah alasan mengapa memilih Unicorn sebagai emulator dibanding Bochs, QEMU (CPU mode), dan Qiling:
- Multi-arsitektur: mendukung berbagai arsitektur CPU seperti x86, x86_64, ARM, ARM64, M68K, MIPS, SPARC, dan lain-lain.
- Emulasi instruksi: Unicorn mampu meniru instruksi prosesor secara akurat, memungkinkan analisis mendalam terhadap kode yang dijalankan.
- Application Programming Interface (API) yang mudah digunakan: menyediakan API yang mudah digunakan dalam bahasa pemrograman seperti C dan Python, yang memudahkan pengembangan skrip dan alat analisis.
- Efisien dan ringan: dibangun di atas QEMU, Unicorn dioptimalkan untuk kecepatan dan efisiensi, memberikan kemudahan dalam analisis.

2.4. Penelitian Terdahulu

Penelitian terbaru dalam bidang keamanan informasi telah mengeksplorasi berbagai pendekatan dalam upaya untuk dekripsi data yang terenkripsi oleh malware. Babiano et al. dalam penelitiannya terhadap algoritma yang mirip Salsa20, berhasil membuat metode untuk menemukan kunci enkripsi melalui analisis algoritma Salsa20 dan *memory forensics*. Hasilnya, 95% kunci enkripsi Salsa20 berhasil ditemukan dan selanjutnya digunakan untuk dekripsi [17]. Peneliti lainnya, Lee et al. dalam penelitian terhadap lima jenis *ransomware*, menyatakan bahwa ada kemungkinan untuk melakukan dekripsi melalui analisis algoritma enkripsi, *memory forensics*, dan adanya kesalahan pada manajemen kunci enkripsi [18]. Pada tahun yang berbeda, Lee et al. dalam penelitian terhadap *ransomware* Magniber v2, berhasil mendapatkan kunci enkripsi dengan melakukan analisis pada algoritma enkripsi dan berhasil menemukan kelemahan pada Pseudo Random Number Generator (PRNG) yang digunakan Magniber v2 [19]. Yustre et al. berhasil melakukan dekripsi data yang terenkripsi oleh *ransomware* Avaddon. Kunci enkripsi ditemukan melalui analisis algoritma enkripsi dan *memory forensics*, alat dekripsi juga dibuat dan disediakan *open source* untuk melakukan dekripsi [20]. Kim et al. melakukan analisis pada *ransomware* Hive dan menemukan kelemahan pada algoritma enkripsi yang dibuat sendiri oleh *threat actor*. Hasilnya, 95% data yang terenkripsi oleh *ransomware* Hive dapat didekripsi melalui pembuatan ulang kunci enkripsi [21]. Pada tahun yang berbeda, Kim et al. berhasil melakukan dekripsi pada data yang terenkripsi oleh *ransomware* Rhysida melalui analisis algoritma enkripsi, kunci dekripsi berhasil ditemukan dengan pembuatan ulang PRNG [22].

Pada penelitian ini, akan dilakukan analisis terhadap *malware* yang menggunakan algoritma enkripsi RC4. Analisis algoritma enkripsi dilakukan melalui metode *reverse engineering*, hasil analisis diimplementasikan melalui emulasi menggunakan Unicorn dengan tujuan untuk melakukan dekripsi. Penelitian ini dan penelitian sebelumnya sama-sama melakukan analisis algoritma enkripsi menggunakan metode *reverse engineering* dan memiliki tujuan yang sama, yaitu untuk melakukan dekripsi. Namun terdapat perbedaan pada jenis malware dan algoritma enkripsi yang diteliti serta metode yang digunakan untuk dekripsi.

3. METODE PENELITIAN

Penelitian ini bertujuan untuk dekripsi data yang terenkripsi oleh *malware* dengan menggunakan metode *reverse engineering* dan emulasi dengan Unicorn. Dalam penelitian ini, peneliti berusaha menjelaskan (*explanatory research*) bagaimana metode *reverse engineering* dan emulasi dengan Unicorn dapat digunakan untuk mendekripsi data yang terenkripsi oleh *malware*. Selain itu, penelitian ini juga mengeksplorasi (*exploratory research*) potensi Unicorn untuk dekripsi algoritma RC4 pada *malware*.

2.5. Perangkat keras dan perangkat lunak

Berikut merupakan spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam melakukan analisis dan dekripsi *malware*:
a. Laptop HP dengan spesifikasi: Windows 11 OS, Intel(R) Celeron(R) 6305 @ 1.80GHz, RAM 12 GB, SSD 512 GB.
b. Virtual machine: VirtualBox 7.1.4
c. Alat analisis statis, decompiler, dan disassembler: Ghidra, IDAPro
d. Emulator: Unicorn 2
e. Python3

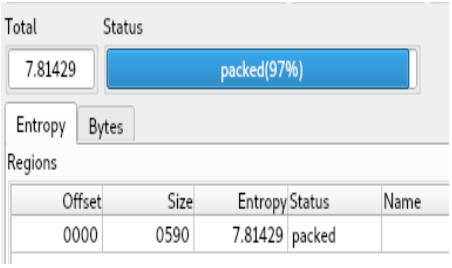
2.6. Alur penelitian

Berikut merupakan alur penelitian:

Gambar 1. Alur penelitian

Diagram alir di atas menggambarkan tahapan metodologi yang digunakan dalam penelitian. Setiap tahapan dalam diagram memiliki fungsi penting untuk memastikan proses penelitian berjalan sistematis dan terarah. Berikut penjelasan detail tiap langkahnya:

a. Mulai: tahapan ini menandai awal dimulainya proses penelitian.
b. Identifikasi masalah: dalam tahapan identifikasi masalah pada penelitian ini, fokus utama adalah untuk memahami kompleksitas dan tantangan yang terkait dengan dekripsi data yang terenkripsi oleh malware yang menggunakan algoritma RC4.



Gambar 2. Entropi data

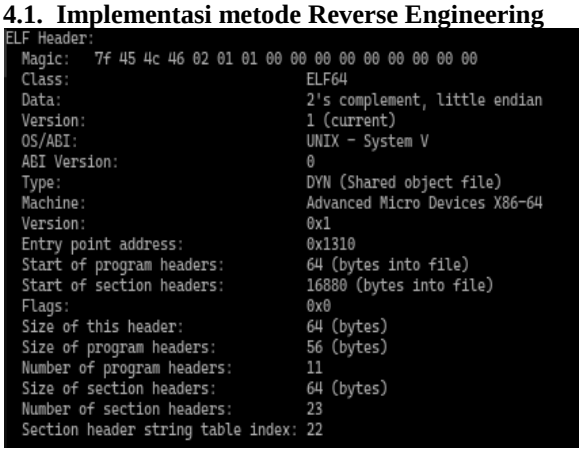
c. Data yang terenkripsi memiliki entropi di skala 7.8. Entropi dalam konteks komputasi dan keamanan siber adalah ukuran kerandoman atau ketidakpastian dalam data. Semakin tinggi entropi, semakin acak dan tidak terprediksi datanya. Masalah ini melibatkan sejumlah aspek, termasuk kompleksitas algoritma RC4, serta keterbatasan dalam penggunaan alat dan teknik konvensional untuk analisis dan dekripsi. Identifikasi masalah ini diperlukan untuk merumuskan pendekatan yang tepat dalam penelitian, memastikan bahwa metodologi yang digunakan dapat mengatasi tantangan yang dihadapi selama proses dekripsi. Berdasarkan hal tersebut entropi yang tinggi merupakan masalah utama dalam penelitian ini.
d. Studi literatur: dalam penelitian ini dimulai dengan identifikasi dan penelusuran terhadap sumber-sumber yang relevan tentang teknik *reverse engineering* emulasi *byte code*, dan dekripsi algoritma RC4 pada *malware*. Tahapan ini mencakup analisis mendalam terhadap penelitian terdahulu yang telah dilakukan dalam bidang tersebut, termasuk metodologi, temuan, dan tantangan yang dihadapi.
e. Implementasi metode *reverse engineering*: setelah memahami teori yang mendasari, peneliti mulai melakukan *reverse engineering* terhadap sampel *malware*. *Reverse engineering* dilakukan dengan menganalisis struktur biner *malware* untuk menemukan bagian kode yang bertanggung jawab terhadap proses enkripsi. Tahapan ini sangat krusial karena memberikan pemahaman teknis terkait cara kerja *malware* dalam mengenkripsi data.
f. Implementasi emulasi dengan Unicorn: berdasarkan hasil *reverse engineering*, peneliti mengimplementasikan hasil analisis menggunakan emulator Unicorn. Unicorn memungkinkan peneliti menjalankan bagian kode *malware* dalam lingkungan terkendali tanpa risiko infeksi terhadap sistem utama. Emulasi ini membantu untuk memvalidasi hasil analisis serta mengamati perilaku aktual malware saat mengenkripsi data.
g. Pengembangan alat dekripsi: setelah berhasil memahami proses enkripsi dan struktur algoritma yang digunakan oleh *malware*, peneliti mengembangkan alat bantu (skrip/program) yang dapat digunakan untuk mendekripsi data yang telah dienkripsi oleh malware. Alat ini dirancang berdasarkan logika algoritma enkripsi yang diperoleh melalui *reverse engineering* dan pengujian melalui emulasi.
h. Analisis dan interpretasi hasil: pada tahap ini, alat dekripsi diuji terhadap data yang telah dienkripsi oleh *malware*. Peneliti melakukan evaluasi terhadap keberhasilan dekripsi dan efektivitas metode yang digunakan. Hasil dekripsi dianalisis dan diinterpretasikan untuk menarik kesimpulan apakah metode yang dikembangkan mampu digunakan secara praktis dalam skenario keamanan informasi.
i. Selesai: tahap akhir dari proses penelitian. Peneliti menyusun laporan akhir yang berisi rangkuman

hasil temuan, kesimpulan, serta rekomendasi untuk penelitian selanjutnya.

4. HASIL DAN PEMBAHASAN

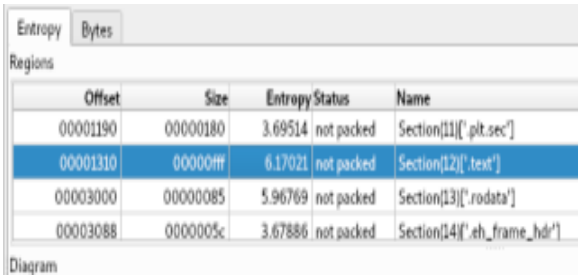
Berikut merupakan hasil dan pembahasan dari penelitian ini.

4.1. Implementasi metode Reverse Engineering



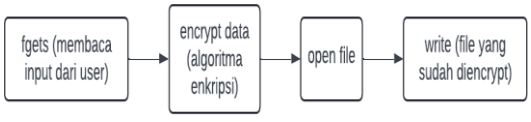
Gambar 3. Analisis statis

Dalam penelitian ini, tahapan implementasi metode reverse engineering dimulai dengan melakukan analisis statis terhadap malware.



Gambar 4. Entropi algoritma enkripsi pada malware

Analisis ini bertujuan untuk memahami struktur malware, termasuk pengidentifikasian lokasi algoritma enkripsi RC4. Lokasi algoritma enkripsi dapat diketahui dengan melihat tingkat entropi yang tinggi pada *malware*. Untuk mengetahui bagian dengan entropi yang tinggi diperlukan *tool* Detect it easy (DIE), diketahui lokasi dengan entropi tinggi berada di *addrerss* 0x00001310 dengan *size* 0x00000fff.



Gambar 5. Proses malware untuk encrypt data

Berdasarkan hasil reverse engineering proses enkripsi yang dilakukan oleh malware adalah seperti gambar diatas. Setelah itu, dilakukan proses disassemble dan decompile menggunakan IDAPro untuk mempermudah pemahaman algoritma enkripsi. Langkah berikutnya adalah menentukan komponen RC4 seperti *byte code* KSA dan PRGA yang akan digunakan pada proses emulasi. Proses ini bertujuan untuk menganalisis perilaku *malware* dan ekstraksi informasi terkait enkripsi. Berikut merupakan *pseudocede* hasil decompile:

a. *Pseudocode* fungsi fgets() yaitu fungsi yang digunakan untuk menerima *input* dari user (seperti *input* dari *keyboard*):
   - global_fgets_pointer (qword_5110): *pointer* ke fungsi fgets, disimpan setelah *resolved* dengan dlsym.
   - global_flag (dword_15A28): semacam *flag* kontrol, jika aktif maka blok penyimpanan *buffer* dilewati.
   - buffer_start (unk_5940) dan buffer_end (unk_10000): batas awal dan akhir *buffer* untuk menyimpan *input* dari terminal.
   - buffer_index (dword_15940): posisi saat ini dalam *buffer*.
   - safe_memcpy (__memcpy_chk): digunakan untuk memastikan penyalinan aman (*buffer overflow check*).

```
01  function fgets(src, size, file):
02      if global_fgets_pointer is not initialized:
03          global_fgets_pointer = dlsym(RTLD_DEFAULT,
"fgets")
04          if dlerror() is not null:
05              return null
06
07      result = global_fgets_pointer(src, size, file)
08      if result is null:
09          return null
10
11      if global_flag == 0:
12          fd = fileno(file)
13          if isatty(fd):  // input dari terminal
14              length = strlen(src)
15              remaining_space = buffer_end - buffer_index
16              dest_buffer = buffer_start + buffer_index
17
18              if length <= remaining_space:
19                  memcpy(dest_buffer, src, length)
20                  buffer_index += length
21              else:
22                  copy_part = remaining_space
23                  memcpy(dest_buffer, src, copy_part)
24                  openwrite(1, buffer_start, buffer_end)
25
26                  remaining_data = src + copy_part
27                  remaining_length = length - copy_part
28
29                  if remaining_length <= buffer_max_size:
30                      safe_memcpy(buffer_start, remaining_data,
remaining_length)
31                      buffer_index = remaining_length
32                  else:
33                                  openwrite(1,  remaining_data,
remaining_length)
34                      buffer_index = 0
35
36      return result
```

b. *Pseudocode* fungsi createkey() untuk membuat *keystream*:
- Fungsi ini mengumpulkan informasi sistem dan *user*, lalu menyusun data tersebut ke dalam *buffer* (v48).
- Dua bagian data (dari uname dan getpwuid) disisipkan setelah di-obfuscate dengan operasi XOR.
- Kemudian, jika ada argumen tambahan, semua disisipkan ke buffer juga.
- *Buffer* digunakan untuk membuat semacam "kunci terenkripsi", melalui proses mirip KSA (Key Scheduling Algorithm seperti di RC4).
- Di akhir, data dan hasil pengacakan dikirim ke fungsi *openwrite*.

```
function createkey(arg_count, arg_values):
    wait until stack guard is set (dummy loop)
    byte_5928 = 2
    uid = getuid()
    pid = getpid()
    v48[0] = 1
    buffer_index = 1
    key_seed = byteswap((pid << 16) | uid)

    if uname(system_info) succeeded:
        obfuscate bytes with xor loop #1
        append formatted system_info to buffer v48
        clear temp buffer

    passwd_info = getpwuid(uid)
    if passwd_info exists:
        obfuscate bytes with xor loop #2
        append formatted passwd_info to buffer v48
        clear temp buffer

    append "[c]" to buffer v48

    if arg_count > 0:
        for each string in arg_values:
            append " <arg>" to buffer v48

    if buffer length <= 0xFFFF:
        null-terminate buffer
    else:
        v51 = 0

        initialize key_schedule array (dword_5528) with descending bytes

    for several rounds:
        perform complex key mixing using state arrays and obfuscated values

            return openwrite(0, v48, total_length, index, key_schedule, 0)
```

c. Pseudocode fungsi openwrite() menyimpan *file* yang sudah *diencypt*:
- Fungsi ini memeriksa apakah sudah ada kesalahan (via dword_15A28). Jika ada, langsung keluar.
- *Buffer (buf)* akan dialokasikan atau diubah ukurannya sesuai dengan parameter a3.
- Proses selanjutnya mengamankan dan mengenkripsi data (terutama jika ukuran data lebih besar dari 8 *byte*).
- Fungsi juga membuka *file* dengan kontrol terhadap hak akses dan ukuran *file*.
- Penguncian *file* dilakukan, data ditulis, dan kemudian *file* ditutup dengan pengecekan kesalahan.

```
function openwrite(a1, a2, a3):
    if dword_15A28 is non-zero:
        return
    byte_5929 = a1
    dword_592E = byteswap(a3)
    if buf is not null:
        if a3 > size:
            resize buf to a3
            if buf is null:
                goto LABEL_18
        else if a3 is zero:
            goto LABEL_11
        else:
            continue normal processing
    else:
        allocate buf with initial size
    if buf is null:
        goto LABEL_18
    if a3 is non-zero:
                if (a3 & 0x80000000) == 0 and (a3 & 0xFFFFFFF8) != 0:
            process data in blocks of 8 bytes
        if (a3 & 7) != 0:
            encrypt remaining data
LABEL_11:
     obfuscate bytes using XOR with some pattern (loop over 24 steps)
    if file exists and has correct permissions:
        if file size is too large:
            goto LABEL_18
        open file in write mode
        if file opening failed:
            goto LABEL_18
    lock the file
    if writing 10 bytes header and data fails:
        mark error (dword_15A28 = 1)
    unlock the file and close it
          if any error during file handling, mark error (dword_15A28 = 1)
LABEL_18:
    mark error (dword_15A28 = 1)
    return
```

d. *Pseudocode* fungsi encrypt() untuk *mengencrypt inputan* dari *user*:
- Fungsi ini melakukan enkripsi XOR berdasarkan panjang data yang ditentukan oleh a1 dengan menggunakan *buffer* dword_5528.
- Proses enkripsi dilakukan dengan mengacak dan menukar nilai pada dword_5528, kemudian meng-enkripsi data byte demi byte.
- Hasil enkripsi disalin ke dalam a3.
- Fungsi ini mengelola pemrosesan enkripsi berdasarkan ukuran data (a1), mulai dari 1 hingga 8 *byte*.

- Fungsi mengupdate nilai qword_5520 yang berfungsi untuk melacak keadaan enkripsi.

```
function encrypt(a1, a2, a3):
    v3 = qword_5520
    result = HIDWORD(qword_5520)

    if a1 >> 3:
            // swap values from dword_5528 and perform XOR encryption for 8 bytes at a time
        // repeat the process for 8 bytes (a3[0] to a3[7])
    else:
        if a1 is non-zero:
            // swap values from dword_5528 for each byte (a3[0] to a3[6])
            // perform XOR encryption for each byte of a2 with dword_5528
            // adjust result based on a1 value to control the number of bytes encrypted (a1: 0-6)

    update qword_5520 with new values of v3 and result
    return result
```

Dari proses dan *code* diatas yang diubah adalah *input* dari *user*, *input* yang digunakan adalah *input chipertext* (data yang sudah dienkripsi), hal ini dikarenakan algoritma enkripsinya termasuk dalam algoritma enkripsi simetrik, sehingga algoritma yang sama dapat digunakan untuk enkripsi dan dekripsi. *Malware* yang diteliti tidak dapat melakukan proses dekripsi, sehingga perlu dilakukan manual oleh peneliti menggunakan cara emulasi.

4.2. Implementasi emulasi dengan Unicorn

Pada tahap ini akan dianalisa API Unicorn yang diperlukan untuk melakukan emulasi. Informasi dari tahap *reverse engineering* digunakan untuk mengidentifikasi API yang akan digunakan. Beberapa API utama yang akan digunakan meliputi uc_open untuk membuka konteks emulasi, uc_mem_map untuk memetakan memori, uc_reg_read dan uc_reg_write untuk mengakses dan memodifikasi register CPU, uc_mem_read dan uc_mem_write untuk membaca dan menulis data dari dan ke memori yang telah dipetakan, serta uc_emu_start dan uc_emu_stop untuk memulai dan menghentikan eksekusi emulasi. Implementasi emulasi dimulai dengan membuka konteks emulasi yang sesuai dengan arsitektur CPU yang dianalisis, seperti x86_64 untuk arsitektur 64-bit. Selanjutnya, memori yang diperlukan untuk menjalankan sampel *malware*, termasuk kode, data, dan stack, dipetakan menggunakan uc_mem_map. Kode biner dari *malware* kemudian dimuat ke dalam memori yang telah dipetakan.

4.3. Pengembangan alat dekripsi

Data yang telah dikumpulkan dari proses sebelumnya akan digunakan untuk mengembangkan alat dekripsi. Alat dekripsi ini akan dibuat menggunakan emulator Unicorn sebagai alat emulasi dan bahasa pemrograman Python untuk mengimplementasikan logika dekripsi. Logika dekripsi akan mengikuti tahapan algoritma RC4. Proses dekripsi dengan Unicorn akan melibatkan beberapa tahapan sebagai berikut:

a. Memuat *byte code* algoritma RC4: memuat *byte code* atau kode biner yang mewakili algoritma RC4 ke dalam *memory* yang telah dialokasikan.
b. Memuat *input (chipertext)*: memuat teks yang dienkripsi (*chipertext*) ke dalam *memory* yang telah dialokasikan.
c. Menginisialisasi *register* CPU: mengatur *register* CPU yang diperlukan untuk menjalankan *byte code* algoritma RC4.
d. Menjalankan emulasi untuk melakukan dekripsi: memulai emulasi untuk menjalankan *byte code* algoritma RC4, yang akan mengambil *chipertext* sebagai *input* dan menghasilkan *plaintext* sebagai output. Proses emulasi akan berlangsung hingga dekripsi selesai dilakukan.

Berikut merupakan *code* python yang digunakan untuk emulasi:

```
def stage_1(pid, length):
    # stage 1 (bytecode fungsi createkey())
    stage_1_sc = bytes.fromhex(
        "b8 02 00 00 00 48 8d 15 02 38 00 ...")
    mu = Uc(UC_ARCH_X86, UC_MODE_64)
    address = 0x1000000
    mu.mem_map(address, 2 * 1024 * 1024)
    mu.mem_write(address, stage_1_sc)
            mu.reg_write(UC_X86_REG_RSP, address + 0x100000)
    mu.mem_write(0x1003c10, pid)
    mu.reg_write(UC_X86_REG_RDI, length)
    signal.signal(signal.SIGINT, interrupt_handler)
    try:
        mu.emu_start(address, address + len(stage_1_sc))
    except UcError:
        pass
    keystream = mu.mem_read(0x100380e, 0x400)
    return keystream

def stage_2(length, data, keystream):
    stage_2_sc = bytes.fromhex(
        "bf 08 00 00 00 e8 0e 00 00 00 48 ...")
    mu = Uc(UC_ARCH_X86, UC_MODE_64)
    address = 0x1000000
    mu.mem_map(address, 2 * 1024 * 1024)
    mu.mem_write(address, stage_2_sc)
            mu.reg_write(UC_X86_REG_RSP, address + 0x100000)
    mu.mem_write(0x1004210, bytes(keystream))
    rsi_addr = 0x1005000
    mu.mem_write(rsi_addr, data)
    mu.reg_write(UC_X86_REG_RSI, rsi_addr)
    r12_addr = 0x1005000 + length
    mu.reg_write(UC_X86_REG_R12, r12_addr)
    rdx_addr = 0x1020000
    mu.reg_write(UC_X86_REG_RDX, rdx_addr)
    signal.signal(signal.SIGINT, interrupt_handler)
    try:
        mu.emu_start(address, address + len(stage_2_sc))
    except UcError:
```

```
40          pass
41      rdx_data = mu.mem_read(rdx_addr, length)
42      print(bytearray(rdx_data).decode("utf8"))
43
44  def read_file(file_path):
45      with open(file_path, 'rb') as file:
46          for offset in data_offset:
47              pid = offset + 2
48              file.seek(pid)
49              pid = file.read(4)
50              pid_bwap = int.from_bytes(pid, byteorder='big')
51              length = offset + 7
52              file.seek(length)
53                          length = int.from_bytes(file.read(3),
byteorder='big')
54              data = offset + 10
55              file.seek(data)
56              data = file.read(length)
57              keystream = stage_1(pid, length)
58              stage_2(length, data, keystream)
```

4.4. Analisis dan interpretasi hasil

Proses ini dimulai dengan pengumpulan data dari eksperimen yang dilakukan, termasuk hasil dari alat dekripsi. Selanjutnya, hasil kuantitatif dianalisis untuk menentukan tingkat keberhasilan alat dekripsi yang digunakan. Kemudian, data kualitatif yang diperoleh selama proses dekripsi, seperti tahapan-tahapan yang diikuti.

a. Tahapan proses dekripsi

Berikut merupakan tahapan dan data yang digunakan dalam proses dekripsi:

- Pemasangan *malware*: LD_PRELOAD adalah variabel lingkungan (environment variable) yang digunakan oleh sistem operasi Linux (dan Unix-like) untuk memuat sebuah pustaka bersama (*shared library*) sebelum pustaka-pustaka lainnya ketika menjalankan suatu program. Perintah yang digunakan adalah LD_PRELOAD=/tmp/malw.elf.
- *Input* data untuk dienkripsi:

```
root@remnux:/tmp# mkdir newDir
root@remnux:/tmp# cp /etc/passwd newDir/
root@remnux:/tmp# touch secret.txt
root@remnux:/tmp# echo passw > secret.txt
root@remnux:/tmp# echo passwd2 >> secret.txt
root@remnux:/tmp# echo passwd23 >> secret.txt
root@remnux:/tmp# ping 192.168.56.120
PING 192.168.56.120 (192.168.56.120) 56(84) bytes of data.
64 bytes from 192.168.56.120: icmp_seq=1 ttl=128 time=2.10 ms
64 bytes from 192.168.56.120: icmp_seq=2 ttl=128 time=2.95 ms
^C
--- 192.168.56.120 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max/mdev = 2.101/2.527/2.954/0.426 ms
root@remnux:/tmp# ssh root@192.168.56.120
The authenticity of host '192.168.56.120 (192.168.56.120)' can't be established.
ECDSA key fingerprint is SHA256:RRJQhhNEyAOQsvVdnhDBwHnDYByq3ByUyGAnBBI82dg.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.56.120' (ECDSA) to the list of known hosts.
root@192.168.56.120's password:
Permission denied, please try again.
root@192.168.56.120's password:

root@remnux:/tmp# ssh 'flare\bravo@192.168.56.120'
```

Gambar 6. *Input* data untuk dienkripsi

Data yang digunakan adalah *input* dari keyboard.

- Data yang sudah dienkripsi:

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0:0000   02 00 06 90 00 00 00 00 00 85 F7 51 84 74 3D 88  ..........÷Q„t=ˆ
0:0010   EF 27 AE EB B7 3C 9E D4 AC 50 77 96 B1 EA D3 E9  ï'®ë·<žÔ¬Pw–±êÓé
0:0020   DE E7 2A 16 3C A1 99 A8 FF 79 15 37 DA F9 35 8C  Þç*.<¡™¨ÿy.7Úù5Œ
0:0030   E1 B4 3C 1A AD A8 79 68 71 BD 26 F5 07 61 73 DA  á´<.¨yhq½&õ.asÚ
0:0040   90 55 FD 65 02 04 09 CD 24 20 A1 0A B4 2C 71 8C  .Uýe...Í$ ¡.´,qŒ
0:0050   23 71 98 C3 B1 BA 2C D5 AF 4A 1C C8 E7 5D 43 A3  #q˜Ã±º,Õ¯J.Èç]C£
0:0060   44 22 BA 44 F3 81 40 10 4A C4 4E 02 A3 F1 B9 A9  D"ºDó.@.JÄN.£ñ'©
0:0070   10 39 DD CE 6E 50 FD 90 D2 E6 93 CB AD 7A 2C 9D  .9ÝÎnPý.Òæ"Ë-z,.
0:0080   23 7E A3 8A BF 69 6D 57 38 F2 4E 4B 1F 25 77 02  #~£Š¿imW8òNK.%w.
0:0090   00 06 91 00 00 00 00 00 87 8E 6A DE FB EA 37 87  ..'.....‡ŽjÞûê7‡
0:00A0   C4 C0 3D 10 E8 62 E2 14 97 22 EB D9 25 2E C5 A0  ÄÀ=.èbâ.—"ëÙ%.Å
0:00B0   87 3D 03 57 52 1A 98 1C F2 C8 8E 1C 22 A2 D8 8C  ‡=.WR.˜.òÈŽ."¢Ø
0:00C0   E2 76 1C 90 A1 CF 74 91 E5 73 1D 11 31 67 3E B4  âv..¡Ït'ås..1g>´
0:00D0   53 BA 31 92 44 C7 BB 8E 6D 06 8F 71 71 07 02 35  S°1'DÇ»Žm..qq..5
0:00E0   E2 12 A9 BF 38 F8 64 13 8E E4 52 90 83 56 C8 CA  â.©¿8ød.ŽäR.ƒVÈÊ
0:00F0   8B F6 9B 18 9D 2A 33 D7 FE 8F 22 C6 69 41 F1 8C  ‹ö›..*3×þ."ÆiAñŒ
0:0100   5B 58 9F BC 3B 85 75 27 D6 EB 84 28 25 BE 0B 6E  [XŸ¼;…u'Öë„(%¾.n
0:0110   42 5B C0 3A 2E 96 34 57 FA DB 9F C2 4F E4 93 81  B[À:.–4WúÛŸÂOä".
0:0120   02 00 06 92 00 00 00 00 00 9A 7A D4 D8 35 67 C3  ...'.....šzÔØ5gÃ
```

Gambar 7. Data yang sudah dienkripsi (*chipertext*)

Gambar diatas merupakan tampilan hex salah satu data yang sudah dienkripsi.

Tabel 1. Data yang sudah dienkripsi

Nama data	Jumlah *byte*	Entropi
data1	69079 b	8.0
data2	65710 b	8.0
data3	65886 b	8.0
data4	67000 b	8.0
data5	2358 b	7.83
data6	8950 b	7.89

Tabel diatas merupakan data yang digunakan pada penelitian ini.

b. Hasil dekripsi

Berikut merupakan tampilan hex dari data yang sudah didekripsi.

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000     01 5B 6E 5D 20 4C 69 6E 75 78 3A 52 48 45 4C 37  .[n] Linux:RHEL7
0010     37 3A 33 2E 31 30 2E 30 2D 31 30 36 32 2E 65 6C  7:3.10.0-1062.el
0020     37 2E 78 38 36 5F 36 34 3A 23 31 20 53 4D 50 20  7.x86_64:#1 SMP
0030     54 68 75 20 4A 75 6C 20 31 38 20 32 30 3A 32 35  Thu Jul 18 20:25
0040     3A 31 33 20 55 54 43 20 32 30 31 39 3A 78 38 36  :13 UTC 2019:x86
0050     5F 36 34 00 5B 69 5D 20 72 6F 6F 74 3A 78 3A 30  _64.[i] root:x:0
0060     3A 30 3A 72 6F 6F 74 3A 2F 72 6F 6F 74 3A 2F 62  :0:root:/root:/b
0070     69 6E 2F 62 61 73 68 00 5B 63 5D 20 2F 62 69 6E  in/bash.[c] /bin
0080     2F 62 61 73 68 20 2D 63 20 65 78 65 63 20 2F 72  /bash -c exec /r
0090     6F 6F 74 2F 64 65 62 75 67 2F 6C 6F 61 64 5F 66  oot/debug/load_f
00A0     69 70 73 20 00 0A 01 5B 6E 5D 20 4C 69 6E 75 78  ips ...[n] Linux
00B0     3A 52 48 45 4C 37 37 3A 33 2E 31 30 2E 30 2D 31  :RHEL77:3.10.0-1
00C0     30 36 32 2E 65 6C 37 2E 78 38 36 5F 36 34 3A 23  062.el7.x86_64:#
00D0     31 20 53 4D 50 20 54 68 75 20 4A 75 6C 20 31 38  1 SMP Thu Jul 18
00E0     20 32 30 3A 32 35 3A 31 33 20 55 54 43 20 32 30   20:25:13 UTC 20
00F0     31 39 3A 78 38 36 5F 36 34 00 5B 69 5D 20 72 6F  19:x86_64.[i] ro
0100     6F 74 3A 78 3A 30 3A 30 3A 72 6F 6F 74 3A 2F 72  ot:x:0:0:root:/r
0110     6F 6F 74 3A 2F 62 69 6E 2F 62 61 73 68 00 5B 63  oot:/bin/bash.[c
```

Gambar 8. Data yang sudah didekripsi (*plaintext*)

Gambar diatas merupakan tampilan hex salah satu data yang sudah didekripsi.

Tabel 2. Data yang sudah didekripsi

Nama data	Jumlah *byte*	Entropi	Hasil dekripsi
data1	69079 b	4.7	94.42%
data2	65710 b	4.7	94.26%
data3	65886 b	4.7	94.12%
data4	67000 b	4.7	94.82%
data5	2358 b	3.2	94.27%
data6	8950 b	3.2	94.45%

Tabel diatas merupakan data yang sudah didekripsi.

5. KESIMPULAN DAN SARAN

Penelitian ini bertujuan untuk menjawab pertanyaan utama: Apakah emulasi menggunakan Unicorn dapat digunakan untuk mendekripsi algoritma RC4 yang digunakan dalam *malware*? Berdasarkan hasil eksperimen dan analisis yang dilakukan, tujuan tersebut berhasil dicapai. Emulasi menggunakan Unicorn Engine, yaitu *framework* CPU emulator berbasis QEMU, terbukti mampu mengeksekusi instruksi mesin secara dinamis sehingga memungkinkan pengamatan terhadap proses dekripsi internal pada sampel *malware* yang dianalisis.

Dalam pengujian, data terenkripsi sebesar 278.983 *byte* berhasil didekripsi dengan tingkat keberhasilan rata-rata sebesar 94,46%, menunjukkan bahwa sebagian besar data dapat dipulihkan secara akurat. Dengan demikian, dapat disimpulkan bahwa pendekatan ini efektif dan layak digunakan dalam konteks analisis *malware* berbasis emulasi.

Penelitian ini membuka peluang untuk pengembangan lebih lanjut dalam bidang analisis *malware*, khususnya dalam penggunaan teknik emulasi untuk membongkar algoritma enkripsi. Peneliti selanjutnya disarankan untuk menerapkan pendekatan yang sama, yaitu menggunakan emulasi berbasis Unicorn, untuk mendekripsi algoritma enkripsi lain yang lebih kompleks dan modern seperti AES (Advanced Encryption Standard), DES (Data Encryption Standard), atau ChaCha20. Dengan pengembangan lebih lanjut, pendekatan berbasis emulasi ini berpotensi menjadi alat penting dalam *reverse engineering* dan forensik digital untuk melawan *malware* yang semakin canggih dan mengandalkan teknik enkripsi untuk menyembunyikan aktivitas berbahaya.

DAFTAR PUSTAKA

[1] Kaspersky, "Kaspersky Security Bulletin 2022," Nov. 2022. [Online]. Available: https://go.kaspersky.com/rs/802-JN-240/images/KSB_statistics_2022_en_final.pdf. [Accessed: Apr. 2024].

[2] NTT, "2022–global-threat-intelligence-report-v8.pdf," Jun. 2022. [Online]. Available: https://www.security.ntt/pdf/2022-global-threat-intelligence-report-v8.pdf. [Accessed: Apr. 2024].

[3] A. Stojakowski, "The Impact Of Malware On The Internet," Jul. 2023. [Online]. Available: https://www.researchgate.net/publication/372458600_THE_IMPACT_OF_MALWARE_ON_THE_INTERNET. [Accessed: Apr. 2024].

[4] FBI, CISA, "Iranian State Actors Conduct Cyber Operations," Sep. 2022. [Online]. Available: https://www.ic3.gov/media/news/2022/220921.pdf. [Accessed: Apr. 2024].

[5] HP WOLF SECURITY, "Threat Insights Report," Aug. 2022. [Online]. Available: https://threatresearch.ext.hp.com/wp-content/uploads/2022/08/HP-Wolf-Security-Threat-Insights-Report-Q2-2022.pdf. [Accessed: Apr. 2024].

[6] Kaspersky, "Common TTPs of attacks against industrial organizations. Implants for remote access," Jul. 2023. [Online]. Available: https://ics-cert.kaspersky.com/media/Kaspersky-ICS-CERT-Common-TTPs-of-attacks-against-industrial-organizations-implants-for-remote-access-En.pdf. [Accessed: Apr. 2024].

[7] Q. M. Kenan Begovic, "Cryptographic ransomware encryption detection: Survey," Computers & Security, no. 2023, p. 5, 2023.

[8] C. Madarro-Capó, "Information Theory Based Evaluation of the RC4 Stream Cipher Outputs," Entropy, p. 1, 2021.

[9] A. Suhad M. Kareem, "A Modification on Key Stream Generator for RC4 Algorithm," Engineering and Technology Journal, 2020.

[10] P. Bentley, "A Taxonomy of Encryption and Encoding Algorithms Used by Advanced Persistent," Jun. 2023. [Online]. Available: https://eprints.glos.ac.uk/id/eprint/12874. [Accessed: Apr. 2024].

[11] CISA, "Malware Analysis Report," Jun. 2022. [Online]. Available: https://www.cisa.gov/sites/default/files/publications/MAR-10325820-01.r1.WHITE.pdf. [Accessed: Apr. 2024].

[12] unicorn-engine, "unicorn," GitHub, Nov. 2022. [Online]. Available: https://github.com/unicorn-engine/unicorn. [Accessed: Apr. 2024].

[13] Trellix, "Emulating Code with Unicorn," Trellix, Apr. 2022. [Online]. Available: https://www.trellix.com/assets/docs/atp-library/treat-emulating-code-with-unicorn.pdf. [Accessed: Apr. 2024].

[14] L. Babiano, "Evaluation of live forensic techniques, towards Salsa20-Based cryptographic ransomware mitigation," Forensic Science International: Digital Investigation, vol. 46, 2023.

[15] S. Lee, "A Study on Encryption Process and Decryption of Ransomware in 2019," Journal of the Korea Institute of Information Security & Cryptology, 2019.

[16] J. Lee, "Magniber v2 Ransomware Decryption: Exploiting the Vulnerability of a Self-Developed Pseudo Random Number Generator," Electronics, 2021.

[17] S. Yuste, "Avaddon ransomware: An in-depth analysis and decryption of infected systems," Cryptography and Security, 2021.

[18] Giyoon Kim, "A Method for Decrypting Data Infected with Hive Ransomware," Cryptography and Security, 2022.
[19] G. KCim, "A Method for Decrypting Data Encrypted with Rhysida Ransomware," Cryptography and Security, 2022.
[20] M. H. Hasan, "Image Encryption using Modified RC4 Algorithm," 2023 IEEE 3rd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA), 2023.
[21] S. A. Abdulameer, "A cryptosystem for database security based on RC4 algorithm," Journal of Al-Qadisiyah for Computer Science and Mathematics 15.1, p. 189, 2023.
[22] A. Amira, "A survey of malware analysis using community detection algorithms," ACM Computing Surveys 56.2, pp. 1–29, 2023