REPUBLIC OF ALBANIA
NATIONAL CYBER SECURITY AUTHORITY
DIRECTORATE OF CYBER SECURITY ANALYSIS

# Technical Analysis of a Malicious File

***RaLord-0xb***

**Version: 1.0**
**Date: 13/01/2026**

## TABLE OF CONTENTS

***This report has limitations and should be interpreted with caution!***

Identified Limitations:

**Phase One:**
*Information Sources:* This report is based on the information available at the time of its preparation. Certain aspects may differ from current developments.

**Phase Two:**
*Analysis Details:* Due to resource constraints, some aspects of the malicious file may not have been analyzed in depth. Any additional undiscovered information may lead to changes in the report.

**Phase Three:**
*Information Security:* To protect sources and confidential information, some details have been generalized or omitted. This decision was made to preserve data integrity and security.

**The National Cyber Security Authority (AKSK) reserves the right to amend, update, or modify any part of this report without prior notice.**

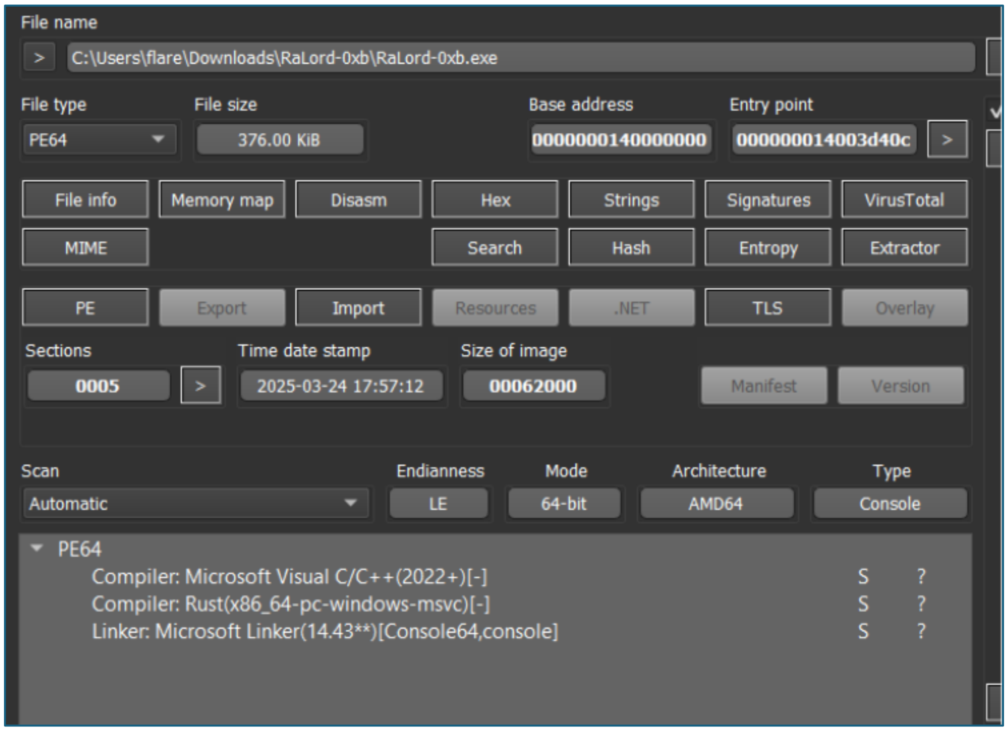
*This report is not a final document.*
*Its findings are based on information available during the investigation and analysis period. No guarantees are provided regarding potential changes or updates to the reported information. The authors bear no responsibility for misuse or consequences arising from decisions based on this report.*

## Technical Information

The circulation of a cyberattack campaign using ransomware attributed to the **NOVA Ransomware Group** has been identified, characterized by new techniques and tactics. This group first appeared in 2024, and its activity demonstrates a focused attack methodology. The group combines system encryption with the exfiltration of sensitive data, using threats of public exposure to increase and reinforce ransom demands.
From an operational perspective, Nova prioritizes efficiency over innovation, employing established techniques such as disabling backups, stopping security services, and abusing legitimate administrative tools to move laterally within compromised networks. Although still maintaining a low profile, Nova demonstrates how new ransomware actors can rapidly conduct impactful cyberattacks by exploiting known weaknesses in information infrastructure defenses.

### Analysis of the RaLord-0xb File

The **RaLord-0xb** file is a Windows executable developed using the **RUST programming language**.

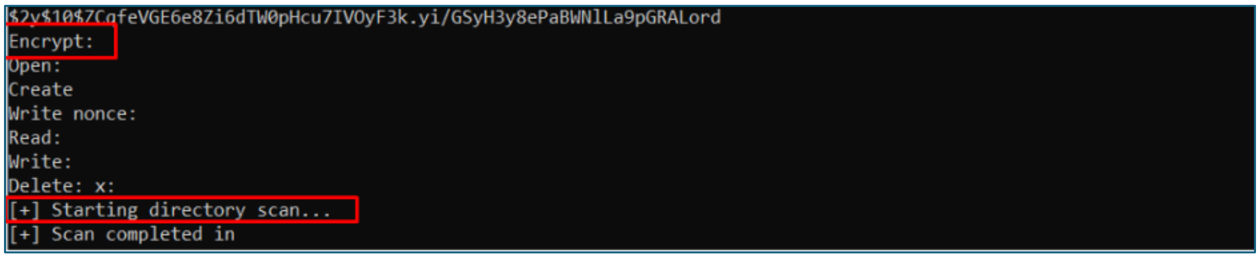


File name: C:\Users\flare\Downloads\RaLord-0xb\RaLord-0xb.exe
File type: PE64
File size: 376.00 KiB
Base address: 0000000140000000
Entry point: 000000014003d40c
Sections: 0005
Time date stamp: 2025-03-24 17:57:12
Size of image: 00062000
Scan: Automatic
Endianness: LE
Mode: 64-bit
Architecture: AMD64
Type: Console

```
PE64
    Compiler: Microsoft Visual C/C++(2022+)[-]              S   ?
    Compiler: Rust(x86_64-pc-windows-msvc)[-]               S   ?
    Linker: Microsoft Linker(14.43**)[Console64,console]    S   ?
```

*Figure 1. RUST programming language*

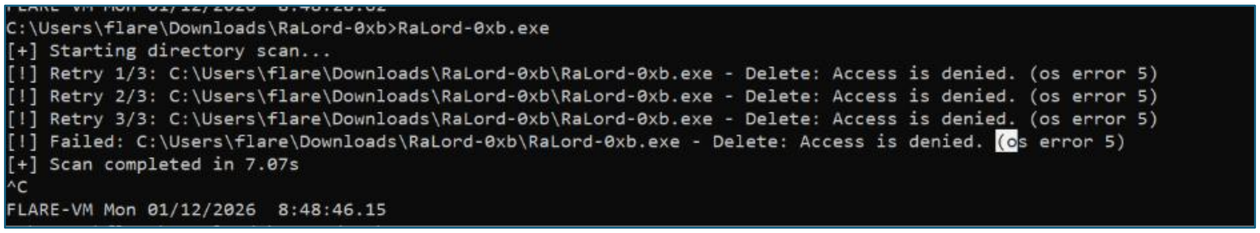
Analysis of the file strings reveals that the malware initiates directory scanning on the operating system and begins encrypting files based on the execution privileges granted to it.



```
$2y$10$7CqfeVGE6e8Zi6dTW0pHcu7IVOyF3k.yi/GSyH3y8ePaBWNlLa9pGRALord
Encrypt:
Open:
Create
Write nonce:
Read:
Write:
Delete: x:
[+] Starting directory scan...
[+] Scan completed in
```

*Figure 2. File strings*

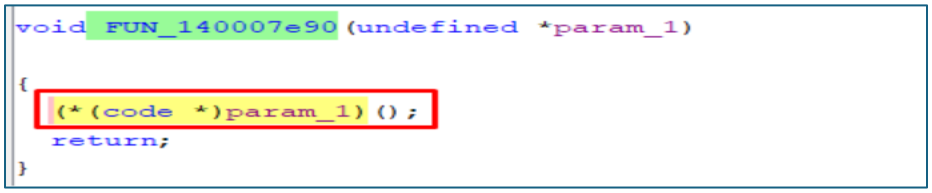
This behavior is also observed when executing the file via **CMD**, where directory enumeration is followed by the encryption process.



```
C:\Users\flare\Downloads\RaLord-0xb>RaLord-0xb.exe
[+] Starting directory scan...
[!] Retry 1/3: C:\Users\flare\Downloads\RaLord-0xb\RaLord-0xb.exe - Delete: Access is denied. (os error 5)
[!] Retry 2/3: C:\Users\flare\Downloads\RaLord-0xb\RaLord-0xb.exe - Delete: Access is denied. (os error 5)
[!] Retry 3/3: C:\Users\flare\Downloads\RaLord-0xb\RaLord-0xb.exe - Delete: Access is denied. (os error 5)
[!] Failed: C:\Users\flare\Downloads\RaLord-0xb\RaLord-0xb.exe - Delete: Access is denied. (os error 5)
[+] Scan completed in 7.07s
^C
FLARE-VM Mon 01/12/2026  8:48:46.15
```

*Figure 3. Execution via CMD*

During analysis, the function **FUN_140007e90** was identified as the main function responsible for invoking the malicious code. This is evident from the portion of the code where it expects the parameter param_1. Static analysis reveals that param_1 = &LAB_140006230, meaning the relevant code is located at the label **LAB_140006230.**



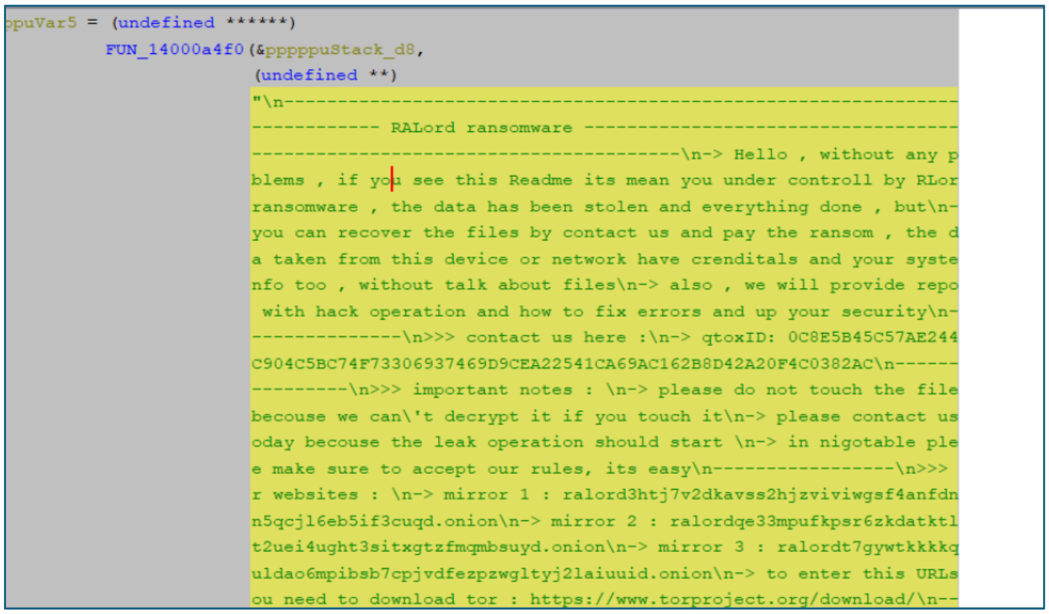
```
void FUN_140007e90(undefined *param_1)

{
  (*(code *)param_1)();
  return;
}
```

*Figure 4. Code invocation*

At this label, the malicious code becomes visible, including the hardcoded ransomware note embedded within the file. The same function also contains the previously identified strings:

"[+] Starting directory scan..."
"[+] Scan completed in ..."
"[!] Error creating README: ..."
The encryption process begins with the function **FUN_14000c8e0.**

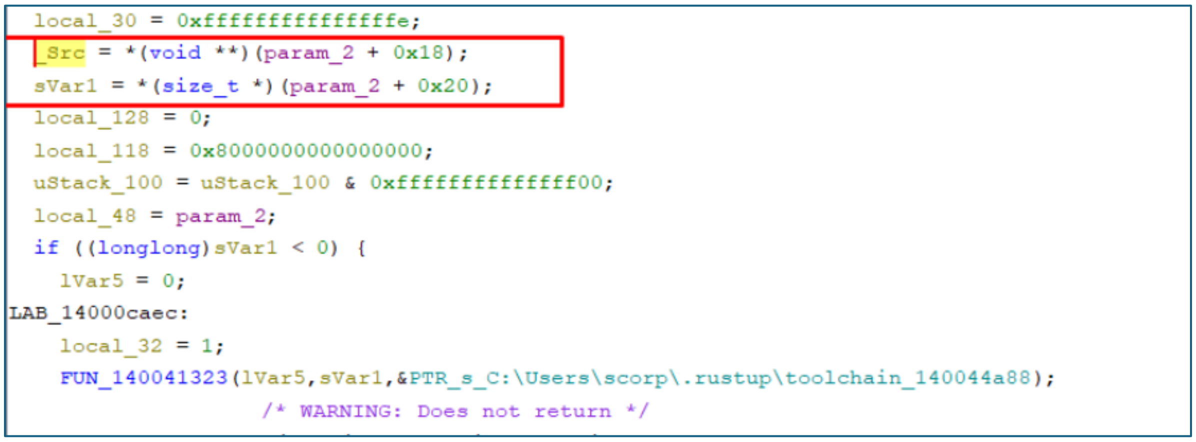


```
ppuVar5 = (undefined ******)
          FUN_14000a4f0(&pppppuStack_d8,
                        (undefined **)
                        "\n-------------------------------------------------------------------
                        ------------ RALord ransomware -----------------------------------
                        ----------------------------------------\n-> Hello , without any p
                        blems , if you see this Readme its mean you under controll by RLor
                        ransomware , the data has been stolen and everything done , but\n-
                        you can recover the files by contact us and pay the ransom , the d
                        a taken from this device or network have crenditals and your syste
                        nfo too , without talk about files\n-> also , we will provide repo
                         with hack operation and how to fix errors and up your security\n-
                        --------------\n>>> contact us here :\n-> qtoxID: 0C8E5B45C57AE244
                        C904C5BC74F73306937469D9CEA22541CA69AC162B8D42A20F4C0382AC\n------
                        ---------\n>>> important notes : \n-> please do not touch the file
                        becouse we can\'t decrypt it if you touch it\n-> please contact us
                        oday becouse the leak operation should start \n-> in nigotable ple
                        e make sure to accept our rules, its easy\n-----------------\n>>>
                        r websites : \n-> mirror 1 : ralord3htj7v2dkavss2hjzviviwgsf4anfdn
                        n5qcjl6eb5if3cuqd.onion\n-> mirror 2 : ralordqe33mpufkpsr6zkdatktl
                        t2uei4ught3sitxgtzfmqmbsuyd.onion\n-> mirror 3 : ralordt7gywtkkkkq
                        uldao6mpibsb7cpjvdfezpzwgltyj2laiuuid.onion\n-> to enter this URLs
                        ou need to download tor : https://www.torproject.org/download/\n--
```

*Figure 5. Ransomware note embedded in the file*

**FUN_14000c8e0** represents the starting point of actual file processing. However, the encryption algorithm itself is not implemented directly in this function. Instead, file data is copied into a private buffer, as shown below, followed by the preparation of metadata / nonce / IV (cryptographic header or auxiliary encryption structure).

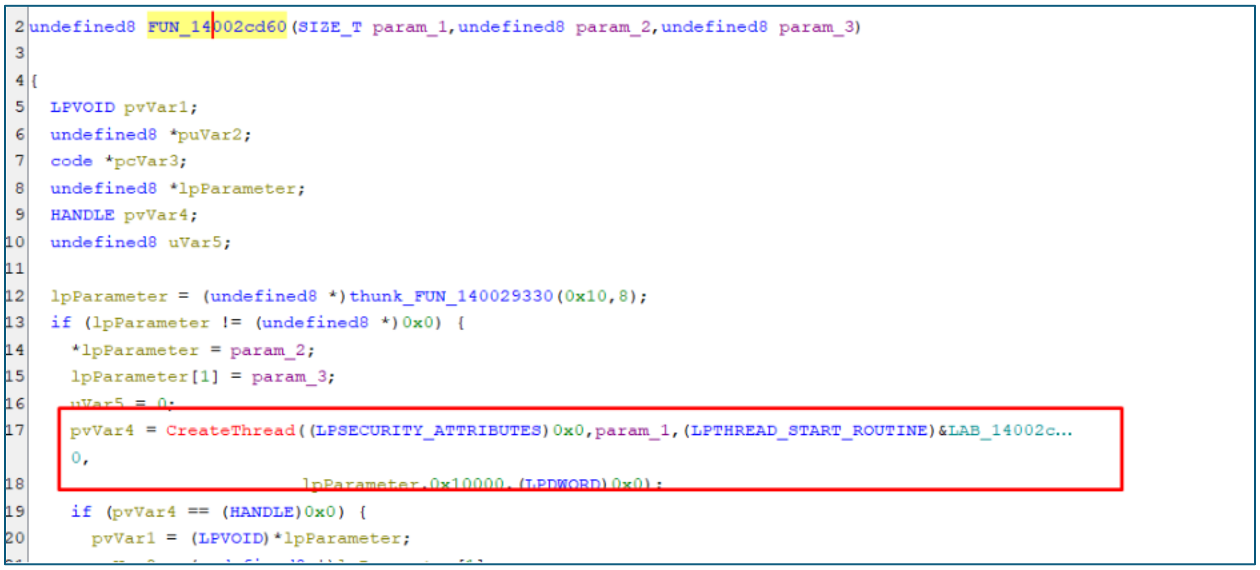
**_Src = *(void **)(param_2 + 0x18);**
**sVar1 = *(size_t *)(param_2 + 0x20);**



```
  local_30 = 0xfffffffffffffffe;
  _Src = *(void **)(param_2 + 0x18);
  sVar1 = *(size_t *)(param_2 + 0x20);
  local_128 = 0;
  local_118 = 0x8000000000000000;
  uStack_100 = uStack_100 & 0xffffffffffffff00;
  local_48 = param_2;
  if ((longlong)sVar1 < 0) {
    lVar5 = 0;
LAB_14000caec:
    local_32 = 1;
    FUN_140041323(lVar5,sVar1,&PTR_s_C:\Users\scorp\.rustup\toolchain_140044a88);
                    /* WARNING: Does not return */
```

*Figure 6. Copying file data into buffer*

A critical function is **FUN_14002cd60**, which creates a new thread and assigns it a function to execute. **LAB_14002cf00** serves as the thread's entry point, initiating the encryption chain.



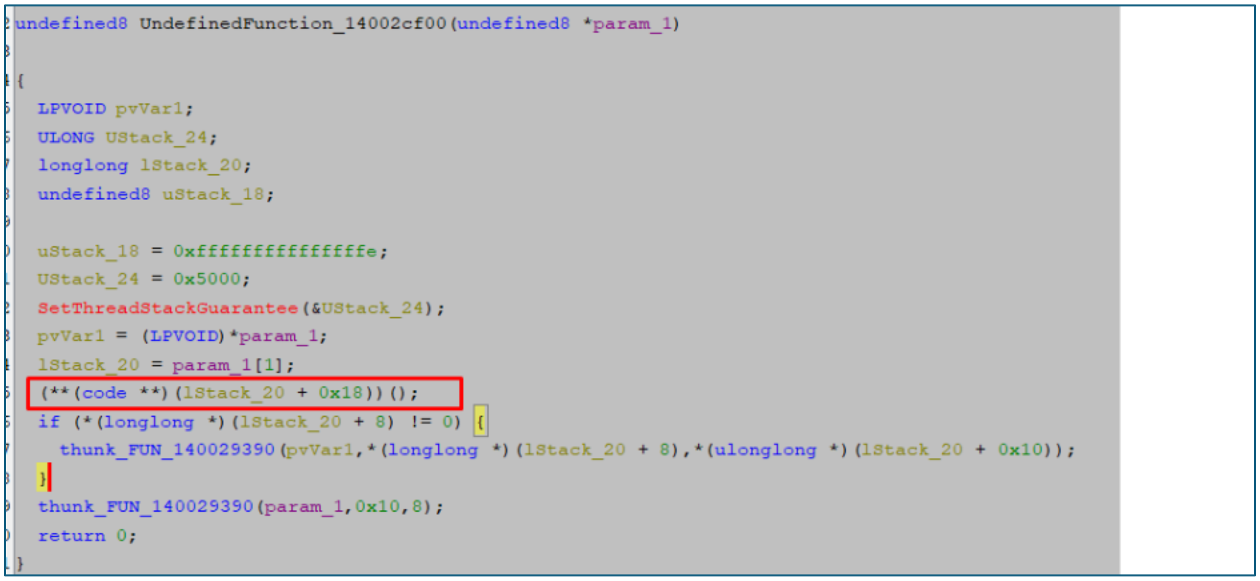
```
undefined8 FUN_14002cd60(SIZE_T param_1,undefined8 param_2,undefined8 param_3)

{
  LPVOID pvVar1;
  undefined8 *puVar2;
  code *pcVar3;
  undefined8 *lpParameter;
  HANDLE pvVar4;
  undefined8 uVar5;

  lpParameter = (undefined8 *)thunk_FUN_140029330(0x10,8);
  if (lpParameter != (undefined8 *)0x0) {
    *lpParameter = param_2;
    lpParameter[1] = param_3;
    uVar5 = 0;
    pvVar4 = CreateThread((LPSECURITY_ATTRIBUTES)0x0,param_1,(LPTHREAD_START_ROUTINE)&LAB_14002c...
    0,
                          lpParameter,0x10000,(LPDWORD)0x0);
    if (pvVar4 == (HANDLE)0x0) {
      pvVar1 = (LPVOID)*lpParameter;
```

*Figure 7. Thread creation*

At this stage, functions are executed sequentially until the final function containing the encryption algorithm is reached.



```
undefined8 UndefinedFunction_14002cf00(undefined8 *param_1)

{
  LPVOID pvVar1;
  ULONG UStack_24;
  longlong lStack_20;
  undefined8 uStack_18;

  uStack_18 = 0xfffffffffffffffe;
  UStack_24 = 0x5000;
  SetThreadStackGuarantee(&UStack_24);
  pvVar1 = (LPVOID)*param_1;
  lStack_20 = param_1[1];
  (**(code **)(lStack_20 + 0x18))();
  if (*(longlong *)(lStack_20 + 8) != 0) {
    thunk_FUN_140029390(pvVar1,*(longlong *)(lStack_20 + 8),*(ulonglong *)(lStack_20 + 0x10));
  }
  thunk_FUN_140029390(param_1,0x10,8);
  return 0;
}
```

*Figure 8. Main code invocation*

By identifying the correct address, the parameter **lStack_20** is observed, and by performing a calculation **qword at PTR_FUN_140045848 + 0x18 = 0x140011790; 0x0000000140011790 is the correct address, as illustrated in the figure below**.

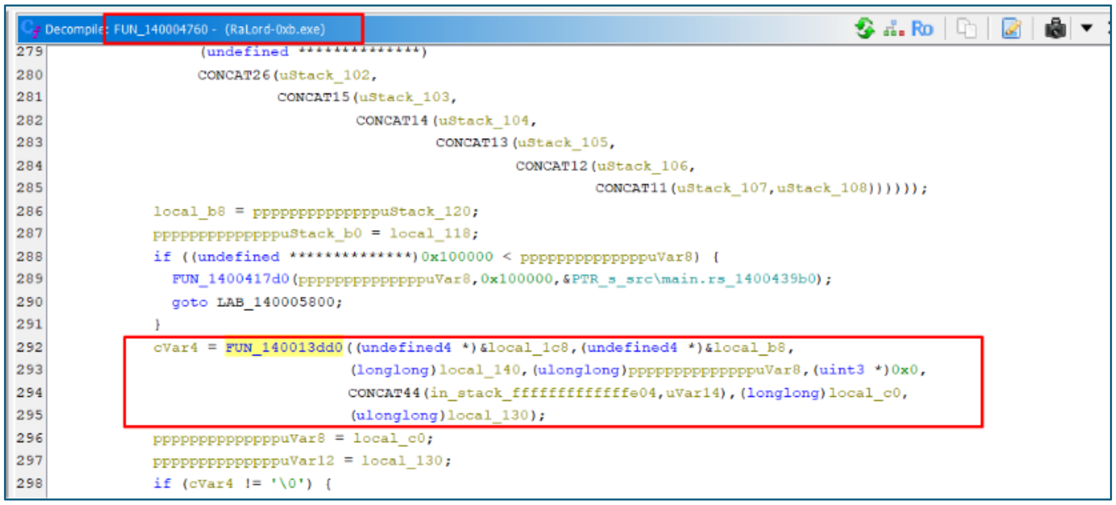
```
?  ->  140011790
```

*Figure 9. Calculated address*

Following the function flow step by step, encryption occurs within **FUN_140004760**, which:

- Opens the file (Read / Open)
- Creates a 0x100000 (1 MB) buffer
- local_140 = malloc(0x100000);
- Main loop (file processing loop)

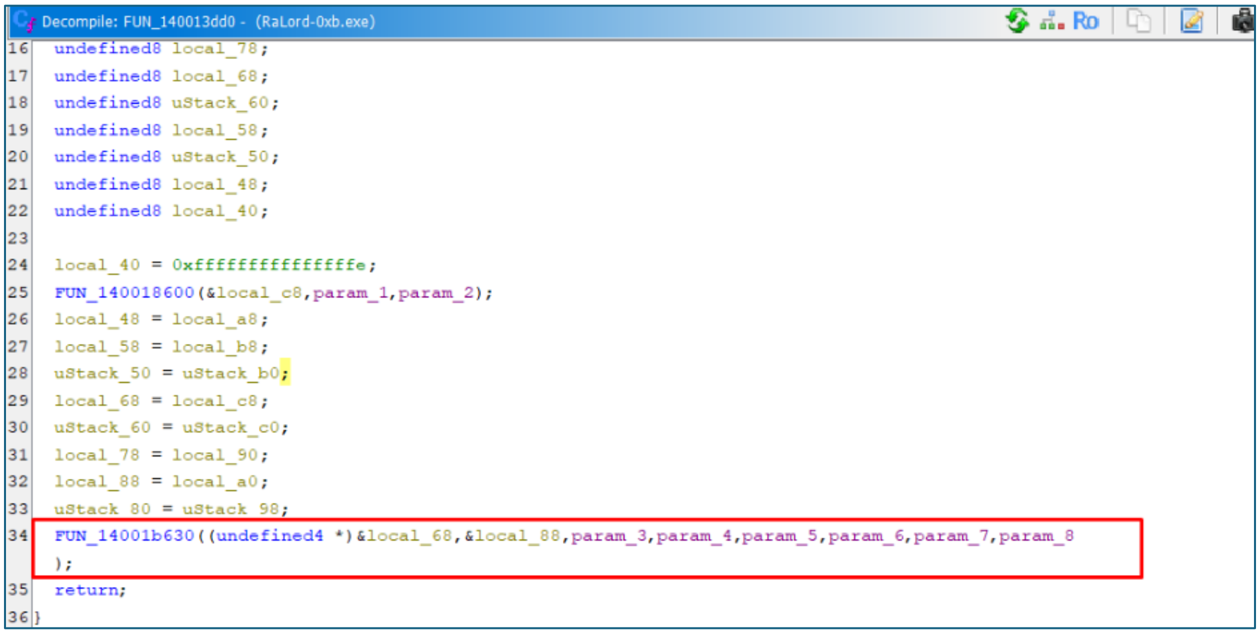
Specifically, the function **FUN_140013dd0** calls the core function **FUN_14001b630**, which contains the complete encryption logic.



Decompile: FUN_140004760 - (RaLord-0xb.exe)

```
                    (undefined **************)
                    CONCAT26(uStack_102,
                             CONCAT15(uStack_103,
                                      CONCAT14(uStack_104,
                                               CONCAT13(uStack_105,
                                                        CONCAT12(uStack_106,
                                                                 CONCAT11(uStack_107,uStack_108))))));
        local_b8 = ppppppppppppppuStack_120;
        ppppppppppppppuStack_b0 = local_118;
        if ((undefined **************)0x100000 < ppppppppppppppuVar8) {
          FUN_1400417d0(ppppppppppppppuVar8,0x100000,&PTR_s_src\main.rs_1400439b0);
          goto LAB_140005800;
        }
        cVar4 = FUN_140013dd0((undefined4 *)&local_1c8,(undefined4 *)&local_b8,
                              (longlong)local_140,(ulonglong)ppppppppppppppuVar8,(uint3 *)0x0,
                              CONCAT44(in_stack_fffffffffffffe04,uVar14),(longlong)local_c0,
                              (ulonglong)local_130);
        ppppppppppppppuVar8 = local_c0;
        ppppppppppppppuVar12 = local_130;
        if (cVar4 != '\0') {
```

*Figure 10. Encryption function call*



Decompile: FUN_140013dd0 - (RaLord-0xb.exe)

```
  undefined8 local_78;
  undefined8 local_68;
  undefined8 uStack_60;
  undefined8 local_58;
  undefined8 uStack_50;
  undefined8 local_48;
  undefined8 local_40;

  local_40 = 0xfffffffffffffffe;
  FUN_140018600(&local_c8,param_1,param_2);
  local_48 = local_a8;
  local_58 = local_b8;
  uStack_50 = uStack_b0;
  local_68 = local_c8;
  uStack_60 = uStack_c0;
  local_78 = local_90;
  local_88 = local_a0;
  uStack_80 = uStack_98;
  FUN_14001b630((undefined4 *)&local_68,&local_88,param_3,param_4,param_5,param_6,param_7,param_8
               );
  return;
}
```

*Figure 11. Main function where encryption actually occurs*

If we analyze the core function **FUN_14001b630**, where file encryption takes place, we can conclude that it uses **ChaCha20 for encryption** combined with **Poly1305 for authentication (MAC).**

- **64-byte (0x40) blocks, typical of ChaCha20**

**if (local_148 < 0x40) { ... }**
**local_148 = local_148 - 0x40;**
**lVar20 = local_140 + 0x40;**
**lVar24 = lVar24 + 0x40;**

- **XORs the keystream with the plaintext**

***_Dst = *_Dst ^ *puVar1;**
**_Dst[1] = _Dst[1] ^ uVar25;**

- **The counter is incremented for each block**

**FUN_140017000(local_128, iVar29, (uint *)&local_188, 0x40);**
iVar29 = block counter

- **Generates a keystream for each block**

**ChaCha20 counter-based keystream**

- **Poly1305 MAC në fund (16 byte TAG)**

**if (param_8 < param_4 + 0x10) return error;**
**FUN_140018e30((longlong)local_d8, ..., 0x10);**

- **+0x10 = 16 byte tag**
- **Authentication check**

**Poly1305 = 16-byte MAC**

## Indicators of Compromise (IoCs)

| | |
|---|---|
| **456B9ADAABAE9F3DCE2207AA71410987F0A571CD8C11F2E7B41468501A863606** | **RaLord-0xb.exe** |
| **794807D348783FDA909C94D28458A7325D2C57644DF17289729BC8158BB4B3AA** | **RaLord-0xb.zip** |
| **a9f701ff27899e33e6948a9d63b02d58e53d95ff54dfdf70f4ce0a5b4faaf9ef** | **README-XeBY311RpMRQ.txt** |

## Recommendations

**The National Cyber Security Authority recommends:**

- Immediate blocking of the above-mentioned Indicators of Compromise on security devices.
- Continuous analysis of logs generated by SIEM (Security Information and Event Management) systems.
- Training non-technical staff on phishing attacks and methods to avoid infection.
- Deployment of network perimeter security devices capable of deep traffic inspection, based not only on access control rules but also on behavioral analysis (Next-Generation Firewalls).
- Segmentation of identified systems into different VLANs, applying Access Control Lists across the entire network perimeter. Web services should be separated from their databases, and Active Directory should reside in a dedicated VLAN.
- Implementation of LAPS (Local Administrator Password Solution) for Microsoft systems to manage local administrator passwords.
- Application of traffic filtering for remote access to hosts (employees / third parties / clients).
- Deployment of solutions that filter, monitor, and block malicious traffic between web applications and the internet, such as Web Application Firewalls (WAF).
- Behavioral-level traffic analysis for endpoint devices through EDR/XDR solutions, enabling malware detection beyond signature-based methods.
- Design and implementation of Identity and Access Management (IAM) solutions to control user identities and privileges in real time, based on the Zero Trust principle.