Tukkan Consultancy Analytics

Revil Ransomware Analysis

tukkan
-malware lab-

Non Confidential

Overview of Sample

MD5: **a4331ff805b0a8f2a2892777c224b65e**

Sha 256:
329983dc2a23bd951b24780947cb9a6ae3fb80d5ef546e8538dfd9459b176483

REvil, also known as Sodinokibi, was one of the most infamous and prolific **ransomware-as-a-service (RaaS)** operations. Operating from approximately 2019 to 2021, the Russian-speaking cybercrime syndicate pioneered the "double extortion" tactic. This involved not only encrypting a victim's sensitive data but also exfiltrating it beforehand, threatening to publish the stolen files on their "Happy Blog" leak site if the ransom wasn't paid. REvil's affiliates targeted thousands of organizations worldwide, from small businesses to major corporations, and were responsible for high-profile attacks against companies like the meat processor **JBS Foods** and the IT software provider **Kaseya**. The group's activities caused widespread disruption and financial damage until a coordinated international law enforcement effort in late 2021 led to the seizure of their servers and the arrest of several key members, effectively dismantling the operation.

Analysis Team

Malware R&D Laboratory by Tukkan

John Smith, CEO

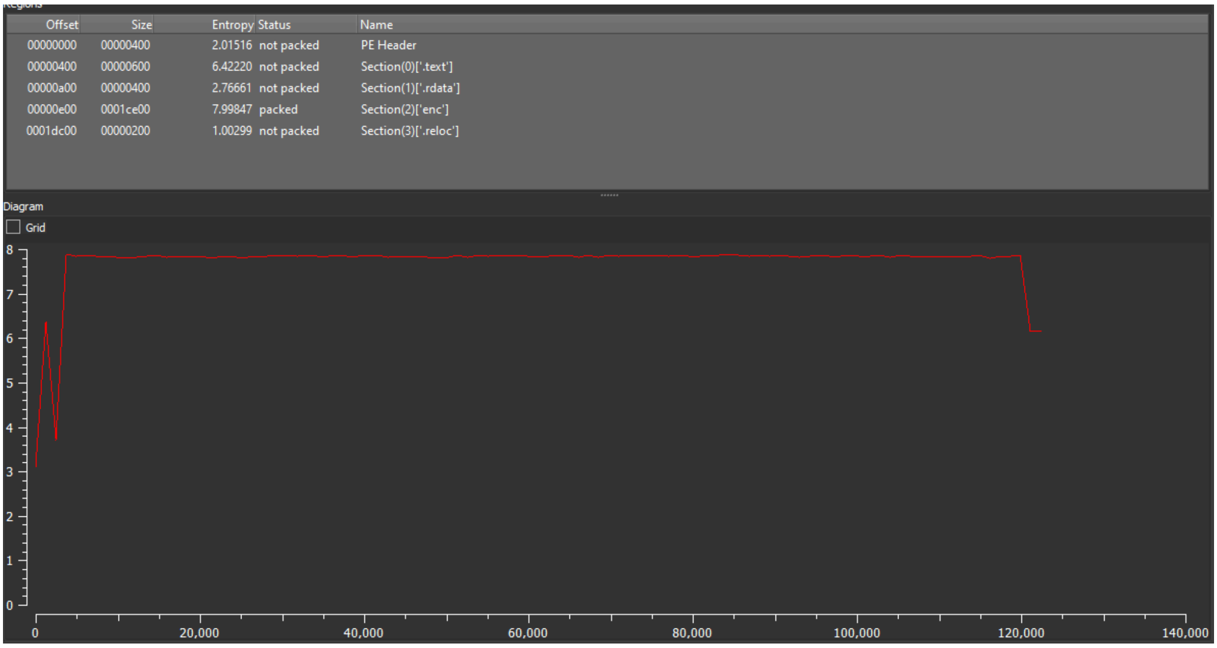
Jane Smith, Head of Malware Lab

Bernard Smith, SANS GREM, eCTHP, eCDFP

Technical Analysis

This sample of Revil Ransomware consists of 2 stages. First Stage Decrypts the second stage and execute stage 2.

When we look into sections of the malware, we can see that there is unique section called ‘**enc**’. This section has 7.99 entropy, which is a common indicator for encrypted data.

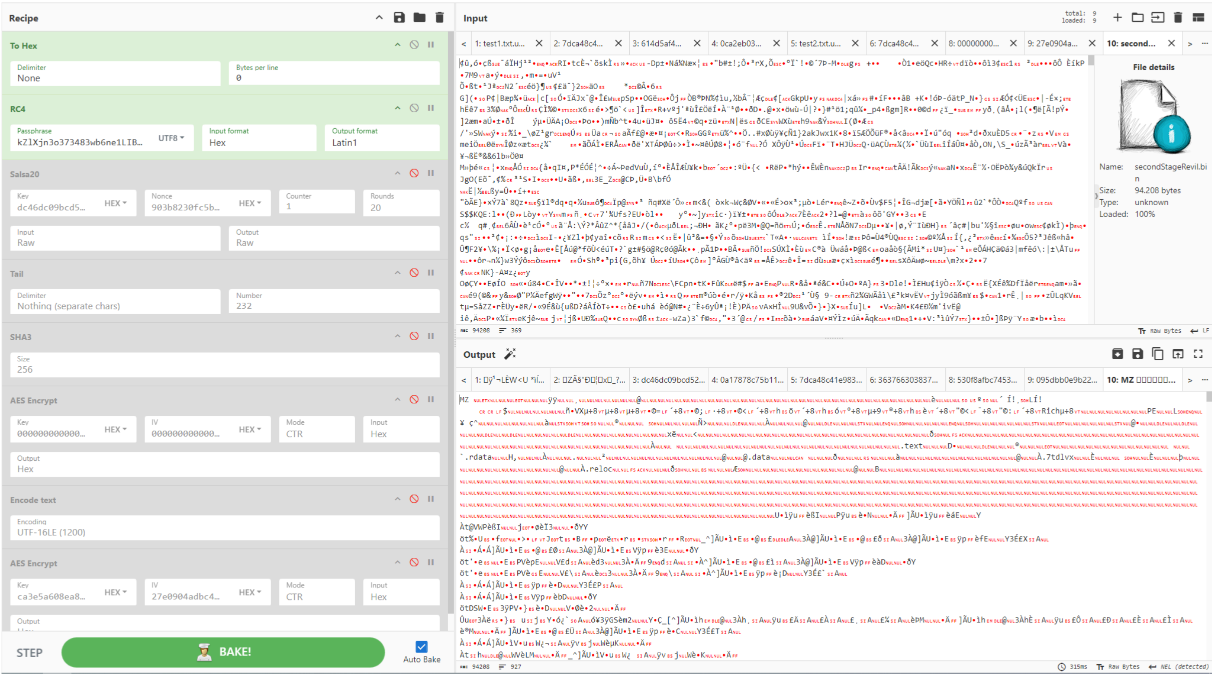


We can see the general structure of the executable below, After Entry function initializes command line arguments, it calls Loader function. Loader function first initializes the key for encrypted section ‘**enc**’, then decrypts it using RC4 encryption algorithm. Both second stage and first stage of the ransomware doesn’t use Wincrypt API calls to encrypt or decrypt data, all encryption algorithms are included in the malwares. This makes it harder to recognize encryption algorithms included in the ransomware. After decrypting stage 2 with rc4, LoadRansomware function is called.

```
BOOL __thiscall Loader(_DWORD *this)
{
  int v2; // ecx
  char buffer[256]; // [esp+4h] [ebp-120h] BYREF
  char key[32]; // [esp+104h] [ebp-20h] BYREF

  setBufferWithSizeTo0(buffer, 256);
  qmemcpy(key, "kZlXjn3o373483wb6ne1LIBNWD3KWBEK", sizeof(key));
  RC4_init(v2, key);
  RC4_crypt(buffer);
  return LoadRansomware(this) == 0;
}
```

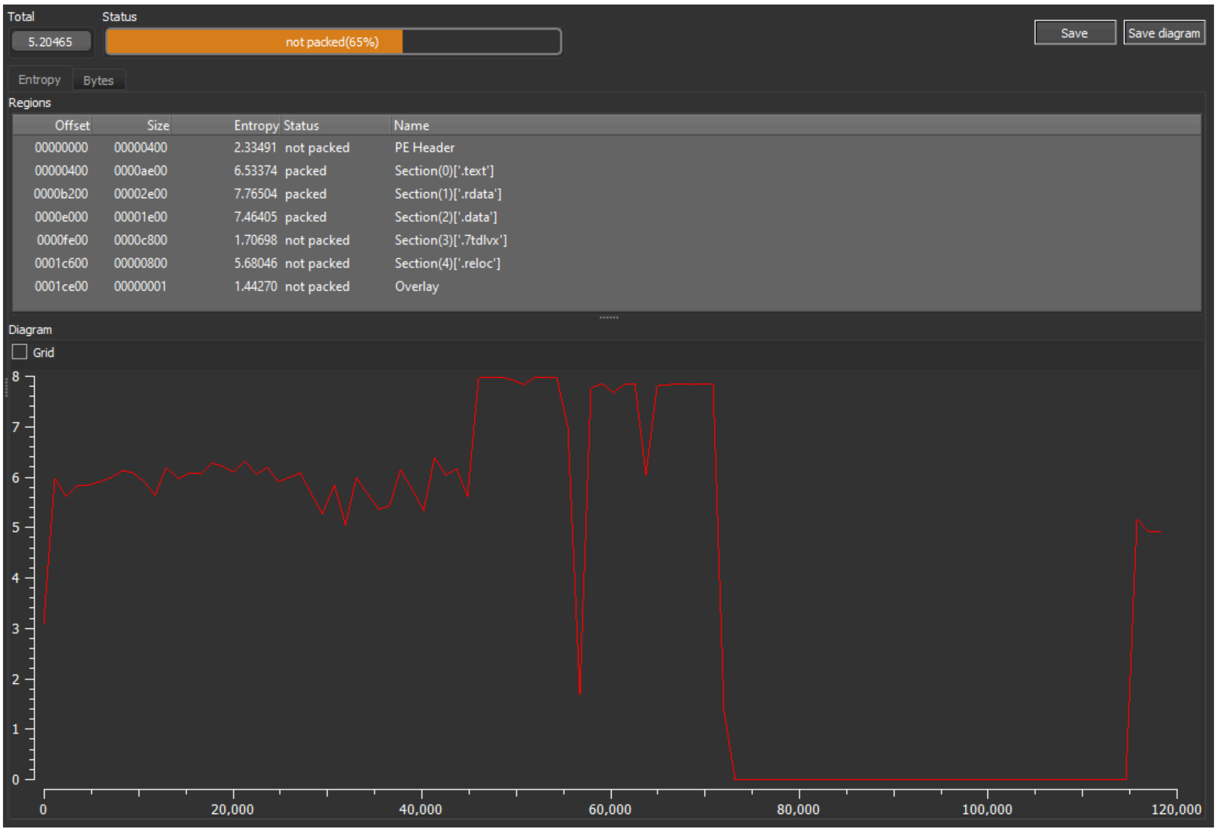
We can use the rc4 key to obtain second stage of the ransomware



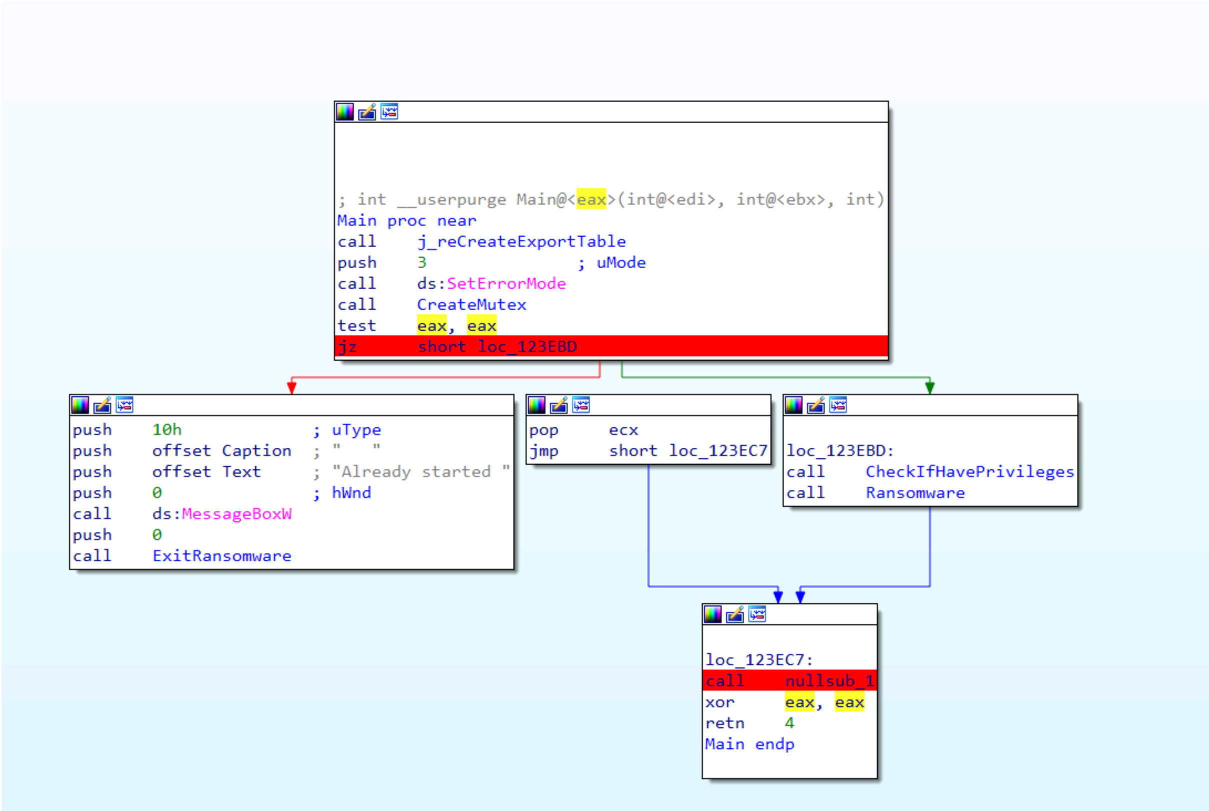
LoadRansomware function first calls **NtAllocateVirtualMemory** and **NtWriteVirtualMemory** functions to allocate and write the decrypted payload, the verifies if the decrypted block is a valid execute pe program. Finally, it rebases the executable and runs it from its entrypoint.

```
reBaseProgram(v30);
removeImports(v30);
entrypoint = &v30[v8[4]];
if ( (_WORD)v23 )
  ((void (__stdcall *)(char *, int, _DWORD))entrypoint)(entrypoint, 1, 0);
else
  ((void (__stdcall *)(_DWORD, _DWORD, _DWORD, _DWORD))entrypoint)(*a1, a1[1], a1[2], a1[3]);
return 0;
```

When we look the second stage, we see **.rdata** and **.data** sections have entropy close to 8 for most of its data, which could indicate most of its data is encrypted. When we investigate the sample further, we confirm that most of this data is encrypted using rc4 encryption algorithm.



We can see the main execution flow of the ransomware below, first it restores its export table using a technique called api hashing. This technique is used to hide import tables in static analysis by recreating imports on runtime by using hashed addresses of the used functions.



```
  char v13; // [esp+1Dh] [ebp-4Fh]
  CHAR ProcName[4]; // [esp+20h] [ebp-4Ch] BYREF
  char v15; // [esp+34h] [ebp-38h]
  CHAR v16[4]; // [esp+38h] [ebp-34h] BYREF
  char v17; // [esp+48h] [ebp-24h]
  CHAR v18[4]; // [esp+4Ch] [ebp-20h] BYREF
  char v19; // [esp+5Ah] [ebp-12h]
  int dll_ptr[3]; // [esp+5Ch] [ebp-10h] BYREF
  char v21; // [esp+6Ah] [ebp-2h]

  for ( i = 0; i < 0x294; i += 4 )
    *(&RtlAdjustPrivilege + i) = apiHashing(*(&RtlAdjustPrivilege + i));
  rc4Crypt(poi_array, 2440, 6, 21, a5);
  v13 = 0;
  rc4Crypt(poi_array, 2409, 6, 14, dll_ptr);
  v21 = 0;
  rc4Crypt(poi_array, 779, 4, 20, ProcName);
  v15 = 0;
  rc4Crypt(poi_array, 1911, 13, 16, v16);
  v17 = 0;
  rc4Crypt(poi_array, 2653, 15, 14, v18);
  v19 = 0;
  v1 = restoreImports(a5);
  CreateStreamOnHGlobal = GetProcAddress(v1, v7);
  v2 = restoreImports(dll_ptr);
  CoInitializeEx = GetProcAddress(v2, v8);
  v3 = restoreImports(ProcName);
  CoInitializeSecurity = GetProcAddress(v3, v9);
  v4 = restoreImports(v16);
  CoCreateInstance = GetProcAddress(v4, v10);
  v5 = restoreImports(v18);
  result = GetProcAddress(v5, v11);
  CoUninitialize = result;
  return result;
}
```

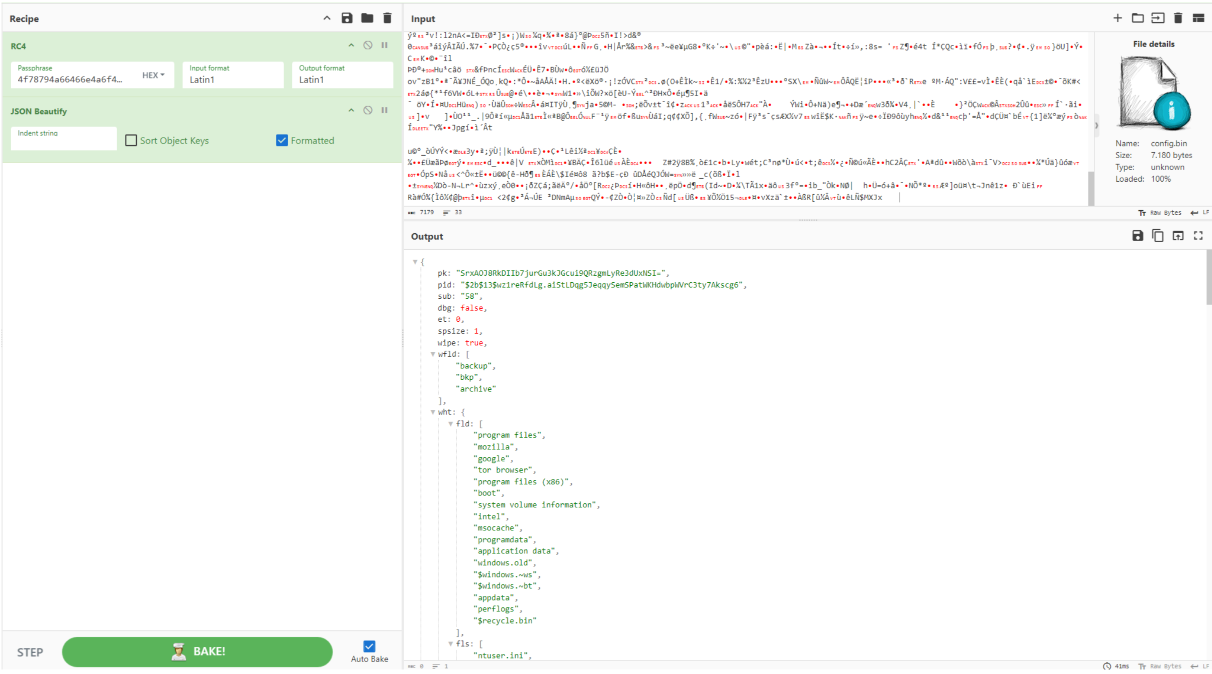
After restoring the import table, the ransomware ensures that only one instance is running by creating a mutex with the name: **Global\530D4C9F-32A8-6FCB-DFF6-A5DE7490E287**. It then checks whether it has been executed with administrator privileges on the target system. If the necessary privileges are not present, the ransomware attempts to re-execute itself with elevated (administrator) rights. This process involves a basic UAC (User Account Control) prompt without any bypass techniques, indicating that the threat actor likely already has administrator access on the systems where the ransomware is deployed.

```
17 db     0
18 aGlobal530d4c9f:
18 text "UTF-16LE", 'Global\530D4C9F-32A8-6FCB-DFF6-A5DE7490E287'
5E db     0
5F db     0
```

After necessary conditions are met, ransomware starts by initializing configuration. Configuration data is stored in the malware as encrypted. Configuration is a Json object, ransomware parses this object to initialize necessary settings, then other settings that is not included in the config is decrypted and initialized.

```
RansomConfig = RansomwareConfigDecryptor();
result = 0;
if ( RansomConfig )
{
  v14[0] = 0;
  v14[1] = 0;
  v14[4] = 0;
  v14[5] = 0;
  v14[2] = createHeapWithSize;
  v14[3] = FreeHeap;
  ConfigSize = strlen(RansomConfig);
  v3 = sub_12AE74(v14, RansomConfig, ConfigSize);
  if ( v3 )
  {
    rc4Crypt(&p_poi_array, 135, 7, 2, &a5);
    BYTE2(a5) = 0;
    rc4Crypt(&p_poi_array, 273, 4, 3, &dll_ptr);
    HIBYTE(dll_ptr) = 0;
    rc4Crypt(&p_poi_array, 88, 4, 3, &v41);
    HIBYTE(v41) = 0;
    rc4Crypt(&p_poi_array, 2489, 7, 3, &v40);
    HIBYTE(v40) = 0;
    rc4Crypt(&p_poi_array, 918, 16, 3, &v39);
    HIBYTE(v39) = 0;
    rc4Crypt(&p_poi_array, 1699, 5, 3, &v38);
    HIBYTE(v38) = 0;
    rc4Crypt(&p_poi_array, 287, 15, 3, &v37);
    HIBYTE(v37) = 0;
    rc4Crypt(&p_poi_array, 2906, 12, 3, &v36);
    HIBYTE(v36) = 0;
    rc4Crypt(&p_poi_array, 635, 15, 3, &v35);
    HIBYTE(v35) = 0;
    rc4Crypt(&p_poi_array, 3027, 15, 5, &v30);
    v31 = 0;
    rc4Crypt(&p_poi_array, 187, 11, 5, &v28);
    v29 = 0;
    rc4Crypt(&p_poi_array, 1609, 4, 3, &v34);
    HIBYTE(v34) = 0;
    rc4Crypt(&p_poi_array, 1767, 9, 2, &v45);
    BYTE2(v45) = 0;
    rc4Crypt(&p_poi_array, 32, 16, 6, &v26);
    v27 = 0;
    rc4Crypt(&p_poi_array, 363, 5, 3, &v33);
    HIBYTE(v33) = 0;
000008ED ConfigInitalization:53 (1214ED)
```

Configuration file contains a whitelist folders and files, debug mode setting, service and process list and other settings that is used in ransomware.



While some of these settings meanings are still unknown after analysis, we can see that this configuration contains debug mode switch, folders to be wiped, whitelist for folders and files that would corrupt the OS in case of encryption, extensions that won’t be meaningful to encrypt, services and processes to stop to achieve maximum efficiency from encryption process, ransom note and ransomware note format.


```
"pk": "SrxAOJ8RkDIIb7jurGu3kJGcui9QRzgmLyRe3dUxNSI=",
"pid": "$2b$13$wz1reRfdLg.aiStLDqg5JeqqySemSPatWKHdwbpWVrC3ty7Akscg6",
"sub": "58",
"dbg": false,
"et": 0,
"spsize": 1,
"wipe": true,
"wfld": [
    "backup",
    "bkp",
    "archive"
],
```

```json
"wht": {
    "fld": [
        "program files",
        "mozilla",
        "google",
        "tor browser",
        "program files (x86)",
        "boot",
        "system volume information",
        "intel",
        "msocache",
        "programdata",
        "application data",
        "windows.old",
        "$windows.~ws",
        "$windows.~bt",
        "appdata",
        "perflogs",
        "$recycle.bin"
    ],
```

```json
"fls": [
    "ntuser.ini",
    "autorun.inf",
    "ntldr",
    "iconcache.db",
    "ntuser.dat",
    "boot.ini",
    "bootsect.bak",
    "desktop.ini",
    "ntuser.dat.log",
    "bootfont.bin",
    "thumbs.db"
],
```

```
"ext": [
    "dll",
    "scr",
    "icns",
    "ics",
    "nomedia",
    "sys",
    "ps1",
    "hlp",
    "lock",
    "spl",
    "msi",
    "mpa",
    "wpx",
    "ocx",
    "drv",
    "msp",
    "cmd",
    "rtp",
    "key",
    "deskthemepack",
    "bat",
    "ico",
    "mod",
    "prf",
    "diagcfg",
    "cpl",
    "adv",
    "hta",
    "ani",
    "386",
    "bin",
    "diagcab",
    "msu",
    "rom",
    "diagpkg",
    "shs",
    "themepack",
    "theme",
    "com",
    "cab",
    "msc",
    "icl",
    "exe",
    "idx",
    "nls",
    "lnk",
    "msstyles",
    "cur"
]
```

```
"svc": [
    "QBCFMonitorService","thebat",
    "dbeng50","winword","dbsnmp",
    "VeeamTransportSvc","disk+work",
    "TeamViewer_Service.exe","firefox",
    "QBIDPService","steam","onenote",
    "CVMountd","cvd","VeeamDeploymentSvc",
    "VeeamNFSSvc","bedbh","mydesktopqos",
    "avscc","infopath","cvfwd","excel",
    "beserver","powerpnt","mspub",
    "synctime","QBDBMgrN","tv_w32.exe",
    "EnterpriseClient","msaccess","ocssd",
    "mydesktopservice","sqbcoreservice",
    "CVODS","DellSystemDetect","oracle",
    "ocautoupds","wordpad","visio","SAP",
    "bengien","TeamViewer.exe","agntsvc",
    "CagService","avagent","ocomm",
    "outlook","saposcol","xfssvccon",
    "isqlplussvc","pvlsvr","sql",
    "tbirdconfig","vxmon","benetns",
    "tv_x64.exe","encsvc","sapstartsrv",
    "vsnapvss","raw_agent_svc",
    "thunderbird","saphostexec"
],
```

```
"prc": [
    "vsnapvss","EnterpriseClient","firefox","infopath",
    "cvd","tv_x64.exe","VeeamTransportSvc","steam","encsvc",
    "mydesktopservice","outlook","synctime","ocssd","SAP",
    "cvfwd","bengien","vxmon","bedbh","ocomm","ocautoupds",
    "raw_agent_svc","oracle","disk+work","powerpnt","saposcol",
    "sqbcoreservice","sapstartsrv","beserver","saphostexec",
    "dbeng50","isqlplussvc","CVODS","DellSystemDetect",
    "CVMountd","TeamViewer.exe","dbsnmp","thunderbird","mspub",
    "wordpad","visio","benetns","QBCFMonitorService","TeamViewer_Service.exe",
    "tv_w32.exe","QBIDPService","winword","thebat","VeeamDeploymentSvc",
    "avagent","QBDBMgrN","mydesktopqos","xfssvccon","sql","tbirdconfig",
    "CagService","pvlsvr","avscc","VeeamNFSSvc","onenote","excel",
    "msaccess","agntsvc"
],
```

```
"dmn": "",
"net": false,
"exp": false,
"arn": false,
"nbody": "LQAtAC0APQA9AD0AIABXAGUAbABjAG8AbQBlAC4AIABBAGcAYQBpAG4ALgAgAD0APQA9AC0ALQAtAA0ACgANAAoAWwAr
"nname": "{EXT}-README.txt",
"img": "QQBsAGwAIABvAGYAIAB5AG8AdQByACAAZgBpAGwAZQBzACAAYQByAGUAIABlAG4AYwByAHkAcAB0AGUAZAAhAA0ACgANAA
```

After initialization is complete, ransomware first check if the language used in the target system is one of the languages in language whitelist. If the language is on this array ransomware exits directly. This gives us an idea about threat actors origin and motives.

Decimal	LangID (Hex)	Language (Locale)
1049	`0x0419`	Russian (Russia)
1058	`0x0422`	Ukrainian (Ukraine)
1059	`0x0423`	Belarusian (Belarus)
1064	`0x0428`	Tajik (Cyrillic, Tajikistan)
1067	`0x042B`	Armenian (Armenia)
1068	`0x042C`	Azeri (Cyrillic, Azerbaijan)
1079	`0x0437`	Georgian (Georgia)
1087	`0x043F`	Kazakh (Kazakhstan)
1088	`0x0440`	Kyrgyz (Kyrgyzstan)
1090	`0x0442`	Turkmen (Turkmenistan)
1091	`0x0443`	Uzbek (Latin, Uzbekistan)
1092	`0x0444`	Tatar (Russia)
2072	`0x0818`	Romanian (Moldova)

Malware stop services and kill processes found on configuration to make sure when it tries to encrypt files, files opened by these processes and services are accessible to the ransomware. Afterwards, it deletes shadow copies existing on the system to make sure target can't recover their files from copy.

```
}
v0 = ConfigInitalization();
if ( v0 )
{
  if ( !dword_130FF0 && DisarmRansomwareByLanguage() )
    ExitRansomware(0);
  sub_1228C3();
  v6 = 0;
  RtlAdjustPrivilege(20, 1, 0, &v6);
  if ( dword_13100C )
  {
    Thread = CreateThread(0, 0, firstThread, 0, 0, 0);
    CloseHnd(Thread);
    StopServicesOnConfig();
    KillProcessesOnConfig(0, 0, sub_12297D);
    DeleteShadowCopiesUsingPowershell();
  }
  RtlAdjustPrivilege(9, 1, 0, &v6);
  if ( sub_122FE1() )
  {
    if ( !dword_131008 )
      sub_12419B();
    sub_1255AB(0, 1);
    if ( !dword_131008 && dword_130FEC )
      sub_12577D(dword_130F58, 59, 0, sub_1229A4);
  }
}
```

Malware starts encryption process by starting a thread for encrpyting the files, then calls a function that creates keys for each file to be encrypted and writes ransom notes to all folders.

```
    concat_1(v2, L"*");
    LODWORD(v5) = FindFirstFileW(v2, &FindFileData);
    hFindFilea = v5;
    if ( v5 != -1 )
    {
      do
      {
        if ( sub_125AA1(FindFileData.cFileName, ".")
          && sub_125AA1(FindFileData.cFileName, L"..")
          && (FindFileData.dwFileAttributes & 0x400) == 0 )
        {
          concat_0(&v2[v12], FindFileData.cFileName);
          if ( (FindFileData.dwFileAttributes & 0x10) != 0 )
          {
            concat_1(v2, L"\\");
            if ( (hFindFile)[1](v2, FindFileData.cFileName) )
            {
              sub_12705D(&v13, v2);
              *(hFindFile + 3) += (hFindFile[10])(hFindFile[3], v2, FindFileData.cFileName);
            }
          }
          else
          {
            nFileSizeLow = FindFileData.nFileSizeLow;
            nFileSizeHigh = FindFileData.nFileSizeHigh;
            if ( (hFindFile[2])(v2, FindFileData.cFileName, FindFileData.nFileSizeLow, FindFileData.nl
              *(hFindFile + 4) += (hFindFile[11])(hFindFile[4], v2, FindFileData.cFileName, nFileSizel
          }
        }
      }
      while ( !*hFindFile && FindNextFileW(hFindFilea, &FindFileData) );
      LODWORD(v5) = FindClose(hFindFilea);
      goto LABEL_20;
    }
  }
  while ( v4 )
  {
    v8 = v4;
    v4 = *(v4 + 4);
    FreeHeap(*v8);
    LODWORD(v5) = FreeHeap(v8);
  }
  return v5;
}
0000632A findFilesToEncrypt:74 (126F2A)
```

Key generation process can be seen below: Key generation algorithm makes use of Advanced Encryption Standard (AES) algorithm in counter mode and Elliptic curve cryptography with 25519 donna implementation. ECC functions used in the ransomware matches with the code from “**https://github.com/agl/curve25519-donna**” repository, we can assume threat actor copied the code from this repository to reduce of making mistakes in implementation of the encryption algorithm. To create encrpytion keys, Ransomware uses aes in an unconventional way. Ransomware creates a random using an embedded random generator, and if they fail, it uses wincrypt api to generate random keys.

<whole secret generation process here>

```
int __cdecl RandomCreatorWithSize(unsigned int *ptrAddr, int size)
{
  int result; // eax
  BYTE v3[48]; // [esp+4h] [ebp-30h] BYREF

  if ( !startCheck )
  {
    result = InitalizeAes();
    if ( !result )
      return result;
    getCritObj2(&critObj1);
    startCheck = 1;
  }
  getCritObj(&critObj1);
  if ( aesCtr > 0x1000000 && (!GetRandomNumber(v3, 0x30u) || !createIvForAesUsingRandoms(&AesRounds, v3, 48, 0, 0))
    || !ivCreate(&AesRounds, ptrAddr, size) )
  {
    return 0;
  }
  LeaveCritObj(&critObj1);
  return 1;
}
```

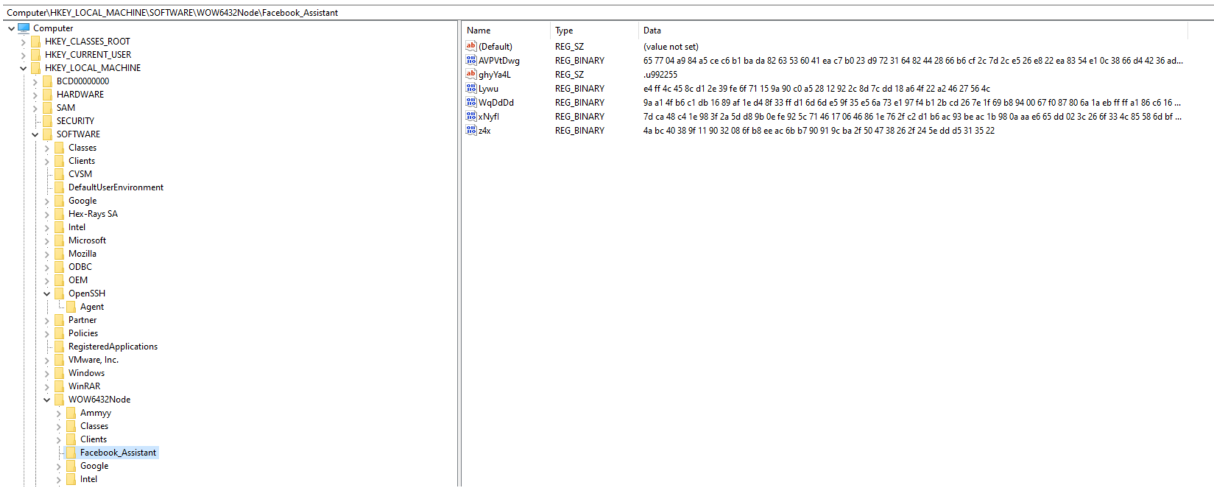
```
void __cdecl fileEncryptionKeyGenerationModule(encryptCtx *encryptCtx)
{
  int k[8]; // [esp+Ch] [ebp-40h] BYREF
  char secret[32]; // [esp+2Ch] [ebp-20h] BYREF

  qmemcpy(encryptCtx->data1, data1, sizeof(encryptCtx->data1));
  qmemcpy(encryptCtx->data2, data2, sizeof(encryptCtx->data2));
  calculatePublicKeyEcc(secret, encryptCtx->eccPubKey);
  keyCreator_(secret, data3, k);
  nullifyBuffer(secret, 32);
  salsa20KeySetup(encryptCtx[1].unkBlock, k, 256);
  nullifyBuffer(k, 32);
  RandomCreatorWithSize(encryptCtx->salsaNonce, 8);
  Salsa20ivSetup(encryptCtx[1].unkBlock, encryptCtx->salsaNonce);
  encryptCtx->eccCrc32Checksum = hashingAlgorithm(0, encryptCtx->eccPubKey, 32);
  encryptCtx->size1 = unk1;
  encryptCtx->size2 = unk2;
  encryptCtx->unk1 = 0;
  Salsa20Encrypt_Decrypt(encryptCtx[1].unkBlock, &encryptCtx->unk1, &encryptCtx->unk1, 4u);
}
```

After Key generation for a file is completed, files are encrypted using a thread that accesses files asynchronously to increase encrpytion speed. To encrypt files, ransomware uses **salsa20** encryption algorithm with the key and nonce created before. Some parts of the encryption context is written to end of encrpyted files to provide necessary information for decryption. This information contains, shared key that is generated secret used to encrypt the file, crc32 hash of this shared key, other information that is asssumed to be used in decryption process and nonce for salsa20 encryption.

```
switch ( v2[17].Internal )
{
  case 1u:
    readFileAndPostIO(a1, v2, 2);
    break;
  case 2u:
    v3 = 1;
    if ( unk1 == 1 )
      v3 = 3;
    salsaAndWrite(v2, v6, v3);
    break;
  case 3u:
    WriteToAFile_Thread(v2, 4u);
    break;
  case 4u:
    concatAndMove_thread(a1, v2);
    break;
  }
}
```

Part of the encryption information embedded in files are also written to **HKLM\SOFTWARE\WOW6432Node\Facebook_Assistant** key on **64-bit** systems. This information is same in all encrypted files.



Conclusion

Ransomware uses ECC to keep secret hidden on the endpoint and nullifies any used secrets from the memory after key generation is complete. This key is a public key of an asymmetric encryption algorithm, allowing only holder of the private key to decrypt the secret that is used to generate file encryption key. Usage of the AES in unconventional way reduces the chances of recreation of encryption keys by following same procedures, considering all encryption keys are created in a nested structure. At this moment it does not seems possible to decrypt files without the mentioned private key.