From LukaLocker to Nitrogen

Lightning talk – BOTCONF 2025

GLIMPS

Reporting

We had a report / question from the editor in chief of LeMagIT, Valery Marchive

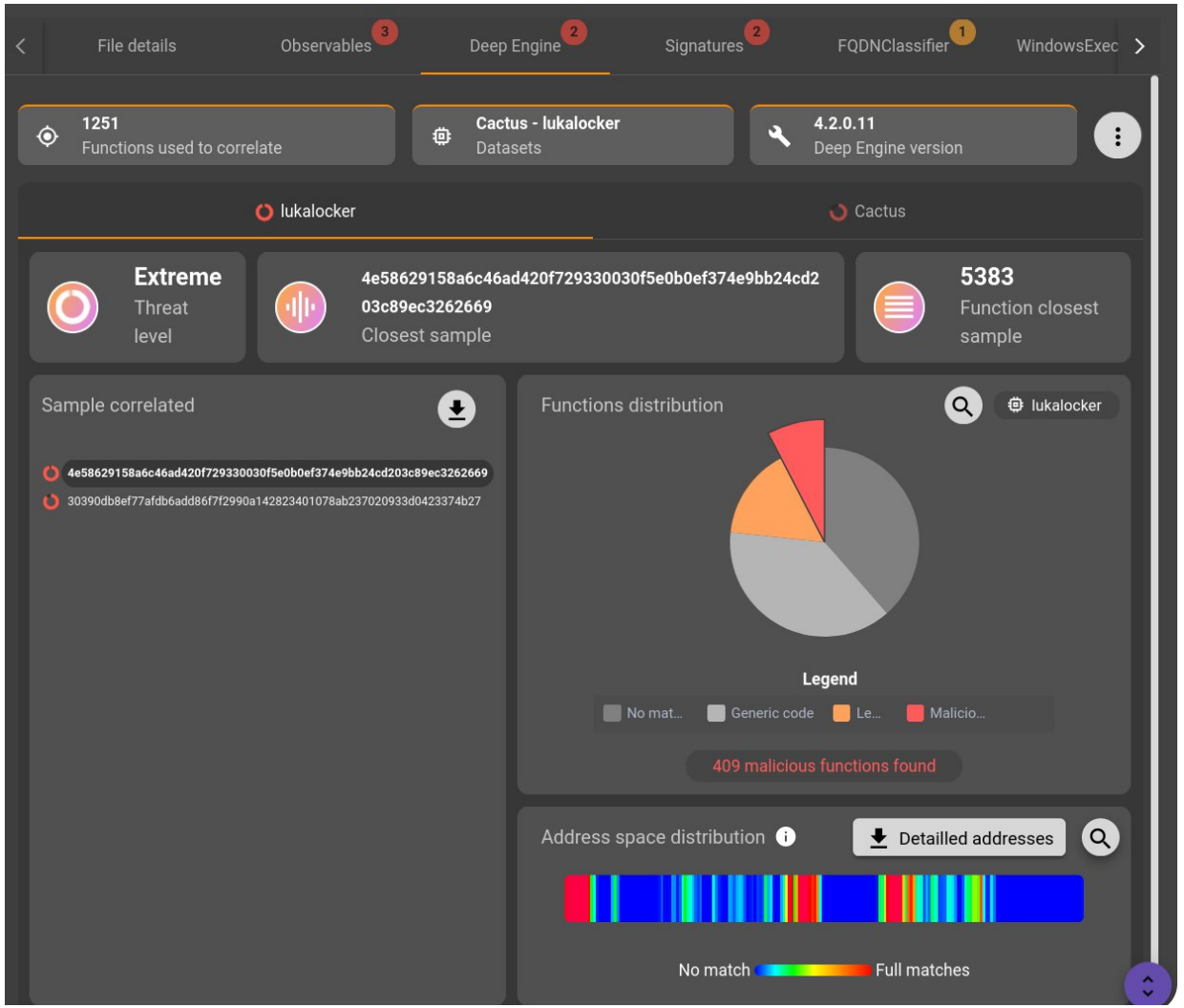
« Look at these ransomware samples, do they share some features? »

SHA256	Family	Date
0a8088e2ba539541f476836c6f4e5812c4ae5c52133801faa1bc3806a4ade683	Cactus	November 2023
30390db8ef77afdb6add86f7f2990a142823401078ab237020933d0423374b27	LukaLocker1	May 2024
4e58629158a6c46ad420f729330030f5e0b0ef374e9bb24cd203c89ec3262669	LukaLocker2	June 2024
e6a498b89aa04d7c25cbfa96599a4cd9bdcc79e73bf7b09906e5ca85bda2bff6	Nitrogen	March 2025

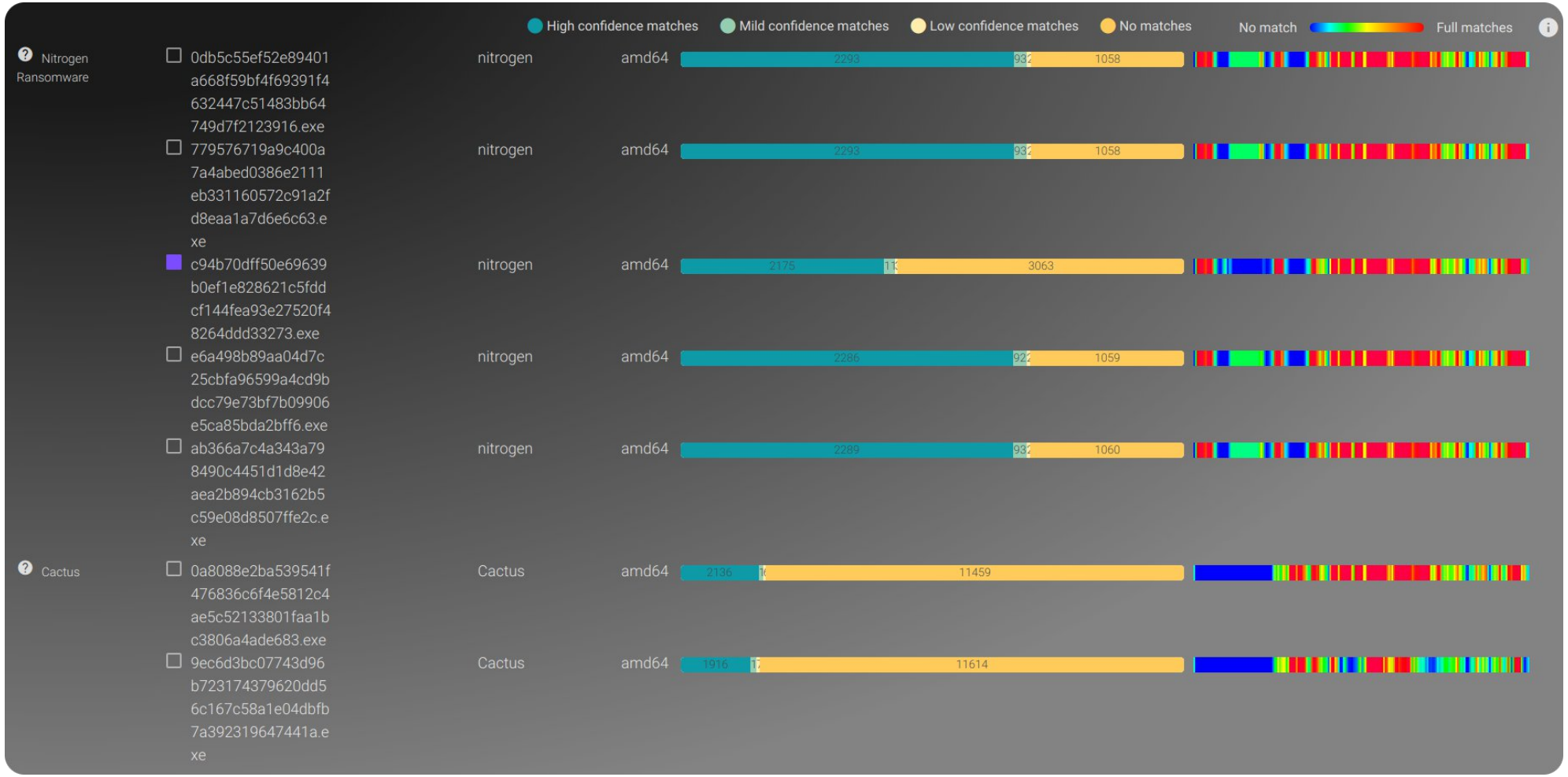
Families were qualified by paiement links inside the ransom notes.

Correlations

Lukalocker X Nitrogen



Similarities



Cactus, Lukalocker and Nitrogen seem to share some functions.

Can you explain this Ransomware?

Functions

54.6%

Here you will find the functions that have weighted in the prediction of the malconv model. We also wonder if these functions were found by the correlator.

Highlight the correlations of malconv and the correlator

Offset	Virtual Address	Malconv score	Correlator score	Total score	CAPA Rules	MITRE ATT&CK
0×79ac0	5369210560	0.00000	0.00008	0.00008	-	-
0xcf210	5369560592	0.09945	-	0.09945	encode data using XOR parse PE header	T1027 : Defense Evasion (Obfuscated Files or Information) T1027.005 : Defense Evasion (Obfuscated Files or Information) ↳ Indicator Removal from Tools T1129 : Execution (Shared Modules)
0×173d0	5368807376	0.03512	-	0.03512	encode data using XOR contain obfuscated stackstrings	T1027 : Defense Evasion (Obfuscated Files or Information) T1027.005 : Defense Evasion (Obfuscated Files or Information) ↳ Indicator Removal from Tools T1129 : Execution (Shared Modules)
0×94360	5369319264	0.02745	-	0.02745	-	-
0xb67b0	5369459632	0.02293	-	0.02293	-	-

« ‹ 1 2 3 4 5 › » 5

Here we can spot interesting functions in the binaries

Let’s have a look!

The samples work exactly the same way:

1 > mutex creation

2 > calling the file iteration process (in charge of file encryption)

3 > exits with an immediate shutdown

The fun fact here: they use the same mutex fjhv6esdvsx

```
sub_14004CAE7();
sub_140002360();
strcpy(v20.m128i_i8, "GetCommandLineW");
strcpy(v18.m128i_i8, "kernel32.dll");
v0 = (__int64 (*)(void))FunctionLibCall_thelib_function(&v18, &v20);
v1 = v0();
sub_140035E80(v1);
if ( !byte_14015B2E0 )
{
  strcpy(v21, "fjhv6esdvsx");
  v20 = _mm_loadu_si128((const __m128i *)&xmmword_1401225F0);
  strcpy(v18.m128i_i8, "CreateMutexA");
  strcpy((char *)v17, "kernel32.dll");
  v6 = (__int64 (__fastcall *)(_QWORD, __int64, __m128i *))FunctionLibCall_thelib_function(v17, &v18);
  v7 = v6(0i64, 1i64, &v20);
  strcpy(v21, "ecl");
  strcpy(v18.m128i_i8, "kernel32.dll");
  v20 = _mm_loadu_si128((const __m128i *)&xmmword_140122610);
  v8 = (unsigned int (__fastcall *)(__int64, _QWORD))FunctionLibCall_thelib_function(&v18, &v20);
  if ( v8(v7, 0i64) )
    return 1i64;
}
if ( ((unsigned int)sub_14000BCC0() == 11 || (unsigned int)sub_14000BCC0() == 10)
{
  for ( i = (_QWORD *)v17[0]; i; i = (_QWORD *)i[4] )
  {
    v10 = i[1];
    v18.m128i_i64[0] = (__int64)v19;
    sub_140034100(&v18, *i, *i + 2 * v10);
    fileIterationAndProcess(
      (unsigned int)&v18,
      v11,
      v12,
      v13,
      v17[0],
      v17[1],
      v18.m128i_i64[0],
      v18.m128i_i64[1],
      v19[0]);
    sub_1400F31D0(&v18);
  }
}
if ( (unsigned __int8)sub_140014420(0i64) )
  sub_140014450(0i64);
nullsub_3();
sub_1400340B0("all files crypted, exit 0\n");
if ( (unsigned __int8)sub_14000BB60() )
{
  system(Command);
  system("shutdown -r -t 0");
}
raise(2);
return 0i64;
}
```

Compare

```
LABEL_10:
  v114 = v18;
  *(_WORD *)((char *)v17 + v16) = 0;
  sub_140011610(&Block);
  if ( Block != v115 )
    j_j_free_0_3(Block);
  v19 = _mm_cvtsi32_si128(0x6C6C642Eu);         // lld.
  v20 = v102[0];
  strcpy((char *)v111, "FindFirstFileW");
  BYTE4(v109[1]) = 0;
  v109[0] = (void *)0x32336C656E72656Bi64;      // 23lenrek
  LODWORD(v109[1]) = _mm_cvtsi128_si32(v19);
  v21 = (__int64 (__fastcall *)(void *, void **))FunctionLibCall_thelib_function(v109, v111);
  HIDWORD(v86) = 0;
  v22 = v21(v20, &Block);
  if ( v22 != -1 )
  {
    v23 = _mm_cvtsi32_si128(0x2C002Cu);         // ,,
    v24 = _mm_cvtsi32_si128(0x656C6946u);       // eliF
    while ( 1 )
    {
      LODWORD(v106) = 70;
      v25 = v1 & 0xFFFFFFFFFFFFi64;
      BYTE4(v111[1]) = 0;
      strcpy((char *)v109, "lstrcmpW");
      LODWORD(v111[1]) = _mm_cvtsi128_si32(v19);
      LOWORD(v25) = 104;
      v111[0] = (void *)0x32336C656E72656Bi64;  // 23lenrek
      v1 = v25 & 0xFFFF00000000FFFFui64 | 0x6600470000i64;
      HIDWORD(v106) = v1 ^ 0x470046;
      LODWORD(v107) = WORD2(v1) ^ 0x48;
      v26 = (unsigned int (__fastcall *)(wchar_t *, char *))FunctionLibCall_thelib_function(v111, v109);
      if ( v26(String, (char *)&v106 + 4) )
      {
        v96 = 2;
        v95[3] = _mm_cvtsi128_si32(v23);
        v97 = 2;
        v98 = 2;
        LODWORD(v111[0]) = 2;
        BYTE4(v109[1]) = 0;
        strcpy((char *)v95, "lstrcmpW");
        v109[0] = (void *)0x32336C656E72656Bi64;// 23lenrek
        HIDWORD(v111[0]) = v95[3] ^ 0x20002;
```

```
LABEL_11:
  v101 = v20;
  *(_WORD *)((char *)v19 + v18) = 0;
  sub_14000DCB0(&Block);
  if ( Block != v10 )
    j_j_free_0_3(Block);
  v16 = v98;
  v10 = (__int64 *)v96;
  v21 = _mm_cvtsi32_si128(0x6C6C642Eu);
  v22 = v91;
  BYTE4(v96[1]) = 0;
  strcpy((char *)v98, "FindFirstFileW");
  v96[0] = (void *)0x32336C656E72656Bi64;       // 23lenrek
  LODWORD(v96[1]) = _mm_cvtsi128_si32(v21);
  v23 = (__int64 (__fastcall *)(void *, void **))FunctionLibCall_thelib_function(v96, v98);
  v15 = (void *)v23(v22, &Block);
  if ( v15 != (void *)-1i64 )
  {
    si128 = _mm_load_si128((const __m128i *)&xmmword_1400C9550);
    v25 = _mm_load_si128((const __m128i *)&xmmword_1400C9560);
    v17 = 0x4F00000000i64;
    while ( 1 )
    {
      v93 = (void *)0x2E00000061i64;
      strcpy((char *)v96, "lstrcmpW");
      BYTE4(v98[1]) = 0;
      v98[0] = (void *)0x32336C656E72656Bi64;   // 23lenrek
      LODWORD(v98[1]) = _mm_cvtsi128_si32(v21);
      v94[0] = 46;
      v27 = (unsigned int (__fastcall *)(wchar_t *, char *))FunctionLibCall_thelib_function(v98, v96);
```

```
LABEL_10:
  v96 = v14;
  *(_WORD *)((char *)v13 + v12) = 0;
  sub_1400157E0(&Block);
  if ( Block != v97 )
    j_j_free_0_3(Block);
  v15 = _mm_cvtsi32_si128(0x6C6C642Eu);
  v16 = v87;
  strcpy((char *)v93, "FindFirstFileW");
  BYTE4(v91[1]) = 0;
  v91[0] = (void *)0x32336C656E72656Bi64;       // kernel32.dll
  LODWORD(v91[1]) = _mm_cvtsi128_si32(v15);
  v17 = (__int64 (__fastcall *)(void *, void **))FunctionLibCall_thelib_function(v91, v93);
  v77 = 0;
  v18 = v17(v16, &Block);
  if ( v18 != -1 )
  {
    v19 = _mm_cvtsi32_si128(0x300030u);
    while ( 1 )
    {
      BYTE4(v93[1]) = 0;
      strcpy((char *)v91, "lstrcmpW");
      v93[0] = (void *)0x32336C656E72656Bi64;
      v80 = 0x2E0000002Ei64;
      LODWORD(v93[1]) = _mm_cvtsi128_si32(v15);
      v20 = (unsigned int (__fastcall *)(wchar_t *, __int64 *))FunctionLibCall_thelib_function(v93, v91);
```

The samples share the same function import obfuscation.

Obfuscation

```
LABEL_15:
      BYTE4(v109[1]) = 0;
      strcpy((char *)&v111[1] + 4, "W");
      v111[0] = (void *)0x7478654E646E6946i64;  // txeNdniF
      LODWORD(v111[1]) = _mm_cvtsi128_si32(v24);// v24=eliF
      v109[0] = (void *)0x32336C656E72656Bi64;  // 23lenrek
      LODWORD(v109[1]) = _mm_cvtsi128_si32(v19);
      v27 = (unsigned int (__fastcall *)(__int64, void **))FunctionLibCall_thelib_function(v109, v111);// call FindNextFileW from kernel32.dll
      if ( !v27(v22, &Block) )
      {
        BYTE4(v111[1]) = 0;
        v111[0] = (void *)0x32336C656E72656Bi64;// 23lenrek
        strcpy((char *)v109, "FindClose");
        LODWORD(v111[1]) = _mm_cvtsi128_si32(v19);
        v28 = (void (__fastcall *)(__int64))FunctionLibCall_thelib_function(v111, v109);
        v28(v22);
        goto LABEL_17;
      }
    }
```

The called function and library are stored reverse [::-1] and Base64 encoded.

Here, we have the example of how the obfuscated function « FindNextFileW » is called from « kernel32.dll »

This method can evade imports detection rules based as they are « rebuilt » during running.

Conclusion

- We can assess with good confidence that Lukalocker and Nitrogen are, at least, variants.
- Unless they seem to be different families, they are probably both operated by Volcano Demon

Contact

Sylvio HOARAU

sylvio.hoarau@glimps.re

Link to the article:

https://www.glimps.re/en/resource/nitrogen-correlation-with-lukalocker-cactus/

Find us at SSTIC 2025:

From Black Box to Clear Insights: Explainable AI in Malware Detection with MalConv2