국세청 개인 정보 유출 협박, 락빗(LockBit) 그룹의 LockBit Green 랜섬웨어

요약

1. 2019년 등장한 락빗(LockBit) 그룹

- 개발한 랜섬웨어를 공격자에게 판매하여 수입을 얻는 RaaS(Ransomware as a Service) 형태로 운영

2. 공격방식

- 입사지원서로 위장한 메일에 락빗 랜섬웨어 파일첨부 및 유포
- 파일명과 확장자 사이에 다수의 공백을 추가하고 문서 아이콘으로 위장해 의심없이 실행유도

3. 최근 발견 사례

- 23년 1월, 'LockBit Green 버전 빌더 페이지'가 새롭게 공개됨
- 랜섬웨어 실행 시 파일 복구가 불가능하도록 백업본을 삭제함
- 데이터베이스 파일은 전체 암호화, 가상머신 파일은 20%만 암호화 변조

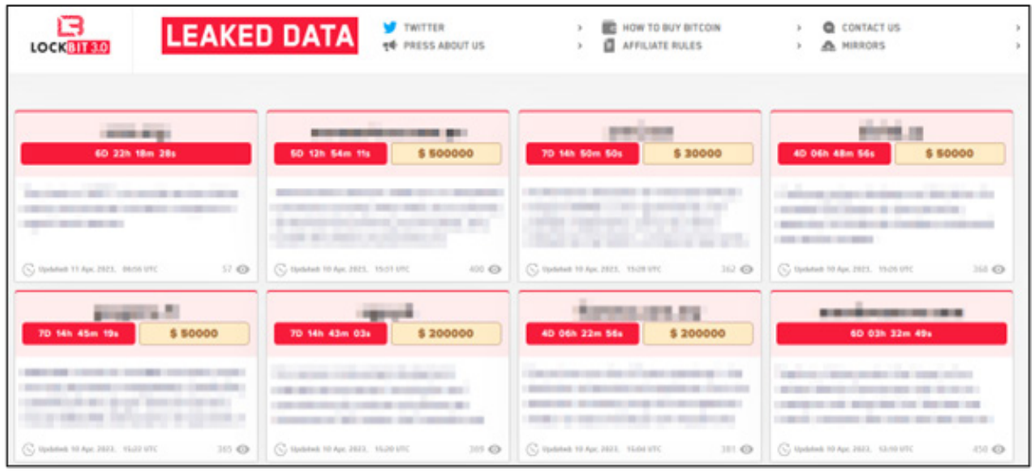
대응 방안

1. 논리적 망분리를 적용하여 악성코드 PC 유입을 원천 차단한다.
2. AV(패턴기반탐지) + EDR(행위기반탐지) 솔루션을 최신 형상으로 유지한다.
3. PC 취약점을 주기적으로 점검, 보완한다.
4. 신뢰할 수 없는 메일의 첨부파일은 실행을 금지한다.
5. 비 업무 사이트 및 신뢰할 수 없는 웹사이트의 연결을 차단한다.
6. OS나 어플리케이션은 최신 형상을 유지한다.

목차

1. 개요

1.1 배경



[그림 1] 락빗 랜섬웨어 그룹 다크웹 페이지

락빗(LockBit)은 2019년 등장한 랜섬웨어 그룹이다. 이들은 랜섬웨어를 개발하고 공격자들에게 판매하여 수입을 얻는 RaaS(Ransomware as a Service) 형태로 운영해 위험부담을 낮췄다. 피해 기업은 민간이 가장 많았으며 정부와 공공 또한 그 대상이 되기도 하였다. 락빗 다크웹 페이지에 피해 기업의 정보와 지불금액, 제한시간이 **[그림 1]**과 같이 게시된다.

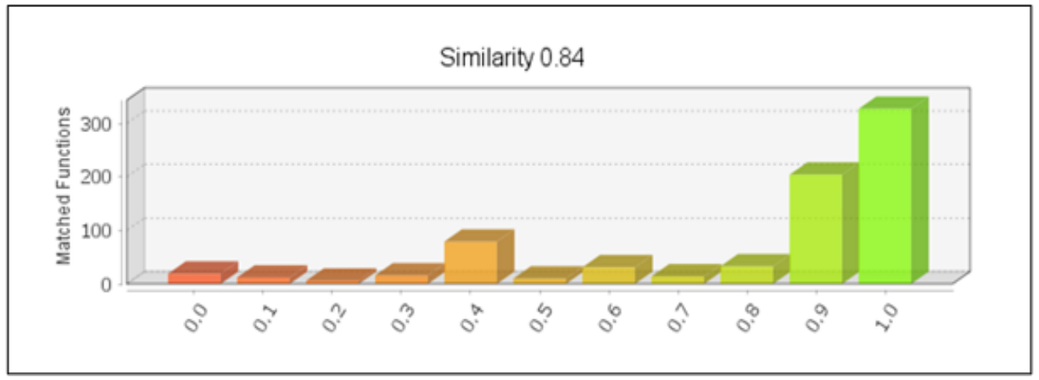
Builder (LockBit Black)	Builder (LockBit Green)
• Running one • Local disks • Network shares • Same encryption key • Maximum decryptor protection • Self-spread • Spread method • Delete event logs • Encrypt file name • Language check • Network shares encryption • Desktop wallpaper • Shut-down the system • Kill defender • Skip hidden folders • Encryption mode • Impersonation • Kill services • Kill processes • Print a note • Set icon • Self-delete • Wipe free space • White folders • White files • White extensions • White hosts • Processes to kill • Services to kill • Accounts for impersonations	• Running one • Local disks • Network shares • Same encryption key • Maximum decryptor protection • Select the number of builds • Select encryption size

[표 1] LockBit Black과 Green 빌더에서 지원하는 정책 비교

기존의 락빗 랜섬웨어 최신버전은 LockBit Black이었으나,
최근 트위터에서 LockBit Green 버전의 빌더[01] 페이지가 공개되었다.
Black 버전의 빌더[02] 와 비교했을 때 가장 눈에 띄는 차이점은 빌더 페이지에서 제공되는 정책의 개수이다.
Black 버전은 30개의 커스터마이징 요소를 제공하는 반면, Green 버전은 7개로 간결하다.

01 LockBit Green 버전의 빌더 페이지

02 Black 버전의 빌더 페이지



[그림 2] 콘티 랜섬웨어 3.0과의 유사도

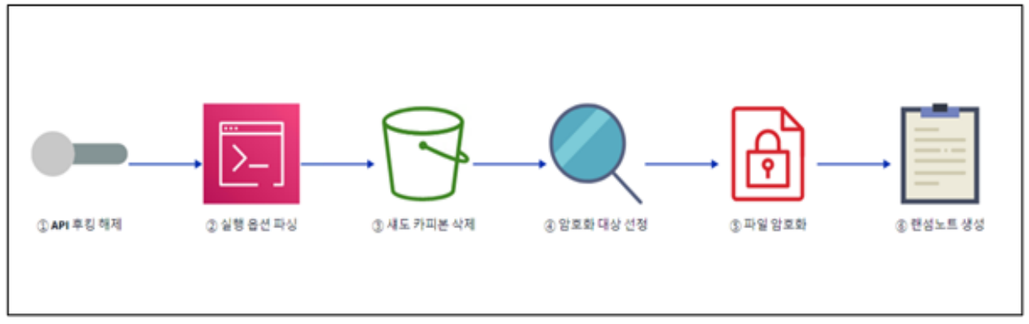
분석 과정에서 Green 버전이 작년에 유출된 콘티(Conti) 랜섬웨어 3.0 버전의 소스코드를 수정하여 작성했다는 점을 발견했다. Visual Studio 2022를 사용해 콘티 랜섬웨어 3.0 소스코드를 릴리즈 모드로 빌드하여 락빗과 실행파일을 비교해 본 결과, 84%의 유사도가 나왔다.
이는 락빗 랜섬웨어의 빌드 환경과 컴파일러 버전이 다르다는 것을 감안하면 매우 높은 유사도이며, 실제로 분석을 진행했을 때도 유사한 코드가 상당수 존재했다.
이전에 LockBit Black 버전이 블랙매터(BlackMatter) 랜섬웨어와 코드가 유사했다는 점을 감안하면 이번에 Green 버전이 콘티 랜섬웨어의 소스코드를 기반으로 작성한 것 역시 새로운 일은 아니다.

1.2 파일정보

Name	〈unknown〉
Type	PE32
SHA-256	45c317200e27e5c5692c59d06768ca2e7eeb446d6d495084f414d0f26 1f75315
Description	LockBit Green Ransomware

[표 2] LockBit Green 랜섬웨어 파일 정보

2. 분석



[그림 3] LockBit Green 랜섬웨어 실행 순서도

① API 후킹 해제

- DLL에 API 후킹이 적용되어 있다면 해제한다.

② 실행 옵션 파싱

- 락빗 랜섬웨어가 어떤 옵션과 함께 실행됐는지 확인하기 위해 명령줄에서 파싱한다.

③ 섀도 카피본 삭제

- 시스템을 원래대로 복구하지 못하도록 섀도 카피본을 삭제한다.

④ 암호화 대상 선정

- 파일 암호화 대상 경로를 수집한다. 특정 파일 확장자와 폴더는 암호화 대상에서 제외된다.

⑤ 파일 암호화

- 파일을 사용할 수 없도록 데이터를 암호화한다. 파일 암호화는 Chacha20 대칭키 암호 알고리즘으로 수행하며 대칭키와 Nonce는 하드코딩 된 RSA 공개키로 암호화된다.

⑥ 랜섬노트 생성

- 피해자에게 연락처를 제공하기 위해 랜섬노트를 생성한다.

2.1 API 리졸빙

```
pFunc = (DWORD *)g_ApiCache[CacheIndex];
for ( v2 = 2 - (_DWORD)pFunc + 4458501; !(v2 % 3); ++v2 )
  ;
if ( pFunc )
  return pFunc;
pFunc = GetProcAddressEx(0, ModuleId, hash);
```

[그림 4] 캐싱 여부 확인

락빗(LockBit Green) 랜섬웨어는 WinAPI를 사용할 때, API의 주소를 런타임에 계산하여 사용한다.
각 API는 최초 호출 시에만 주소를 계산하며, 두 번째 호출부터는 캐시에 저장된 주소를 사용한다.
캐시는 4,096 바이트를 할당하므로 최대 1,024개의 API 주소를 저장할 수 있다.

```
for ( j = 0; j < len; ++j )
{
  v13 = v23[j];
  if ( (unsigned __int8)(v13 - 65) <= 0x19u )
    v23[j] = v13 + 32;                          // API 이름에서 대문자는 소문자로 치환
}
v22 = len;
v14 = v23;
v15 = 0xB801FCDA;
if ( len >= 4 )
{
  len -= 4 * v7;
  do
  {
    v16 = 0x5BD1E995 * *(_DWORD *)v14;
    v14 += 4;
    v15 = (0x5BD1E995 * (v16 ^ HIBYTE(v16))) ^ (0x5BD1E995 * v15);
    --v7;
  }
  while ( v7 );
}
```

[그림 5] MurmurHash2A 해시 함수 의사코드

API의 주소를 계산할 땐 특정 DLL의 Export 테이블을 순회하면서 일치하는 함수명이 있는지 비교한다.
함수명을 소문자로 치환한 뒤 MurmurHash2A[03] 해시 함수로 계산하고,
하드코딩된 해시값과 일치하는 함수명이 있는지 비교한다.

03 MurmurHash2A 해시 함수

2.2 API 후킹 해제

```
kernel32_LoadLibraryA = GetProcAddressEx2(v3, 0xFu, 0x439C7E33u, 4);
(kernel32_LoadLibraryA)(v2);
kernel32_GetModuleFileNameW = GetProcAddressEx2(v5, 0xFu, 0xEBD9A617, 8);
kernel32_GetModuleFileNameW(hModule, hash, 260);
kernel32_CreateFileW = GetProcAddressEx2(v7, 0xFu, 0x17610E8u, 97);
v1 = (kernel32_CreateFileW)(hash, GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
if ( !v1 )
  return v1;
kernel32_GetFileSizeEx = GetProcAddressEx2(v10, 0xFu, 0x43BA1FFBu, 104);
(kernel32_GetFileSizeEx)(v1, &dllSize);
lodword = dllSize;
if ( !dllSize
  || (kernel32_CreateFileMappingW = GetProcAddressEx2(v13, 0xFu, 0x90B93FDA, 7),
      (v16 = (kernel32_CreateFileMappingW)(v1, 0, 2, 0, 0, 0)) == 0) )
{
ABEL_37:
  ProcAddressEx2 = GetProcAddressEx2(v13, 0xFu, 0xCA528872, 91);
  return ProcAddressEx2(v1);
}
kernel32_MapViewOfFile = GetProcAddressEx2(v13, 0xFu, 0xAB3C572u, 6);
v18 = (kernel32_MapViewOfFile)(v16, 4, 0, 0, lodword);
```

[그림 6] 후킹되지 않은 DLL 파일을 공유 메모리에 매핑

- kernel32.dll
- ws2_32.dll
- advapi32.dll
- rstrtmgr.dll
- ole32.dll
- netapi32.dll
- iphlpapi.dll
- shlwapi.dll
- shell32.dll
- ntdll.dll

DLL의 Export 테이블을 순회하며 함수의 프롤로그에 후킹이 설치되어 있는지 검사한다.
대상 DLL은 윈도우즈 기본 DLL로 위 목록과 같다. 또한, 락빗은 디스크에 존재하는 (후킹되지 않은)
위 DLL들을 각각 공유 메모리에 매핑시켜 추후 원본 코드를 패치하기 위한 용도로 활용한다.
매핑이 끝나면 기존에 로딩 되어있던 위 DLL들의 Export 테이블을 순회하며 후킹 여부를 검사한다.

```
if ( pFunc )
{
  v39 = *pFunc;
  if ( *pFunc == -23 || v39 == (char)0xFF && pFunc[1] == 0x25 )
  {
```

[그림 7] API 후킹 여부 검사

후킹 여부는 함수 프롤로그의 최대 2바이트를 보고 구분한다.
만약, 첫 1바이트가 E9로 시작하거나 또는 첫 2바이트가 FF 25로 시작하는 경우,
해당 API는 후킹되었다고 간주한다. 두 개의 코드 패턴은 코드 케이브로 분기하기 위한
JMP 명령어로 후킹 시 자주 활용된다.

```
kernel32_VirtualProtect = (BOOL (__cdecl *)(LPVOID, SIZE_T, DWORD, PDWORD))GetProcAddressE
                                                                            v37,
                                                                            0xFu,
                                                                            0xA7E8A5u,
                                                                            5);
v1 = (char *)kernel32_VirtualProtect((LPVOID)pFunc, 64, PAGE_EXECUTE_READWRITE, (PDWORD)&o
if ( !v1 )
  return v1;
*(_QWORD *)pFunc = *(_QWORD *)v33;
*(_WORD *)(pFunc + 8) = *(_WORD *)(v33 + 8);// 함수 프롤로그 10바이트만큼 복사
v44 = oldProtect;
kernel32_VirtualProtect_1 = (BOOL (__cdecl *)(LPVOID, SIZE_T, DWORD, PDWORD))GetProcAddres
                                                                              v43,
                                                                              0xFu,
                                                                              0xA7E8A5u,
                                                                              5);
v1 = (char *)kernel32_VirtualProtect_1((LPVOID)pFunc, 64, v44, (PDWORD)&v56);
if ( !v1 )
  return v1;
```

[그림 8] 원본 코드로 패치

후킹된 함수의 메모리 권한을 PAGE_EXECUTE_READWRITE로 수정해 원본 코드로 패치한다.
패치는 후킹되지 않은(공유 메모리에 매핑시켰던) DLL의 동일한 함수 첫 10바이트를 복사하여 이루어진다.


```
                ; void __stdcall GetNativeSystemInfoStub(LPSYSTEM_INFO lpSystemInfo)
                                public _GetNativeSystemInfoStub@4
_GetNativeSystemInfoStub@4 proc near    ; DATA XREF: .rdata:6B88274F↓o
                                        ; .rdata:off_6B892C98↓o

lpSystemInfo    = dword ptr  4

FF 25 B4 18+            jmp     ds:__imp__GetNativeSystemInfo@4 ; GetNativeSystemInfo(x)
88 6B       _GetNativeSystemInfoStub@4 endp
```

[그림 9] kernel32.dll의 GetNativeSystemInfo API 코드

여기서 문제가 있다. 락빗은 후킹되지 않은 DLL 파일을 공유 메모리에 매핑시킬 때
재배치를 수행하지 않기에, 패치 대상 함수의 첫 10바이트에 절대주소를 참조하는 코드가 있다면
유효하지 않은 메모리 주소로 분기하는 문제가 발생할 수 있다.
보통 DLL이 또다른 DLL로 포워딩해주는 경우가 여기에 해당한다.
Import 함수를 호출할 때 IAT(Import Address Table)를 절대주소로 참조하기 때문이다.

```
KERNEL32.DLL:76B51EB0
KERNEL32.DLL:76B51EB0             kernel32_GetNativeSystemInfo proc near
KERNEL32.DLL:76B51EB0 FF 25 B4 18+jmp     dword ptr ds:6B8818B4h
KERNEL32.DLL:76B51EB0 88 6B       kernel32_GetNativeSystemInfo endp
KERNEL32.DLL:76B51EB0                                       dword ptr ds:6B8818B4h=00000000
```

[그림 10] 잘못 패치된 API

잘못 패치된 API는 호출 시 유효하지 않은 메모리 주소로 분기하므로 크래시가 발생한다.
이 버그는 콘티 랜섬웨어의 소스코드에도 존재했으며,
추후에 발견될 샘플에서는 올바르게 수정되어 있을 확률이 매우 높다.

2.3 실행 옵션 파싱

옵션	설명
-p 〈PATH〉 (optional)	파일 암호화를 시작할 최상위 경로. 이 옵션을 사용하면 PATH_ENCRYPT 모드로 동작한다. 이 옵션은 -m 보다 우선 순위가 높다.
-m 〈MODE〉 (optional)	-p 옵션을 설정하지 않았다면 이 옵션으로 정해진 범위를 암호화할 수 있다. -m 옵션은 다음 네 가지 모드를 지원한다. • all (ALL_ENCRYPT) • local (LOCAL_ENCRYPT) • net (NETWORK_ENCRYPT) • backups (BACKUPS_ENCRYPT) 기본값은 NETWORK_ENCRYPT 모드다. BACKUPS_ENCRYPT 모드는 파일 암호화 없이 섀도 카피본만 삭제하는 옵션이다.
-log 〈PATH〉 (optional)	로그 파일 경로. 기본값은 로그 파일을 생성하지 않는 것이다.
-size 〈PERCENT〉 (deprecated)	파일 암호화 백분율. 락빗 랜섬웨어는 이 값을 50으로 고정했기 때문에 옵션 사용이 무의미하다. 이는 이전에 콘티 랜섬웨어에서 사용하던 옵션이다.
-nomutex (deprecated)	중복실행 허용 여부. 락빗 랜섬웨어는 이 값을 TRUE로 고정했기 때문에 옵션 사용이 무의미하다. 이는 이전에 콘티 랜섬웨어에서 사용하던 옵션이다.

[표 3] 락빗 랜섬웨어의 실행 옵션

[표 3]은 락빗 랜섬웨어에서 지원하는 옵션과 그에 대한 설명이다.
콘티 랜섬웨어의 소스코드를 가져다 썼기에 옵션 또한 비슷하다.
단, 락빗은 다섯 가지 중 두 가지 옵션을 더 이상 사용하지 않는다.

실행 모드	설명
ALL_ENCRYPT	LOCAL_ENCRYPT와 NETWORK_ENCRYPT를 모두 수행한다.
LOCAL_ENCRYPT	마운트된 모든 논리 디스크를 순회하며 암호화한다.
NETWORK_ENCRYPT	공유 폴더를 암호화한다.
PATH_ENCRYPT	지정된 최상위 경로부터 암호화한다.
BACKUPS_ENCRYPT	파일 암호화는 하지 않고 섀도 카피본만 삭제한다.

[표 4] 락빗 랜섬웨어의 실행 모드

락빗 랜섬웨어의 실행 모드에 대한 추가 설명은 **[표 4]**와 같다.

2.4 섀도 카피본 삭제

락빗을 포함한 대부분의 랜섬웨어는 피해자가 파일을 복구하지 못하도록
시스템에 존재하는 백업본을 삭제한다. 볼륨 섀도 카피 서비스(Volume Shadow Copy Service)는
윈도우즈의 대표적인 백업 기능이다.

```
SELECT * FROM Win32_ShadowCopy
```

[표 5] 섀도 카피본 쿼리 명령

WQL(WMI Query Language)을 사용해 감염PC에 존재하는 섀도 카피본 목록을 쿼리한다.
위와 같이 쿼리하면 섀도 카피본의 ID를 획득할 수 있다.

```
cmd.exe /c C:\Windows\System32\wbem\WMIC.exe shadowcopy where
"ID=<SHADOW_ID>" delete
```

[표 6] 섀도 카피본 삭제 명령 형식

CreateProcessW API를 호출해 위와 같은 형식으로 명령을 실행한다.
섀도 카피본들을 순회하며 앞서 구했던 ID를 채워넣어 삭제한다.
32비트 실행파일은 WoW64 파일시스템 리다이렉션에 의해 System32 폴더에 접근할 수 없으므로
락빗은 리다이렉션을 일시적으로 해제하고 위 명령을 실행한다.

2.5 암호화 대상 선정

실행 모드	최상위 경로
LOCAL_ENCRYPT	A부터 Z까지의 문자 중 마운트 된 모든 논리 디스크
NETWORK_ENCRYPT	특정 IPv4 주소 대역에 포함되는 SMB(445번 포트) 서버의 공유 폴더
PATH_ENCRYPT	실행 인자(-p 옵션)로 입력했던 경로

[표 7] 실행 모드에 따른 최상위 경로

락빗은 암호화 대상 파일 경로를 수집하기 위해 주어진 최상위 경로부터 최하위까지 파일을 재귀적으로 탐색한다. 이 때, 락빗의 실행 모드에 따라 재귀 탐색을 시작할 최상위 경로가 **[표 7]**과 같이 설정된다.

```
if ( g_EncryptMode == ALL_ENCRYPT )
  dwNumberOfProcessors = v48.dwNumberOfProcessors;// ALL_ENCRYPT 모드인 경우 스레드는
else
  dwNumberOfProcessors = 2 * v48.dwNumberOfProcessors;// ALL_ENCRYPT 모드가 아닌 경우
```

[그림 11] 실행 모드에 따른 스레드 개수 할당

파일 탐색과 암호화는 병렬적으로 수행한다. 감염PC에서 사용 가능한 스레드의 개수를 구하여
실행 모드가 ALL_ENCRYPT인 경우, LOCAL_ENCRYPT를 수행하는 스레드 풀과
NETWORK_ENCRYPT를 수행하는 스레드 풀에 각각 절반씩 할당한다.
ALL_ENCRYPT가 아닌 경우 단일 스레드 풀에 사용 가능한 스레드를 모두 할당한다.

tmp, winnt, temp, thumb, $Recycle.Bin, $RECYCLE.BIN, System Volume Information, Boot, Windows, Trend Micro, perflogs

[표 8] 파일 탐색에서 생략되는 폴더명

.exe, .dll, .lnk, .sys, .msi, !!!-Restore-My-Files-!!!.txt, CONTI_LOG.txt, .bat

[표 9] 파일 탐색에서 생략되는 파일 확장자

파일 탐색 시 특정 폴더와 파일 확장자는 생략된다.
일반적으로 운영체제 구동에 필요한 시스템 파일이나 파일의 크기가 커서
암호화 속도를 지연시키는 실행 파일, 그리고 랜섬웨어에 이미 감염된 파일 등이 여기에 포함된다.
[표 9]에 CONTI_LOG.txt가 포함된 것을 통해
락빗 랜섬웨어가 콘티 랜섬웨어의 소스코드를 수정하여 작성됐다는 것을 알 수 있다.

2.6 파일 암호화

조건	암호화 대상
1MiB 이하	전체 암호화.
1MiB 초과 5MiB 이하	앞의 1MiB만 암호화.
5MiB 초과	실행 옵션(-size)에서 설정한 백분율만큼 암호화. 락빗은 해당 옵션이 50으로 고정이기에 절반만큼 암호화한다.

[표 10] 파일 크기에 따른 암호화 정도

조건	암호화 대상
데이터베이스 파일	전체 암호화. 파일 확장자는 아래 표 참고.
가상머신 파일	20%만큼 암호화. 파일 확장자는 아래 표 참고.

[표 11] 파일 확장자에 따른 암호화 정도

락빗은 항상 파일의 모든 데이터를 암호화하지는 않는다.
파일의 확장자와 크기 등으로 대상의 종류를 구분하여 파일 암호화 정도의 백분율을 결정한다.

.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 12] 데이터베이스 파일 확장자

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

[표 13] 가상머신 파일 확장자

데이터베이스 또는 가상머신 관련 파일을 구분할 때 검사하는 파일 확장자는 위 표와 같다.

```
RmShutdown = GetProcAddressEx2(v28, 0x13u, 0xE7D62D41, 43);
result = RmShutdown(v32, RmForceShutdown, 0) == ERROR_SUCCESS;
j___free_base(v19);
```

[그림 12] 대상 파일을 사용 중인 프로세스 강제 종료

암호화를 위해 파일 핸들을 얻을 때 다른 프로세스가 이미 해당 파일을 사용 중
(FILE_SHARE_WRITE를 활성화하지 않았다고 가정)이라면 암호화할 수 없다.
따라서, 락빗은 ERROR_SHARING_VIOLATION 또는 ERROR_LOCK_VIOLATION이 발생한다면
해당 파일을 사용 중인 프로세스를 강제 종료한다.
강제 종료는 윈도우즈에서 제공하는 Restart Manager API를 통해 수행하며
파일 탐색기(explorer.exe) 프로세스는 종료 대상에서 제외된다.


```
0365F440  00 00 00 00 00 00 00 00  00 00 00 00 65 78 70 61  ............expa
0365F450  6E 64 20 33 32 2D 62 79  74 65 20 6B 6B 22 52 23  nd·32-byte·kk"R#
0365F460  45 8F 95 20 56 7B 1D 7B  90 36 CC 65 CB 70 8E 26  E..·V{.{.6.....&
0365F470  4E 27 DC 11 51 36 53 A6  66 77 DB 89 00 00 00 00  N'..Q6S.fwj·....
0365F480  00 00 00 00 E2 63 45 37  D0 8C A4 70 E2 63 45 37  .....7Ќ·.p...7
```

[그림 13] Chacha20 키 스트림

오프셋	크기	설명
+0	16 (0x10)	Chacha20 키 스트림에 항상 포함되는 문자열 상수
+16 (+0x10)	32 (0x20)	Chacha20 대칭키 (CryptGenRandom API에 의해 무작위로 생성됨)
+48 (+0x30)	8	Chacha20 카운터 값
+56 (+0x38)	8	Chacha20 Nonce 값 (CryptGenRandom API에 의해 무작위로 생성됨)

[표 14] Chacha20 키 스트림 구성

파일 암호화는 대칭키 암호 알고리즘 중 Chacha20을 사용한다.
16바이트 크기의 키와 8바이트 크기의 Nonce 값은 CryptGenRandom API를 사용해
매 파일마다 무작위로 생성한다. Chacha20 키 스트림 구성과 설명은 위의 [표 14]와 같다.

```
.data:00035790 ; BYTE g_PublicKey[4096]
.data:00035790 g_PublicKey     db 30h, 82h, 2, 22h, 30h, 0Dh, 6, 9, 2Ah, 86h, 48h, 86h
.data:00035790                                         ; DATA XREF: ThreadpoolHandler+5C↑o
.data:00035790                                         ; ThreadpoolHandler+E0↑o
.data:0003579C                 db 0F7h, 0Dh, 3 dup(1), 5, 0, 3, 82h, 2, 0Fh, 0, 30h, 82h
.data:000357AA                 db 2, 0Ah, 2, 82h, 2, 1, 0, 0D8h, 48h, 71h, 0E9h, 0BAh
.data:000357B6                 db 8Ah, 39h, 4Eh, 31h, 0A0h, 0E0h, 6Bh, 0CDh, 12h, 9Bh
.data:000357C0                 db 0DEh, 49h, 0AAh, 76h, 65h, 0E8h, 44h, 0F0h, 6Eh, 98h
.data:000357CA                 db 73h, 0CBh, 0C7h, 0B3h, 90h, 48h, 38h, 3Bh, 0D7h, 3Dh
.data:000357D4                 db 77h, 65h, 0AAh, 8Ch, 0EDh, 57h, 73h, 0CFh, 64h, 5Fh
.data:000357DE                 db 74h, 0DCh, 29h, 0DFh, 3Ah, 50h, 45h, 4Bh, 8Eh, 71h
```

[그림 14] 대칭키를 암호화할 때 사용되는 RSA 공개키

파일을 암호화할 때마다 대칭키가 다르기 때문에, 무작위로 생성된 키와 Nonce 값을 저장해두지 않으면 이후 복호화가 불가능하다. 락빗은 하드코딩된 RSA 공개키를 가지고 Chacha20 키와 Nonce 값을 암호화한 뒤 감염된 파일의 끝 부분에 저장한다.

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
0000D2B0	A3	7D	96	23	6C	82	11	5E	66	38	96	B2	98	F0	7F	EC	£}–#l‚.^f8–²˜ð.ì
0000D2C0	87	CE	D6	DD	34	EE	1E	94	80	DD	1E	84	5A	E2	81	F0	‡ÎÖÝ4î.”€Ý.„Zâ.ð
0000D2D0	85	C8	2A	53	0F	44	90	8F	9E	9C	1C	24	4E	DA	FA	7B	…È*S.D..žœ.$NÚú{
0000D2E0	72	72	4B	C3	6E	F0	3F	28	27	62	6D	77	71	27	B9	7C	rrKÃnð?('bmwq'¹|
0000D2F0	77	0F	9F	BC	E4	42	09	10	76	D5	EE	EB	03	4D	59	E2	w.Ÿ¼äB..vÕîë.MYâ
0000D300	7B	3F	D8	9B	D5	5A	B1	80	78	3D	C5	1A	16	FD	96	F9	{?Ø›ÕZ±€x=Å..ý–ù
0000D310	71	A1	D1	BF	46	05	09	A0	5B	5F	DD	C7	AD	67	2A	ED	q¡Ñ¿F.. [_ÝÇ.g*í
0000D320	08	B1	36	1B	DA	BF	DF	9C	A4	21	FA	C7	6C	D4	EE	A4	.±6.Ú¿ßœ¤!úÇlÔî¤
0000D330	B6	E6	37	1F	1F	CC	11	29	A5	AC	DB	D7	77	37	C2	E3	¶æ7..Ì.)¥¬Û×w7Âã
0000D340	D0	0B	86	DA	4E	04	AC	31	48	5B	1F	D4	5A	D8	CF	10	Ð.†ÚN.¬1H[.ÔZØÏ.
0000D350	77	82	B4	4D	3A	AB	12	D2	28	4C	B8	B9	AC	9F	48	D1	w‚´M:«.Ò(L¸¹¬ŸHÑ
0000D360	09	94	0F	75	75	9D	76	BF	5B	3E	8D	41	E2	5E	E9	A6	.”.uu.v¿[>.Aâ^é¦
0000D370	00	71	86	BA	F7	37	7D	3D	C1	39	55	9A	A4	95	AE	CB	.q†º÷7}=Á9Uš¤•®Ë
0000D380	DD	38	14	93	1C	13	3A	B0	B1	F9	24	FA	A7	62	8E	0A	Ý8.“..:°±ù$ú§bŽ.
0000D390	68	5A	C9	F7	69	23	A8	DC	55	F3	81	0A	8D	6D	0A	36	hZÉ÷i#¨ÜUó...m.6
0000D3A0	FC	4A	D4	5F	AD	B0	66	07	0B	81	40	73	20	A3	5A	F9	üJÔ_.°f...@s £Zù
0000D3B0	DA	BD	C8	BC	D2	FD	DF	91	FC	97	78	E2	18	96	3F	AB	Ú½È¼Òýßˈüxâ.–?«
0000D3C0	2B	B0	FF	E3	56	36	25	D9	A1	9A	7F	0A	4C	5A	38	13	+°ÿãV6%Ù¡š..LZ8.
0000D3D0	4D	28	71	8D	F1	B6	E3	77	21	36	98	76	14	73	A5	44	M(q.ñ¶ãw!6˜v.s¥D
0000D3E0	5A	60	99	16	3A	00	B3	9D	86	74	2C	B6	CA	A5	FE	1F	Z`™.:.³.†t,¶Ê¥þ.
0000D3F0	28	D9	AF	1A	22	C5	09	87	5F	2C	93	11	D0	07	18	77	(Ù¯."Å.‡_,“.Ð..w
0000D400	D9	33	6F	47	F5	78	48	1F	F0	44	4C	45	A4	A2	0C	D0	Ù3oGõxH.ðDLE¤¢.Ð
0000D410	FA	E3	E7	78	2B	DC	BE	91	5E	6A	22	FE	BD	FD	2C	88	úãçx+Ü¾‘^j"þ½ý,ˆ
0000D420	85	1A	50	76	89	B0	4A	93	A4	03	0A	89	5E	5B	8B	64	….Pv‰°J“¤..‰^[‹d
0000D430	3F	A8	20	4B	C1	DB	61	6B	3E	E3	96	4E	02	55	A1	E2	?¨ KÁÛak>ã–N.U¡â
0000D440	BA	25	E8	12	86	99	D4	6F	02	3A	CE	73	CB	5F	96	5B	º%è.†™Ôo.:ÎsË_–[
0000D450	26	9A	DF	F6	29	12	A4	A0	26	EA	09	0F	7F	AC	E2	B6	&šßö).¤ &ê...¬â¶
0000D460	A5	F3	EC	7F	52	76	45	01	9F	71	41	2D	7B	B1	31	CF	¥óì.RvE.ŸqA-{±1Ï
0000D470	15	04	1A	A7	EE	CE	14	27	DE	9A	41	CF	D0	55	FC	50	...§îÎ.'ÞšAÏÐUüP
0000D480	3B	2D	09	44	F7	C4	A7	63	16	09	22	CC	33	EF	EC	99	;-.D÷Ä§c.."Ì3ïì™
0000D490	86	8E	2D	BE	08	EE	E7	C9	89	85	F2	2E	E9	9C	0C	6D	†Ž-¾.îçÉ‰…ò.éœ.m
0000D4A0	DD	7B	44	AF	BD	53	0D	86	DD	08	D5	DB	D3	97	92	47	Ý{D¯½S.†Ý.ÕÛÓ—’G
0000D4B0	0F	5F	8C	85	7B	48	91	FE	DF	72	7F	8B	4D	62	A0	79	._Œ…{H‘þßr.‹Mb y
0000D4C0	7F	1E	42	34	70	23	98	B5	55	64	51	4D	01	FF	AE	FA	..B4p#˜µUdQM.ÿ®ú
0000D4D0	87	2D	65	CE	FE	C1	C4	83	F6	9A	56	86	62	42	00	00	‡-eÎþÁÄƒöšV†bB..
0000D4E0	00	00	00	00	00	00	00	00	00	00	24	00	DE	D2	00	00	..........$.ÞÒ..
0000D4F0	00	00	00	00													....

[그림 15] 감염 파일 메타데이터

오프셋	크기	설명
+0	524 (0x20C)	RSA 공개키로 암호화된 Chacha20 키와 Nonce 값
+524 (+0x20C)	1	파일 암호화 모드 • FULL_ENCRYPT = 36 (0x24) • PARTLY_ENCRYPT = 37 (0x25) • HEADER_ENCRYPT = 38 (0x26)
+525 (+0x20D)	1	파일 암호화 백분율
+526 (+0x20E)	8	원본 파일 크기

[표 15] 감염 파일 메타데이터

감염 파일의 끝부분에 추가되는 메타데이터 형식은 위와 같다.
파일 암호화 모드의 FULL_ENCRYPT는 데이터 전체를 암호화,
PARTLY_ENCRYPT는 설정된 백분율만큼 암호화,
그리고 HEADER_ENCRYPT는 앞의 1MiB만 암호화하는 것을 말한다.
백분율 필드는 암호화 모드가 PARTLY_ENCRYPT일 때만 유효한 값이다.

2.7 랜섬노트 생성

```
.data:000368E0 g_RansomNoteSize dd 2B40h                ; DATA XREF: DecryptNote+1E2↑r
.data:000368E4 g_RansomNoteContent db 95h, 5Dh, 29h, 71h, 66h, 10h, 0E3h, 15h, 2, 0B5h, 20h
.data:000368E4                                          ; DATA XREF: DecryptNote+29E↑o
.data:000368EF                  db 3Ch, 9Bh, 96h, 0CBh, 52h, 2 dup(79h), 1Dh, 7Fh, 0F6h
.data:000368F9                  db 1Fh, 0B3h, 44h, 16h, 34h, 0E5h, 3Dh, 0C5h, 65h, 14h
.data:00036903                  db 0CEh, 0D5h, 0F2h, 7Fh, 0D5h, 9Ah, 57h, 3Eh, 6Dh, 5
.data:0003690D                  db 96h, 12h, 14h, 66h, 0EFh, 70h, 0AFh, 0DDh, 13h, 0B1h
.data:00036917                  db 0A2h, 66h, 5Dh, 79h, 32h, 88h, 69h, 65h, 0EDh, 0A2h
.data:00036921                  db 9Ah, 2 dup(42h), 56h, 5Ah, 55h, 0E5h, 16h, 13h, 51h
.data:0003692B                  db 4Bh, 9Bh, 0B1h, 72h, 4Dh, 96h, 3Ah, 71h, 0E4h, 2Ah
.data:00036935                  db 5, 95h, 52h, 2, 37h, 67h, 0C0h, 0BBh, 0F5h, 8Ah, 0F8h
.data:00036940                  db 38h, 0E1h, 0A5h, 0D1h, 0B5h, 0B9h, 0CBh, 68h, 0B3h
```

[그림 16] 암호화된 랜섬노트 본문

C6 ED 38 3C 61 A6 05 5F 91 A9 01 E9 5E 8D 20 9C FC 1B 8F A6 5B 20 91 20 11 54 F1 8D 7C 7F 35 E8

[표 16] 랜섬노트 복호화에 사용되는 AES-256 키

D0 AC FF 07 F8 2B 52 0A 76 ED 61 29 E1 CF B8 E2

[표 17] 랜섬노트 복호화에 사용되는 AES-256 IV

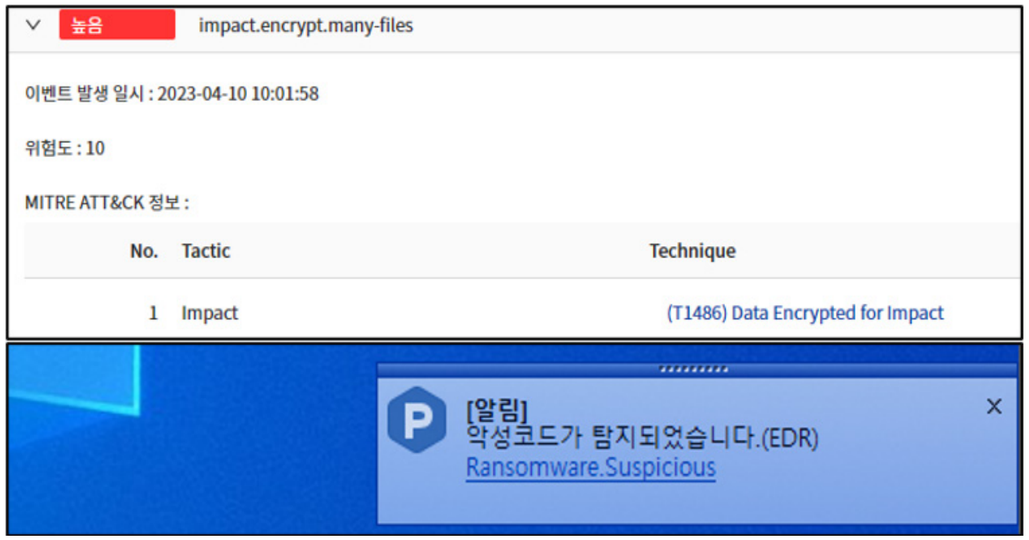
랜섬노트는 실행파일에 AES 알고리즘에 의해 암호화된 채로 하드코딩 되어있다.
마찬가지로 하드코딩 된 AES-256의 키와 IV를 사용해 랜섬노트를 복호화한다.

```
http://lockbitsupt7nr3fa6e7xyb73lk6bw6rcneqhoyblniiabj4uwvzapqd.onion
http://lockbitsupuhswh4izvoucoxsbnotkmgq6durg7kficg6u33zfvq3oyd.onion
http://lockbitsupxcjntihbmat4rrh7ktowips2qzywh6zer5r3xafhviyhqd.onion

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>> Your personal ID: [redacted] <<<<<
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

>>>>> Warning! Do not delete or modify encrypted files, it will lead to problems with decrypt

>>>>> Don't go to the police or the FBI for help and don't tell anyone that we attacked you.
```

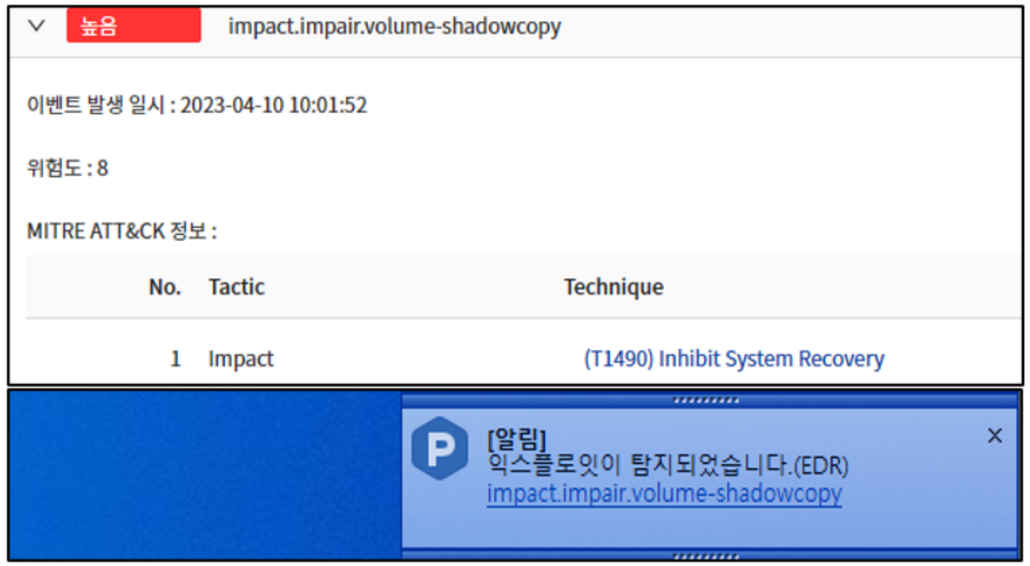
[그림 17] 복호화된 랜섬노트 본문

복호화된 랜섬노트는 **[그림 17]**과 같다. 랜섬노트엔 피해자를 식별하기 위한 ID도 함께 포함되어 있다.

3. Privacy-i EDR 탐지 정보



[그림 18] Privacy-i EDR 파일 암호화 탐지



[그림 19] Privacy-i EDR 섀도 카피본 삭제 탐지

Privacy-i EDR은 행위 기반 탐지 엔진을 통해 LockBit Green 랜섬웨어의 파일 암호화와 볼륨 섀도 카피본 삭제 행위를 위와 같이 탐지하고 차단하였다.

4. 대응

1. 논리적 망분리를 적용하여 악성코드 PC 유입을 원천 차단한다
2. AV(패턴기반탐지) + EDR(행위기반탐지) 솔루션을 최신 형상으로 유지한다.
3. PC 취약점을 주기적으로 점검, 보완한다.
4. 신뢰할 수 없는 메일의 첨부파일은 실행을 금지한다.
5. 비 업무 사이트 및 신뢰할 수 없는 웹사이트의 연결을 차단한다.
6. OS나 어플리케이션은 최신 형상을 유지한다.

5. 별첨

5.1 국내 피해 현황

일시	기업명	피해 규모
2023.03.30	대한민국 국세청	락빗이 탈취한 정보를 공개하겠다고 밝혔으나 국세청은 공식적인 피해가 없다고 밝힘,
2022.09.29	모 대기업	22년 7월 정보 복구비용으로 345 비트코인 요구 (한화 약 100억원) 22년 8월 탈취 데이터 공개로 1,350만 달러 재요구 (한화 약 180억원) 협상 결렬로 데이터 공개 (주간보고서, 임원회의록, 사업계획서 등 2TB 크기의 데이터)

5.2 국제 피해 현황

일시	기업명	피해 규모
2022.12.12	미국 캘리포니아 주 재무부	기밀 데이터, 재무문서, 재판 소송 관련 문서 등 76GB 크기의 데이터

5.3. 참고 자료

LockBit 랜섬웨어 다크웹
hxxp://lockbitapt72iw55njgnqpymggskg5yp75ry7rirtdg4m7i42artsbqd.onion

궁금하신 점이나 문의사항은 malware@somansa.com 으로 문의주십시오