시나리오 기반의 Blind SSRF 위협 및 대응 방안 분석

정준영, 송종근*
동서대학교 정보보안학과, *동서대학교 정보보안학과

nagara1126@naver.com, *ssongjg@dongseo.ac.kr

Scenario-Based Analysis of Blind SSRF Threats and Mitigation Srategies

JEONG JUN YEONG, SONG JONG GEUN*

Dongseo Univ , *Dongseo Univ

요 약

본 논문은 Blind SSRF 공격의 대표적 시나리오를 설명하고, Docker 기반 격리 실습으로 Blind SSRF 을 재현 및 분석한다. 기존 대응 방안은 내부 대역 차단 · 리졸브 · 리다리렉트 검증과 같은 다양한 다단계 검증을 권하지만, 검증이 늘어날수록 관리 부담과 오탐이 커지는 문제가 있다. 이를 보완하기 위해 핵심 민감 대역과 비허용 프로토콜을 최소한의 하드 검증으로 고정하고, 그 외 요청에 대해서는 URL/도메인 특성을 기반으로 하여 머신러닝 기반 분류기로 실시간 탐지 방안을 제시한다.

Ⅰ. 서 론

서버 측 요청 위조 공격 (Server-Side Request Forgery, SSRF)은 웹 애플리케이션 보안에서 현실적이고 심각한 위협으로 자리 잡았으며, 특히 내부 네트워크나 클라우드 메타데이터 같은 보호된 자원에 무단으로 접근할 수 있다는 점에서 그 위험성이 매우 크다.[1] SSRF 의 다양한 변형 중 Blind SSRF 는 서버가 공격자가 제공한 URL 을 호출하지만 그 결과가 공격자에게 직접 반환되지 않는 유형으로, 공격자는 DNS/HTTP 콜백 또는 타임스탬프와 같은 OOB(Out-of-Band)을 통해 공격의 성공 여부를 판단하고 추가 공격을 시도할 수 있다. Blind SSRF 는 단방향 공격 특성을 가지고 있으며 완전히 알려진 SSRF 공격보다 피해가 낮은 경우가 많다. Backend 시스템에서 민감한 데이터를 추출하는 데 악용될 수는 없지만, 경우에 따라 완전한 원격 코드 실행을 위해 악용될 수 있다.[2]

본 연구에서는 Docker 기반의 격리된 실습 환경을 구축하여 Blind SSRF 공격의 발생 과정을 단계별로 분석한다. 피해자 서버는 Flask 와 Gunicorn 으로 구현된 웹 애플리케이션으로 구성하고, 공격자 서버는 interactsh 와 Log Collector 을 사용하여 OOB(Out-of-Band) 통신을 실시간으로 탐지한다.

실습을 통해 실제적인 보안 위험을 평가하고, 기존의 대응방안의 한계를 심층적 분석한다. 나아가 본 연구는 블랙리스트 및 다단계 검증의 한계점을 보완하기 위한 머신러닝 기반 실시간 URL/도메인 판별 구조를 제안하며, 불필요한 차단을 최소화하면서 동시에 탐지 효율성을 개선하는 대응 방안을 제시한다.

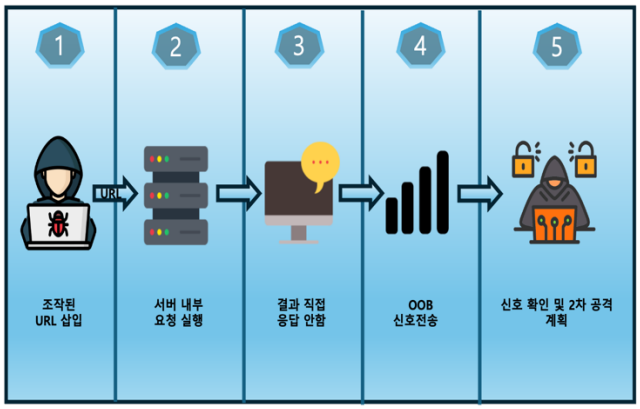

그림 1. Blind SSRF 공격 시나리오

Ⅱ. 본론

2.1 공격 시나리오 설명

Blind SSRF 공격은 취약한 웹 애플리케이션을 대상으로 진행된다. 공격자는 URL 입력란에 내부망이나 클라우드 메타데이터를 목표로 하면서 동시에 자신이 제어하는 도메인으로의 간접 통신을 유도하는 조작된 URL 을 삽입한다. 서버는 해당 URL 을 통해 내부 요청을 실행하지만, 그 결과를 공격자에게 직접적으로 반환하지 않는다. 대신 요청 처리 중에 DNS 쿼리나 HTTP 콜백과 같은 OOB 신호를 공격자 서버로 보낸다. 공격자는 자신의 로그에서 이러한 신호를 확인해 Blind SSRF 성공 여부를 간접적으로 판단한다.

도구	실습에 사용된 기능	버전
Docker	피해자 컨테이너 · 공격자 컨테이너 실행 및 격리 환경 구성	28.0.1
Docker Compose	다중 컨테이너 구성	V2.33.1-desktop.1
Python Flask	취약 웹 애플리케이션 구현	3.1.2
Gunicorn	로그 및 워크러 생성 관찰	23.0.0
Attacker(Log Cllector)	공격자 역할의 HTTP 서버/로그 수집기	3.1.2
Interactsh-client (ProjectDiscovery)	OOB 상호작용 캡쳐(DNS/HTTP)	1.2.4

표 1. 실습에 활용된 도구

2.2 Docker 를 활용한 실습 환경 구축

본 연구에서는 Docker 및 Docker Compose 를 활용하여 Blind SSRF 취약성 실증을 위한 로컬 격리 실습 환경을 구축한다. 이 환경의 핵심인 피해자 서버는 Python Flask 를 기반으로 개발되며, Gunicorn 으로 WSGI 를 처리하여 호스트 8080 포트에 매핑한다. 특히, 외부 HTTP 호출 기능은 requests 라이브러리를 사용하도록 의도적으로 구현하는데, 이는 요청 시 자동으로 포함되는 User-Agent 헤더를 통해 서버의 내부 기술 스택 정보가 유출될 수 있음을 증명하기 위함이다. 이 서버는 URL 입력 시 즉시 응답을 반환하고 외부 호출은 Backend 로 수행하여 Blind 환경을 구성한다. OOB 콜백을 수신 및 기록하는 리스너로는 전문 도구인 interactsh 를 활용하여[3], 피해자 서버로부터 발생하는 DNS 쿼리 및 HTTP 콜백을 탐지하고 실험의 증거로 활용한다.

2.3 Blind SSRF 실습 과정

공격자는 외부로부터의 통신을 탐지하기 위해 OOB(Out-of-Band) 리스너인 interactsh 를 실행하고 고유한 외부 도메인 주소를 확보한다.[3]

확보된 도메인 주소를 Docker 로 실행된 Python Flask 기반의 취약한 웹 애플리케이션의 URL 입력 파라미터에 삽입하고 서버로 전송한다.

피해자 서버는 공격자가 요청한 URL 을 수신한 뒤 사용자에게 즉시 정상적인 응답을 반환한다. 실제 외부로의 통신은 서버의 내부 로직에 따라 Backend 에서 별도로 처리되기에, 공격자는 이 단계에서 직접적인 결과를 확인할 수 없다. 서버가 Backend 에서 외부 도메인의 IP 주소를 확인하는 과정에서 DNS 쿼리가 발생한다.

DNS 쿼리가 바로 공격자의 interactsh 리스너로 전송되는 핵심적인 외부 신호 역할을 한다. 공격자는 자신의 interactsh 실시간 로그를 확인하여 피해자 서버로부터 발생한 DNS 상호작용 기록을 찾아낸다. 특히, 함께 수신된 HTTP 콜백의 User-Agent 헤더를 통해 서버가 내부적으로 requests 라이브러리를 사용한다는 정보가 유출되었음을 확인하며 공격을 성공시킨다.

2.4 위협 및 대응방안

본 실습에서 공격자는 로그에 기록된 타임스탬프와 헤더 정보를 분석하여 Blind SSRF 취약점이 존재를 증명한다. 이번 실습에서 공격을 받아 유출된 HTTP 헤더의 User-Agent 정보(python-requests/2.32.5)를 통해 피해자 서버가 특정 HTTP 클라이언트 라이브러리를 사용하고 있다는 내부 정보를 파악하였다.

이에 대한 현재 OWASP 에서 권하는 현실적 대응방법으로는 단일 기법이 아닌, 블랙리스트 내부 민감 대역을 확실히 차단하고, DNS 리졸브 · 리다이렉트 검증 · 프로토콜 제한 · 토큰 기반 적법성 검증 같은 다단계 검증 흐름을 결합하는 것을 권고하고 있다.[4] 다단계 검증은 필요하지만, 요소가 늘어날수록 운영 복잡도와 오탐이 커진다. 본 연구에서 제안할 대응방안은 내부 대역 · 비허용 프로토콜을 최소 하드 룰로 묶음 처리하고, 나머지 요청에 대해서는 URL 과 도메인을 머신러닝 기반 분류기로 허용할 요청을 실시간으로 판단하는 방식을 제안한다.

이는 고비용 평판 · 콘텐츠 검증은 비동기로 분리해 지연을 억제할 수 있다. 제시한 대응 방안은 새로운 패턴에 더 빨리 적응하면서 회색 영역(Gray area) 트래픽을 세분화해 불필요한 차단을 줄일 수 있는 장점이 있다. 하지만 보완해야 할 사항은 회피 기법과 데이터 드리프트에 대비한 주기적 재학습 · 앙상블이 필요하며, PII 최소화 및 예외 처리 절차를 함께 운용해야 하는 보완사항이 존재한다.

Ⅲ. 결론

Blind SSRF 는 Blind 상황에서도 OOB 통신을 이용하여 로그 · 헤더 분석을 통해 내부 정보 유출을 유발할 수 있음을 실습으로 입증했다. 다단계 규칙 기반 대응은 필요하지만, 요소가 늘어날수록 운영 복잡도와 오탐이 커진다. 제안한 최소 하드룰, 경량 분류기, 비동기 심층 검증을 통해 운영 부담은 줄이면서도 동적 위협에 대응할 수 있는 실무적 대안이 된다. 향후 공개 피드와 합성 시나리오를 이용한 오프라인 평가 및 섀도 테스트로 제안안의 정략적 성능을 검증할 필요가 있다.

참 고 문 헌

[1] S.-H. Moon and S.-H. Han, "Differences between XSS vs CSRF vs SSRF," Proceedings of the Korea Institute of Information and Communication Sciences (KICS) Conference, pp. 329–331, 2024.

[2] J. Kettle, "Cracking the lens: targeting HTTP's hidden attack-surface," PortSwigger Research, 2017, (https://portswigger.net/research/cracking-the-lens-targeting-https-hidden-attack-surface#remoteclient).

[3] ProjectDiscovery, "interactsh – Open Source Out-of-Band Interaction Server," GitHub Repository, 2025, (https://github.com/projectdiscovery/interactsh).

[4] OWASP Foundation, "Server-Side Request Forgery Prevention Cheat Sheet," OWASP Cheat Sheet Series, 2024, (https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html).