

칼리 리눅스와 모의해킹 입문

도구 지도, 절차, 방어 관점, 파이썬 객체지향까지 연결한 확장 보고서

기초 자료: 260625 - 모의해킹 메모

구성 범위: Kali Linux 개념, 모의해킹 절차, 핵심 도구 21 종, 무선 보안, Python 객체지향 및 자동화 설계

작성일	2026-06-25
작성 목적	흩어진 수업 메모를 학습용 보고서와 읽을거리로 재구성
대상 독자	칼리 리눅스와 모의해킹, Python OOP를 함께 배우는 입문자
사용 범위	본인 소유 또는 명시적으로 허가된 실습/업무 환경에 한정

머리말

이 문서는 원문 메모의 빠른 요약을 넘어서, 모의해킹을 처음 접하는 사람이 “도구 이름을 외우는 단계”에서 “전체 절차를 이해하고 보고서로 말할 수 있는 단계”로 넘어가도록 구성했다. 칼리 리눅스와 보안 도구는 강력하지만, 도구 자체가 목적은 아니다. 실제 모의해킹의 가치는 허가된 범위 안에서 증거를 수집하고, 영향도를 설명하며, 방어자가 실행할 수 있는 개선안을 남기는 데 있다.

원문의 방향성인 “자주 쓰이는 툴 정리”와 “Python으로 자동화된 보안 점검 프로그램을 만들기”를 살리되, 공격 자동화가 아닌 합법적 진단 자동화라는 관점으로 재정리했다. 각 도구는 무엇을 확인하는지, 어떤 산출물을 남기는지, 방어 관점에서는 무엇을 봐야 하는지를 함께 설명한다.

사용 전제

모든 스캔, 진단, 실습은 본인 소유 시스템 또는 명시적으로 허가받은 테스트 범위에서만 수행한다.

민감한 도구는 계정 잠금, 서비스 장애, 개인정보 노출, 법적 책임으로 이어질 수 있으므로 실습 랩과 승인된 업무 외에는 실행하지 않는다.

목차

1. 1. 전체 요약 - 모의해킹은 공격이 아니라 검증과 보고의 과정이다
2. 2. 칼리 리눅스 이해 - 해킹툴 모음 이상의 의미
3. 3. 모의해킹 라이프사이클 - 대상 선정부터 문서화까지
4. 4. 단계별 핵심 도구 지도 - Nmap부터 Chisel/Ligolo-ng 까지
5. 5. 무선 보안과 패킷 주입 - Aircrack-ng 를 이해하는 안전한 방식
6. 6. Python 객체지향 핵심 - self, dunder, property, static, slots
7. 7. Python으로 설계하는 안전한 보안 점검 프레임워크
8. 8. 보고서 작성법 - 발견 사항을 방어 가능한 개선안으로 바꾸기
9. 9. 학습 로드맵과 체크리스트
10. 부록 A. 용어 사전 / 부록 B. 참고자료

CHAPTER 1

전체 요약

흩어진 도구 목록을 하나의 사고 체계로 묶기

모의해킹을 처음 배우면 도구 이름이 먼저 눈에 들어온다. Nmap, Wireshark, Burp Suite, Metasploit, Hydra, SQLmap 처럼 강력한 도구는 많다. 그러나 도구를 아는 것과 모의해킹을 아는 것은 다르다. 모의해킹은 “어떻게 뚫는가”만 묻는 일이 아니라, “무엇을 테스트할 권한이 있는가”, “무엇이 실제 위험인가”, “증거는 충분한가”, “어떻게 고쳐야 하는가”를 끝까지 추적하는 평가 활동이다.

한 문장 정의

모의해킹은 허가된 범위 안에서 공격자의 관점으로 시스템을 검증하고, 그 결과를 방어자가 실행 가능한 개선안으로 번역하는 보안 평가 절차다.

이 보고서의 핵심 메시지

- **도구보다 절차가 먼저다:** 대상 선정, 정보 수집, 취약점 분석, 검증, 영향 평가, 방어 분석, 문서화가 하나의 흐름을 이룬다.
- **스캔 결과는 결론이 아니다:** 열린 포트, 배너, 취약점 후보는 “가설”이다. 실제 영향도와 재현 가능성, 오탐 여부를 함께 평가해야 한다.
- **공격 단계는 항상 최소화한다:** 권한 탈취나 데이터 접근은 목적이 아니라 위험을 증명하기 위한 제한적 수단이다. 업무에서는 사전 승인과 중지 조건이 필요하다.
- **Python 자동화는 공격 자동화가 아니라 점검 자동화여야 한다:** 범위 검증, 속도 제한, 로깅, 예외 처리, 리포팅을 갖춘 안전한 도구 구조가 중요하다.
- **보고서가 최종 산출물이다:** 좋은 모의해킹은 “취약했다”가 아니라 “무엇이 왜 위험하고, 어떻게 우선순위를 두고 고칠지”를 남긴다.

도구를 배우는 순서

초보 단계에서는 “한 번에 많은 도구”보다 “각 단계에서 대표 도구 하나를 제대로 이해하는 것”이 더 효율적이다. 예를 들어 네트워크 정보 수집은 Nmap, 패킷 이해는 Wireshark, 웹 프록시는 Burp Suite, 웹 서버 기본 점검은 Nikto, 디렉토리 탐색은 Gobuster 또는 FFUF, 보고서 자동화는 Python 으로 시작하면 된다.

학습 축	대표 질문	대표 도구/개념
네트워크	어떤 호스트와 포트가 살아 있는가?	Nmap, Wireshark, TCP/IP, DNS
웹	요청/응답 흐름과 입력값 검증은 안전한가?	Burp Suite, Nikto, FFUF, OWASP WSTG
취약점 검증	탐지 결과가 실제 영향으로 이어지는가?	Metasploit은 랩 중심, 수동 검증, 영향도 평가
인증/계정	암호 정책과 계정 보호가 충분한가?	정책 승인 하의 패스워드 감사, 계정 잠금 정책
사후 분석	침투 이후 어떤 통제 미비가 드러나는가?	권한 상승 점검, 로그, EDR/탐지 관점
자동화	반복 점검과 보고서를 어떻게 안정화할 것인가?	Python OOP, 범위 검증, 결과 모델링, 리포트 생성

CHAPTER 2

칼리 리눅스 이해

“해킹툴 모음”을 넘어 보안 평가 작업대라고 보기

칼리 리눅스는 오픈소스, Debian 기반의 리눅스 배포판이며, 고급 침투 테스트와 보안 감사에 사용할 수 있도록 설계되어 있다. 공식 문서도 Kali를 BackTrack의 후속으로 설명하면서, 여러 플랫폼에서 동작하고 정보보안 전문가와 학습자가 접근할 수 있다고 설명한다. [R1]

원문 메모의 “해킹툴을 모아놓은 리눅스”라는 표현은 입문자에게 직관적이지만, 실무적으로는 조금 더 넓게 이해해야 한다. Kali는 단순히 도구를 설치해 둔 OS가 아니라, 포렌식, 리버스 엔지니어링, 취약점 탐지, 무선 보안 평가처럼 정보보안 업무에 필요한 환경과 패키지를 한곳에 묶은 작업대다. 공식 문서도 여러 도구, 구성, 스크립트, 산업별 수정 사항을 통해 사용자가 보안 과업에 집중할 수 있도록 돕는다고 설명한다. [R1]

Kali의 특징을 정확히 이해하기

- **Debian 기반:** 패키지 관리, 파일시스템 구조, 리눅스 명령어, 서비스 관리 방식은 Debian 계열의 흐름을 따른다.
- **보안 도구 중심:** Nmap, Wireshark, Metasploit, Burp Suite, Aircrack-ng 등 여러 도구를 쉽게 설치하고 운용할 수 있다. Kali Tools 페이지는 보안 전문가용 오픈소스 도구 생태계와 연결되어 있다. [R2]
- **무선 평가 친화성:** 공식 문서에는 무선 평가를 위해 injection patch가 포함된 커널과 다양한 무선 장치 지원이 특징으로 제시된다. [R1]
- **범용 데스크톱과는 목적이 다름:** Kali는 일상 업무용 OS라기보다 테스트, 분석, 교육, 평가를 위한 전문 환경에 가깝다.
- **도구가 책임을 대신하지 않음:** 도구가 강력할수록 범위 승인, 로그 기록, 실습 격리, 데이터 보호가 더 중요하다.

개념 정정: packet injection

“Kali라서 무조건 패킷 주입이 된다”가 아니라, OS/커널/드라이버/무선 랜카드가 함께 지원해야 한다. Kali는 이 작업을 위한 도구와 커널 패치, 장치 지원을 잘 갖춘 배포판으로 이해하는 편이 정확하다.

Kali를 배울 때 먼저 잡아야 할 기본기

기본기	왜 중요한가	학습 포인트
리눅스 파일시스템	룰 설정, 로그, 워드리스트, 바이너리 위치를 이해해야 한다.	/etc, /usr/share, /var/log, 권한과 소유자
네트워크 기초	스캔 결과는 TCP/IP와 DNS를 알아야 해석할 수 있다.	TCP 3-way handshake, UDP, ICMP, DNS, ARP
패키지 관리	도구 설치/업데이트/의존성 문제를 해결해야 한다.	apt, repository, GPG key, version pinning
권한 모델	root 권한과 일반 사용자 권한의 차이를 이해해야 한다.	sudo, 파일 권한, capabilities
보고 습관	나중에 재현하려면 명령, 옵션, 시간, 타겟, 결과를 남겨야 한다.	작업 로그, 스크린샷, 원본 결과 파일 보존

Kali를 실습 환경으로 쓸 때의 안전 원칙

- 공격 대상은 반드시 본인 소유 또는 허가된 랩으로 제한한다.
- 클라우드, 회사망, 학교망, 공용 와이파이에서 무단 스캔을 하지 않는다.
- 도구별로 서비스 장애 가능성을 확인하고, 속도 제한과 중지 조건을 정한다.
- 실습 VM은 스냅샷을 사용해 원상 복구가 가능하게 한다.
- 보고서 작성까지 한 세트로 실습한다. 도구 실행 결과만 모으면 학습 효과가 반감된다.

CHAPTER 3

모의해킹 라이프사이클

대상 선정부터 문서화까지 이어지는 7 단계

원문 메모의 모의해킹 순서는 매우 좋은 뼈대다. 여기서는 0 단계 대상 선정부터 6 단계 문서화까지를 실무 보고서 구조에 맞게 확장한다. NIST SP 800-115 도 정보보안 테스트의 계획, 수행, 결과 분석, 완화 전략 개발을 돕기 위한 실무 권고를 제공한다고 설명한다. [R3]

단계	목표	핵심 질문	대표 산출물
0. 대상 선정	허가된 범위와 중지 조건을 정의한다.	무엇을 테스트할 수 있고, 무엇을 제외되는가?	범위 문서, 승인 내역, 일정, 연락 체계
1. 정보 수집	타겟의 공개 정보와 노출면을 파악한다.	도메인, IP, 서비스, 기술 스택은 무엇인가?	자산 목록, 포트/서비스 목록, OSINT 메모
2. 취약점 분석	약점 후보를 찾고 오탐 가능성을 줄인다.	버전, 설정, 입력값, 인증 흐름에 취약점 후보가 있는가?	스캐너 결과, 수동 검증 메모, 우선 순위 초안
3. 검증/침투	허가된 범위 안에서 실제 영향 가능성을 확인한다.	취약점이 실제 권한, 데이터, 서비스 영향으로 이어지는가?	재현 절차, 증거, 제한된 PoC 결과
4. 영향 평가	기술 결과를 비즈니스 위험으로 번역한다.	누가 무엇을 잃고, 얼마나 심각한가?	영향도 설명, 위험 등급, 공격 경로
5. 방어 분석	수정 방안과 탐지/대응 개선안을 제시한다.	어떤 설정, 코드, 정책, 모니터링을 바꿔야 하는가?	권고안, 우선순위, 검증 방법
6. 문서화	의사결정 가능한 보고서로 정리한다.	경영진과 실무자가 각각 무엇을 해야 하는가?	요약 보고서, 기술 부록, 재검증 계획

단계별 해설

0 단계 대상 선정은 “공격 전에 하는 행정 절차”가 아니다. 이 단계가 없으면 이후 모든 행위는 보안 테스트가 아니라 무단 접근으로 오해될 수 있다. IP 대역, 도메인, 테스트 시간, 금지 행위, 데이터 접근 제한, 장애 발생 시 연락처, 보고서 배포 범위를 문서화해야 한다.

1 단계 정보 수집에서는 가능한 한 덜 침습적인 방법부터 시작한다. DNS, 인증서, 공개 저장소, 검색 엔진, 조직 정보, 포트 스캔 결과를 모아 공격 표면을 지도처럼 만든다. 이 지도는 이후 단계에서 “어디를 깊게 볼지”를 정하는 근거가 된다.

2 단계 취약점 분석에서는 자동 스캐너가 유용하지만, 자동 결과를 그대로 결론으로 쓰면 위험하다. 오래된 버전이 탐지되어도 실제로 백포트 패치가 적용되었을 수 있고, 스캐너가 취약하다고 표시해도 인증 조건이나 방화벽 때문에 실제 영향이 없을 수 있다. 반대로 자동 스캐너가 놓치는 비즈니스 로직 결함도 많다.

3 단계 검증/침투는 “가능성을 증명”하는 단계이지, 무제한 장악을 시도하는 단계가 아니다. 업무에서는 최소 권한, 최소 접근, 최소 증거 수집 원칙을 지켜야 한다. 데이터가 민감하면 원본을 추출하지 않고 개수, 일부 마스킹된 샘플, 해시, 스크린샷 등으로 대체하는 방법을 우선 고려한다.

4 단계 영향 평가는 기술 언어를 의사결정 언어로 바꾸는 과정이다. “SQL Injection 가능”보다 “인증 없이 고객 테이블의 특정 필드에 접근할 가능성이 있으며, 개인정보 노출과 규제 리스크가 발생한다”가 더 좋은 설명이다.

5 단계 방어 분석에서는 패치, 설정 변경, 입력값 검증, 인증/인가 강화, 네트워크 분리, 모니터링, 로그 정책, 교육까지 연결한다. 좋은 권고안은 “무엇을 바꿔야 하는지”뿐 아니라 “바꾼 뒤 어떻게 검증할지”까지 포함한다.

6 단계 문서화는 마지막에 몰아서 하는 일이 아니라 전 과정에서 계속 축적해야 한다. 명령어, 시간, 타겟, 버전, 스크린샷, 원본 출력, 판단 근거를 남기면 보고서 품질이 크게 올라간다.

읽을거리: 해킹툴을 아는 것과 모의해킹을 아는 것

해킹툴은 “가능한 행동”을 늘려 준다. 모의해킹 역량은 “그 행동을 언제, 왜, 어디까지 해야 하는지”를 판단하는 능력에서 나온다. 초보자가 가장 빠르게 성장하는 방법은 도구 실행 결과를 매번 보고서 언어로 번역해 보는 것이다.

CHAPTER 4

단계별 핵심 도구 지도

21 개 도구를 목적, 산출물, 방어 관점으로 재분류

이 장은 원문 메모의 도구 목록을 기반으로 한다. 단, 악용 가능성이 큰 도구는 명령어 중심이 아니라 역할, 산출물, 방어 분석 포인트 중심으로 설명한다. 웹 애플리케이션 테스트는 OWASP WSTG 처럼 체계적인 테스트 프레임워크와 함께 읽어야 도구 결과를 올바르게 해석할 수 있다. [R4]

도구 사용 안전 등급

낮음: 주로 읽기/관찰 중심이지만, 대량 스캔은 여전히 허가 필요.

중간: 서버 부하나 계정 잠금 가능성이 있어 속도 제한과 범위 확인 필요.

높음: 인증 우회, 권한 상승, 자격 증명 탈취, 피싱, 내부망 이동에 가까운 기능이 있어 승인된 랩/업무 외 실행 금지.

도구	단계	주요 역할	방어 관점	안전 등급
Nmap	정보 수집	포트, 서비스, OS 추정	서비스 노출면과 방화벽 정책 확인	낮음-중간
Wireshark	정보 수집/분석	패킷 캡처와 프로토콜 분석	평문 프로토콜, 비정상 트래픽, 네트워크 흐름 확인	낮음
Nikto	취약점 분석	웹 서버 설정/버전/위험 파일 점검	오래된 서버, 불필요한 파일, 헤더 설정 개선	중간
Burp Suite	웹 진단	HTTP/HTTPS 트러쉬와 요청 변조	입력값 검증, 세션, 인증/인가 흐름 점검	중간
Metasploit	검증/랩	취약점 검증 프레임워크	패치 필요성, 영향도 검증, 탐지률 확인	높음
Hydra	인증 감사	로그인 정책 점검	계정 잠금, MFA, 비밀번호 정책, 탐지 로그	높음
SQLmap	웹 진단	SQL Injection 자동 탐지/검증	파라미터 바인딩, ORM 사용, WAF/로그 점검	높음
Aircrack-ng	무선 보안	Wi-Fi 패킷 캡처/분석 도구 모음	WPA 설정, 암호 강도, 관리 프레임 보호	높음
Gobuster	정보 수집	숨겨진 경로/디렉토리 탐색	불필요한 관리자 페이지, 백업 파일 제거	중간
Maltego	OSINT	도메인/이메일/인프라 관계 시각화	공개 노출 정보 최소화, 자산 인벤토리 검증	낮음-중간
John the Ripper	암호 감사	오프라인 해시 강도 검증	암호 정책, 해시 알고리즘, 솔트/반복 횟수 개선	높음
Hashcat	암호 감사	GPU 기반 해시 크래킹	약한 암호와 오래된 해시 정책 확인	높음
LinPEAS/WinPEAS	권한 상승 점검	로컬 설정 미비 탐색	취약 권한, 패치 누락, 비밀 정보 노출 개선	높음
Mimikatz	Windows 보안 검증	인증정보 노출 위험 검증	LSASS 보호, Credential Guard, 권한 최소화	높음
FFUF	웹 퍼징	경로/파라미터/헤더 퍼징	숨겨진 엔드포인트, 입력 처리 안정성 확인	중간
WPScan	CMS 진단	WordPress 플러그인/테마/사용자 점검	업데이트, 플러그인 최소화, 사용자 노출 방지	중간
Responder	내부망/AD 검증	LLMNR/NBT-NS 악용 가능성 확인	LLMNR 비활성화, SMB signing, NTLM 제한	높음
BeEF	브라우저 보안	클라이언트 사이드 취약성 검증	브라우저 보안정책, 피싱 대응, CSP	높음
SET	사회공학 훈련	피싱 시나리오 테스트 자동화	보안 인식 교육, 이메일 보안, 신고 프로세스	높음
NetExec	AD/내부망	대규모 Windows 네트워크 평가	권한 분리, 계정 재사용 제거, 도메인 로그 분석	높음
Chisel/Ligolo-ng	터널링/피벗	분리망 접근 경로 검증	네트워크 분리, egress 제어, 트러쉬 로그 점검	높음

4.1 정보 수집 도구

Nmap

네트워크 스캔의 표준 도구다. 포트 오픈 여부, 서비스 배너, 일부 OS 추정, NSE 스크립트 기반 점검을 통해 공격 표면을 만든다. 초보자는 -sV가 “서비스 버전 추정”이고 -O가 “운영체제 추정”이라는 점부터 이해하면 좋다. 결과 해석에서는 포트 번호보다 “왜 열려 있고 누가 써야 하는가”를 묻는 습관이 중요하다.

Wireshark

패킷을 눈으로 보는 도구다. 네트워크 문제 해결에도 널리 쓰이며, 보안 관점에서는 평문 로그인, 비정상 DNS, ARP 이상, TLS 핸드셰이크, 프로토콜 오류를 확인한다. 캡처 자체가 민감 정보를 포함할 수 있으므로 저장/공유 시 마스킹과 보관 정책이 필요하다.

Gobuster / Dirb / Dirbuster

숨겨진 디렉토리, 백업 파일, 관리자 페이지, 서브도메인 후보를 사전 기반으로 찾는다. 좋은 결과를 얻으려면 워드리스트 선택, 확장자, 응답 코드, 응답 길이 필터링을 이해해야 한다. 방어 관점에서는 노출된 백업 파일, .env, 테스트 페이지, 기본 관리자 경로를 제거해야 한다.

Maltego

도메인, IP, DNS, 이메일, 조직 정보의 관계를 그래프로 보여 주는 OSINT 도구다. 공격자 관점에서는 공개 정보가 얼마나 많이 연결되는지 보여 주고, 방어자 관점에서는 외부 노출 자산과 직원 정보의 과다 노출을 줄이는 근거가 된다.

```
# 허가된 랩/업무 범위에서만 사용하는 기본 예시
nmap -sV -O [target-ip]
nikto -h http://[authorized-target]
gobuster dir -u http://[authorized-target] -w [wordlist]
ffuf -u http://[authorized-target]/FUZZ -w [wordlist]
```

4.2 웹 애플리케이션 진단 도구

Nikto

웹 서버의 알려진 취약 파일, 위험한 설정, 오래된 버전, 기본 파일을 빠르게 확인한다. 장점은 빠르고 단순하다는 점이고, 한계는 결과가 거칠고 오탐이 있다는 점이다. Nikto 결과는 “정밀 분석 후보”로 보아야 한다.

Burp Suite

브라우저와 서버 사이에 프록시로 위치해 요청/응답을 관찰하고 수정한다. 웹 해킹 학습의 핵심은 Burp에서 보이는 HTTP 요청 구조를 이해하는 것이다. 쿠키, 세션, CSRF 토큰, Authorization 헤더, JSON 파라미터, 리다이렉션을 읽을 수 있어야 한다.

SQLmap

SQL Injection 가능성을 자동으로 탐지하고 검증하는 도구다. 운영 시스템에서는 위험도가 높으므로 사전 승인, 테스트 계정, 제한된 파라미터, 낮은 강도, 데이터 추출 금지 원칙이 필요하다. 방어 관점에서는 파라미터 바인딩, ORM 사용, 최소 권한 DB 계정, 에러 메시지 통제가 핵심이다.

FFUF

웹 경로, 파라미터, 헤더, 가상 호스트 등을 빠르게 퍼징한다. Gobuster보다 유연한 위치 치환이 가능해 숨겨진 엔드포인트와 파라미터 찾기에 유용하다. 단, 속도가 빠른 만큼 부하와 로그 폭증을 유발할 수 있어 제한이 필요하다.

WPScan

WordPress 사이트에서 플러그인, 테마, 버전, 사용자 노출 등을 점검한다. 방어 관점에서는 플러그인 최소화, 자동 업데이트 정책, 관리자 계정명 노출 방지, 웹방화벽보다 근본적인 패치 관리가 중요하다.

4.3 검증, 권한 상승, 사후 행위 도구

이 범주의 도구는 학습 가치가 크지만 악용 위험도 크다. 따라서 여기서는 구체적 공격 절차보다 “무엇을 검증하는 도구 인지”와 “방어자가 무엇을 고쳐야 하는지”에 초점을 둔다.

Metasploit Framework

취약점 검증과 랩 실습에 널리 쓰이는 프레임워크다. 익스플로잇과 페이로드의 개념을 이해하는 데 도움이 되지만, 실제 업무에서는 승인된 취약점에 대해 최소한의 영향 검증으로 제한해야 한다. 방어자는 패치 우선순위, 취약 서비스 제거, EDR 탐지, 네트워크 분리 정책을 함께 점검한다.

Hydra

인증 인터페이스의 비밀번호 정책을 검증하는 도구로 알려져 있다. 운영 환경에서는 계정 잠금, 서비스 장애, 로그 폭증을 유발할 수 있으므로 매우 조심해야 한다. 방어 관점의 핵심은 MFA, 계정 잠금, 지연 응답, 비밀번호 재사용 방지, 인증 실패 모니터링이다.

John the Ripper / Hashcat

유출되었거나 감사 목적으로 확보한 해시의 강도를 확인하는 오프라인 암호 감사 도구다. 취약한 암호가 빨리 복구된다면 비밀번호 정책, 해시 알고리즘, 솔트, 반복 횟수, MFA 정책을 개선해야 한다.

LinPEAS / WinPEAS

침투 후 로컬 시스템에서 권한 상승 가능성을 찾는 점검 스크립트다. 과도한 sudo 권한, 취약한 서비스 권한, 민감 파일 노출, 패치 누락을 찾아 방어 개선 목록으로 바꾸는 데 유용하다.

Mimikatz

Windows 인증 정보 보호 상태를 검증하는 대표적 도구로 알려져 있다. 업무 환경에서는 사용 자체가 매우 민감하므로 승인된 랩과 엄격한 절차가 필요하다. 방어자는 LSASS 보호, Credential Guard, 관리자 권한 최소화, 원격 로그인 제한, 탐지 룰을 점검한다.

Responder

Windows 내부망에서 이름 해석 프로토콜과 NTLM 인증 흐름의 약점을 검증하는 데 사용된다. 방어 관점에서는 LLMNR/NBT-NS 비활성화, SMB signing, NTLM 제한, 네트워크 세그먼트 분리, 이벤트 로그 모니터링이 중요하다.

NetExec

Active Directory 환경에서 인증, 권한, 원격 관리 노출을 대규모로 점검하는 도구다. 방어자는 로컬 관리자 비밀번호 재사용, 과도한 도메인 권한, 원격 실행 경로, SMB/RPC 노출, 감사 로그를 체계적으로 확인해야 한다.

Chisel / Ligolo-ng

네트워크 터널링과 피벗팅 시나리오를 검증하는 데 쓰인다. 외부에서 직접 접근할 수 없는 내부 자원이 중간 장비를 통해 접근 가능한지 보여 주므로, 방어 관점에서는 egress 필터링, 프록시 로그, 네트워크 분리, 방화벽 정책 검증이 핵심이다.

BeEF / SET

브라우저와 사람을 대상으로 하는 클라이언트 사이드/사회공학 시나리오를 교육 또는 승인된 훈련에서 검증하는 도구다. 방어자는 보안 인식 교육, 피싱 신고 프로세스, 이메일 보안, 브라우저 하드닝, CSP, 세션 보호를 함께 살펴야 한다.

도구 결과를 보고서 언어로 바꾸는 예

도구 결과 표현	보고서 표현으로 변환	방어 조치
22/tcp open ssh OpenSSH 7.x	SSH가 외부에 노출되어 있고 버전이 오래되었을 가능성이 있다.	접근 IP 제한, 최신 패치, MFA/키 기반 인증, 배너 확인

관리자 경로 /admin 발견	인증 페이지가 예측 가능한 경로에 노출되어 공격 표면이 증가한다.	접근 제어, VPN/SSO 보호, 기본 경로 제거, 모니터링
SQLi 후보 탐지	입력값 검증 미흡으로 DB 접근 위험이 존재할 수 있다.	파라미터 바인딩, DB 최소 권한, 에러 메시지 통제, 재검증
약한 암호 빠른 복구	비밀번호 정책이 공격자의 사전 대입에 취약하다.	MFA, 길이 기반 정책, 유출 암호 차단, 해시 정책 개선
권한 상승 후보 발견	일반 사용자 권한에서 관리자 권한으로 확대될 가능성이 있다.	권한 최소화, 패치, 서비스 권한 재설정, 비밀정보 제거

CHAPTER 5

무선 보안과 패킷 주입

Aircrack-ng 를 “도구 절차”가 아니라 “위험 모델”로 이해하기

무선 보안은 유선 네트워크와 다른 특성이 있다. 전파는 물리적 경계 밖으로 퍼지고, 공격자는 건물 밖에서도 신호를 관찰할 수 있다. 따라서 Wi-Fi 보안은 암호 강도, 인증 방식, 관리 프레임 보호, 게스트망 분리, AP 설정, 클라이언트 보안까지 함께 봐야 한다.

Aircrack-ng Suite의 구성 요소 이해

구성 요소	개념적 역할	방어 관점
airmon-ng	무선 인터페이스를 모니터 모드로 전환하는데 사용된다.	관리되지 않는 무선 어댑터 사용과 비인가 캡처를 통제한다.
airodump-ng	주변 AP와 클라이언트, 채널, 암호화 방식 등 무선 환경을 관찰한다.	불필요한 SSID, 약한 암호화, 예전 표준 사용 여부를 점검한다.
aireplay-ng	무선 프레임 주입 시나리오를 실행/검증하는데 사용될 수 있다.	관리 프레임 보호, AP 로그, 비인가 공격 탐지를 확인한다.
aircrack-ng	캡처된 핸드셰이크 등에 대한 암호 강도 검증에 사용된다.	강한 패스프레이즈, WPA3/SAE, PSK 관리, 게스트망 분리를 강화한다.

실습 경계

무선 보안 실습은 반드시 본인 소유 AP와 실습용 클라이언트에서만 해야 한다. 주변 무선망을 대상으로 캡처, 인증 해제, 핸드셰이크 수집, 암호 추측을 수행하면 법적 문제가 발생할 수 있다.

무선 보안에서 보고해야 할 포인트

- SSID와 보안 방식: WPA2/WPA3, 개인/엔터프라이즈 모드, WPS 사용 여부
- 암호 정책: 길이, 예측 가능성, 공유 방식, 퇴사자/외부자 관리
- 네트워크 분리: 게스트망, 내부망, IoT 망, 관리망 분리 여부
- 관리 프레임 보호: deauthentication 류 공격에 대한 완화 설정
- AP 관리: 기본 관리자 계정, 펌웨어 업데이트, 원격 관리 노출 여부
- 탐지/대응: 비정상 인증 실패, 로밍 이상, AP 이벤트 로그 수집 여부

읽을거리: 무선 보안은 “암호를 깨는 기술”이 아니다

무선 보안을 배울 때 가장 흥미로운 장면은 핸드셰이크 캡처와 암호 검증일 수 있다. 하지만 실무에서 더 중요한 질문은 “왜 이 암호가 약했는가”, “왜 게스트가 내부망에 닿았는가”, “왜 비인가 AP를 탐지하지 못했는가”다. 도구는 현상을 보여주고, 보고서는 원인과 개선책을 말해야 한다.

CHAPTER 6

Python 객체지향 핵심

모의해킹 자동화를 위한 언어 구조 이해

원문 메모의 후반부는 Python 객체지향 개념을 집중적으로 다룬다. Python 으로 보안 점검 도구를 만들려면 단순히 문법을 아는 것보다 “객체가 무엇을 저장하고, 메서드가 어떤 대상에 바인딩되며, 모듈을 어떻게 나눌지”를 이해해야 한다. 공식 Python 문서는 import 가 모듈을 검색하고 그 결과를 현재 스코프의 이름에 바인딩하는 과정이라고 설명한다. [R5]

6.1 import 와 from import

import 는 다른 파일의 코드를 가져오는 것처럼 보이지만, 실제로는 모듈을 찾고, 필요하면 실행해 모듈 객체를 만들고, 그 객체를 현재 이름공간에 묶는 과정이다. 반면 from module import name 은 모듈 안의 특정 이름을 현재 이름공간에 직접 바인딩한다.

```
# 1) 모듈 객체를 가져오기
import scanners.nmap_scanner
scanner = scanners.nmap_scanner.NmapScanner()

# 2) 모듈 안의 이름을 직접 가져오기
from scanners.nmap_scanner import NmapScanner
scanner = NmapScanner()

# 3) import 시 원치 않는 실행을 막는 관용구
def main():
    print('run diagnostics')

if __name__ == '__main__':
    main()
```

Python 문서는 top-level 환경에서 실행된 모듈의 `__name__`이 `__main__`이 된다고 설명한다. 그래서 if `__name__ == "__main__"`: 블록은 파일을 직접 실행할 때만 테스트 코드나 메인 로직이 돌게 하고, 다른 파일에서 import 할 때는 실행되지 않게 하는 방어벽 역할을 한다. [R6]

6.2 Python 에서 “모든 것은 객체다”

Python에서는 숫자, 문자열, 함수, 클래스, 모듈까지 모두 객체로 다룰 수 있다. 객체는 대체로 identity, type, value 를 가진다. 함수를 변수에 담거나, 클래스를 함수 인자로 넘기거나, 모듈의 속성을 런타임에 검사하는 방식이 가능한 이유도 이 객체 모델 때문이다.

대상	객체로 본다는 뜻	보안 도구 설계에서의 의미
함수	변수에 담고 인자로 넘길 수 있다.	체크 로직을 플러그인처럼 등록할 수 있다.
클래스	클래스 자체도 객체다.	Scanner 클래스를 상속해 다양한 점검기를 만들 수 있다.
모듈	모듈도 속성을 가진 객체다.	도구별 모듈을 분리하고 동적으로 로드할 수 있다.
예외	예외 타입도 클래스다.	ScopeError, FindingError 같은 도메인 예외를 설계할 수 있다.

6.3 클래스, 인스턴스, self

클래스는 객체를 만들기 위한 설계도이고, 인스턴스는 그 설계도로 실제 메모리에 만들어진 객체다. 클래스 내부의 일반 메서드는 객체를 통해 호출될 때 첫 번째 인자로 자신을 자동 전달받는다. 관례적으로 이 첫 번째 인자를 `self` 라고 쓴다.

```
class Scanner:
    def __init__(self, name):
        self.name = name
        self.results = []

    def add_result(self, result):
        self.results.append(result)
```

```
s = Scanner('network')
s.add_result('22/tcp open')
# 내부적으로는 Scanner.add_result(s, '22/tcp open')처럼 동작한다고 이해하면 쉽다.
```

self를 왜 쓰는가

self는 “이 메서드를 호출한 바로 그 객체”에 접근하기 위한 통로다. self.name은 각 인스턴스가 가진 name이고, Scanner.name 같은 클래스 속성과는 다르다.

6.4 생성자, 소멸자, dunder 메서드

__init__, __del__, __str__, __repr__, __len__처럼 양쪽에 언더스코어 두 개가 붙은 이름을 dunder 또는 magic method라고 부른다. 사용자가 직접 호출하기보다는 Python 인터프리터가 특정 상황에서 자동 호출한다. 예를 들어 객체 생성 후 초기화에는 __init__, print 표현에는 __str__ 또는 __repr__ 계열이 관여한다.

원문 메모에는 del a 이후 __del__이 호출되는 흐름이 나온다. CPython에서는 참조 카운팅 때문에 참조가 0이 되는 순간 소멸자가 바로 호출되는 경우가 많지만, 순환 참조나 다른 Python 구현에서는 시점이 달라질 수 있다. 파일, 소켓, 캡처 핸들 같은 자원은 __del__에 의존하기보다 with 문과 context manager로 관리하는 것이 안전하다.

```
class CaptureSession:
    def __enter__(self):
        print('capture start')
        return self

    def __exit__(self, exc_type, exc, tb):
        print('capture stop and close handle')

with CaptureSession() as session:
    pass
```

6.5 클래스 변수와 인스턴스 변수

클래스 바로 아래에 선언한 변수는 클래스 객체에 붙는 클래스 변수다. 모든 인스턴스가 공유하므로, 설정값이나 카운터처럼 공유가 의도된 데이터에 적합하다. 반대로 각 객체가 따로 가져야 하는 값은 __init__ 안에서 self.변수명으로 선언해야 한다.

```
class BadScanner:
    results = [] # 모든 인스턴스가 공유하므로 위험한 예

class GoodScanner:
    default_timeout = 3 # 공유해도 되는 설정값

    def __init__(self):
        self.results = [] # 인스턴스마다 독립된 결과 목록
```

구분	선언 위치	공유 여부	사용 예
인스턴스 변수	__init__ 내부의 self.x	객체마다 별도	스캔 결과, 대상 호스트, 실행 시간
클래스 변수	class 바로 아래의 x	클래스와 모든 인스턴스가 공유	기본 타임아웃, 버전, 전체 카운터
정적 메서드	@staticmethod	자동 인자 없음	URL 정규화, 해시 계산 등 유틸리티
클래스 메서드	@classmethod	cls를 자동 전달	설정 파일에서 객체 생성, 클래스 상태 관리

6.6 @property Getter/Setter

Python의 @property는 메서드를 변수처럼 읽게 하면서도 내부적으로 검증 로직을 유지하게 해 준다. 공식 문서도 property 객체가 getter, setter, deleter 메서드를 데코레이터로 제공한다고 설명한다. [R7]

```
class Target:
```

```
def __init__(self, host):
    self._host = None
    self.host = host

@property
def host(self):
    return self._host

@host.setter
def host(self, value):
    if not value:
        raise ValueError('host must not be empty')
    self._host = value

t = Target('192.0.2.10')
print(t.host) # 변수처럼 읽지만 내부적으로 getter 호출
t.host = 'lab.local' # 대입처럼 보이지만 setter 검증 통과 후 저장
```

이 방식의 장점은 하위 호환성이다. 처음에는 단순한 public 변수처럼 쓰다가 나중에 검증이 필요해져도 외부 코드는 `t.host` 형태를 유지할 수 있다. 보안 도구에서는 타겟, 포트, 스코프, 심각도 같은 값에 검증을 넣을 때 유용하다.

6.7 인스턴스 메서드, 클래스 메서드, 정적 메서드

메서드 종류	첫 번째 인자	호출 형태	적합한 용도
인스턴스 메서드	<code>self</code>	<code>obj.method()</code>	객체별 상태 읽기/수정
클래스 메서드	<code>cls</code>	<code>Class.method()</code> 또는 <code>obj.method()</code>	팩토리 메서드, 클래스 변수 제어
정적 메서드	없음	<code>Class.method()</code> 또는 <code>obj.method()</code>	클래스와 논리적으로 관련된 순수 유틸리티

```
class Finding:
    severity_order = {'low': 1, 'medium': 2, 'high': 3}

    def __init__(self, title, severity):
        self.title = title
        self.severity = severity

    @classmethod
    def from_dict(cls, data):
        return cls(data['title'], data['severity'])

    @staticmethod
    def normalize_severity(value):
        return value.strip().lower()
```

공식 문서에 따르면 `classmethod`는 클래스 또는 인스턴스에서 호출될 수 있고, 파생 클래스에서 호출하면 파생 클래스 객체가 암시적 첫 번째 인자로 전달된다. [R7]

6.8 `__dict__`, 동적 속성, `__slots__`

일반적인 사용자 정의 객체는 내부적으로 `__dict__`를 사용해 속성을 저장한다. 그래서 런타임에 `obj.new_attr = 1`처럼 새 속성을 추가할 수 있다. 이 유연성은 디버깅과 메타프로그래밍에는 좋지만, 대량 객체 생성에서는 메모리 부담이 된다.

Python 공식 데이터 모델 문서는 `__slots__`가 인스턴스가 가질 데이터 멤버를 명시적으로 선언하고, `__dict__`와 `__weakref__`의 자동 생성을 막을 수 있다고 설명한다. 또한 `__dict__`가 없으면 `slots`에 등록되지 않은 새 변수 할당은 `AttributeError`가 된다. [R8]

```
class OptimizedTarget:
    __slots__ = ['host', 'port']

    def __init__(self, host, port):
        self.host = host
        self.port = port

obj = OptimizedTarget('lab.local', 443)
# obj.memo = 'new' # AttributeError: slots에 없는 속성
```

`__slots__`는 만능이 아니다

객체 수가 매우 많아 메모리 최적화가 필요할 때는 유용하다. 하지만 동적 속성 추가가 막히고, 상속과 다중 상속에서 주의점이 많다. 입문 단계에서는 먼저 `__dict__` 모델을 이해한 뒤 필요할 때만 사용한다.

6.9 상속, super, 오버라이딩, 다형성

상속은 공통 기능을 부모 클래스에 두고, 자식 클래스가 필요한 부분만 확장하도록 만드는 방법이다. 다형성은 같은 메서드 이름을 호출했을 때 객체 종류에 따라 다른 동작을 하게 하는 성질이다. 보안 점검 프레임워크에서는 Scanner 라는 공통 인터페이스를 두고 PortScanner, WebScanner, ConfigScanner 가 각자 scan 을 구현하는 방식이 자연스럽다.

```
class Scanner:
    def scan(self, target):
        raise NotImplementedError

class PortScanner(Scanner):
    def scan(self, target):
        return f'port scan for {target}'

class WebScanner(Scanner):
    def scan(self, target):
        return f'web check for {target}'

for scanner in [PortScanner(), WebScanner()]:
    print(scanner.scan('authorized-lab'))
```

6.10 OOP 개념을 보안 점검 도구에 매핑하기

객체지향을 배울 때 가장 좋은 방법은 문법 이름을 외우는 것이 아니라, 실제 문제의 명사를 클래스로 바꿔 보는 것이다. 모의해킹 보고서 자동화에서는 Target, Scope, Evidence, Finding, Scanner, Reporter 같은 명사가 자연스럽게 클래스 후보가 된다.

OOP 개념	보안 점검 프로젝트에서의 대응	주의할 점
캡슐화	Finding 내부에 심각도 계산과 증거 마스킹 로직을 넣는다.	외부에서 내부 리스트를 마음대로 수정하지 못하게 한다.
상속	Scanner 부모 클래스를 만들고 WebScanner/PortScanner 가 확장한다.	상속이 깊어지면 복잡해지므로 공통 인터페이스만 추출한다.
다형성	reporter.export(finding) 호출은 PDF/Markdown/HTML 구현체별로 다르게 동작한다.	반환 형식과 예외 규칙은 통일해야 한다.
클래스 변수	공통 severity_order, default_timeout 같은 값을 공유한다.	mutable 리스트/딕셔너리를 공유하면 의도치 않은 결과 섞임이 생긴다.
정적 메서드	URL 정규화, 포트 범위 검증, 문자열 마스킹 같은 유틸리티를 둔다.	상태가 필요한 기능을 역지로 static 으로 만들지 않는다.
클래스 메서드	Finding.from_dict(), Target.from_url()처럼 대체 생성자를 만든다.	cls 를 사용하면 상속 클래스에서도 자연스럽게 동작한다.
__slots__	Target 객체가 수십만 개일 때 메모리 사용을 줄인다.	디버깅 편의성과 동적 확장성이 줄어든다.

입문자가 자주 하는 실수	왜 문제인가	개선 방향
메서드에 self 를 빼먹음	객체로 호출될 때 첫 번째 인자가 자동 전달되어 TypeError 가 난다.	인스턴스 상태를 쓰는 메서드는 def m(self, ...) 형태로 작성한다.
class 아래에 results = [] 선언	모든 인스턴스가 같은 리스트를 공유한다.	__init__ 에서 self.results = [] 로 만든다.
import 시 테스트 코드가 실행됨	모듈 재사용이 어렵고 자동화 파이프라인이 오염된다.	if __name__ == "__main__": 가드를 사용한다.
__del__ 에 자원 정리를 의존	소멸 시점이 예측과 다를 수 있다.	with 문과 명시적 close 를 사용한다.
모든 것을 상속으로 해결	구조가 경직되고 테스트가 어려워진다.	작은 클래스와 조합, 명확한 인터페이스를 우선한다.

CHAPTER 7

Python 으로 설계하는 안전한 보안 점검 프레임워크

“자동화된 공격 프로그램”을 “통제된 진단 자동화”로 바꾸기

원문 메모에는 궁극적 목표로 “자동화된 공격 프로그램을 Python 으로 만드는 것”이 언급되어 있다. 실무적으로는 이 목표를 “승인된 범위 안에서 반복 가능한 보안 점검을 수행하고 보고서를 자동 생성하는 프레임워크”로 재정의하는 것이 안전하고 가치 있다. 자동화가 강력할수록 스코프 검증, 속도 제한, 로깅, 실패 처리, 데이터 최소화가 필수다.

자동화의 안전 기준

1. 허가된 범위 외 타겟은 실행 전에 차단한다.
2. 서비스 장애를 막기 위해 동시성, 요청 속도, 타임아웃을 제한한다.
3. 위험한 검증은 기본 비활성화하고 명시적 승인 플러그가 있어야 한다.
4. 민감 데이터는 원본 저장을 피하고 마스킹/해시/요약으로 기록한다.
5. 모든 입력, 명령, 결과, 예외를 감사 로그로 남긴다.

7.1 권장 아키텍처

컴포넌트	역할	객체지향 연결
TargetScope	허가된 도메인, IP, 포트, 시간대를 검증한다.	@property 로 입력 검증, contains 메서드
Target	개별 점검 대상을 표현한다.	host, port, scheme 같은 인스턴스 변수
Scanner	모든 점검기의 공통 인터페이스다.	추상 부모 클래스, scan 메서드
Check	특정 취약점 또는 설정 항목을 점검한다.	다형성, 플러그인 구조
Evidence	증거를 구조화해 저장한다.	데이터 클래스, 마스킹 메서드
Finding	위험 등급, 영향, 권고안을 담는다.	@classmethod from_evidence
Reporter	Markdown/PDF/HTML 보고서를 생성한다.	출력 형식별 하위 클래스
AuditLog	실행 기록과 예외를 보존한다.	컨텍스트 매니저, 파일 핸들 관리

7.2 안전한 스코프 검증 예시

```
class ScopeError(Exception):
    pass

class TargetScope:
    def __init__(self, allowed_hosts):
        self.allowed_hosts = set(allowed_hosts)

    def assert_allowed(self, host):
        if host not in self.allowed_hosts:
            raise ScopeError(f'out of scope: {host}')

class SafeScanner:
    def __init__(self, scope, timeout=3):
        self.scope = scope
        self.timeout = timeout

    def scan(self, host):
        self.scope.assert_allowed(host)
        # 실제 점검 로직은 여기에서 제한적으로 수행
        return {'host': host, 'status': 'checked'}

scope = TargetScope(['lab.local'])
scanner = SafeScanner(scope)
print(scanner.scan('lab.local'))
```

이 예시는 공격 코드를 보여 주기 위한 것이 아니라 구조를 보여 주기 위한 것이다. 핵심은 scan 전에 항상 스코프 검증을 강제하고, 예외 타입을 명확히 분리하며, 결과를 딕셔너리나 객체로 구조화한다는 점이다.

7.3 결과 모델링 예시

```
class Evidence:
    def __init__(self, source, summary, raw=None):
        self.source = source
        self.summary = summary
        self.raw = raw

    def safe_summary(self):
        return self.summary.replace('password=', 'password=***')

class Finding:
    def __init__(self, title, severity, evidence, recommendation):
        self.title = title
        self.severity = severity
        self.evidence = evidence
        self.recommendation = recommendation

    @classmethod
    def from_evidence(cls, evidence):
        return cls(
            title='노출된 관리 경로',
            severity='medium',
            evidence=evidence,
            recommendation='관리 경로 접근을 VPN/SSO 뒤로 이동하고 로그를 강화한다.'
        )
```

7.4 플러그인 방식으로 확장하기

점검 도구가 커질수록 if 문으로 모든 도구를 처리하기보다 Scanner 인터페이스를 만들고, 각 도구가 같은 반환 형식을 지키도록 설계하는 편이 좋다. 이렇게 하면 Nmap 기반 점검, 웹 헤더 점검, TLS 설정 점검, 보고서 생성기를 서로 교체하기 쉬워진다.

나쁜 구조	좋은 구조
한 파일에 모든 스캔 로직과 출력 로직이 섞임	scanner, models, reporters, config 모듈로 분리
결과가 print 문자열로만 남음	Finding/Evidence 객체로 구조화
실행 중 실패하면 전체 중단	예외를 잡아 실패한 점검만 기록
타겟 범위 검증이 함수마다 제각각	TargetScope 를 공통 관문으로 강제
보고서 작성은 마지막에 수작업	실행 중 증거를 누적해 자동 초안 생성

7.5 자동화가 위험해지는 순간

자동화는 반복 실수를 줄이지만, 잘못 설계하면 피해 규모를 키운다. 특히 전체 인터넷 대역 스캔, 인증 대입 반복, 위험한 페이로드 자동 실행, 민감 데이터 원본 저장, 범위 검증 없는 URL 입력은 피해야 한다. 자동화된 보안 점검기는 기본적으로 보수적이어야 하고, 위험한 동작은 여러 단계의 명시적 승인 없이는 실행되지 않아야 한다.

CHAPTER 8

보고서 작성법

발견 사항을 방어 가능한 개선안으로 바꾸기

모의해킹의 마지막 단계는 문서화이지만, 실제로는 모든 단계에서 문서화를 해야 한다. 좋은 보고서는 공격 시연을 자랑하는 문서가 아니라, 조직이 위험을 줄이기 위해 무엇을 해야 하는지 결정하게 해 주는 문서다.

8.1 보고서의 기본 구조

구성	목적	포함 내용
Executive Summary	비기술 의사결정자를 위한 요약	전체 위험 수준, 핵심 발견, 우선순위, 개선 방향
Scope & Method	테스트 범위와 방법의 투명성 확보	대상, 기간, 제외 범위, 도구, 제약 조건
Findings	개별 취약점 설명	제목, 심각도, 영향, 증거, 재현 개요, 권고안
Risk Matrix	우선순위 결정	가능성, 영향도, 탐지 난이도, 수정 난이도
Remediation Plan	실행 가능한 개선 계획	담당팀, 예상 난이도, 단기/중기 조치
Appendix	기술 세부 자료 보존	원본 스캔 결과, 스크린샷, 명령 로그, 용어 설명

8.2 Finding 작성 템플릿

항목	작성 질문	예시 표현
제목	한 줄로 무엇이 문제인가?	관리자 페이지가 인증 없이 외부에 노출됨
심각도	영향과 가능성을 고려하면 우선순위는?	High / Medium / Low 또는 조직 기준
영향	공격자가 무엇을 할 수 있는가?	권한 없는 사용자가 설정 정보를 조회할 수 있음
증거	무엇으로 확인했는가?	요청/응답 캡처, 화면, 로그, 스캔 결과
재현 개요	어떤 조건에서 재현되는가?	허가된 테스트 계정과 지정된 URL 에서 확인
원인	왜 발생했는가?	경로 접근 제어가 라우터 레벨에서 누락됨
권고안	무엇을 고쳐야 하는가?	서버 측 인증/인가 검증 추가, VPN 뒤로 이동
재검증	어떻게 수정 완료를 확인할 것인가?	비인가 요청이 401/403 으로 차단되는지 확인

8.3 좋은 문장과 나쁜 문장

피해야 할 문장	개선된 문장
사이트가 털릴 수 있음.	인증되지 않은 사용자가 /admin 경로에 접근할 수 있어 관리자 기능 노출 가능성이 확인됨.
SQLmap 으로 DB 가 나옴.	id 파라미터에서 SQL Injection 후보가 확인되었으며, 제한된 범위의 검증에서 DB 오류 기반 반응이 재현됨.
비밀번호가 약함.	승인된 해시 감사에서 사전 기반 추측에 취약한 패턴이 확인되어 길이 기반 정책과 유출 암호 차단이 필요함.
방화벽 설정을 잘해야 함.	외부 접근이 필요한 포트를 식별하고, SSH 는 관리 IP/VPN 으로 제한하며, 나머지 서비스는 내부망으로 이동할 것을 권고함.

8.4 증거 수집 원칙

- 증거는 필요한 만큼만 수집한다. 민감 데이터 원본 덤프는 피하고 마스킹된 샘플을 우선한다.
- 명령어, 옵션, 타겟, 실행 시각, 결과 파일 경로를 함께 기록한다.
- 화면 캡처는 URL, 계정, 토큰, 개인정보를 마스킹한다.
- 자동 스캐너 결과는 원본 파일로 보관하되, 보고서에는 의미 있는 부분만 요약한다.
- 오탐 가능성, 제약 조건, 테스트하지 못한 영역을 정직하게 적는다.

보고서의 목적

보고서는 “내가 무엇을 할 수 있었는지”를 보여 주는 문서가 아니다. “조직이 무엇을 고쳐야 하는지”를 결정하게 해 주는 문서다.

8.5 보고서 리뷰 체크리스트

리뷰 질문	좋은 상태	나쁜 상태
범위가 명확한가?	대상, 기간, 제외 범위, 계정, 테스트 제한이 표기됨	스캔 결과만 있고 무엇을 테스트했는지 불명확
증거가 충분한가?	스크린샷, 요청/응답, 로그, 원본 결과가 연결됨	주장만 있고 재현 근거가 없음
영향이 설명되는가?	기술 취약점이 데이터/권한/서비스 영향으로 번역됨	도구가 위험하다고 했다는 설명에 그침
권고안이 실행 가능한가?	담당자가 바로 수정 방향을 이해할 수 있음	보안을 강화하라는 추상적 문장만 있음
오탐과 제약을 적었는가?	검증하지 못한 조건과 오탐 가능성을 분리함	모든 자동 결과를 확정 취약점처럼 기록
재검증이 가능한가?	수정 후 확인할 요청, 설정, 로그 기준이 있음	언제 끝났다고 볼지 기준이 없음

8.6 경영진 요약용 쓰는 방식

경영진 요약은 도구 이름을 나열하는 곳이 아니다. 핵심은 “현재 가장 큰 위험은 무엇이고, 왜 지금 조치해야 하며, 어떤 순서로 줄일 수 있는가”다. 기술 세부사항은 부록으로 보내고, 본문에는 우선순위와 의사결정 포인트를 둔다.

요약 항목	작성 예시
전체 평가	외부 노출 서비스 관리와 웹 입력값 검증에서 중간 이상 위험이 확인됨.
가장 중요한 위험	관리 경로 노출과 오래된 서비스 버전으로 인해 인증 우회 또는 권한 상승 가능성이 존재함.
단기 조치	외부 접근 제한, 취약 버전 패치, 관리자 경로 접근 제어를 2 주 내 우선 적용.
중기 조치	자산 인벤토리와 취약점 관리 프로세스를 정례화하고, 재검증 자동화를 도입.
재검증 기준	동일 범위에서 재스캔 및 수동 확인 시 취약 경로 접근이 차단되고 로그가 수집되어야 함.

CHAPTER 9

학습 로드맵과 체크리스트

도구 암기에서 구조적 역량으로

원문 메모의 내용은 크게 두 축으로 나뉜다. 하나는 칼리와 모의해킹 도구, 다른 하나는 Python 객체지향이다. 두 축은 따로 배우는 것이 아니라 “도구 결과를 Python 객체로 구조화하고 보고서로 자동화한다”는 목표로 연결할 수 있다.

9.1 6 주 학습 로드맵

주차	학습 목표	실습 산출물
1 주차	Linux/Kali 기본기와 네트워크 기초	명령어 노트, 네트워크 용어 정리, 실습 VM 스냅샷
2 주차	정보 수집과 포트/서비스 해석	Nmap 결과표, 열린 포트별 위험 설명
3 주차	웹 요청/응답과 Burp Suite 기초	HTTP 요청 분석 메모, 쿠키/세션 흐름도
4 주차	웹 서버/경로/파라미터 점검	Nikto/Gobuster/FFUF 결과 요약 보고서
5 주차	Python OOP 로 결과 모델링	Target, Evidence, Finding 클래스 구현
6 주차	미니 보안 점검 보고서 자동 생성	Markdown 또는 PDF 보고서 초안 자동 생성

9.2 매 실습 후 남겨야 할 기록

- 실습 목적: 오늘 확인하려는 질문은 무엇인가?
- 대상 범위: 어떤 호스트, 도메인, 계정, 시간대가 허가되었는가?
- 도구와 옵션: 어떤 도구를 어떤 옵션으로 실행했는가?
- 결과 요약: 발견한 사실과 모르는 사실은 무엇인가?
- 오탐/제약: 자동 결과 중 신뢰하기 어려운 부분은 무엇인가?
- 방어 조치: 운영자가 바로 할 수 있는 개선은 무엇인가?
- 재검증 방법: 수정 후 무엇을 다시 확인할 것인가?

9.3 개인 프로젝트 아이디어

프로젝트	설명	안전 장치
Nmap 결과 파서	XML/텍스트 결과를 읽어 열린 포트와 서비스 표를 만든다.	입력 파일만 처리하고 네트워크 요청 없음
웹 헤더 점검기	허가된 URL의 보안 헤더 존재 여부를 확인한다.	도메인 allowlist와 요청 속도 제한
취약점 후보 정리기	스캐너 결과를 Finding 템플릿으로 변환한다.	민감 데이터 자동 마스킹
리포트 생성기	Evidence/Finding 객체를 Markdown/PDF로 출력한다.	원본 로그와 요약본 분리
학습 랩 대시보드	내 실습 결과를 시간순으로 모아 학습 진도를 시각화한다.	외부 타겟 입력 차단

9.4 최종 체크리스트

체크 항목	완료 기준
범위 승인	테스트 대상, 기간, 제외 범위, 중지 조건이 문서화되어 있다.
도구 선택	각 도구를 왜 쓰는지 설명할 수 있다.
실행 기록	명령, 옵션, 시간, 결과 파일이 남아 있다.
결과 해석	스캐너 결과를 오탐/실제 위험으로 구분했다.
영향 평가	기술 취약점을 업무 위험으로 번역했다.
방어 권고	수정 조치와 재검증 방법을 함께 제시했다.

자동화 안전	스코프 검증, 속도 제한, 로그, 민감정보 마스킹이 있다.
보고서 품질	경영진 요약과 기술 부록을 분리했다.

CHAPTER A

용어 사전

메모를 읽을 때 자주 등장하는 핵심 개념

용어	설명
Reconnaissance	공격 표면을 파악하기 위한 정보 수집 단계.
OSINT	공개 출처를 통해 도메인, 조직, 이메일, 인프라 정보를 수집하는 활동.
Port Scan	호스트에서 열려 있는 네트워크 포트를 확인하는 과정.
Service Banner	서비스가 노출하는 이름/버전/서버 정보.
Vulnerability	보안 약점. 설정 오류, 버그, 설계 결함, 운영 미비를 포함.
Exploit	취약점을 이용해 영향 가능성을 검증하는 코드나 절차.
Payload	취약점 이용 후 실행되는 동작 또는 코드.
Privilege Escalation	낮은 권한에서 더 높은 권한으로 올라가는 과정.
Post-Exploit	초기 접근 이후 영향도 확인, 권한/경로 검증, 증거 수집 단계.
Pivoting	이미 접근한 시스템을 경유해 다른 내부 자원 접근 가능성을 검증하는 과정.
Packet Injection	네트워크 프레임을 의도적으로 생성해 전송하는 행위. 무선 보안 평가에서 자주 언급된다.
Handshake	통신 시작 시 인증/키 교환을 위해 주고받는 절차 또는 캡처 데이터.
Brute Force	가능한 조합을 반복적으로 시도하는 방식. 승인 없는 사용은 금지된다.
Wordlist	사전 대입에 쓰는 문자열 목록.
Hash	원문을 고정 길이 값으로 변환한 결과. 암호 저장에는 솔트와 강한 KDF가 필요하다.
MFA	비밀번호 외 추가 인증 요소를 요구하는 다중 인증.
WAF	웹 공격 패턴을 탐지/차단하는 웹 애플리케이션 방화벽.
CSP	브라우저에서 스크립트 실행과 리소스 로딩을 제한하는 보안 정책.
Finding	보고서에 기록하는 개별 발견 사항.
Evidence	발견 사항을 뒷받침하는 증거.
Remediation	취약점을 수정하거나 위험을 낮추는 조치.
False Positive	취약하지 않은데 취약한 것으로 탐지된 오탐.
Scope	테스트가 허가된 대상과 범위.
Dunder Method	__init__처럼 양쪽에 언더스코어 두 개가 붙은 Python 특수 메서드.
__dict__	일반 Python 객체가 속성을 저장하는 딕셔너리형 이름공간.
__slots__	허용할 인스턴스 속성을 제한해 메모리 사용을 줄이는 Python 기능.

CHAPTER B

참고자료

공식 문서와 원문 메모

이 보고서는 사용자가 제공한 수업 메모를 기반으로 구성하고, 일부 정의와 방법론 설명은 공식 문서를 참고해 보강했다.

[R1] Kali Linux Documentation, "What is Kali Linux?". <https://www.kali.org/docs/introduction/what-is-kali-linux/>

[R2] Kali Linux Tools. <https://www.kali.org/tools/>

[R3] NIST SP 800-115, "Technical Guide to Information Security Testing and Assessment".
<https://csrc.nist.gov/pubs/sp/800/115/final>

[R4] OWASP Web Security Testing Guide. <https://owasp.org/www-project-web-security-testing-guide/>

[R5] Python Documentation, "The import system". <https://docs.python.org/3/reference/import.html>

[R6] Python Documentation, "__main__ - Top-level code environment". https://docs.python.org/3/library/__main__.html

[R7] Python Documentation, "Built-in Functions - property, classmethod, staticmethod".
<https://docs.python.org/3/library/functions.html>

[R8] Python Documentation, "Data model - __slots__". <https://docs.python.org/3/reference/datamodel.html>

[S1] 사용자 제공 원문 메모, "260625 - 모의해킹". 업로드 파일: Pasted text(1).txt

마무리

모의해킹 학습의 최종 목표는 "더 강한 공격"이 아니라 "더 정확한 검증과 더 실용적인 방어 개선"이어야 한다. Kali Linux와 Python은 그 목표를 달성하기 위한 도구다. 도구 실행, 결과 해석, 보고서 작성, 자동화 설계를 하나의 흐름으로 묶으면 입문 단계에서도 빠르게 실무형 사고방식에 가까워질 수 있다.