

수업 내용 정리 보고서

작성일: 2026년 6월 25일

챕터 구성

- 모의해킹 라이프사이클 및 단계별 핵심 토픽
- 파이썬 객체지향 기초: 객체 생명주기와 메서드 설계
- 파이썬 객체지향 심화: 캡슐화, 상속 및 메모리 최적화
- 실무 적용을 위한 주요 점검 및 오답 포인트

모의해킹 라이프사이클 및 단계별 핵심 툴셋

모의해킹(Penetration Testing)은 단순히 시스템을 공격하는 행위가 아니라, 대상의 취약점을 식별하고 이를 통해 발생할 수 있는 위험을 평가하여 방어 전략을 수립하는 체계적인 프로세스입니다. 성공적인 모의해킹을 위해서는 각 단계의 목적을 명확히 이해하고, 그 단계에 최적화된 도구를 적재적소에 활용하는 능력이 필수적입니다.

1. 모의해킹의 6단계 프로세스

효과적인 공격과 보고를 위해 현업에서는 일반적으로 다음과 같은 라이프사이클을 따릅니다.

- 대상 선정 (Target Selection):** 공격 범위를 확정하고 허가된 자산인지 확인합니다.
- 정보 수집 (Information Gathering):** 타겟의 네트워크 구조, 서비스, 도메인 등 기초 데이터를 수집합니다.
- 취약점 분석 (Vulnerability Analysis):** 수집된 정보를 바탕으로 시스템의 약점(버전 오류, 설정 미비 등)을 탐색합니다.
- 침투 및 공격 (Exploit):** 발견된 취약점을 이용하여 실제 시스템 제어권을 획득하거나 데이터를 탈취합니다.
- 영향 평가 (Impact Assessment):** 공격 성공 시 비즈니스에 미치는 실제적 피해 규모를 산정합니다.
- 분석 및 문서화 (Analysis & Documentation):** 방어 방안을 도출하고, 발견된 모든 과정을 보고서로 작성합니다.

2. 단계별 핵심 도구(Toolsets) 요약

각 단계에서 가장 빈번하게 사용되는 메이저 툴들을 정리하였습니다. 이 도구들은 칼리 리눅스(Kali Linux) 환경에서 표준적으로 사용됩니다.

단계	주요 도구	핵심 용도 및 특징
정보 수집 (Reconnaissance)	Nmap, Wireshark, Gobuster, Maltego, FFUF	포트 스캐닝, 패킷 분석, 숨겨진 디렉토리 탐색, 관계 시각화, 초고속 웹 퍼징(Fuzzing)
취약점 분석 (Vulnerability Analysis)	Nikto, Burp Suite, WPScan	웹 서버 설정 오류 스캔, HTTP 패킷 변조 및 인터셉트, 워드프레스 전용 취약점 진단
침투 및 공격 (Exploit)	Metasploit, Hydra, SQLmap, BeEF, SET	종합 공격 플랫폼, 무차별 대입(Brute-force), SQL 인젝션 자동화, 브라우저 제어, 사회공학적 공격
권한 상승 및 사후 행위 (Post-Exploit)	LinPEAS, Mimikatz, Responder, NetExec, Chisel	관리자 권한 탈취 스크립트, 윈도우 인증 정보 탈취, 내부망 자격 증명 스푸핑, 네트워크 터널링(Pivoting)

전문가 팁: 정보 수집 단계가 정교할수록 다음 단계인 취약점 분석과 공격의 성공률이 비약적으로 상승합니다. 특히 *Gobuster*나 *FFUF*와 같은 퍼징 도구를 통해 눈에 보이지 않는 관리자 페이지나 백업 파일을 찾아내는 것이 초기 침투의 핵심입니다.

파이썬 객체지향 기초: 객체 생명주기와 메서드 설계

모의해킹 자동화 프로그램을 개발하기 위해서는 파이썬의 객체지향 프로그래밍(Object-Oriented Programming, OOP) 개념을 반드시 마스터해야 합니다. 모든 것을 객체로 다루는 파이썬의 특성을 이해하면, 복잡한 공격 시나리오를 구조화된 코드로 구현할 수 있습니다.

1. 클래스(Class)와 인스턴스(Instance)의 관계

클래스는 객체를 만들기 위한 '설계도'이며, 인스턴스는 그 설계도를 바탕으로 메모리에 실제로 구현된 '실체(객체)'입니다. 예를 들어, '공격 도구'라는 클래스를 만들었다면, 이를 통해 생성된 'Nmap 객체', 'Hydra 객체' 등이 각각의 인스턴스가 됩니다.

2. 객체의 생명주기: 생성자와 소멸자

객체가 메모리에 생성되고 사라지는 과정은 파이썬의 특수 메서드(Magic Method/Dunder Method)를 통해 관리됩니다.

- `__init__(self, ...)` (생성자):** 객체가 생성될 때 자동으로 호출되며, 객체의 초기 상태(변수 값 등)를 설정합니다.
- `__del__(self)` (소멸자):** 객체의 참조 횟수가 0이 되어 메모리에서 해제될 때 자동으로 호출됩니다.

3. 파이썬 메서드의 3가지 유형

메서드는 설계된 목적과 호출 방식에 따라 세 가지로 구분됩니다. 이를 잘못 사용하면 객체의 상태를 제어할 수 없거나 불필요한 메모리가 낭비될 수 있습니다.

종류	데코레이터	첫 번째 인자	주요 용도
인스턴스 메서드	없음	self (객체 자신)	개별 객체의 고유한 데이터(인스턴스 변수)를 읽거나 수정할 때 사용
클래스 메서드	@classmethod	cls (클래스 자체)	클래스 변수를 제어하거나, 객체 생성을 위한 별도의 경로(Factory)를 만들 때 사용
정적 메서드	@staticmethod	없음	클래스와 논리적으로는 연결되어 있으나, 내부 데이터를 사용하지 않는 유틸리티 함수

4. 코드 실행 흐름 분석 실습

아래의 예시 코드는 클래스의 생성, 메서드 호출, 그리고 소멸 과정을 보여줍니다.

```
class A:
    b = 1 # 클래스 변수 (Static 성격)

    def __init__(self):
        print("A 생성자")

    def a(self):
        print(self)

    def __del__(self):
        print("A 소멸자")

    @staticmethod
    def c():
        print("static c")

if __name__ == '__main__':
    a = A()
    a.a()
    del a
    A.c()
    print('끝')
```

위 코드를 실행했을 때의 출력 결과와 그 의미를 분석하면 다음과 같습니다.

```
A 생성자
<__main__.A object at 0x7f1d...>
A 소멸자
static c
끝
```

실행 단계별 상세 해석:

- 1. `a = A():` 인스턴스가 생성되며 `__init__` 이 호출됩니다. (출력: "A 생성자")
- 2. `a.a():` 인스턴스 메서드가 호출됩니다. 파이썬은 내부적으로 `A.a(a)`로 변환하여 실행하며, `self`를 통해 현재 객체의 메모리 주소를 출력합니다.
- 3. `del a:` 변수 `a`를 삭제하여 참조를 끊습니다. 참조 횟수가 0이 되면 가비지 컬렉터가 `__del__`을 호출합니다. (출력: "A 소멸자")
- 4. `A.c():` 정적 메서드는 인스턴스 없이 클래스 이름으로 직접 호출 가능합니다. (출력: "static c")

파이썬 객체지향 심화: 캡슐화, 상속 및 메모리 최적화

전문적인 공격 프로그램을 설계할 때는 코드의 재사용성을 높이고, 데이터의 무결성을 유지하며, 대규모 스캐닝 시 메모리 효율을 극대화하는 기술이 필요합니다.

1. 캡슐화(Encapsulation)와 Getter/Setter

객체 내부의 중요한 변수를 외부에서 직접 수정하지 못하도록 보호하는 것이 캡슐화입니다. 파이썬에서는 변수명 앞에 언더바(`_` 또는 `__`)를 붙여 이를 표현합니다.

단순히 변수를 숨기는 것에 그치지 않고, `@property`와 `@name.setter`를 사용하여 데이터의 유효성을 검증하는 것이 권장되는 방식(Pythonic Way)입니다.

- **장점 1 (데이터 무결성):** Setter 내부에서 조건문을 통해 잘못된 데이터(예: 빈 문자열, 잘못된 IP 형식)가 저장되는 것을 방지할 수 있습니다.
- **장점 2 (하위 호환성):** 처음에는 일반 변수로 설계했다라도, 나중에 로직이 필요할 때 외부 코드의 수정 없이 내부적으로 Getter/Setter를 도입할 수 있습니다.

2. 상속(Inheritance)과 다형성(Polymorphism)

모의해킹 도구의 구조를 설계할 때 매우 유용하게 사용되는 개념입니다.

- **상속:** Parent 클래스의 기능을 Child 클래스가 물려받는 것입니다. 이때 `super()`를 호출하여 부모의 생성자를 반드시 실행해줘야 부모가 가진 자원을 온전히 물려받을 수 있습니다.
- **메서드 오버라이딩(Overriding):** 부모로부터 물려받은 메서드를 자식 클래스에서 용도에 맞게 재정의하는 것입니다. 예를 들어, `Scanner` 클래스의 `scan()`

메서드를 StealthScanner에서 우회 기술을 포함하도록 재정의하여 구현할 수 있습니다.

3. 메모리 최적화: __dict__와 __slots__

파이썬 객체는 기본적으로 새로운 속성을 실시간으로 추가할 수 있도록 __dict__라는 딕셔너리 구조를 가집니다. 이는 매우 편리하지만, 수만 개의 객체를 생성해야 하는 대규모 스캐너 개발 시에는 엄청난 메모리 낭비를 초래합니다.

구분	__dict__ 사용 (기본값)	__slots__ 사용 (최적화)
속성 추가	실시간으로 자유롭게 추가 가능	선언된 변수 외 추가 불가능 (에러 발생)
메모리 효율	낮음 (딕셔너리 오버헤드 발생)	높음 (정해진 크기만큼만 할당)
사용 사례	일반적인 애플리케이션 개발	수만 개의 객체를 생성하는 고성능 툴

실무 적용을 위한 주요 점검 및 오답 포인트

학습한 내용을 실무 코드에 적용할 때 흔히 발생하는 실수와 점검해야 할 핵심 사항을 정리하였습니다.

1. 파이썬 OOP 개발 체크리스트

- **self 누락 확인:** 인스턴스 메서드를 정의할 때 첫 번째 파라미터로 self를 명시했는가? (누락 시 `TypeError` 발생)
- **Static 변수 선언 위치:** 인스턴스마다 고유한 값을 가져야 하는 변수를 클래스 바로 아래(Static)에 선언하지 않았는가? 인스턴스 변수는 반드시 `__init__` 내부에서 self. 변수명으로 선언해야 합니다.
- **Getter/Setter 호출 방식:** @property를 사용한 경우, 호출할 때 `p.name()`이 아닌 `p.name`과 같이 일반 변수처럼 접근하고 있는가?
- **부모 생성자 호출:** 상속을 구현할 때 `super().__init__()`을 호출하여 부모 클래스의 초기화 로직을 누락하지 않았는가?

2. 모의해킹 툴 활용 시 주의사항

- **권한 및 범위 준수:** 모든 모의해킹 도구는 반드시 **사전에 승인된 대상**에 대해서만 사용해야 합니다. 허가되지 않은 대상에 대한 스캐닝 및 공격은 법적 처벌의 대상이 됩니다.
- **도구의 특성 파악:**
 - 단순 정보 수집에는 Nmap과 Gobuster를 사용하십시오.
 - 웹 취약점의 상세 분석이 필요할 때는 Burp Suite를 통해 패킷을 직접 조작하십시오.
 - 내부망 이동(Lateral Movement)이 목적이라면 NetExec나 Chisel을 고려하십시오.
- **시스템 영향도 고려:** Hydra와 같은 무차별 대입 공격이나 FFUF와 같은 고속 퍼징은 대상 서버의 서비스 중단(DoS)을 유발할 수 있으므로 주의 깊게 실행해야 합니다.