

수업 내용 정리 보고서

작성일: 2026년 7월 29일

챕터 구성

- 웹 리다이렉션 메커니즘과 Open HTTP Redirect 취약점
- File Inclusion(LFI/RFI) 원리 및 공격 시나리오
- 보안 강화 및 통합 방어 체크리스트

웹 리다이렉션 메커니즘과 Open HTTP Redirect 취약점

웹 애플리케이션을 운영하다 보면 사용자를 특정 페이지로 이동시켜야 하는 상황이 빈번하게 발생합니다. 이를 **리다이렉션(Redirection)**이라고 하며, 서버가 클라이언트에게 "요청한 페이지가 다른 곳에 있으니 그쪽으로 가라"고 안내하는 메커니즘을 의미합니다. 본 장에서는 리다이렉션의 기본 원리와 이를 악용한 **Open HTTP Redirect** 취약점, 그리고 입력값 검증 미흡으로 인해 발생하는 **File Inclusion** 취약점에 대해 상세히 학습합니다.

1. HTTP 리다이렉션의 메커니즘

리다이렉션은 웹 서버가 클라이언트의 요청을 처리한 후, 응답 헤더를 통해 클라이언트가 다른 URL로 다시 요청을 보내도록 유도하는 방식입니다. 이는 크게 WAS(Web Application Server) 내부에서 처리되는 방식과 웹 서버(Nginx, Apache 등) 단계에서 처리되는 방식으로 나뉩니다.

- **WAS 내부 자동 변환:**
 - **RedirectMatch:** Apache 모듈 등을 사용하여 특정 패턴의 URL을 다른 주소로 영구 이동(301)시킵니다.
 - **RewriteEngine:** Apache의 mod_rewrite 모듈을 사용하여 조건(RewriteCond)에 따라 규칙(RewriteRule)을 정의하고 URL을 변조하거나 리다이렉트합니다.
- **웹 서버(Nginx) 활용:**
 - 80(HTTP) 포트로 들어오는 요청을 443(HTTPS)으로 강제 전환하거나, 특정 경로를 다른 경로로 리다이렉트할 때 사용합니다.
 - 프록시(Proxy) 설정을 통해 외부에는 특정 주소로 보이지만, 실제로는 내부 WAS로 연결되는 '겉데기 WAS' 역할을 수행하게 할 수도 있습니다.

리다이렉션이 발생할 때 클라이언트는 **HTTP 응답 코드(Status Code)**를 통해 서버의 처리 결과를 확인합니다. 주요 코드는 다음과 같습니다.

코드 범위	의미	상세 내용 및 예시
100번대	정보 제공	요청이 접수되었음을 알림
200번대	요청 성공	요청이 성공적으로 처리됨 (개발자 관점의 정상 상태)
300번대	리다이렉션	301 (Moved Permanently): 페이지가 영구적으로 이동됨 302 (Found): 페이지가 임시로 이동됨 304 (Not Modified): 수정되지 않았으므로 캐시 사용 권장
400번대	클라이언트 오류	사용자의 요청이 잘못됨 (예: 401 Unauthorized, 403 Forbidden)
500번대	서버 오류	서버 내부 문제로 요청 처리 실패 (개발자 관점의 오류)

참고: HTTP 응답 코드는 약속된 규약이므로, 개발자가 의도적으로 특정 코드를 보낼 수 있습니다. 예를 들어 법적 검열을 나타내는 **451 Unavailable For Legal Reasons**나 이스터 에그인 **418 I'm a teapot** 같은 특이한 코드도 존재합니다.

2. Open HTTP Redirect 취약점

Open HTTP Redirect란 웹 애플리케이션이 사용자가 입력한 외부 URL을 검증 없이 신뢰하여, 해당 주소로 사용자를 이동시키는 취약점을 말합니다. 공격자는 신뢰할 수 있는 웹 사이트의 도메인을 이용해 사용자를 악성 사이트로 유인할 수 있습니다.

공격의 주요 목적 및 활용 시나리오

1. **보안 솔루션 우회:** 보안 장비가 악성 도메인을 차단하더라도, 이미 신뢰받고 있는 정상적인 도메인 뒤에 리다이렉션 파라미터를 붙이면 필터링을 통과할 가능성이 높습니다.
2. **심리적 기만:** 사용자는 URL의 앞부분(예: naver.com, google.com)만 보고 링크를 클릭하는 경향이 있어, 악성 사이트임에도 의심 없이 접속하게 됩니다.
3. **기술적 제약 우회 (OAuth 토큰 탈취):** 가장 치명적인 시나리오로, OAuth(Open Authorization) 프로토콜의 동작 방식을 악용합니다.

[OAuth 토큰 탈취 메커니즘]

OAuth 인증 과정에서는 인증이 완료된 후 사용자를 돌려보낼 주소인 **redirect_uri**가 사용됩니다. 현대적인 OAuth 2.0 방식은 보안을 위해 사전에 등록된 주소(Exact Match)로만 토큰을 전달하도록 설계되어 있습니다. 하지만 서비스 A에 Open Redirect 취약점이 있다면 공격자는 다음과 같이 행동할 수 있습니다.

- 공격자는 인증 요청 시 **redirect_uri**에 "**서비스 A의 정상 주소 + 공격자 서버 주소**"와 같은 형태의 값을 주입합니다.
- 인증 서버는 서비스 A의 주소가 포함되어 있으므로 정상적인 요청으로 판단하고, 인증을 마친 뒤 **임시 인증 코드(Authorization Code)**를 포함하여 서비스 A로 사용자를 보냅니다.
- 이때 서비스 A의 리다이렉션 취약점이 작동하여, 브라우저는 임시 코드가 붙은 상태로 공격자의 서버로 이동하게 됩니다.
- 공격자는 탈취한 임시 코드를 사용하여 실제 Access Token을 교환받고, 사용자의 계정 세션이나 정보를 탈취할 수 있습니다.

또한, 리다이렉션 과정에서 발생하는 **Referer 헤더**를 통해서도 민감한 정보가 노출될 수 있습니다. 브라우저는 리다이렉션 시 "어디에서 왔는지"를 알려주기 위해 이전 페이지의 URL을 Referer 헤더에 담아 전송하는데, 이 URL에 세션 ID나 중요한 파라미터가 포함되어 있다면 공격자가 이를 획득할 수 있습니다.

Open HTTP Redirect 방어 대책

- **화이트리스트(Whitelisting):** 리다이렉트 가능한 URL 목록을 사전에 정의하고, 해당 목록에 있는 주소만 허용합니다.
- **상대 경로 사용:** 외부 절대 경로(http://...) 대신 내부의 상대 경로(/dashboard)만 허용하여 외부 사이트로의 이탈을 원천 차단합니다.
- **완전 일치 검증:** OAuth 사용 시 와일드카드(*)를 사용한 패턴 매칭 대신, 등록된 URL과 정확히 일치하는지 확인합니다.

3. 입력값 검증 미흡에 따른 추가 취약점: File Inclusion

웹 애플리케이션이 파라미터를 통해 파일 경로를 전달받고 이를 적절히 검증하지 않을 경우, **File Inclusion** 취약점이 발생합니다. 이는 공격자가 서버 내부의 민감한 파일을 읽거나 악성 스크립트를 실행하게 만드는 매우 위험한 취약점입니다.

- **LFI (Local File Inclusion):** 서버 내부의 로컬 파일을 포함시켜 읽는 공격 (예: /etc/passwd 탈취).
- **RFI (Remote File Inclusion):** 외부의 악성 서버에 있는 파일을 다운로드하여 서버에서 실행시키는 공격.

공격자는 파라미터에 ../와 같은 경로 이동 문자열을 삽입하여 상위 디렉토리로 접근하거나, **User-Agent** 헤더 등에 PHP 코드(예: <?php passthru(' 명령어 '); ?>)를 삽입하여 웹셸(Web Shell)을 실행할 수 있습니다.

실습: LFI 취약점 공격 수행

DVWA 환경에서 page 파라미터에 경로 이동 문자열을 사용하여 시스템 파일에 접근하는 실습을 진행하였습니다. 공격자는 다음과 같은 페이로드를 입력합니다.

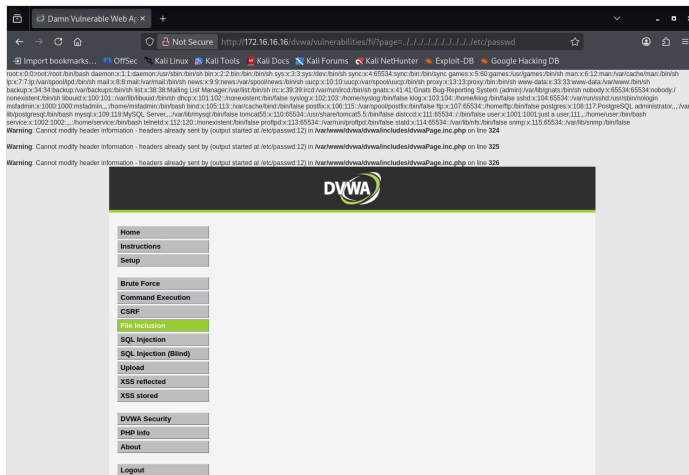
```
http://172.16.16.16/dvwa/vulnerabilities/fi/?page=../../../../../../../../../../../../etc/passwd
```

실습 결과, 웹 페이지 상단에 시스템의 사용자 계정 정보가 담긴 /etc/passwd 파일 내용이 출력되는 것을 확인할 수 있었습니다.

실습 화면 해석

- **입력 값:** page=../../../../../../../../../../../../etc/passwd
- **확인된 결과:** 서버 시스템의 /etc/passwd 파일 내용 출력
- **오류 메시지:** Warning: Cannot modify header information... (파일 내용이 이미 출력되어 헤더를 수정할 수 없다는 경고)

URL의 경로 탐색 파라미터와 화면 상단에 출력된 passwd 파일 데이터 및 PHP Warning 메시지



이 절의 실습 결과를 확인하는 화면

위 화면은 공격자가 입력한 경로 이동 문자열이 필터링되지 않고 시스템에 전달되어, 원래 의도된 **include.php**가 아닌 시스템 핵심 파일이 실행되었음을 증명합니다. 또한, 출력된 텍스트 상단의 Warning 메시지는 파일 내용이 이미 브라우저로 전송되기 시작했기 때문에 서버가 추가적인 HTTP 헤더(Cookie 등)를 설정할 수 없는 상태를 보여줍니다.

File Inclusion 방어 대책

- **입력값 필터링:** ../, ./와 같은 경로 이동 문자열을 엄격히 차단합니다.
- **함수 사용 주의:** include(오류 시 계속 실행)와 require(오류 시 중지)의 차이를 이해하고, eval과 같은 위험한 함수 사용을 지양합니다.
- **PHP 설정 강화:** php.ini에서 display_errors = Off로 설정하여 공격자에게 시스템 경로 등 민감한 정보가 노출되지 않도록 합니다.
- **사전 검증(Fuzzing):** 개발 단계에서 사용자 입력값이 파일 경로로 유효한지, 허용된 디렉토리 내에 있는지 퍼징(Fuzzing) 등을 통해 사전에 검증해야 합니다.

종합 점검 체크리스트

웹 애플리케이션 개발 및 보안 점검 시 다음 항목을 반드시 확인해야 합니다.

1. 리다이렉션 관련:

- 사용자가 입력한 URL이 리다이렉트 대상인가? (YES → 위험)
- 리다이렉션 대상 URL이 화이트리스트에 등록되어 있는가?

- 상대 경로만을 허용하도록 설계되었는가?

2. OAuth 관련:

- `redirect_uri`가 사전 등록된 주소와 100% 일치하는가? (와일드카드 사용 금지)

3. 파일 포함 관련:

- 사용자 입력값이 파일 경로 파라미터로 사용되는가?
- `../`와 같은 경로 이동 문자열에 대한 필터링이 존재하는가?
- 시스템 에러 메시지가 사용자에게 그대로 노출되고 있지는 않은가?

File Inclusion(LFI/RFI) 원리 및 공격 시나리오

1. File Inclusion의 정의와 분류

File Inclusion 취약점이란 웹 애플리케이션이 외부에서 입력받은 값을 검증 없이 파일 경로로 사용하여, 의도하지 않은 파일을 서버 내부에서 읽거나 실행할 때 발생하는 보안 약점입니다. 공격자는 이 취약점을 통해 서버 내의 민감한 설정 파일이나 시스템 파일을 탈취하거나, 악성 스크립트를 실행시켜 서버의 제어권을 획득할 수 있습니다.

공격 대상이 되는 파일의 위치에 따라 크게 두 가지 유형으로 분류됩니다.

- **LFI (Local File Inclusion):** 공격자가 서버 내부(Local)에 이미 존재하는 파일을 포함하도록 유도하는 공격입니다. 예를 들어, 사용자 계정 정보가 담긴 `/etc/passwd`나 웹 애플리케이션의 설정 파일 등을 읽어오는 행위가 이에 해당합니다.
- **RFI (Remote File Inclusion):** 공격자가 외부(Remote)에 위치한 악성 서버의 스크립트 주소를 입력하여, 서버가 해당 파일을 다운로드하고 실행하도록 만드는 공격입니다. 이는 주로 악성코드 유포나 웹셸(Web Shell) 실행 시나리오에서 활용됩니다.

2. 공격 메커니즘과 주요 시나리오

File Inclusion 공격은 주로 `include` 또는 `require`와 같은 PHP 함수가 사용자 입력값(GET, POST 파라미터 등)을 직접 인자로 받을 때 발생합니다.

대표적인 공격 방법은 다음과 같습니다.

- **경로 이동(Path Traversal):** `../`(상위 디렉토리 이동 문자열)을 반복적으로 사용하여 웹 루트 디렉토리를 벗어나 시스템의 최상위 디렉토리까지 접근하는 방식입니다.
- **웹셸(Web Shell) 실행:** 공격자가 서버에 악성 스크립트(예: `name.php`)를 업로드한 후, File Inclusion 취약점을 이용해 해당 파일을 호출함으로써 시스템 명령어를 실행할 수 있는 권한을 얻습니다.
- **헤더 변조 공격:** 파라미터뿐만 아니라 User-Agent와 같은 HTTP 헤더 필드에 `<?php passthru('명령어'); ?>`와 같은 구문을 삽입하여 명령어를 실행시키는 방식도 가능합니다.

3. 실습: DVWA를 이용한 LFI 공격 수행

DVWA(Damn Vulnerable Web Application) 환경에서 File Inclusion 취약점을 이용하여 리눅스 시스템의 핵심 파일인 `/etc/passwd` 내용을 탈취하는 실습을 진행하였습니다. 공격자는 `page` 파라미터에 경로 이동 문자열을 사용하여 웹 디렉토리의 제한을 우회하였습니다.

실습 과정에서 입력한 공격 페이로드는 다음과 같습니다.

`http://172.16.16.16/dvwa/vulnerabilities/fi/?page=../../../../../../../../../../../../etc/passwd`

이 페이로드는 현재 위치에서 상위 디렉토리로 계속 이동하여 시스템 루트에 도달한 후, 해당 경로에 있는 `etc/passwd` 파일을 읽어오도록 설계되었습니다. 요청 결과, 브라우저에는 시스템의 사용자 계정 정보가 포함된 파일 내용이 노출되었습니다.

이 화면은 LFI(Local File Inclusion) 공격이 성공하여 서버 내부의 민감한 시스템 파일이 외부로 유출되었음을 증명합니다. 특히 화면에 나타난 **'Warning: Cannot modify header information'** 메시지는 매우 중요한 분석 단서를 제공합니다. 이는 `include` 함수를 통해 `/etc/passwd` 파일의 내용이 이미 브라우저로 출력(Output)된 이후에, PHP 스크립트가 HTTP 헤더(예: 쿠키나 리다이렉트 정보)를 수정하려고 시도했기 때문에 발생하는 오류입니다. 즉, 파일 내용이 먼저 출력되었다는 점을 통해 공격이 성공적으로 이루어졌음을 확인할 수 있습니다.

4. 취약점 방어 및 보안 대책

File Inclusion 취약점을 방어하기 위해서는 개발 단계부터 입력값 검증과 서버 설정 최적화가 병행되어야 합니다.

구분	대책 내용	상세 설명
코드 레벨 방어	입력값 화이트리스트팅 (Whitelisting)	허용된 파일 목록만을 배열로 관리하고, 그 외의 입력은 모두 차단합니다.
경로 제어	경로 이동 문자열 필터링	../, ./ 와 같은 경로 조작 문자열이 포함되어 있는지 검사합니다.
함수 구분	include vs require	include는 오류 발생 시 스크립트를 계속 실행하지만, require는 실행을 중단하므로 중요 파일 처리 시에는 require가 권장됩니다.
서버 설정	php.ini 설정 최적	display_errors = Off로 설정하여 공격자에게 시스템 정보가 노출되는 것을 방지합니다.

또한, 시스템의 권한을 제한하기 위해 **chroot** 기술을 사용하여 프로세스가 특정 디렉토리를 루트(root)로 인식하게 함으로써, 상위 디렉토리에 대한 접근 자체를 물리적으로 차단하는 구조적 방어 전략을 검토할 수 있습니다.

5. 학습 내용 요약 및 점검 리스트

File Inclusion 취약점 점검 및 방어 시 다음 항목을 반드시 확인해야 합니다.

- **점검 항목:**
 - 사용자 입력값이 `include`, `require`, `eval` 등의 함수 인자로 직접 전달되는가?
 - 입력값에 `../`와 같은 경로 이동 문자열에 대한 필터링이 존재하는가?
 - 서버 설정에서 에러 메시지가 상세하게 출력되고 있는가? (`display_errors` 확인)
- **실무 포인트:**
 - 단순히 특수문자를 막는 것을 넘어, 허용된 파일 목록(Whitelist)만을 처리하도록 설계하는 것이 가장 강력한 방어 수단입니다.
 - RFI 공격을 막기 위해서는 `php.ini`에서 `allow_url_include` 설정을 `Off`로 유지해야 합니다.

보안 강화 및 통합 방어 체크리스트

웹 애플리케이션 보안의 핵심은 **사용자로부터 입력받은 데이터에 대한 불신**입니다. 오늘 학습한 File Inclusion과 Open HTTP Redirect 취약점은 모두 개발자가 사용자의 입력값(파라미터)을 검증 없이 신뢰하여 발생합니다. 이러한 취약점들은 단순히 파일을 보여주는 것을 넘어, 서버 권한 탈취나 사용자 계정 도용 (OAuth)으로 이어질 수 있으므로 설계 단계부터 통합적인 방어 전략이 필요합니다.

1. File Inclusion 취약점 방어 전략

File Inclusion 취약점은 서버 내부의 파일을 포함(include)하거나 외부의 악성 스크립트를 가져와 실행하는 공격입니다. 공격 방식에 따라 다음 두 가지로 구분됩니다.

- **LFI (Local File Inclusion):** 서버 내부에 이미 존재하는 민감한 파일(예: `/etc/passwd`, 설정 파일 등)에 접근하는 공격
- **RFI (Remote File Inclusion):** 공격자가 제어하는 외부 서버의 악성 스크립트를 다운로드하여 서버에서 실행시키는 공격

실습을 통해 파라미터 변조를 통한 LFI 공격이 어떻게 수행되는지 확인하였습니다. 공격자는 경로 이동 문자열(`../`)을 반복 사용하여 루트 디렉토리까지 접근한 뒤, 시스템의 핵심 파일에 접근을 시도합니다.

실습 화면 해석

공격자는 `page` 파라미터에 상위 디렉토리 접근을 위한 `../` 문자를 삽입하여 서버의 `/etc/passwd` 파일을 호출하였습니다.

- **요청 URL:** `.../fi/?page=../../../../../../../../../../../../../../../../etc/passwd`
- **출력 결과:** `root:x:0:0:root:/root:/bin/bash`와 같은 시스템 사용자 계정 정보가 브라우저에 노출됨
- **오류 메시지:** `Warning: Cannot modify header information...` 발생. 이는 이미 `/etc/passwd` 파일의 내용이 출력되어 버렸기 때문에, PHP가 이후에 헤더 정보를 수정할 수 없음을 의미함

위 화면은 경로 제한이나 파라미터 필터링이 부재할 때 발생하는 전형적인 사례입니다. 이를 방어하기 위한 체크리스트는 다음과 같습니다.

1. **입력값 필터링 및 화이트리스트팅:** `../`, `./`와 같은 경로 이동 문자열을 차단하거나, 허용된 파일 목록(Whitelist) 내에서만 호출이 가능하도록 로직을 구성해야 합니다.
2. **함수 사용 주의:** 사용자 입력값을 `include`, `require`, `eval` 등의 함수에 직접 전달하는 것을 금지해야 합니다.
 - `include`: 오류 발생 시 에러를 출력하고 스크립트를 계속 실행하므로 공격이 지속될 위험이 있음
 - `require`: 오류 발생 시 실행을 중지하므로 `include`보다는 상대적으로 안전할 수 있으나, 근본적인 해결책은 아님
3. **chroot 활용:** 프로세스의 루트 디렉토리를 현재 설정 페이지로 제한하여, 공격자가 상위 디렉토리로 접근하는 것을 물리적으로 차단합니다.
4. **서버 설정(PHP.ini):** `display_errors = Off`로 설정하여, 공격자가 에러 메시지를 통해 서버의 내부 구조를 유추(Fuzzing)하지 못하도록 해야 합니다.

2. Open HTTP Redirect 및 OAuth 보안 방어 전략

Open HTTP Redirect는 웹 애플리케이션이 사용자가 입력한 외부 URL로 검증 없이 리다이렉션(Redirection)할 때 발생합니다. 이는 단순히 엉뚱한 사이트로 이동하는 것을 넘어, 다음과 같은 심각한 위협을 초래합니다.

- **보안 솔루션 우회:** 신뢰할 수 있는 도메인을 거쳐가기 때문에 보안 필터링을 통과할 수 있습니다.
- **사회 공학적 공격:** 사용자가 이미 알고 있는 도메인을 통해 이동하므로 악성 사이트임을 인지하기 어렵습니다.
- **OAuth 토큰 탈취:** 인증 서버가 인증 코드를 전달하는 `redirect_uri`를 공격자의 서버로 변조하여 사용자의 세션이나 액세스 토큰을 가로챌 수 있습니다.

특히 OAuth 프로토콜을 사용하는 환경에서는 '**신뢰할 수 있는 주소로만 토큰을 보낸다**'는 원칙이 깨질 경우 서비스 계정이 도용될 수 있습니다. 공격자는 서비스 A의 Open Redirect 취약점을 이용해, 인증 서버가 발급한 임시 코드를 자신의 서버로 가로채는 시나리오를 구성할 수 있습니다.

이를 방지하기 위한 보안 강화 포인트는 다음과 같습니다.

1. **URL 화이트리스트:** 리다이렉션될 대상 URL을 사전에 정의된 목록과 대조하여 검증해야 합니다.
2. **상대 경로 사용 권장:** 외부 도메인으로의 이동을 차단하기 위해, 외부 주소(`http://...`)가 아닌 내부 경로(`/dashboard` 등)만 허용하도록 설계합니다.
3. **OAuth 설정 최적화:** `redirect_uri`를 등록할 때 와일드카드(`*`)를 사용하여 범위를 넓히지 말고, 반드시 **완전 일치(Exact Match)**하는 URL만 허용해야 합니다.
4. **Referer 헤더 모니터링:** 리다이렉션 시 발생하는 `Referer` 헤더에 민감한 정보(URL 파라미터 등)가 포함되어 유출되지 않는지 점검해야 합니다.

3. 통합 방어 체크리스트 (Summary)

개발 및 운영 단계에서 반드시 확인해야 할 보안 점검 항목을 정리합니다.

구분	점검 항목	세부 내용 및 권장 사항
입력값 검증	경로 이동 문자열 차단	<code>../</code> , <code>./</code> , <code>%2e%2e%2f</code> 등 변조된 경로 패턴 필터링
	URL 화이트리스트	허용된 도메인 및 내부 경로 외의 리다이렉션 차단
코드 구현	위험 함수 사용 제한	사용자 입력값을 <code>include</code> , <code>require</code> , <code>eval</code> 에 직접 전달 금지
	OAuth URI 매칭	<code>redirect_uri</code> 설정 시 와일드카드 사용 금지 및 완전 일치 적용
서버 설정	에러 노출 제어	<code>php.ini</code> 내 <code>display_errors = Off</code> 설정으로 정보 유출 방지
	디렉토리 격리	필요 시 <code>chroot</code> 를 사용하여 프로세스의 접근 범위 제한

실무 포인트: 보안은 "막는 것"만큼이나 "어떻게 검증할 것인가"가 중요합니다. 파일 포함 기능이 반드시 필요하다면, 파일명을 파라미터로 받는 대신 `page_id=1`과 같이 숫자를 받고 서버 내부의 매핑 테이블을 통해 실제 파일명을 결정하는 방식이 가장 안전합니다.