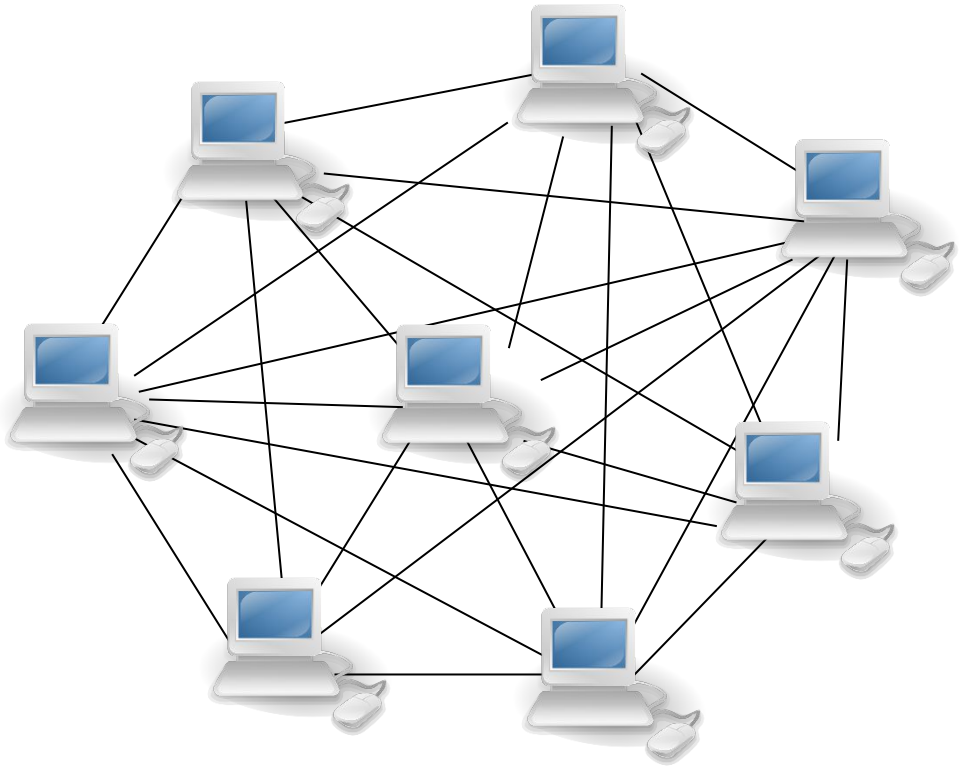
시큐어코딩

인터넷 작동 방식

▶ 인터넷

- 전 세계의 컴퓨터들이 네트워크를 통해 연결되어 정보를 공유하는데 목적을 두며 프로토콜을 이용하여 통신(TCP, IP, 도메인 이름, HTTP)



▶ 프로토콜 스위트

- 인터넷 초기 데이터 교환에 신뢰성이 없었기에 이를 해결 하고자 전송 제어 프로토콜(TCP) 개발하였다.
- 이러한 TCP를 포함한 20여개의 네트워크 프로토콜을 프로토콜 스위트라 한다.

▶ 프로토콜 스위트

TCP에서 발신측에서는 보내는 메시지(패킷)에 목적지의 주소를 담아 보내진다.
수신측에서는 전송받은 메시지(패킷)를 순서에 맞게 재조립 한다.

데이터 패킷은 인터넷 프로토콜(ip) 주소로 전송된다.
ip 주소는 고유해야 한다.

사람들이 기억하기 쉽게 ip주소 대신 DNS(Domain Name System)을 이용하여 주소를 변환해 이용한다.

▶ 상태 저장 연결

- HTTP는 어떤 사용자에서 어떤 요청이 오는지 구분하지 못한다.
- 현대 사이트는 다른 페이지들과 상호작용 하면서 활동을 추적해야 할 필요가 있어 HTTP 통신 상태가 지속적으로 유지되기 위해 핸드셰이크를 수행한다.
- 세션추적

▶ 암호화

- HTTP 요청&응답시 중간공격이 가능하므로 암호화가 필요하다.
- 인코딩, TLS(Transport Layer Security)
- TLS를 이용한 HTTP통신을 HTTP Secure(https)라 한다.

브라우저 작동 방식

▶ 웹 페이지 렌더링

렌더링 파이프라인(Rendering Pipeline): html을 화면에 보이는 시각적 표현으로 변환하는 것을 담당하는 웹 브라우저 소프트 웨어

- 브라우저가 HTTP 응답을 수신 하면 HTML 페이지를 구문 분석(DOM)
- 각 DOM 노드는 대략 원본과 동일하고 트리 구조로 설명
- CSS파일 등을 이용해 스타일링 정보 결정

브라우저는 DNS 주소를 확인 및 캐시, 보안 인증서 해석 및 검증, 웹 서버 지시에 따라 쿠키 저장 및 전송 등의 다른 작업도 진행 한다.

웹 서버 작동 방식

▶ 정적 자원

- HTML 파일, 이미지 파일 등
- 인터넷 초기 웹사이트는 정적 자원으로 구성, 개발자들은 손으로 HTML 파일을 코드화 했고 사이트는 개별 HTML 들로 구성됐다.

▶ 정적 자원

✔ URL 결정

url의 자원 이름 및 요청된 대로 자원 파일을 반환

```
/images/test_ing.png
```

✔ 콘텐츠 전송 네트워크(CDN)

정적자원 복사본을 전 세계 데이터 센터에 저장 후 전달

✔ 콘텐츠 관리 시스템

제작 도구를 제공하는 콘텐츠 관리 시스템 이용(WYSIWYG 등)

▶ 동적 자원

대부분의 현대 웹사이트는 동적 자원 사용

해석하는 코드, 템플릿 등.

▶ 동적 자원

✔ 템플릿

직관적이지 않은 동적 자원를 구성하는 코드때문에 템플릿 사용하여 페이지 작성

✔ 데이터 베이스

- 웹 서버는 동적 자원 코드 실행 시 종종 데이터베이스에서 데이터 로드
- 웹 서버와 데이터베이스 사이의 인터페이스는 해커들의 목표 대상이 된다.
- SQL 데이터 베이스, NoSql 데이터 베이스

프로그래머 작동 방식

▶ 소프트웨어 개발 생명주기

개발 과정에서 버그 및 보안 취약점의 위험 줄이는 개발 습관을 들인다.

소프트웨어 개발 생명주기에 따라 코드 작성을 한다.

1. 설계 및 분석
2. 코드 작성
3. 배포 전 테스트
4. 릴리스 프로세스
5. 릴리스 후 테스트 및 관찰

(추가로 종속성 관리도 고려해야 한다.)

인젝션 공격

▶ 인젝션 공격

✔ 인젝션 공격

외부 코드를 주입해 애플리케이션을 통제하거나 민감한 정보를 읽으려 하는 행위

- SQL 인젝션 공격
- 커맨드 인젝션 공격
- 파일 업로드 취약점

▶ SQL 인젝션 공격

✔ SQL

SQL(Structured Qeury Language)은 관계형 데이터베이스에서 데이터 및 데이터 구조를 추출하고 조작하는데 사용하는 프로그래밍 언어이다.

✔ SQL 인젝션 공격

- 웹 서버가 데이터베이스 드라이버에 전달하는 sql문을 불안정하게 구성할 때 발생한다.
- HTTP 요청을 통해 파라미터를 전달하고 드라이버가 개발자의 의도와 다른 작업을 수행할 수 있다.

  - 이메일 매개 변수를 billy@gmail. com' -- 로 전달

```
String sql = "SELECT * FROM users WHERE email='" + email +
             "' AND encrypted_password='" + password + "'";
```

```
"SELECT * FROM users
WHERE email='billy@gmail. com' -- AND encrypted _password=' Z5DSA92H0'"
```

▶ 인젝션 공격 조치 방안

✔ 매개변수화 된 구문 사용

바인딩 매개변수를 사용해 SQL 문을 구성. Prepared Statement

✔ 객체 관계 매핑 사용

SQL문의 명시적 구성을 추상화하고 객체 관계 매핑(Object Relational Mapping)을 사용

▶ 커맨드 인젝션 공격 & 파일 업로드 취약점

웹 어플리케이션이 시스템 명령어를 실행할 때 취약점을 이용하여 공격자가 의도한 시스템 명령을 실행하게 만드는 공격

▶ 커맨드 인젝션 공격 & 파일 업로드 취약점

✔ 커맨드 인젝션 공격

웹 어플리케이션이 시스템 명령어를 실행할 때 취약점을 이용하여 공격자가 의도한 시스템 명령을 실행하게 만드는 공격

- 적절한 이스케이프 문자를 활용하여 조치

✔ 파일 업로드 취약점

웹 어플리케이션에서 사용자로 부터 파일을 업로드 받을 때, 제대로된 검증을 하지 않아 악성 파일이 서버에 업로드하고 실행할 수 있는 취약점

- CDN에 파일 업로드를 하거나 업로드된 파일 내용을 체크하여 조치

크로스 사이트 스크립팅 공격

▶ 크로스 사이트 스크립팅 공격

크로스 사이트 스크립팅(Cross-Site Scripting,XSS)

악의적인 자바스크립트를 사용자의 브라우저에서 실행하도록 하는 공격

- 스토어드 크로스 사이트 스크립팅 공격
- 리플렉티드 크로스 사이트 스크립팅 공격
- DOM 기반 크로스 사이트 스크립팅 공격

▶ 스토어드 크로스 사이트 스크립팅 공격(Stored XSS)

데이터 베이스에 자바스크립트 코드를 주입하여 브라우저에서 HTML코드가 렌더링 될때 스크립트가 작성되고 실행되게 하는 공격 유형

```
<script>alert('XSS')</script>
```

다른 사용자가 이 스크립트가 출력된 페이지를 방문할 때 마다 스크립트 실행

▶ 조치방안

✔ 이스케이프 처리

사용자 입력을 서버에서 철저히 검증하고

HTML, JavaScript, CSS 등에서 특수문자를 적절히 이스케이프 처리

✔ 콘텐츠 보안 정책 구현

콘텐츠 보안 정책(Content Security Policy)을 사용해 사이트에서 자바스크립트 실행을 차단

```
response.setHeader("Content-Security-Policy", "default-src 'self'; script-src 'self' https://test.com;");
```

▶ 리플렉티드 크로스 사이트 스크립팅 공격(Reflected XSS)

HTTP 요청의 URL이나 GET 파라미터에 악의적인 자바스크립트를 주입하여

이를 포함한 링크를 피해자가 클릭하면 스크립트가 실행되는 공격

```
http://example.com/search?q=<script>alert('XSS')</script>
```

이 링크를 클릭하면 브라우저는 스크립트를 실행

✔ 조치 방안

XSS 공격 취약점을 조치하는 것과 동일한 방법으로 이스케이프 조치

▶ DOM 기반 크로스 사이트 스크립팅 공격

URI 조각으로 악의적인 자바스크립트를 웹 페이지로 삽입하여 실행하는 공격

클라이언트 측에서만 발생하며 브라우저가 DOM을 조작할때 발생

```
https://example.com/page.html#section1
```

일치하는 id 속성이 있다면 페이지를 연 후 해당 태그로 스크롤

✔ 조치 방안

URI 조각에서 동적 콘텐츠 이스케이프 조치

사이트간 요청 위조 공격

▶ 사이트 간 요청 위조 공격

사이트 간 요청 위조 공격(Cross-Site Request Forgery,CSRF)

사용자가 인증된 세션을 가지고 있는 웹 애플리케이션에서, 공격자가 사용자의 의지와는 상관없이 특정 요청을 보내도록 유도하여 악의적인 작업을 수행하게 만드는 공격

✔ 공격 분석

일반적으로 상태를 변경하는 GET 요청을 구현하는 사이트를 공격

GET 요청은 요청 내용 전체를 URL에 포함하는 유일한 HTTP 요청 유형이라 CSRF 공격에 취약

▶ 조치 방안

✔ REST 원칙을 따른다

GET 요청이 서버 상태를 변경하지 않도록 한다.

자원을 가져온다면 GET , 서버에 새 개체를 만든다면 PUT, 자원을 수정을 한다면 POST, 자원을 삭제를 한다면 DELETE 요청을 사용 ⇒ REST

✔ 안티 CSRF 공격 쿠키 구현

클라이언트가 요청할 때 쿠키와 함께 CSRF 토큰을 전송하게 하고, 서버 측에서 이를 검증하여 요청의 정당성을 확인하는 방식

쿠키 : http 헤더의 브라우저와 웹서버간에 전달되는 작은 텍스트

```
<input type="hidden" name="_xsrf" value="123123e1fa">
```

서버가 헤더값에 쿠키를 포함하여 응답을 반환하면 브라우저는 다음 요청에 동일한 정보를 담아 돌려준다.

▶ 조치 방안

✔ SameSite 쿠키 속성 사용

쿠키를 설정 할 때 SameSite 속성을 지정하여 다른 웹사이트에서 생성된 사이트에 대한 요청의 쿠키 제어

```
Set-Cookie: _xsrf=123123e1fa; SameSite=Strict;
```

Strict(같은 사이트),Lax(같은사이트&일부안전한크로스사이트요청),None(모든) 속성값 활용

✔ 추가적인 조치

중요 작업 수행 시 로그인 세부 정보 등을 다시 확인. 재인증

인증 손상

▶ 인증 구현

인증은 HTTP의 일부.

인증 문제가 발생하면 응답에 401 상태코드 담아 반환

- 기본인증 방식
- 다이제스트 인증방식

✔ 인증

- HTTP 네이티브 인증 : HTTP에 내장
- 네이티브하지 않은 인증 : 개발자 원하는 대로 작성

▶ 조치 방안

1. 서트파티 인증 사용

   인증 시스템을 직접 구현하는 대신 서트파티 서비스 사용

2. 자체 인증 시스템 보호

   이름,암호등을 수동 선택할 수있는 방법도 필요

3. Single Sign-on과 통합

세션 하이제킹

▶ 세션 작동 방식

사용자가 HTTP에서 자신을 인증하면 웹 서버는 로그인 프로세스 중에 사용자에게 세션 식별자를 할당

웹 서버는 각 요청과 함께 제공된 세션ID를 인식해 사용자에게 매핑하고 작업 수행

✔ 서버 측 세션

웹 서버는 세션 상태를 메모리에 유지 하고 웹 서버와 브라우저는 세션 식별자를 앞뒤로 전달하는데 이를 서버측 세션이라 한다.

```
// 세션을 가져오거나 없으면 새로 생성
HttpSession session = request.getSession();

// 세션 ID를 가져옴
String sessionId = session.getId();
```

```
// 요청에서 모든 쿠키 가져오기
Cookie[] cookies = request.getCookies();
String sessionId = null;

// 쿠키가 존재하는지 확인
if (cookies != null) {
    for (Cookie cookie : cookies) {
        // 세션 쿠키 확인 (세션 쿠키의 이름은 JSESSIONID)
        if (cookie.getName().equals("JSESSIONID")) {
            sessionId = cookie.getValue();
            break;
        }
    }
}
```

▶ 세션 작동 방식

✔ 클라이언트 측 세션

서버에 상태를 저장하는 서버 측 세션과는 달리, 클라이언트 측 세션은 사용자의 브라우저에 직접 데이터를 저장하고, 클라이언트가 서버에 요청할 때마다 해당 데이터를 활용

쿠키, 웹스토리지(로컬스토리지,세션스토리지), JWT

▶ 세션 하이제킹 방법

✔ 쿠키 도난

쿠키 헤더의 값을 도용해 세션 하이재킹을 수행

```
Set-Cookie: session_id=1233210123; HttpOnly; Secure; SameSite=Lax;
```

쿠키 보안 속성 설정을 통해 보안 강화

1. 크로스 사이트 스크립팅 공격 : 브라우저에 주입된 자바스크립트를 사용해 쿠키를 읽고 제어
2. 중간자 공격
3. CSRF 공격 : 공격하려는 대상을 속여 사이트 링크 클릭

▶ 세션 하이제킹 방법

✔ 쿠키 고정

URL에 추가된 세션ID를 가로채 공격.

인터넷 초기 브라우저는 쿠키를 지원하지 않아 URL에 세션ID 노출되는 경우가 있다.

✔ 취약한 세션 id 활용

공격자가 취약한 세션ID에 액세스하여 사용자 세션을 가로챌 수 있다.

추측이 어려운 세션ID를 사용하는 지등의 과정이 필요하다.

권한

▶ 권한 상승

✔ 수직적 확대

자신의 계정보다 더 넓은 권한을 가진 계정에 액세스 할 수 있다.

✔ 수평적 확대

자신의 계정과 유사한 권한을 가진 다른 계정에 액세스 할 수 있다.

▶ 접근 제어

✔ 접근 제어 전략

- 인증 : 사용자가 사이트로 돌아올 때 사용자를 올바르게 식별
- 사용자가 자신을 확인한 후 수행해야 하는 작업과 해서는 안 되는 작업 결정
- 권한: 사용자가 작업을 수행하려고 할 때 권한 평가 확인

✔ 인증 모델 설계

사용자에게 적용하는 방법을 문서화 하는것이 중요하다.(합의된 규칙 집합)

권한 부여 규칙을 모델링하는 방법: 접근제어목록, 화이트&블랙리스트,역할 기반 접근제어, 소유권 기반 접근 제어

▶ 접근 제어

✔ 접근 제어 구현

액세스 규칙을 정의한 후에는 코드로 규칙을 구현해야 한다.

✔ 접근 제어 테스트

공격자 처럼 허점을 찾아 테스트를 진행 한다.

✔ 감사용 기록 추가

사용자가 사이트를 탐색할 때 로깅 문을 추가하여 작업을 수행할 때마다 기록

▶ 디렉터리 접근 공격

✔ 접근 제어 전략

URL에 파일 경로를 설명하는 매개변수가 포함되어 있다면 공격자는 디렉터리로 접근 공격이 가능

- 파일경로 및 상대 파일 경로 : 서버측 코드로 공격자가 파일 이름 대신 상대 파일 경로를 전달 및 평가할 수 있다면 파일 시스템에서 파일을 제어할 수 있다.
- 디렉터리 접근 공격 분석 : 해커가 파일을 다운로드하려고 임으로 경로로 대체하면 정보가 노출 될 수 있다.

```
?menu=abc.pdf
->
?menu=../../../etc/passwd
```

▶ 디렉터리 접근 공격

✔ 조치 방안

1. 웹 서버 신뢰 : 웹 서버가 정적 콘텐츠 URL을 확인 하는 방법을 숙지.
2. 호스팅 서비스 사용 : CDN 활용
3. 간접 파일 참조 사용 : 일 경로에 해당하는 불투명한 ID를 각 파일에 할당한 다음 모든 URL이 해당 ID로 각 파일을 참조
4. 파일 참조 삭제 : 경로 구분 문자가 포함된 파일 참조를 금지.

정보 누출

▶ 정보 누출

정보를 누출함으로써 웹사이트의 추약점을 광고하고 있을 수 있으니 이 취약점을 즉시 해결

✔ 조치 방안

1. 숨길 수 없는 서버 헤더 사용 안함 : HTTP 응답 헤더를 비활성화
2. 깔끔한 URL 사용 : .jsp, .php와 같은 파일 접미사 사용을 자제
3. 일반 쿠키 매개변수 사용 : 오류 페이지를 활용
4. 클라이언트 측 오류 보고 안 함
5. 자바스크립트 파일 최소화 또는 난독화 : 자바스크립트 코드를 압축하고 난독화를 하여 소스코드를 읽기 어렵게 만든다
6. 클라이언트 측 파일 삭제 : 템플릿 파일과 HTML에서 너무 많은 정보를 제공하는 설명을 삭제

암호화

▶ 인터넷 프로토콜의 암호화

- 인터넷으로 전송되는 메시지는 데이터 패킷으로 분할되고 TCP를 통해 최종목적지로 전달. 수신자 컴퓨터는 전송된 메시지의 TCP 패킷을 원래 메시지로 재조립.
- TCP 통신은 악의적인 서트파티가 패킷을 가로채어 읽는 중간자 공격에 취약하여 TLS등으로 보호해야 하는데 TLS를 사용해 수행도니 HTTP 통신을 HTTPS 통신이라 한다.

✔ 암호화 알고리즘

입력데이터를 가져와서 보안 통신을 시작하고자 하는 두 당사자 간에 공유되는 암호키를 사용해 데이터를 스크램블. 스크램플된 값은 복호화 키가 없으면 해독이 불가능.

- 대칭암호화 알고리즘
- 비대칭암호화 알고리즘
- 해시함수

▶ 인터넷 프로토콜의 암호화

✔ TLS 핸드 셰이크

TLS로 전달되는 데이터 패킷은 블록암호라고 하는 대칭 암호화 알고리즘을 사용하여 암호화

암호 집합: 통신을 보호하는데 사용되는 알고리즘 집합.

1. 키교환알고리즘 : 비대칭 암호화, 서버와 클라이언트가 서로 안전하게 대칭 키(데이터를 암호화하고 복호화하는 데 사용)를 교환
2. 대칭암호화알고리즘: 데이터를 실제로 암호화하고 복호화하는 알고리즘으로, 동일한 키를 양쪽에서 사용
3. MAC 알고리즘: 데이터를 전송하는 동안 그 데이터가 변조되지 않았는지 검증하는 역할.

▶ HTTPS 사용

✔ 디지털 인증서

공용 암호화 키의 소유권을 증명하는데 사용되는 전자 문서

TLS에서 암호화 키를 인터넷 도메인과 연결하는 데 사용

✔ 디지털 인증서 획득

- 키 쌍 및 인증서 서명 요청 생성
- 도메인 검증
- 확장 유효성 검사 인증서
- 인증서 만료 및 해지
- 자체 서명된 인증서

▶ HTTPS 사용

✔ 디지털 인증서 설치

HTTPS를 사용하도록 웹 서버 구성

- 디지털 인증서와 암호화 키는 주로 대부분 웹 서버에 배포
- 표준 HTTPS 포트(443)에서 트래픽을 수신하도록 업데이트
- 표준 HTTP 포트(80)에서 암호화되지 않은 트래픽 처리 방법을 결정

▶ HTTP(및 HTTPS) 공격

✔ 암호화 되지 않은 HTTP 공격 방식

- 무선 라우터
- wi-fi 핫스팟
- 인터넷 서비스 공급자
- 정부 기관

서드파티

▶ 종속성 보호

보안 문제가 공개 되는 즉시 파악하고 소프트웨어를 신속하게 패치해야 한다.

✔ 실행 중인 코드 파악

보안 문제가 공개되는 즉시 파악하고 소프트웨어를 신속하게 패치

종속성 관리 도구, 운영체제 패치, 무결성 검사

✔ 새로운 버전 신속하게 구현할 수 있다

보안 문제에 대응하려면 패치를 신속하게 배포할 수 있어야 한다.

✔ 보안 문제 경계 유지

소셜미디어, 메일, 블로그 자문, 소프트웨어 도구 등을 통해 보안권고사항 빠르게 파악한다.

▶ 구성 보안

소프트웨어, 네트워크, 시스템 등의 설정을 올바르게 구성하고 관리하여 보안 취약점을 예방하는 과정

- 기본 자격 증명 사용 안 함
- 디렉터리 리스팅 비활성화
- 구성 정보 보호
- 테스트 환경 강화
- 보안 관리 프론트 엔드

▶ 사용하는 서비스 보안

액세스 자격 증명을 안전하게 저장하는 방법

✔ api 키 보호

서드 파티 서비스를 사용자가 등록할 때 API키를 발급하는데 이를 안전하게 저장

✔ 웹훅 보호

서비스 공급자가 이벤트가 발생할 때 HTTPS 요청을 보내는 간단한 역API

▶ 공격 벡터로서의 서비스

서드 파티 서비스는 잠재적으로 웹사이트를 공격 할 수 있는 부분이 된다.

- 멀티파이징을 경계악성 코드 전송 방지
- 평판이 좋은 광고 플랫폼 사용
- safeframe을 사용광고 기본 설정 맞춤
- 의심스러운 광고 검토 및 보고

부속품이 되지마라

▶ 이메일 사기

스팸 알고리즘은 이메일에서 악의적인 링크를 찾고 이를 보호하려고 웹 메일 공급자는 유해하다 알려진 도메인의 최신 블랙리스트를 보관한다.

✔ 열린 리다이렉션 방지

사용자를 대상으로 리다이렉션할 목적으로 다른 URL 내에 URL을 인코딩한다면 이러한 인코딩된 URL이 절대 URL이 아닌 상대 URL인지 확인.

✔ 기타 고려 사항

부득이하게 외부 링크를 게시해야 할 경우 구글 세이프 브라우저 API 를 활용하여 유해 사이트 블랙리스트와 대조해보는 등과 같은 방법으로 해결

▶ 이메일에서 악의적인 링크 숨기기

스팸 알고리즘은 이메일에서 악의적인 링크를 찾고 이를 보호하려고 웹 메일 공급자는 유해하다 알려진 도메인의 최신 블랙리스트를 보관한다.

✔ 열린 리다이렉션 방지

사용자를 대상으로 리다이렉션할 목적으로 다른 URL 내에 URL을 인코딩한다면 이러한 인코딩된 URL이 절대 URL이 아닌 상대 URL인지 확인.

✔ 기타 고려 사항

부득이하게 외부 링크를 게시해야 할 경우 구글 세이프 브라우저 API 를 활용하여 유해 사이트 블랙리스트와 대조해보는 등과 같은 방법으로 해결

▶ 클릭재킹

사용자가 의도하지 않은 행동을 하도록 속이는 사이버 공격 기법

사용자 인터페이스를 투명하게 덮거나 조작

✔ 클릭재킹 방지

- Content Security Policy (CSP) 사용 : CSP는 사이트가 로드할 수 있는 리소스를 제어하는 보안 기능
- X-Frame-Options 헤더 사용 : 웹 페이지가 다른 페이지의 iframe에서 로드되는 것을 제한
- 정확한 UI 구성: 용자 인터페이스 디자인에서 중요 버튼이나 링크가 쉽게 클릭될 수 없도록 배치하고, 사용자에게 명확한 행동 유도를 제공

▶ 서버 측 요청 위조

서버 측 요청 위조(Server-Side Request Forgery, SSRF)**는 공격자가 서버를 이용해 내부 리소스에 비정상적인 요청을 전송하도록 유도하는 보안 취약점

✔ 서버 측 위조 방지

- 입력 검증 및 화이트리스트 : 사용자 입력을 철저히 검증하고, 허용된 URL 목록(화이트리스트)만 요청할 수 있도록 설정
- 내부 IP 차단 : 버가 내부 IP 주소(예: 127.0.0.1)로 요청을 보내지 않도록 차단
- CORS 설정 : 교차 출처 리소스 공유(CORS)를 적절히 설정하여, 외부에서 내부 API에 접근하는 것을 제한

서비스 거부 공격

▶ 서비스 거부 공격

특정 서버나 네트워크에 과도한 트래픽이나 리소스 소모 요청을 보내서

정상적인 서비스 제공을 방해하는 공격 방식 DoS, Denial of Service Attack

▶ 서비스 거부공격 유형

✔ 인터넷 제어 메시지 프로토콜 공격

- 네트워크에서 오류 메시지를 보고하거나 네트워크 연결 상태를 진단하기 위해 사용되는 제어 메시지 프로토콜
- 악의적인 크기의 ICMP 패킷(주로 ping 명령으로 전송되는 ICMP Echo Request)을 보내는 공격

✔ 전송 제어 프로토콜 공격

핸드 셰이크를 완료 하지 않고 SYN 메시지(동기화 메시지)로 과부하를 일으켜 합법적 연결시도도 거부하게 만드는 공격

▶ 서비스 거부공격 유형

✔ 애플리케이션 계층 공격

슬로로리스 공격 : 서버의 많은 HTTP 연결을 열고 부분 HTTP 요청을 주기적으로 전송해 연결을 계속 열어 두어 연결 풀이 모두 소모한다.

RUDY(R U Dead Yeat?) 공격은 임의의 긴 Content-Length 헤더값을 가진 서버로 끝없는 POst 요청을 전송해 서버가 의미 없는 데이터를 처리하게 한다.

▶ 서비스 거부공격 유형

✔ 분산 서비스 거부 공격

다수의 협력 소스, 즉 분산 서비스 거부 공격(DDos)을 시도

✔ 의도하지 않은 서비스 거부 공격

인터넷 트래픽 급증이 모두 악의적인 것은 아니다.

▶ 서비스 거부공격 조치 방안

방화벽 및 침입 방지 시스템 활용

분산 서비스 거부 보호 서비스 활용

대규모 트래픽 급증에 대처할 준비