Chapter 10

2026 주요정보통신기반시설 기술적 취약점 분석·평가 방법 상세가이드

Web Application(웹)

Web Application(웹)

01 Web Application(웹) 취약점 분석 · 평가 항목

점검항목	항목 중요도	항목코드
코드 인젝션 (Code Injection)	상	CI
SQL 인젝션 (SQL Injection)	상	SI
디렉터리 인덱싱	상	DI
에러 페이지 적용 미흡	상	EP
정보 누출	상	IL
크로스사이트 스크립트	상	XS
크로스사이트 요청 위조(CSRF)	상	CF
서버사이드 요청 위조(SSRF)	상	SF
약한 비밀번호 정책	상	BF
불충분한 인증 절차	상	IA
불충분한 권한 검증	상	IN
취약한 비밀번호 복구 절차	상	PR
프로세스 검증 누락	상	PV
악성 파일 업로드	상	FU
파일 다운로드	상	FD
불충분한 세션 관리	상	IS
데이터 평문 전송	상	SN
쿠키 변조	상	CC
관리자 페이지 노출	상	AE
자동화 공격	상	AU
불필요한 Method 악용	상	WM

CI (상)	Web Application(웹) 1. 코드 인젝션 (Code Injection)
개요	
점검 내용	웹 애플리케이션 내 다양한 인젝션 공격(LDAP, 운영체제 명령 실행, SSI, XPATH, XML, SSTI 인젝션 등)에 대해 외부 입력값이 쿼리나 명령어로 삽입되어 비인가된 접근이나 코드 실행의 가능 유무 점검
점검 목적	허용되지 않은 코드 및 쿼리 실행을 방지하여 비인가된 접근, 데이터 유출, 시스템 변조, 악성 코드 실행 등의 위협을 차단하여, 데이터 보호와 시스템 안정성을 확보하기 위함
보안 위협	해당 취약점이 존재하는 경우 비인가된 데이터 접근으로 민감 정보가 탈취될 수 있으며, 시스템 명령어나 스크립트가 실행되어 서버 제어나 악성 코드 실행이 가능하고, 데이터 무결성이 훼손되어 정보의 신뢰성이 떨어지며, 서비스 거부 공격(DoS)으로 시스템 가용성이 저하될 수 있음. 따라서 "|", ";", "`", "〈" 등의 특수 문자에 대한 필터링 구현과 함께 입력값 검증, 화이트리스트 적용 등의 추가적인 보안 조치가 필요함
참고	※ **LDAP 인젝션:** 입력값이 LDAP(Lightweight Directory Access Protocol) 쿼리에 삽입되어 디렉터리 서비스 데이터에 대한 비인가된 조회, 수정, 삭제를 유발하는 공격 ※ **운영체제 명령 실행:** 입력값이 시스템 명령어로 실행되어 서버의 운영체제 명령을 비인가로 수행하거나 민감한 시스템 정보를 노출하는 공격 ※ **SSI 인젝션:** 입력값이 서버 사이드 인클루드(Server Side Include) 명령어로 실행되어 웹 애플리케이션 서버에서 비인가된 스크립트 실행이나 파일 접근을 유발하는 공격 ※ **XPath 인젝션:** 입력값이 XPath 쿼리에 삽입되어 XML 데이터의 비인가된 조회, 추가, 삭제를 유발하는 공격 ※ **XXE 인젝션:** 입력값이 XML 문서나 쿼리에 삽입되어 XML 데이터에 대한 비인가된 접근, 외부 엔티티 참조(XML External Entities)를 통한 시스템 정보 유출을 유발하는 공격 ※ **SSTI 인젝션:** 입력값이 서버 사이드 템플릿 엔진에 삽입되어 템플릿 렌더링 과정에서 비인가된 코드 실행이나 서버 내부 데이터 노출을 유발하는 공격 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 방화벽
판단 기준	**양호 :** 임의의 입력값에 대하여 철저한 검증이 이루어져, 허용되지 않은 값이 필터링되고 허용된 값만 처리되는 경우 **취약 :** 임의의 입력값에 대하여 검증 없이 명령이 실행되는 경우
조치 방법	화이트리스트 방식으로 쿼리를 허용된 값만 처리하고, 특수 문자에 대해 입력값 검증
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

● LDAP(Lightweght Directory Access Protocol) 인젝션

- 점검 방법

Step 1) 사용자 입력값에 대하여 변조된 LDAP 쿼리 삽입 후 실행 가능 여부 확인

[LDAP 쿼리 삽입]

- 조치 방법

1. 사용자 입력값을 화이트리스트로 지정하여 영문(a-z, A-Z)과 숫자(0-9)만을 허용
2. 특수문자를 사용해야 하는 경우 입력값(DN에 사용되는 특수문자는 '\'를 붙여 이스케이프 처리, 필터에 사용되는 특수문자는 '= + < > # \ , ; 앞뒤 공백' 등)에 대해서는 실행 명령이 아닌 일반문자로 인식되도록 처리
3. DN과 필터에 사용되는 사용자 입력값에는 특수문자 제거
4. 웹 방화벽에 LDAP 관련 특수문자를 필터링하도록 룰셋 적용

※ 특수문자 필터링 예시

구분	필터링 예시						
변경 전	\	=	+	*	(	)	\0
변경 후	\5c	\3d	\2b	\2a	\28	\29	\00

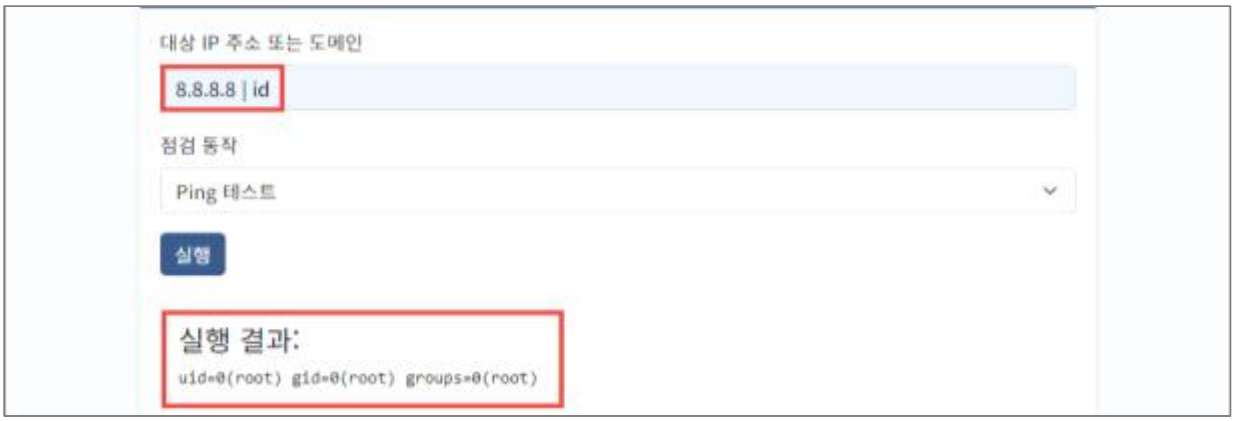
※ 필터링 대상 예시

필터링 대상 예시					
'	"	-	#	(	)
〈	〉	=	/*	*/	+
*	;	&	|	\	\0
:	`	%	user_tables	table_name	column_name
Syscolumns	union	select	insert	drop	update
and	or	If	join	substring	from
where	declare	substr	openrowset	xp_	sysobject

● 운영체제 명령실행

- 점검 방법

Step 1) 웹 애플리케이션 기능 내 전달되는 파라미터 값에 대하여 운영체제 명령어 삽입 후 실행 여부 확인



[임의 운영체제 명령어 삽입]

- 조치 방법

1. 웹 애플리케이션 설계 시 운영체제로부터 명령어를 직접적으로 호출하지 않도록 구현하고, 언어/프레임워크에서 제공하는 안전한 API 사용
2. 명령어를 직접 호출하는 기능이 필요한 경우에는, 데이터가 OS의 명령어 해석기에 전달되기 전에 화이트리스트 기반으로 입력값을 검증/확인하도록 구현
3. 입력값에 대한 파라미터 데이터의 필터링 처리
   - Unix/Linux : &, &&, |, ||, ;, `, $(), <, > 등
   - Windows : &, |, ^, % 등
4. 웹 서버 및 웹 애플리케이션 서버는 공개적으로 알려진 취약점이 제거된 상위 버전으로 업데이트
   KISA 보호나라&KrCERT/CC
   알림마당 > 보안공지(https://www.boho.or.kr/)
5. 웹 방화벽에 모든 사용자 입력값을 대상으로 악용될 수 있는 특수문자 및 키워드 등에 대한 룰셋 적용

※ 특수문자 필터링 처리 문자 설명

구분	상세 설명
&	첫 번째 명령어는 백그라운드에서 실행되며, 두 번째 명령어는 즉시 실행
&&	첫 번째 명령어가 성공했을 때만 두 번째 명령어 실행
|	두 명령어를 연결하여, 첫 번째 명령어의 출력을 두 번째 명령어의 입력으로 전달
;	첫 번째 명령어 실행 후 성공여부와 무관하게 두 번째 명령어 실행
` ` 또는 $()	백틱 또는 괄호 안에 있는 명령어를 실행하고 그 출력을 반환
〉 또는 〉〉	명령 실행 결과를 파일로 생성(덮어쓰기 또는 추가)
〈	파일 내용을 명령어 입력으로 전달
^ (Windows)	명령어 이스케이프/제어 문자로 사용
% (Windows)	환경 변수 치환
\	이스케이프 문자 또는 뒤에 오는 특수문자를 무효화하거나 명령 연결 시 사용

SSI(Server Side Includes) 인젝션

- 점검 방법

Step 1) 사용자가 입력 가능한 파라미터 값에 <!--#echo var="DOCUMENT_ROOT" -->를 삽입하여 전송 후 반환되는 페이지에 사이트의 홈 디렉터리가 표시되는지 확인

[Server Side Includes 지시어 삽입 및 취약점 유무 판단]

Step 2) 사용자가 입력 가능한 파라미터 값에 <!-- #exec cmd="ls -al" -->를 삽입하여 전송 후 반환되는 페이지에 디렉터리의 파일 리스트가 표시되는지 확인

[심화 공격 수행]

Step 3) HTTP 요청(Request) 헤더에 명령어를 삽입하여 실행되는지 확인

```
GET / HTTP/1.0
Referer: 〈!--#exec cmd="/bin/ps ax"--〉
User-Agent: 〈!--#include virtual="/proc/version"--〉
```

- 조치 방법

1. 화이트리스트 방식으로 사용자 입력에 대한 사용 가능한 문자들을 정의하여 정해진 문자를 제외한 나머지 모든 문자들을 필터링 처리
2. 필터링 해야 하는 대상은 GET 질의 문자열, POST 데이터, 쿠키, URL, 그리고 일반적으로 브라우저와 웹 서버가 주고받는 모든 데이터를 포함하며, 아래는 특수문자에 대한 엔티티 형태를 표시한 것임
3. 웹 서버의 SSI 기능을 사용하지 않거나, 웹 방화벽에 특수문자를 필터링하도록 룰셋 적용

구분	필터링 예시						
변경 전	〈	〉	"	(	)	#	&
변경 후	<	>	"	(	)	#	&

XPath 인젝션

- 점검 방법

Step 1) 취약점 존재 유무 판단을 위한 Xpath 쿼리(' or '1'='1, ' and 'a' = 'b 등) 삽입

(※ 예시로 제시한 것으로, 웹 사이트 환경에 맞춰 점검해야 함)

[XPath 구문 삽입 시도 및 취약점 유무 판단]

Step 2) 추가적인 쿼리 질의를 통하여 데이터 추출 등의 타당성 검토

```
' or count(parent::*[position()=1])=0 or 'a'='b
1' or string-length(username)=[COUNT] and '1'='1... 등
```

[심화 공격 수행]

- 조치 방법

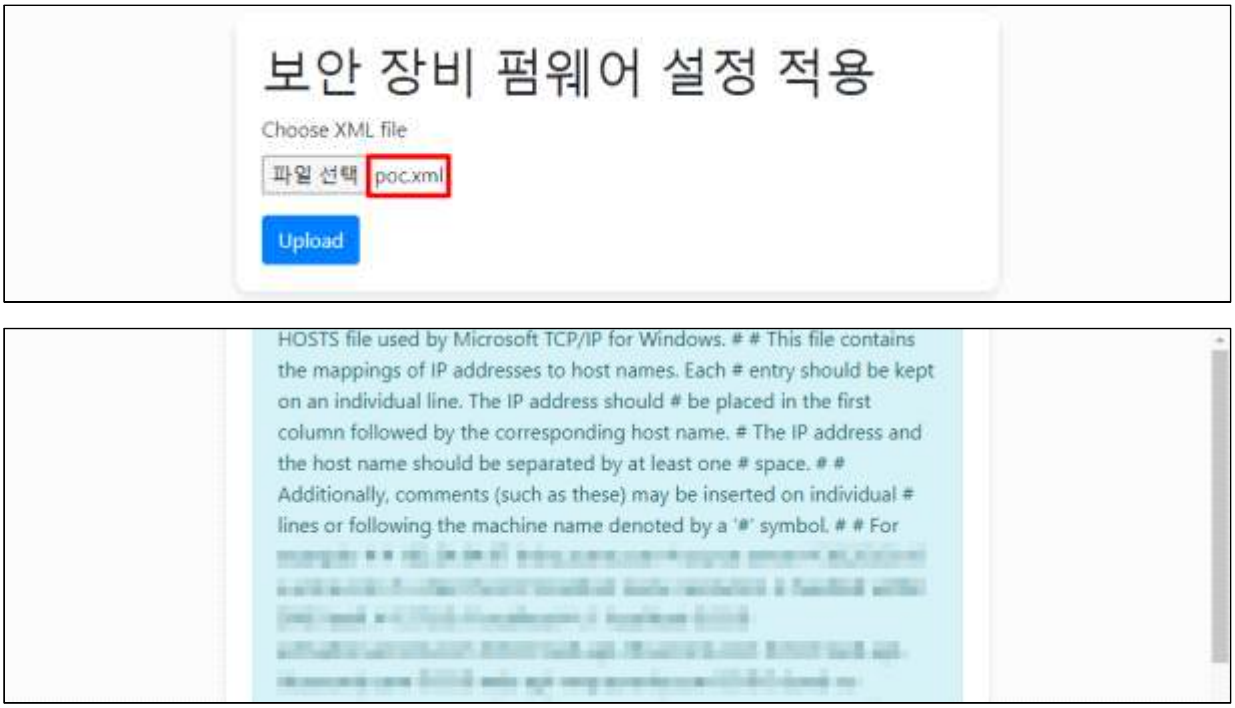
1. XPath 쿼리에 사용자가 값을 입력할 수 있는 경우, 입력값 검증을 통해 필요 문자만을 받아들이게 함. ( ) = ' [ ] : , * / 등의 오동작를 유발하는 특수문자는 제한하며, 특정 특수문자만을 필터링하는 것이 아닌 허용된 문자 이외의 모든 입력을 화이트리스트 방식으로 필터링 처리
2. Xpath 및 XQuery의 쿼리 삽입 시 사용되는 특수문자를 필터링하도록 웹 방화벽 룰셋 적용

XXE(XML External Entities) 인젝션

- 점검 방법

Step 1) 파일 업로드 페이지, API 엔드포인트, HTML Form 등 XML을 파싱하는 페이지 포인트 내 XML 객체 삽입 시도 (※ 예로 제시한 것으로, 웹 사이트 환경에 맞춰 점검해야 함)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///C:/Windows/System32/drivers/etc/hosts" >
]>
<foo>&xxe;</foo>
```



[외부 Entity 객체 삽입 및 XXE 공격 수행]

- 조치 방법

1. 허용된 태그와 속성만 사용하도록 화이트리스트 방식을 이용한 입력값 검증 로직 구현
2. 최근 언어별 XML 파서의 경우, 기본적으로 외부 엔티티 처리가 비활성화되어 있으나 소스코드 상에서 명시적으로 비활성화 처리하는 것이 안전한 방법
3. 의도하지 않은 오동작이 발생할 가능성이 존재하는 외부 엔티티 참조 명령어 및 주요 스키마 등에 대하여 웹 방화벽에 룰셋 적용

※ Java

Java DTD(외부 엔티티) 비활성화 예시

```
...
// 공통 보안 설정
dbf.setXIncludeAware(false);
...
if (mode == SecurityMode.FULL_SECURE) {
    dbf.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);
    dbf.setFeature("http://xml.org/sax/features/external-general-entities", false);
    dbf.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
    dbf.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd", false);
} else if (mode == SecurityMode.LIMITED_SECURE) {
    // DTD를 완전히 비활성화할 수 없는 경우
    dbf.setFeature("http://xml.org/sax/features/external-general-entities", false);
    dbf.setFeature("http://xml.org/sax/features/external-parameter-entities", false);
    dbf.setFeature("http://apache.org/xml/features/nonvalidating/load-external-dtd", false);
}
...
```

※ ASP.NET

ASP.NET DTD(외부 엔티티) 비활성화 예시

```
...
// XmlDocument 객체 생성
XmlDocument doc = new XmlDocument();

// XmlResolver를 null로 설정하여 외부 엔티티의 해석을 비활성화
doc.XmlResolver = null
...
```

※ PHP 8.0 이전

PHP 8.0 이전 DTD(외부 엔티티) 비활성화 예시

```
...
libxml_disable_entity_loader(true);
...
```

※ PHP 8.0 이후

▪ libxml_disable_entity_loader() 함수가 더 이상 사용되지 않으며, 외부 엔티티 로드는 기본적으로 비활성화되어있지만, LIBXML_NOENT 플래그로 인하여 XML 파서가 외부 엔티티를 확장하도록 설정되어있는 경우 XXE 취약점 발생 가능

PHP 8.0 이후 DTD(외부 엔티티) 비활성화 예시

```
...
// DOMDocument 인스턴스 생성
$dom = new DOMDocument();
// 외부 엔티티 비활성화
$dom->loadXML($xmlfile, LIBXML_NONET);
...
```

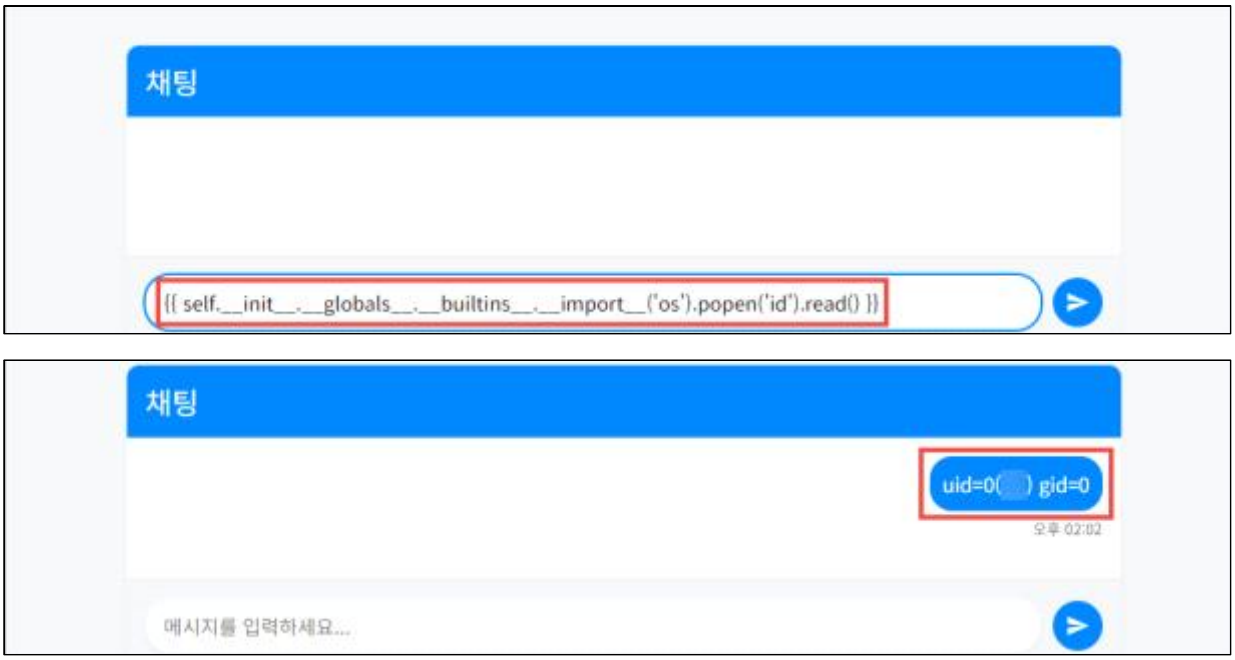
SSTI(Server Side Template Injection)

- 점검 방법

Step 1) 사용자 입력값이 서버 템플릿 엔진 내에서 처리되는지 확인하기 위해 {{7*7}} 등의 수식 삽입 시도

[템플릿 엔진 수식 삽입을 통한 취약점 유무 판단]

Step 2) 상위 컨텍스트 및 객체 접근을 시도하여 원격 코드 실행의 가능 여부를 판별하기 위하여 페이로드 입력 및 실행 유무 확인



[심화 공격 수행]

- 조치 방법

1. 템플릿 언어에서 예약된 의미를 가지는 문자({ } < > % # @ 등)에 대하여 이스케이프 처리
2. 템플릿 엔진의 안전 모드를 사용함으로써 템플릿 내에서 실행할 수 있는 명령을 제한하여 임의 코드 실행 방지
3. 안전 모드 및 내장 함수를 활용하여 보안 조치를 하는 경우 엔진의 버전과 그 버전에서 지원하는 기능을 고려하여 적절한 보안 패치 진행
4. 에러 메시지를 통해 사용된 템플릿 언어 및 관련 취약점에 대한 유의미한 정보를 제공할 가능성이 존재하므로, 노출되는 에러 메시지를 제한
5. 템플릿 엔진 내 사용되는 특수문자를 필터링하도록 웹 방화벽 룰셋 적용

※ Java

사용자 입력값 특수문자 인코딩 처리 예시

```
...
name = org.apache.commons.text.StringEscapeUtils.escapeHtml4(name); // 사용자 입력값 인코딩

// 사용자 입력값 이스케이프 처리
public static String escapeSpecialCharacters(String input) {
```

```
        if (input == null) return null;
        return input.replaceAll("([*{}\\[\\]<>%#@])", "\\\\$1");
}
```

▪ Velocity 템플릿 엔진에서 제공하는 '$esc.html'를 사용하여 사용자 입력을 HTML 인코딩 처리

Java(Velocity) 안전한 템플릿 사용 예시

```
...
#set($userInput = $esc.html($params.get("userInput")))
<p>사용자 입력: $userInput</p>
...
```

▪ FreeMarker 템플릿 엔진의 내장 함수인 '?html'를 사용하여 사용자 입력을 HTML 인코딩 처리

Java(FreeMarker) 안전한 템플릿 사용 예시

```
...
<p>사용자 입력: ${userInput?html}!</p>
...
```

※ ASP.NET

사용자 입력값 특수문자 인코딩 처리 예시

```
// 사용자 입력값 인코딩
...
private static readonly Dictionary<char, string> HtmlEntities = new Dictionary<char, string> {
    { '*', "&#42;" },
    { '{', "&#123;" },
    { '}', "&#125;" },
    { '[', "&#91;" },
    { ']', "&#93;" },
    { '<', "&lt;" },
    { '>', "&gt;" },
    { '%', "&#37;" },
    { '#', "&#35;" },
    { '@', "&#64;" }
};

public static string EscapeHtmlEntities(string input) {
```

```
    if (input == null) return null;
    StringBuilder escapedString = new StringBuilder();
    foreach (char ch in input) {
        if (HtmlEntities.ContainsKey(ch)) {
                escapedString.Append(HtmlEntities[ch]);
            } else {
                escapedString.Append(ch);
            }
        }
        return escapedString.ToString();
    }
...

// 사용자 입력값 이스케이프 처리
public static string EscapeSpecialCharacters(string input) {
        if (input == null) return null;
        return Regex.Replace(input, @"([*{}\[\]<>%#@])", @"\$1");
}
...
```

※ Python

사용자 입력값 특수문자 인코딩 처리 예시

```
# 사용자 입력값 인코딩
def escape_html_entities(input_string):
    if input_string is None:
        return None
...
    html_entities = {
        '*': '&#42;',
        '{': '&#123;',
        '}': '&#125;',
        '[': '&#91;',
        ']': '&#93;',
        '<': '&lt;',
        '>': '&gt;',
        '%': '&#37;',
        '#': '&#35;',
```

```
        '@': '&#64;',
        ...
    }
    return ''.join(html_entities.get(char, char) for char in input_string)
...
# 사용자 입력값 이스케이프 처리
def escape_special_characters(input_string):
    if input_string is None:
        return None
    return re.sub(r'([*{}\[\]<>%#@])', r'\\\1', input_string)
...
```

▪ 사용자 입력을 직접 템플릿에 삽입하는 방식은 SSTI 취약점에 노출될 수 있으므로 사용자 입력을 템플릿 변수로 전달하는 방식을 사용하여 안전하게 처리

Python(jinja2) 안전한 템플릿 사용 예시

```
...
template = "userinput : {{ userinput }}"
return render_template_string(template, userinput=param)
...
```

※ PHP

PHP 사용자 입력값 특수문자 인코딩 처리 예시

```
...
function escape_html_entities($input_string) {
    if ($input_string === null) {
        return null;
    }

    $html_entities = [
        '*' => '&#42;',
        '{' => '&#123;',
        '}' => '&#125;',
        '[' => '&#91;',
        '<' => '&lt;',
        '>' => '&gt;',
        '%' => '&#37;',
```

```
        ...
    ];

    return strtr($input_string, $html_entities);
}

// 특수 문자 앞에 백슬래시를 추가하여 이스케이프 처리
function escape_special_characters($input_string) {
    if ($input_string === null) {
        return null;
    }
    return preg_replace('/([*{}\[\]<>%#@])/', '\\\\$1', $input_string);
}
...

// htmlspecialchars 함수를 이용하여 사용자 입력값을 HTML 인코딩
$userInput = htmlspecialchars($userInput ENT_QUOTES, 'UTF-8');
...
```

SI (상)	Web Application(웹)
	2. SQL 인젝션 (SQL Injection)
개요	
점검 내용	웹 애플리케이션 내 입력값이 SQL 쿼리에 삽입되어 비인가된 데이터베이스 접근과 조작 가능 여부 점검
점검 목적	웹 애플리케이션 내 SQL문으로 해석될 수 있는 입력값 허용을 차단하고 운영 중인 데이터베이스에 대한 비인가된 접근 및 조작을 방지하여, 데이터 무결성과 보안성을 확보하기 위함
보안 위협	해당 취약점이 존재하는 경우, 입력값이 SQL 쿼리에 삽입되어 데이터베이스에 비인가된 접근을 허용하며, 공격자는 민감 데이터의 조회, 수정, 삭제를 포함한 다양한 악의적인 행위가 가능하므로 입력값에 대한 특수문자 필터링을 구현해야함
참고	※ **SQL 인젝션:** 사용자의 입력값으로 웹 사이트 SQL 쿼리가 완성되는 약점을 이용하여, 입력값을 변조해 비정상적인 SQL 쿼리를 조합하거나 실행하는 공격. 이는 개발자가 의도하지 않은 SQL문을 실행하게 하여 데이터베이스를 비정상적으로 조작하고, 민감한 데이터를 조회, 수정, 삭제할 수 있는 공격 ※ SQL 인젝션 공격 관련 코드 검토 필요 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 방화벽
판단 기준	**양호 :** 임의로 작성된 SQL 쿼리 입력에 대한 적절한 검증을 통해 비정상적인 쿼리가 실행되지 않도록 하는 경우
	취약 : 임의로 작성된 SQL 쿼리 입력에 대한 검증이 이루어지지 않아 비정상적인 쿼리가 실행되는 경우
조치 방법	소스코드 내 SQL 쿼리를 입력값으로 받는 함수나 코드를 사용할 경우, 임의의 SQL 쿼리 입력에 대한 검증 로직을 구현하여 서버에 검증되지 않는 SQL 쿼리요청 시 에러 페이지가 아닌 정상 페이지가 반환되도록 필터링 처리하고 웹방화벽에 SQL 인젝션 관련 룰셋을 적용하여 SQL 인젝션 공격을 차단함
조치 시 영향	웹 서비스에서 사용하고 있는 명령어 및 특수문자가 필터링 되어 장애가 발생 될 수 있어 사전 영향도 및 코드 분석이 필요

점검 및 조치 사례

- 점검 방법

Step 1) 사용자 입력값 조건에 따른 참, 거짓 SQL 쿼리를 삽입하여, 응답의 변화(응답시간, 에러메시지, 응답 내용 등) 유무 확인

[DB 쿼리 삽입을 통한 취약점 유무 판단]

Step 2) 인증 페이지(로그인, 비밀번호 검증 등) 내 참이 되는 SQL쿼리를 삽입하여 우회 유무 확인

[인증 페이지 내 DB 쿼리 삽입을 통한 인증 우회 확인]

- 조치 방법

1. SQL 쿼리 내 사용되는 문자열의 유효성을 검증하는 로직 구현
2. 아래와 같은 특수문자에 대하여 사용자 입력값으로 지정 금지

문자	상세 설명
'	문자 데이터 구분 기호
;	쿼리 구분 기호
--, #	해당 라인 주석 구분 기호
/* */	/* 와 */ 사이 구문 주석

3. Prepared Statements를 사용하여 사용자 입력과 SQL 쿼리를 분리하여 처리
4. 시스템에서 제공하는 에러 메시지 및 DBMS에서 제공하는 에러코드가 노출되지 않도록 예외처리
5. 웹 방화벽(WAF)에 대하여 SQL Injection 관련 룰셋 추가

※ Java

SQL 키워드 및 특수문자 필터링 로직 예시

```
...
public static String sanitize(String input) {
    if (input == null) {
        return null;
    }
    // 특수문자 및 키워드들을 공백으로 치환
    String[] sqlKeywords = {"SELECT", "UNION", "INSERT", "UPDATE", "DELETE", "DROP", "--"};
    String pattern = "(?i)\\b(" + String.join("|", sqlKeywords) + ")\\b|['\"\\\\;()<>#/*!]";
    Pattern regex = Pattern.compile(pattern + "|--");
    Matcher matcher = regex.matcher(input);
    return matcher.replaceAll(" ");
}
...
String sanitizedInput = sanitize(userInput);
```

Prepared Statement 사용 로직 예시

```
...
String sql = "SELECT * FROM users WHERE username = ?";
PreparedStatement preparedStatement = connection.prepareStatement(sql);
preparedStatement.setString(1, userInput);
...
ResultSet resultSet = preparedStatement.executeQuery();
...
```

▪ JDBC 표준 예외 클래스를 사용하여 다양한 데이터베이스 시스템에 일관된 방식으로 예외처리

적절한 예외 처리 예시

```
try {
    // 데이터베이스 작업
} catch (SQLException e) {
    // 브라우저에 일반적인 오류 메시지를 반환
    e.printStackTrace();
    System.out.println("An error occurred. Please try again later.");
}
```

▪ 사용자 입력을 직접 쿼리에 포함시킬 시 취약점이 발생하므로, 파라미터 바인딩을 사용하여 구현

▪ 파라미터 바인딩 : 쿼리를 실행할 때, 쿼리 문자열과 사용자 입력값(파라미터)을 분리하여 처리하는 기법. 데이터베이스는 쿼리 문자열을 미리 파싱하고 컴파일하며, 쿼리 실행 시점에 파싱된 쿼리 문자열에 파라미터를 바인딩하여 데이터를 전달하므로 데이터베이스는 파라미터를 데이터로만 인식함

ORM(JPA-Hibernate) 파라미터 바인딩 사용 예시

```
public class ItemService {
    @PersistenceContext
    private EntityManager em;

    public List<Item> findItemsByUserInput(String userInput) {

        // JPQL을 사용하여 SQL Injection 방지
        String jpql = "SELECT i FROM Item i WHERE i.itemID > :userInput";
        Query query = em.createQuery(jpql, Item.class);
        query.setParameter("userInput", userInput);
        return query.getResultList();
    }
}
```

▪ SQL Mapper(Mybatis) 내 '${}' 구문의 경우 사용자 입력값이 SQL 구문으로 해석되기 때문에 파라미터 바인딩('#{}')을 사용하여 구현

SQL Mapper(Mybatis) 파라미터 바인딩 사용 예시

```
...
    <!-- 학생 정보 삽입 -->
    <insert id="insertStudent" parameterType="com.example.Student">
        INSERT INTO STUDENTS (NUM, NAME, AGE, GRADE)
        VALUES (#{num}, #{name}, #{age}, #{grade})
    </insert>

    <!-- 학생 정보 삭제 -->
    <delete id="deleteStudent" parameterType="int">
        DELETE FROM STUDENTS
        WHERE NUM = #{num}
 </delete>
...
```

※ ASP.NET

▪ 정규표현식을 활용하여 SQL 키워드 및 특수문자에 대하여 필터링 로직 구현

SQL 키워드 및 특수문자 필터링 로직 예시

```
public static string Sanitize(string input)
{
    if (input == null)
    {
        return null;
    }
    // 특수문자들을 공백으로 치환
    string[] sqlKeywords = { "SELECT", "UNION", "INSERT", "UPDATE", "DELETE", "DROP", "--" };
    string pattern = @"(?i)\b(" + string.Join("|", sqlKeywords) + @")\b|['""\\;()<>#/!*]";
    return Regex.Replace(input, pattern, " ");
}
string sanitizedInput = Sanitize(userInput);
...
```

Prepared Statement 사용 로직 예시

```
string strQry = "SELECT count(*) FROM users WHERE userName = @username
                                    AND Password = @password";
using (SqlCommand cmd = new SqlCommand(strQry, cnx))
{
    cmd.Parameters.Add(new SqlParameter("@username", SqlDbType.VarChar, 50) {
                    Value = txtUser.Text });
    cmd.Parameters.Add(new SqlParameter("@password", SqlDbType.VarChar, 50) {
                    Value = txtPassword.Text });
    int intRecs = (int)cmd.ExecuteScalar();
    if (intRecs > 0)
    {
        FormsAuthentication.RedirectFromLoginPage(txtUser.Text, false);
    }
    else
    {
        lblMsg.Text = "Login attempt failed.";
    }
}
```

적절한 에러 예외처리 로직 예시

```
catch (SqlException ex)
{
    Logger.LogError(ex);  // 로그 상세 에러 기록
    // 사용자에게 일반적인 메시지 표시
    lblErrorMessage.Text = "데이터베이스 작업 중 오류가 발생했습니다.";
}
```

※ ASP

적절한 에러 예외처리 예시

```
On Error Resume Next ' 에러 발생 시 계속 실행
...
If Err.Number 〈〉 0 Then
    ' 에러가 발생한 경우
    Err.Clear ' 에러 삭제 처리
%〉
    〈script language="javascript"〉
        alert("서버에 문제가 발생하였습니다. 잠시 후 다시 시도해 주세요.");
        location.replace("〈%=local%〉/login/login.asp?ba=search")
    〈/script〉
〈%
    response.end
End If
```

▪ SQL Server의 xp_cmdshell 기능의 경우 SQL Server 2005 버전부터 기본적으로 비활성화 되어있음

xp_cmdshell 비활성화 처리 예시

```
sp_configure 'show advanced options', 1;     // SQL Server의 고급 옵션 활성화
GO
RECONFIGURE;
GO
sp_configure 'xp_cmdshell', 0;                    // xp_cmdshell 비활성화

GO
RECONFIGURE;
GO
```

※ PHP

▪ ereg_replace, eregi_replace 의 경우, PHP 5.3.0 이후 삭제되었으며, addslashes/magic_quotes_gpc의 경우 멀티 바이트 문자 입력 시 우회의 가능성이 존재하므로 preg_replace를 이용하여 구현

SQL 키워드 및 특수문자 필터링 로직 예시

```
function sanitize($input) {
    if ($input === null) {
        return null;
    }
    // 특수문자들을 공백으로 치환
    $sqlKeywords = ["SELECT", "UNION", "INSERT", "UPDATE", "DELETE", "DROP", "--"];
    $pattern = "/(?i)\\b(" . implode("|", $sqlKeywords) . ")\\b|['\"\\\\;()<>#\/!*]/";
    return preg_replace($pattern, " ", $input);
}
$sanitizedInput = sanitize($userInput);
```

Prepared Statement 사용 로직 예시

```
...
$sql = "SELECT * FROM users WHERE username = ?";
$stmt = $pdo->prepare($sql);
$stmt->execute([$userInput]);
...
$results = $stmt->fetchAll(PDO::FETCH_ASSOC);
...
```

적절한 에러 예외처리 예시

```
...
try {
    // 데이터베이스 작업
} catch (PDOException $e) {            // PDO 확장의 표준 예외 클래스를 사용하여 예외처리
    // 브라우저에 일반적인 오류 메시지를 반환
    error_log($e->getMessage());
    echo 'SQL Exception';
}
...
```

DI (상)	Web Application(웹)
	3. 디렉터리 인덱싱
개요	
점검 내용	웹 애플리케이션 서버 내 디렉터리 인덱싱 취약점 존재 여부 점검
점검 목적	디렉터리 인덱싱 취약점을 제거하여 특정 디렉터리 내 불필요한 파일 정보의 노출 및 비인가자가 민감한 파일에 대한 접근을 차단
보안 위협	해당 취약점이 존재할 경우, 브라우저를 통해 특정 디렉터리 내 파일 리스트가 노출되어 응용 시스템의 구조가 외부에 공개될 수 있으며, 민감한 정보가 포함된 설정 파일 등이 노출될 경우 보안상 심각한 위험을 초래할 수 있음
참고	※ **디렉터리 인덱싱 취약점:** 특정 디렉터리의 초기 페이지(index.html, home.html, default.asp 등) 파일이 존재하지 않을 때 디렉터리 목록을 출력하는 취약점
점검 대상 및 판단 기준	
대상	웹 애플리케이션 서버
판단 기준	**양호 :** 디렉터리 파일 리스트가 노출되지 않는 경우
	취약 : 디렉터리 파일 리스트가 노출되는 경우
조치 방법	웹 애플리케이션 서버 설정을 변경하여 디렉터리 파일 리스트가 노출되지 않도록 설정
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

- 점검 방법

Step 1) URL 경로 중 확인하고자 하는 디렉터리 경로에 대하여 주소창에 입력하여 인덱싱 여부 확인

[디렉터리 인덱싱 취약점 유무 판단]

- 조치 방법

- **Apache**
  - httpd.conf 파일 내 Indexes 옵션 제거 후 서버 재기동

Apache 서버 설정 예시

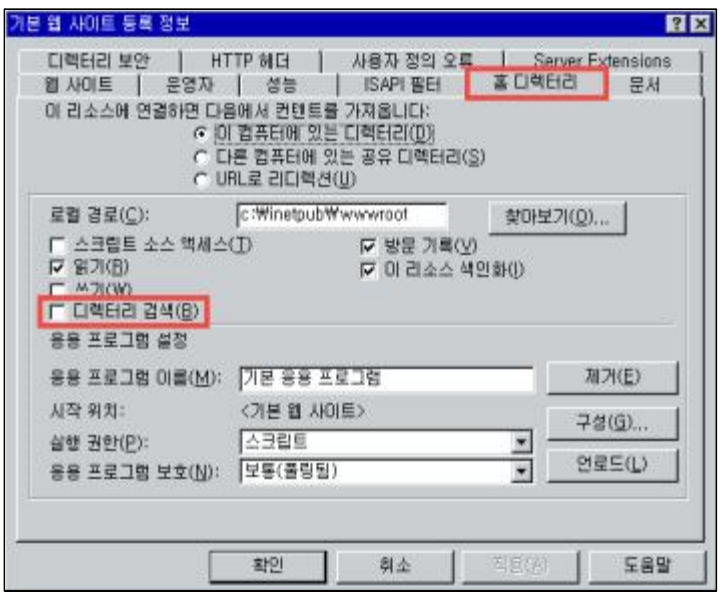
```
<Directory /var/www/html>
        Options Indexes FollowSymLinks
        # Indexes 옵션 제거
        Options FollowSymLinks
</Directory>
```

- **Tomcat**
  - web.xml 파일 내 아래 지시자 수정 후 서버 재기동

Tomcat 서버 설정 예시

```
<init-param>
        <param-name>listings</param-name>
        <param-value>false</param-value>
</init-param>
```

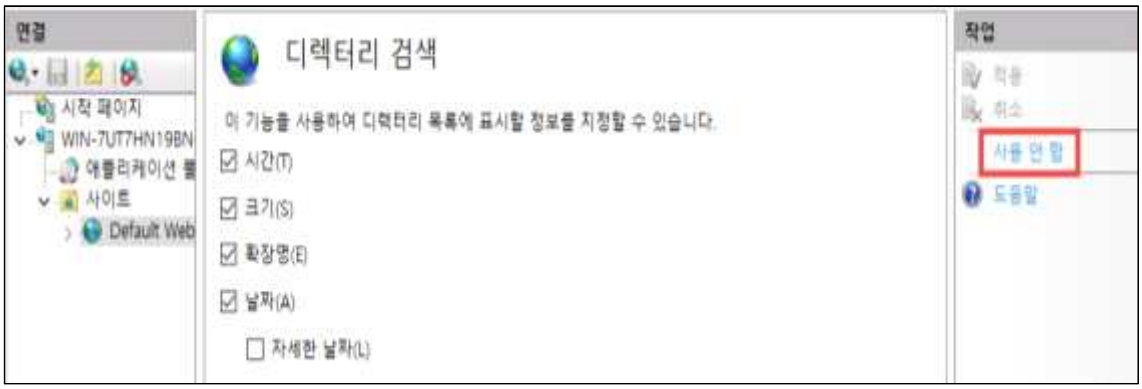
- **IIS (6.0 이하)**
  - 인터넷 정보 서비스 → 등록 정보 → 홈 디렉터리 → 디렉터리 검색 해제



[디렉터리 검색 옵션 해제 설정]

IIS (7.0 이상)

- IIS 관리자 → 디렉터리 검색 → 사용 안 함



[디렉터리 검색 기능 사용 안함 설정]

Nginx

- /etc/nginx/sites-available/default 등 파일 내 아래 지시자 수정 후 서버 재기동

Nginx 서버 설정 예시

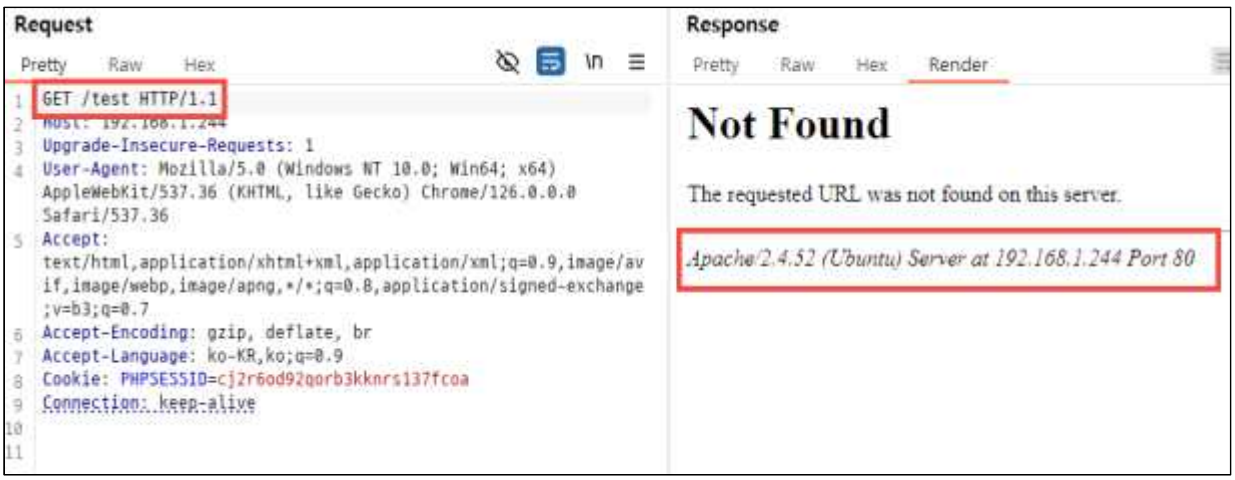
```
...
server {
        location / {
                autoindex off;
        }
}
...
```

EP (상)	Web Application(웹)
	4. 에러 페이지 적용 미흡
개요	
점검 내용	웹 애플리케이션 에러 페이지 내 불필요한 정보 노출 여부 점검
점검 목적	사용자 정의 에러 페이지를 설정하여 기본 서버 에러 페이지 내 불필요한 정보(서버 버전 정보, 시스템 절대 경로, 스택 트레이스 등)의 제공을 차단하기 위함
보안 위협	에러 페이지 내 서버 및 응용 시스템의 상세한 정보를 포함한 경우, 시스템 구조와 스택 트레이스, 데이터베이스 쿼리 등 민감한 정보를 노출시켜 공격 벡터로 악용할 가능성 존재
참고	※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 서버, 웹 방화벽
판단 기준	**양호 :** 에러 발생 시 자체 정의 에러 페이지를 출력하여 과도한 정보가 노출되지 않는 경우
	취약 : 에러 발생 시 기본 에러 페이지가 출력되며, 해당 페이지에 불필요한 정보(서버 버전 정보, 시스템 경로, 스택 트레이스 등)가 노출되는 경우
조치 방법	웹 애플리케이션 서버 내 사용자 정의 에러 페이지를 적용함으로써 불필요한 정보 노출을 방지
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

- 점검 방법

Step 1) 에러 유도 시 에러 페이지 내 불필요한 정보(서버 버전 정보, 시스템 절대 경로, 스택 트레이스 등)가 노출되는지 확인



[에러 페이지 내 서버 버전 정보 노출]

- 조치 방법

Apache

▪ apache2.conf 또는 httpd.conf 파일 내 아래 지시자 추가 후 서버 재기동

응답 헤더 내 서버 버전 정보 제거 예시

```
...
ServerTokens Prod
ServerSignature Off
...
```

사용자 에러 페이지 정의 예시

```
# 예) ErrorDocument 404 /main/error.html
ErrorDocument 404 [에러 페이지 경로]
ErrorDocument 405 [에러 페이지 경로]
# 추가적으로 에러 코드 등록하여 설정
...
```

Tomcat

▪ server.xml 파일 내 아래 지시자 추가 후 서버 재기동

응답 헤더 내 서버 버전 정보 제거 예시

```
# server.xml 파일 내 〈Connector〉 요소에 아래 지시자 추가 후 서버 재기동
〈Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443"
           maxParameterCount="1000"
           server=" "
/〉
```

개발용 리포트 비활성화 예시

```
# server.xml 파일 내 아래 지시자 추가 후 서버 재기동
〈Valve className="org.apache.catalina.valves.ErrorReportValve"
       showReport="false"
       showServerInfo="false" /〉
〈/Host〉
```

에러 페이지 매핑 예시

```
<!-- web.xml -->
<error-page>
  <error-code>404</error-code>
  <location>/errors/404</location>
</error-page>

<error-page>
  <error-code>500</error-code>
  <location>/errors/500</location>
</error-page>

<!-- 모든 예외(최상위) 공통 500 처리 -->
<error-page>
  <exception-type>java.lang.Exception</exception-type>
  <location>/errors/500</location>
</error-page>
```

- **Nginx**
  - nginx.conf 파일 내 아래 지시자 추가 후 서버 재기동

응답 헤더 내 서버 버전 정보 제거 예시

```
...
http {
        server_tokens off;
        ...
}
...
```

  - /etc/nginx/sites-available/default 파일 내 아래 지시자 추가 후 서버 재기동

사용자 에러 페이지 정의 예시

```
server {
   listen 80;
   ...
   # 기타 설정
   ...
```

```
    error_page 400 401 402 405 /custom_4xx.html;
    error_page 404 /custom_404.html;
    error_page 500 502 503 504 /custom_5xx.html;
    location = /custom_404.html {
        root /var/www/html;
        internal;
    }

    location = /custom_4xx.html {
        root /var/www/html;
        internal;
    }

    location = /custom_5xx.html {
        root /var/www/html;
        internal;
    }
...
}
```

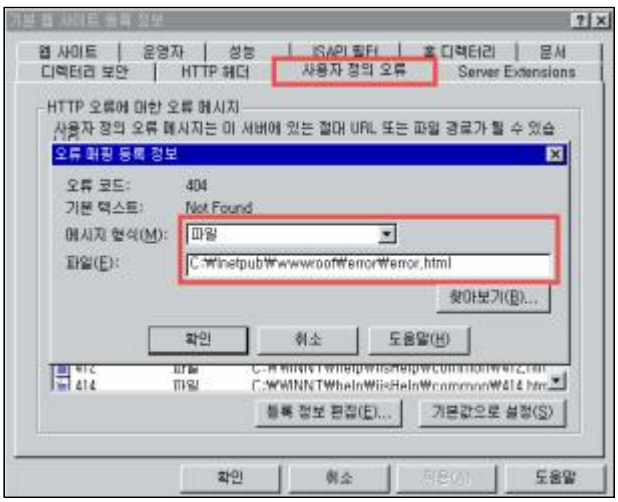
- **IIS (6.0 이하)**

※ 응답 헤더

- Microsoft사의 URLScan 3.1 도구 지원 종료로 인한 서버 버전 제거 제한

※ 에러 페이지

- 인터넷 정보 서비스 → 등록 정보 → 사용자 정의 오류 → 등록 정보 편집 → 별도 에러 페이지 지정

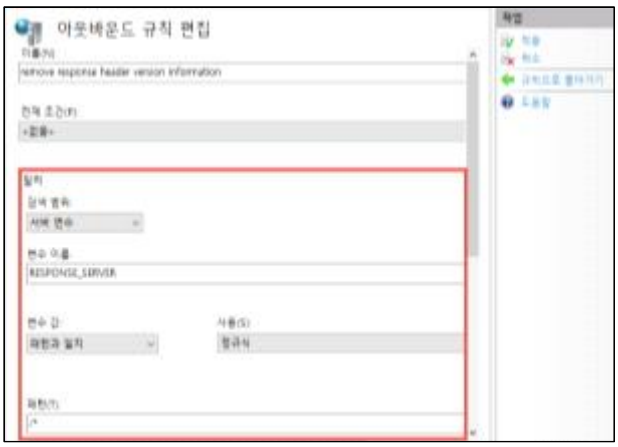


[사용자 정의 에러 페이지 설정]

IIS (7.0 이상)

※ 응답 헤더

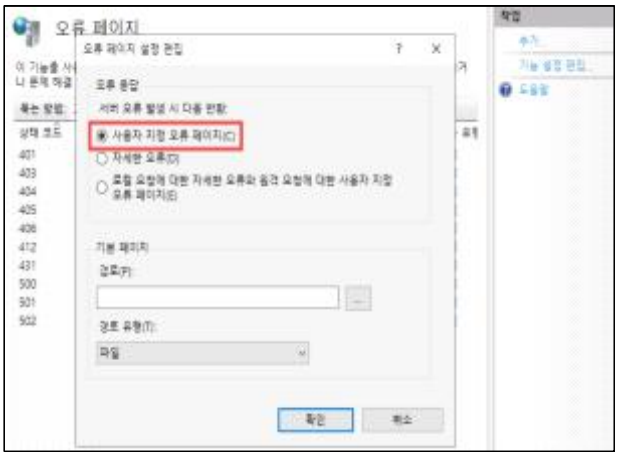
- URL Rewrite 모듈 설치 → IIS 관리자 → URL 재작성 → 서버 변수 보기 → 추가 → RESPONSE_SERVER 변수 추가 → 규칙 추가 → 아웃바운드 규칙(빈 규칙) → 규칙 추가 → 적용
- URL Rewrite 모듈 (https://www.iis.net/downloads/microsoft/url-rewrite)



[응답 헤더 내 서버 버전 정보 제거]

※ 에러 페이지

- IIS 관리자 → 오류 페이지 → 기능 설정 편집 → 사용자 지정 오류 페이지



[사용자 정의 에러 페이지 설정]

IL (상)	Web Application(웹)
	5. 정보 누출
개요	
점검 내용	웹 애플리케이션 내 중요 정보(개인정보, 금융 정보 등) 및 불필요한 정보(주석 내 디버깅 정보, 초기 샘플페이지, 백업 파일 등)의 노출 여부 점검
점검 목적	웹 애플리케이션 내 중요 정보(개인정보, 금융 정보 등)에 대해 마스킹 처리를 하고, 서비스 제공 중 불필요한 정보(주석 내 디버깅 정보, 초기 샘플 페이지, 백업 파일 등)를 삭제하여 민감한 정보와 내부 시스템 정보의 노출을 차단하기 위함
보안 위협	웹 애플리케이션 운영 시 중요 정보(개인정보, 금융 정보 등)가 평문으로 노출될 경우, 개인정보 유출, 신원 도용, 피싱 공격 등의 위협이 발생할 수 있으며, 서버나 WAS의 초기 샘플 페이지, 백업 파일, 압축 파일 등이 노출될 경우, 공격자가 시스템 구조를 파악하고 보안 설정 및 접근 제어 정책을 유출하여 2차 공격에 활용할 수 있음
참고	※ **중요 정보:** 고유식별정보(주민등록번호), 비밀번호(로그인 비밀번호, 계좌 비밀번호, 공인인증서 비밀번호 등), 신용정보(보안카드번호, 카드번호 등) 등 ※ 중요 정보 노출(일회성 노출)이 반드시 필요한 서비스의 경우 본인인증 절차 적용 시 예외 처리 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 애플리케이션 서버
판단 기준	**양호 :** 중요 정보가 마스킹 처리되어 있으며, 서비스 운영 및 서버 사이드의 구조를 파악할 수 있는 불필요한 정보가 노출되지 않는 경우
	취약 : 중요 정보가 평문으로 노출되거나, 서비스 운영 및 서버 사이드의 구조를 파악할 수 있는 과도한 정보가 노출되는 경우
조치 방법	중요 정보를 마스킹 처리하고, 샘플 페이지나 불필요한 정보를 제공하는 요소를 삭제하여 과도한 정보 노출을 최소화함으로써 민감 정보와 내부 시스템 구조의 노출을 방지함
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

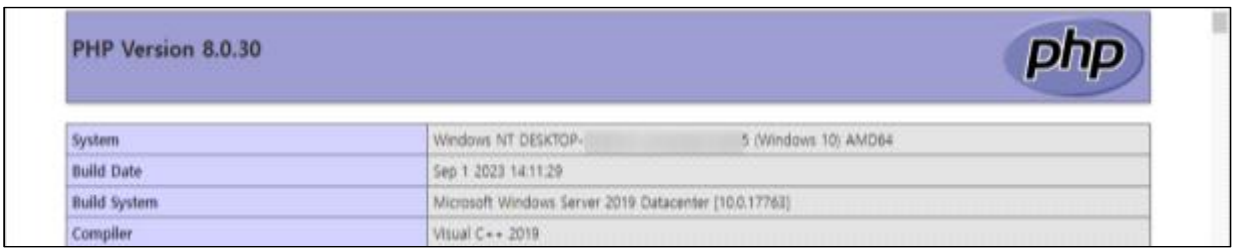
Apache

- 점검 방법

Step 1) 웹 애플리케이션 내 중요 정보(개인정보, 금융정보 등)의 평문 노출 여부 점검

[페이지 내 중요 정보 평문 노출]

Step 2) 서버나 WAS의 초기 샘플 페이지, 백업파일 등 불필요한 정보의 노출 유무 점검



[초기 샘플 페이지 내 중요 정보 노출]

Step 3) HTML 소스코드 내 중요 정보가 코드 및 주석을 통한 노출 여부 점검

```
3       "/admin/settings",
4       "/admin/users",
5       "/admin/logs",
6   ];
7
8   /* ### default id & password ### */
9   // #id - admin
10  // #password - admin
11
```

[HTML 주석 내 중요 정보 노출]

- 조치 방법

1. robots.txt, web.config, nginx.conf 파일 작성을 통해 검색 차단할 디렉터리, 확장자, 페이지 등을 지정할 수 있으며 HTML 태그 내에 META 태그를 추가하여 검색엔진의 인덱싱을 차단함
2. 웹 디렉터리 내 삭제해야 할 파일 태그 확장자에 포함된 백업 파일을 모두 삭제하고, *.txt 확장자와 같이 작업 중 생성된 일반 텍스트 파일이나 이미지 파일 등 불필요한 파일에 대하여 제거
3. 웹 서버 설정 후 초기 페이지와 초기 디렉터리 및 배너를 삭제하여 Banner Grab에 의한 시스템 정보 유출을 차단함
4. 아래 개인정보 마스킹 기준 예시를 참고하여 개인정보 항목의 일부를 마스킹해야 함
5. 개발 중 작성된 주석 문자, 디버그 정보, 시스템 구조 관련 정보 등이 외부에 노출되지 않게 제거

※ 삭제 권고 파일 확장자 예시

삭제 권고 확장자 예시					
*.bak	*.backup	*.org	*.old	*.zip	*.log
*.sql	*.new	*.txt	*.tmp	*.temp	*.!

※ 개인정보 마스킹 기준 예시 (참고 문서 : 개인정보보호위원회_홈페이지 개인정보 노출방지 안내서)

개인정보	설명	예시
성명	성명 중 한 글자 이상	홍*동
주민등록번호	뒤에서부터 6자리	901231-1******
여권번호	뒤에서부터 4자리	12345****
연락처	전화번호 또는 휴대폰 뒤 4자리	010-1234-****
카드번호	7번째에서 12번째자리	9430-82**-****-2393
계좌번호	뒤에서부터 5자리	430-20-1*****

XS (상)	Web Application(웹)
	6. 크로스사이트 스크립트
개요	
점검 내용	웹 애플리케이션 내 악성 스크립트가 다른 사용자의 브라우저에서 실행되는 취약점 존재 여부 점검
점검 목적	사용자 입력값에 대한 검증을 실시하여, 사용자 세션 탈취, 악성 코드 삽입 등의 악의적인 스크립트 실행을 차단하기 위함
보안 위협	사용자 입력값에 대한 필터링이 할 경우, 공격자는 사용자 입력값 내 악의적인 스크립트(JavaScript, VBScript, ActiveX, Flash 등)를 삽입하여 사용자의 쿠키(세션)를 탈취하거나 피싱 사이트로 유도하는 등의 악의적인 공격을 수행할 수 있음
참고	※ **크로스사이트 스크립팅:** 악의적인 스크립트를 웹 페이지에 삽입하여 사용자 세션 탈취, 키로깅, 피싱 공격 등을 유발하는 기법으로, 크게 저장형(Stored)과 반사형(Reflected), DOM 기반 공격 방식으로 나뉨 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 방화벽
판단 기준	**양호 :** 사용자 입력값에 대해 검증 및 필터링이 이루어져, 악의적인 스크립트가 실행되지 않는 경우
	취약 : 사용자 입력값에 대한 검증 및 필터링이 이루어지지 않으며, HTML 코드가 입력 및 실행되는 경우
조치 방법	특수문자에 대해 필터링 처리와 출력값 인코딩(HTML 엔티티, 이스케이프 등)을 적용하여 악성 스크립트 실행을 방지하며, 부득이하게 HTML 코드를 사용해야 하는 경우 화이트리스트 방식을 적용해 허용된 HTML 코드만 처리하도록 설정
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

- 점검 방법

Step 1) 사용자 입력값을 전달받아 HTML 상에 렌더링되는 애플리케이션(게시판, 검색 등)에 스크립트 구문 삽입 후 실행되는지 확인

▪ 게시글 작성 시 스크립트 구문을 삽입하여 글 열람 시 스크립트가 동작하는 경우(Stored XSS)

[Stored(저장형) XSS 취약점 점검]

▪ 입력값이 HTML 상에 렌더링되어 스크립트가 삽입된 URL 접근 시 스크립트가 동작하는 경우(Reflected XSS)

[Reflected(반사형) XSS 취약점 점검]

- 조치 방법

1. 크로스사이트 스크립팅 공격에 사용되는 특수문자에 대하여 입력값 검증 및 필터링(HTML 엔티티, 이스케이프 등)처리 로직을 서버 사이드에서 구현
2. 부득이하게 HTML 코드를 사용해야 하는 경우 화이트리스트 방식을 적용하여 허용된 HTML 코드만 처리
3. 웹 방화벽에서 웹 태그 및 스크립트 관련 특수문자 필터링 룰셋을 적용하여 추가적인 방어 확보
4. 세션 탈취 방지를 위해 쿠키에 HttpOnly, Secure, SameSite 옵션을 설정하여 노출되지 않도록 보호

※ 특수문자 필터링 예시

구분	필터링 예시						
변경 전	〈	〉	"	(	)	#	&
변경 후	<	>	"	(	)	#	&

※ Java

▪ 입력 문자열 내 특수 문자를 HTML Entity로 변환하는 예시

사용자 입력값 HTML Entity 처리 로직 예시

```
...
public static String sanitizeInput(String input) {
    if (input == null) return null;
    return input.replaceAll("&", "&amp;")
                .replaceAll("〈", "&lt;")
                .replaceAll("〉", "&gt;")
                .replaceAll("\"", "&quot;")
                .replaceAll("'", "&#39;");
               /* 필터링 문자 추가*/
}
...
```

※ ASP.NET

▪ 입력 문자열 내 특수 문자를 HTML Entity로 변환하는 예시

사용자 입력값 HTML Entity 처리 로직 예시

```
StringBuilder sb = new StringBuilder();
foreach (char c in input)
{
```

```
    switch (c)
    {
        case '<':
            sb.Append("&lt;");
            break;
       ...
       /* 필터링 문자 추가*/
       ...
        default:
            sb.Append(c);
            break;
    }
}
return sb.ToString();
...
```

※ PHP

▪ 입력 문자열 내 특수 문자를 HTML Entity로 변환하는 예시

사용자 입력값 HTML Entity 처리 로직 예시

```
function escapeHtml($input) {
    return htmlspecialchars($input, ENT_QUOTES | ENT_HTML5, 'UTF-8', false);
}

// 추가적인 문자열 필터링이 필요한 경우
function escapeHtmlExtended($input) {
    $escaped = htmlspecialchars($input, ENT_QUOTES | ENT_HTML5, 'UTF-8', false);
    $additionalEscapes = [
        '\\' => '&#92;',
        '(' => '&#40;',
        ')' => '&#41;',
        '#' => '&#35;'
    ];
    return strtr($escaped, $additionalEscapes);
}
```

CF (상)	Web Application(웹)
	7. 크로스사이트 요청 위조(CSRF)
개요	
점검 내용	웹 애플리케이션 내 사용자의 인증 세션을 악용하여 의도하지 않은 위조 요청 가능 여부 점검
점검 목적	사용자가 인증된 세션을 가진 상태에서 공격자가 의도한 위조 요청이 전송·처리되지 않도록 하여 비정상적인 상태 변경을 방지하기 위함
보안 위협	CSRF 취약점이 존재할 경우, 공격자는 사용자가 로그인된 세션을 악용하여 인증정보 없이도 위조된 요청을 전송할 수 있음. 이로 인해 사용자의 의도와 무관하게 비밀번호 변경, 계좌 이체, 게시글 삭제, 개인정보 수정 등 권한 있는 사용자가 수행할 수 있는 행위가 공격자에 의해 실행될 수 있음
참고	※ CSRF(Cross Site Request Forgery): 사용자가 자신의 의지와 무관하게 공격자가 의도한 행위(수정, 삭제, 등록 등)를 특정 웹사이트에 요청하게 하는 공격 유형으로, 사용자의 인증된 세션을 악용하여 비인가된 행위 수행 가능 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 방화벽
판단 기준	**양호** : 중요한 요청(비밀번호 변경, 송금 등)에 대해 CSRF 방어 토큰이 적용되어 있으며, 토큰 검증이 정상적으로 수행되는 경우
	취약 : 중요한 요청에 대해 CSRF 토큰이 없거나, 토큰 검증을 수행하지 않아 인증된 사용자의 요청 위조가 가능한 경우
조치 방법	중요한 요청에는 CSRF 방어 토큰을 포함하고, 서버 측에서 해당 토큰의 유효성을 검증하도록 설정하며, Referer/Origin 헤더 검증 및 SameSite 쿠키 옵션을 활용하여 불필요한 외부 도메인 요청이 차단되도록 구성
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

Step 1) 크로스사이트 스크립팅(XSS) 취약점 존재 여부 확인 후 데이터 수정 기능(게시글 등록, 비밀번호 변경 등)이 존재하는 요청(Request) 정보를 분석하여 임의의 명령을 수행하는 스크립트 삽입 후 해당 게시글을 타 사용자가 열람하였을 경우 타 사용자의 권한으로 해당 스크립트의 실행 유무 확인

[CSRF 취약점 점검]

- 조치 방법

1. 주요 변경 요청(비밀번호 변경, 송금, 개인정보 수정 등)에 대해 CSRF 토큰을 발급하고 요청 시 토큰을 반드시 포함하도록 구현
2. 서버 측에서 CSRF 토큰의 유효성을 검증하여 토큰이 누락되거나 위조된 요청은 차단하도록 설정
3. Referer 및 Origin 헤더 검증을 통해 외부 사이트에서 발생한 요청 차단
4. SameSite 쿠키 옵션을 적용하여 외부 사이트에서 인증 쿠키가 자동 전송되지 않도록 설정
5. HTTPS 환경에서 쿠키에 Secure, HttpOnly 속성을 적용하여 세션 탈취 및 악용 가능성 최소화

※CSRF Token

- CSRF Token은 서버에서 생성한 고유하고 예측 불가능한 값으로 각 사용자 세션 또는 요청마다 생성되며 생성된 Token은 클라이언트의 요청에 포함되어 서버로 전송되고 서버는 이를 검증하여 요청의 정상 여부 판단

CSRF Token 생성 예시

```
// Java
...
// CSRF Token 생성 및 세션 저장
public String index(Model model, HttpServletRequest request) {
```

CSRF Token 생성 예시

```
    HttpSession session = request.getSession();
    String csrfToken = generateCsrfToken();
    session.setAttribute("csrfToken", csrfToken);
    model.addAttribute("csrfToken", csrfToken);
    model.addAttribute("inputs", inputs);
    return "index";
}

// CSRF Token 생성 함수
private String generateCsrfToken() {
    SecureRandom secureRandom = new SecureRandom();
    byte[] token = new byte[16];
    secureRandom.nextBytes(token);
    return Base64.getUrlEncoder().encodeToString(token);
}

// CSRF Token 검증
@PostMapping("/submit")
public String submit(@RequestParam("input") String input,
    @RequestParam("csrfToken") String csrfToken,
        HttpServletRequest request,
        Model model) {
            HttpSession session = request.getSession();
            String sessionToken = (String) session.getAttribute("csrfToken");

            if (sessionToken == null || !sessionToken.equals(csrfToken)) {
                throw new IllegalStateException("Invalid CSRF token");
            }
            String sanitizedInput = sanitizeInput(input);
            inputs.add(sanitizedInput);
            model.addAttribute("inputs", inputs);
            return "index";
        }

// index.html
〈form action="/submit" method="post"〉
        〈input type="hidden" name="csrfToken" th:value="${csrfToken}" /〉
```

CSRF Token 생성 예시

```
        〈label for="input"〉Enter text:〈/label〉
        〈input type="text" id="input" name="input"〉
        〈button type="submit"〉Add〈/button〉
〈/form〉
...
```

SF (상)	Web Application(웹)
	8. 서버사이드 요청 위조(SSRF)
개요	
점검 내용	입력값을 통해 외부에서 직접적인 접근이 제한된 내부 서버 자원에 접근하여 악의적인 요청을 처리하거나 중요 정보의 유출 여부 점검
점검 목적	입력값 검증을 통해 내부 서버 자원에 대한 비인가 접근을 차단하여 중요 정보(개인정보, 금융 정보 등) 탈취, 데이터 변조, 임의 명령 실행 등 악의적인 행위를 방지하기 위함
보안 위협	서버 간 통신 시 입력값에 대한 검증이 미흡할 경우, 외부에서 접근이 제한된 내부 서버 자원에 대한 정보 수집, 중요 정보(개인정보, 금융 정보, 인사 정보 등) 탈취, 임의 명령 실행, 클라우드 환경 내 메타데이터 수집을 통한 네트워크 인프라 장악이 가능함
참고	※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 애플리케이션 서버, 웹 방화벽, API 서버
판단 기준	**양호 :** 외부 입력값이 화이트리스트 방식으로 검증되어, 허용된 URL 또는 IP 범위 내에서만 처리될 경우
	취약 : 외부 입력값이 검증이 이루어지지 않고 처리되어 허용되지 않는 자원에 임의적인 접근 및 요청이 가능한 경우
조치 방법	입력값 검증 및 화이트리스트를 적용하여 허용된 URL과 IP주소만 접근 가능하도록 설정하며, 네트워크를 분리하여 내부 자원에 대한 비인가 접근을 차단
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

- 점검 방법

Step 1) 사용자 입력을 통해 서버 간 통신이 이루어지는 지점에서 허용되지 않은 주소값을 입력하여 응답, 지연 시간 등을 분석해 취약점 가능성 확인

[서버 간 통신 유무 확인]

```
<div class="section_header section_header_red">
  <div id="about">
  </div>
  It works!
</div>
<div class="content_section_text">
  <p>
    This is the default welcome page used to test the correct
    operation of the Apache2 server after installation on Debian systems.
    If you can read this page, it means that the Apache HTTP server installed at
    this site is working properly. You should <b>
      replace this file
    </b>
     (located at
    <tt>
      /var/www/html/index.html
    </tt>
    ) before continuing to operate your HTTP server.
```

[입력값에 대한 내부 서버 응답값 노출 여부 확인]

Step 2) 습득한 정보를 바탕으로 우회 기법, 포트 스캔, 내부 정보 탈취 등 익스플로잇 시도 및 영향 평가

[심화 공격 수행]

- 조치 방법

1. 외부 요청에 대해 허용된 URL이나 IP주소를 화이트리스트로 정의하여 허용된 대상에만 접근이 가능하도록 설정
2. 내부 네트워크 대역 및 관리용 포트에 대한 요청을 감지하고 차단
3. URL 접근에 실패할 경우 사용자에게 에러 정보나 응답값을 노출하지 않고, 일반적인 에러메시지 출력
4. http, https 외의 다른 프로토콜 (FTP, SMB, SMTP 등)과 URL 스키마(file://, gopher://, data://, dict:// 등)에 대한 접근을 차단해야 하며, 내부 호스트명이 외부에 노출되지 않도록 DNS 설정을 조정
5. 애플리케이션 서버와 중요 내부 시스템간 네트워크 분리를 통하여 불필요한 통신을 제한하여 권한 없는 접근과 외부로부터의 직접적인 접근을 방지

※ Java

화이트 리스트 방식을 이용한 URL 및 IP주소 접근 제한 로직 예시

```
...
private final List<String> allowedDomains = Arrays.asList("example.com", ...);
private final Map<String, List<Integer>> allowedIPsAndPorts = new HashMap<>();

public UrlValidator() {
    allowedIPsAndPorts.put("192.168.1.100", Arrays.asList(80, 443, 8080));
    allowedIPsAndPorts.put("10.0.0.1", Arrays.asList(80, 443));
}
...
public boolean isUrlAllowed(String urlString) {
    try {
```

```
        URL url = new URL(urlString);
        String protocol = url.getProtocol();

        // HTTP와 HTTPS 스키마만 허용
        if (!("http".equalsIgnoreCase(protocol) || "https".equalsIgnoreCase(protocol))) {
            return false;
        }

        String host = url.getHost();
        int port = url.getPort() == -1 ? url.getDefaultPort() : url.getPort();

        if (allowedDomains.contains(host)) {
            return true;
        }

        if (allowedIPsAndPorts.containsKey(host)) {
            return allowedIPsAndPorts.get(host).contains(port);
        }

        return false;
    } catch (Exception e) {
        return false;
    }
}
...
```

※ ASP.NET

화이트 리스트 방식을 이용한 URL 및 IP주소 접근 제한 로직 예시

```
...
private readonly List<string> _allowedDomains = new List<string> { "example.com", ... };
private readonly Dictionary<string, List<int>> _allowedIPsAndPorts = new Dictionary<string, List<int>>
{
    { "127.0.0.1", new List<int> { 80, 443, 8000 } },
    { "10.0.0.1", new List<int> { 80, 443 } }
};
public bool IsUrlAllowed(string urlString)
{
```

```
    if (!Uri.TryCreate(urlString, UriKind.Absolute, out Uri uri))
    {
        return false;
    }
    // HTTP와 HTTPS 스키마만 허용
    if (uri.Scheme != Uri.UriSchemeHttp && uri.Scheme != Uri.UriSchemeHttps)
    {
        return false;
    }

    string host = uri.Host;
    int port = uri.Port;
    // 도메인 확인
    if (_allowedDomains.Contains(host))
    {
        return true;
    }
    // IP주소와 포트 확인
    if (_allowedIPsAndPorts.TryGetValue(host, out List<int> allowedPorts))
    {
        return allowedPorts.Contains(port);
    }
    return false;
}
...
```

※ PHP

화이트 리스트 방식을 이용한 URL 및 IP주소 접근 제한 로직 예시

```
...
function isUrlAllowed($url) {
    $allowedDomains = ['example.com', 'api.example.com'];
    $allowedIPsAndPorts = [
        '192.168.10.10' => [80, 443, 8000],
        '10.0.0.1' => [80, 443]
    ];
    $parsedUrl = parse_url($url);
    if (!$parsedUrl || !isset($parsedUrl['host'])) {
```

```
        return false;
    }
    $host = $parsedUrl['host'];
    $port = isset($parsedUrl['port']) ? $parsedUrl['port'] : ($parsedUrl['scheme'] === 'https' ? 443 : 80);
    // 도메인 확인
    if (in_array($host, $allowedDomains, true)) {
        return true;
    }
    // IP주소와 포트 확인
    if (filter_var($host, FILTER_VALIDATE_IP)) {
        if (array_key_exists($host, $allowedIPsAndPorts)) {
            return in_array($port, $allowedIPsAndPorts[$host], true);
        }
    }
    return false;
}
...
```

- php.ini 파일 내 allow_url_fopen 속성 활성화 시 file_get_contents() 함수를 이용하여 원격 URL에 대하여 접근 및 검색이 가능. 해당 속성을 비활성화하여 원격 URL을 참조할 수 없도록 제한하며 대안으로 cURL 라이브러리 사용

cURL 라이브러리 사용 예시

```
...
function fetchUrl($url) {
    // cURL 세션 초기화
    $ch = curl_init();

    // cURL 옵션 설정
    curl_setopt_array($ch, [
        CURLOPT_URL => $url,
        CURLOPT_RETURNTRANSFER => false,     // 결과를 문자열로 반환
        CURLOPT_FOLLOWLOCATION => false,     // 리다이렉트 제한(기본값: false)
        CURLOPT_MAXREDIRS => 3,              // 최대 리다이렉트 횟수 지정
        CURLOPT_TIMEOUT => 30,               // 세션의 최대 허용 시간 지정 (초)

        // 접근 가능한 프로토콜을 http, https로 제한
        CURLOPT_PROTOCOLS => CURLPROTO_HTTP | CURLPROTO_HTTPS,
```

```
        CURLOPT_SSL_VERIFYPEER => true,     // SSL 인증서 검증
    ]);

    // cURL 실행 및 결과 저장
    $response = curl_exec($ch);
    $error = curl_error($ch);
    $info = curl_getinfo($ch);

    // cURL 세션 종료
    curl_close($ch);

    // 결과 반환
    return [
        'success' => ($error === ''),
        'content' => $response,
        'error' => $error,
        'info' => $info
    ];
}
...
```

▪ allow_url_include 속성 활성화 시 원격 URL을 PHP의 include(), require() 함수를 통해서 사용 가능. 공격자가 악의적인 코드가 포함된 원격 파일을 실행할 수 있는 위험이 존재하므로 php.ini 파일 내 해당 속성 비활성화

allow_url_include 및 allow_url_fopen 속성 비활성화

```
...
;;;;;;;;;;;;;;;;;;
; Fopen wrappers ;
;;;;;;;;;;;;;;;;;;

; Whether to allow the treatment of URLs (like http:// or ftp://) as files.
; https://php.net/allow-url-fopen
allow_url_fopen=Off

; Whether to allow include/require to open URLs (like https:// or ftp://) as files.
; https://php.net/allow-url-include
allow_url_include=Off
...
```

BF (상)	Web Application(웹)
	9. 약한 비밀번호 정책
개요	
점검 내용	웹 애플리케이션 내 로그인 폼 등 비밀번호를 설정하는 단계에서 약한 강도의 문자열 사용 여부를 점검하고, 비밀번호 복잡성 요구사항(최소 길이, 대문자 및 소문자, 숫자 및 특수문자 포함 등)을 준수 여부 점검
점검 목적	유추 가능한 취약한 문자열(이름, 생년월일 등)이나 낮은 복잡성의 비밀번호 사용을 제한하여 계정 및 비밀번호 추측 공격을 방지하기 위함
보안 위협	해당 취약점이 존재할 경우, 유추가 용이한 계정 및 비밀번호 사용으로 인해 사용자 권한 탈취 위험이 있으며, 이를 방지하기 위해 비밀번호의 적절성 및 복잡성을 검증하는 로직을 구현해야 함
참고	※ **약한 비밀번호 정책 취약점:** 웹 사이트에서 취약한 비밀번호로 회원가입이 가능할 경우, 공격자는 비밀번호를 추측하거나 주변 정보를 수집하여 작성한 사전 파일을 이용하여 사용자 계정의 탈취가 가능한 취약점 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드
판단 기준	**양호 :** 관리자 및 사용자 계정의 비밀번호가 유추하기 어려운 값으로 설정되어 있거나 높은 복잡성의 비밀번호 정책이 설정되어 있는 경우
	취약 : 관리자 및 사용자 계정의 비밀번호가 유추하기 쉬운 값으로 설정되어 있거나 낮은 복잡성의 비밀번호 정책이 설정되어 있는 경우
조치 방법	높은 복잡성의 비밀번호 설정 기준을 확립하며, 계정 및 비밀번호의 체크 로직 추가 구현
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

- 점검 방법

Step 1) 웹 애플리케이션 내 추측 가능한 계정이나 비밀번호를 통하여 로그인이 가능한 계정이 존재 여부 확인

[취약한 계정 정보를 통한 로그인 시도]

Step 2) 회원가입 및 사용자 정보 수정 페이지 내 비밀번호 설정 시 높은 복잡도를 요구하는지 확인

[높은 복잡도의 비밀번호 정책 설정 유무 확인]

- 조치 방법

1. 테스트 및 관리자 계정 사용 시 취약한 ID 및 비밀번호 사용을 제한
2. 사용자가 취약한 계정이나 비밀번호를 등록하지 못하도록 비밀번호 규정이 반영된 체크 로직을 회원가입, 정보 변경, 비밀번호 변경 등 적용 필요한 페이지에 모두 구현(아래의 예시와 같이 정규 표현식을 통해 비밀번호 복잡도 검증 로직을 구현)
3. KISA 지식플랫폼 → 법령·가이드라인 → 안내서에서 비밀번호 선택 및 이용 안내서를 통해 안전하게 비밀번호를 생성하고 관리하는 방법 참고
   (https://www.kisa.or.kr/2060305/form?postSeq=14)

※ 취약한 ID/비밀번호 예시

구분	예시
취약한 ID	admin, administrator, manager, guest, tomcat, root, user, operator, anonymous 등
취약한 비밀번호	abcd, 1234, 1111, test, password, public, blank 비밀번호, ID와 동일한 비밀번호 등

※ 규정 예시

규정 예시
Step 1) 다음 각 목의 문자 종류 중 2종류 이상을 조합하여 최소 10자리 이상 또는 3종류 이상을 조합하여 최소 8자리 이상의 길이로 구성 (1) 영문 대문자(26개) (2) 영문 소문자(26개) (3) 숫자(10개) (4) 특수문자(32개) Step 2) 연속적인 숫자나 생일, 전화번호 등 추측하기 쉬운 개인정보 및 아이디와 비슷한 비밀번호는 사용하지 않는 것을 권고 Step 3) 비밀번호에 유효기간을 설정하여 반기별 1회 이상 변경 Step 4) 최근 사용되었던 비밀번호 재사용 금지

※ Javascript

비밀번호 복잡성 검증 예시

```
function isPasswordStrong(password) {
    const minLength = 8;
    const hasUpperCase = /[A-Z]/.test(password);
    const hasLowerCase = /[a-z]/.test(password);
    const hasNumber = /[0-9]/.test(password);
    const hasSpecialChar = /[!@#$%^&*(),.?":{}|<>]/.test(password);
    return password.length >= minLength && hasUpperCase && hasLowerCase && hasNumber &&
hasSpecialChar;
}document.getElementById('password').addEventListener('input', function() {
    const password = this.value;
    const strengthMessage = isPasswordStrong(password) ? 'Strong' : 'Weak';
    document.getElementById('strengthText').textContent = `Strength: ${strengthMessage}`;
});
```

4. 약한 비밀번호 정책을 사용하고, 비밀번호 입력 실패 횟수 제한 설정 또한 미흡한 경우 사전대입 공격 및 무차별 대입 공격을 통한 계정 탈취가 가능하므로 일정 횟수(3~5회) 이상 초과할 경우 계정이 잠기도록 서버 사이드 스크립트(PHP, ASP, JSP 등)를 통하여 임계 로직 구현 권고

IA (상)	Web Application(웹)
	10. 불충분한 인증 절차
개요	
점검 내용	중요 페이지 접근 시 추가 인증 절차 존재 여부 및 인증 로직 우회 여부 점검
점검 목적	중요 페이지 접근 시 추가 인증 절차를 도입하고, 서버 사이드에서 인증 여부를 검증하여 불필요한 정보 노출 및 변조를 차단하기 위함
보안 위협	중요 페이지 및 인증 로직(개인정보 수정, 본인인증, OTP 인증 등)에 대한 인증 절차가 미흡할 경우 무단 접근으로 인해 중요 정보가 유출되거나 변조될 가능성이 있으므로, 해당 구간에는 추가적인 인증 절차를 서버 사이드 방식으로 구현 및 검증
참고	※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드
판단 기준	**양호** : 중요 정보 페이지 접근 시 추가 인증 절차가 존재하며, 인증 로직이 서버 사이드에서 구현되어 우회가 불가능한 경우
	취약 : 중요 정보 페이지 접근 시 추가 인증 절차가 존재하지 않거나, 인증 로직 우회가 가능하여 비인가자가 접근 가능할 경우
조치 방법	중요 페이지에 대한 추가 인증 절차를 도입하고, 서버 사이드에서 인증 여부를 검증하는 로직 구현
조치 시 영향	일반적인 경우 영향 없음

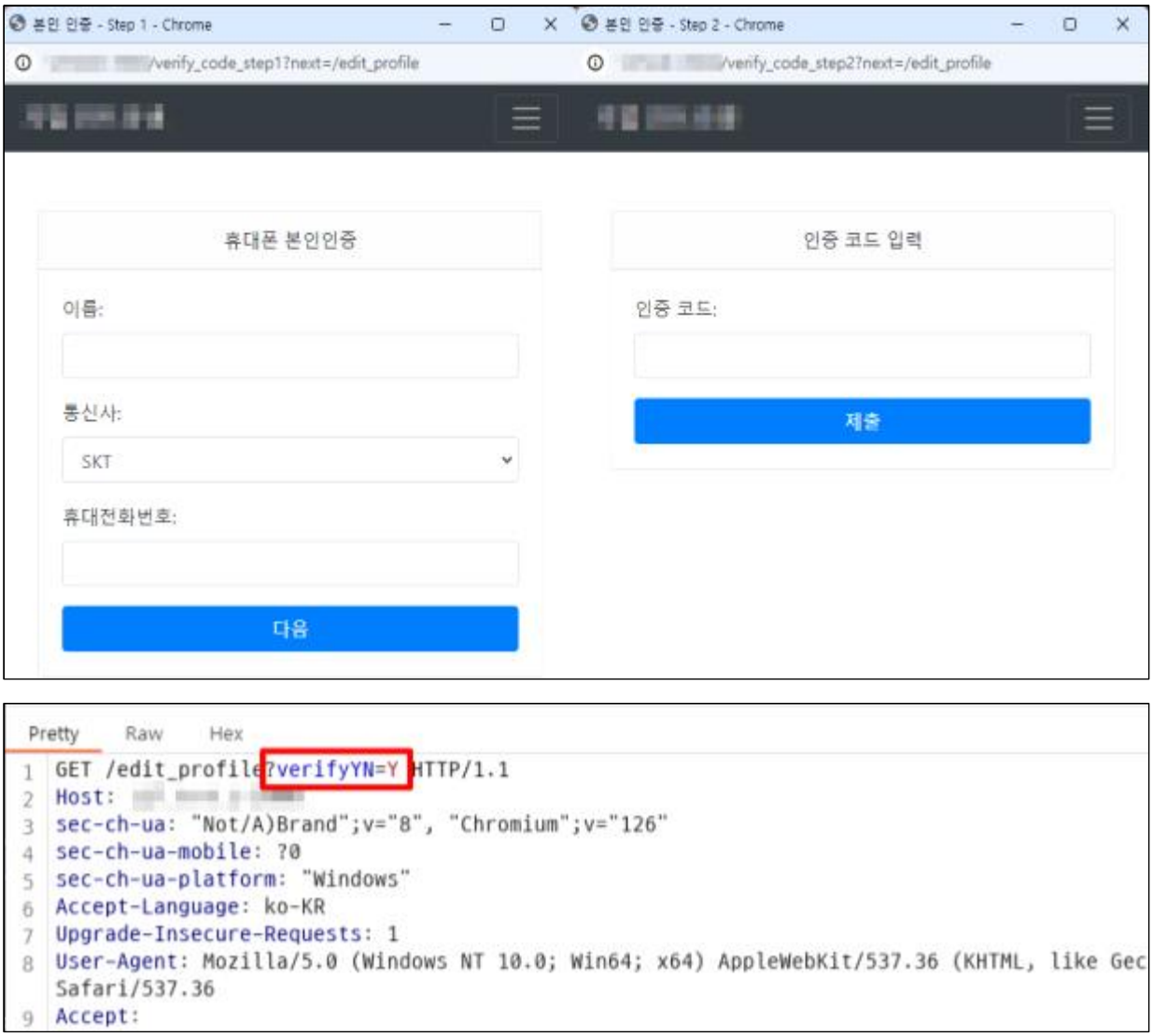
점검 및 조치 사례

- 점검 방법

Step 1) 중요 정보(개인정보 변경 등) 페이지 접근 시 재인증 절차 존재 여부 확인

[2차 인증 유무 확인]

Step 2) 로직 변조, 파라미터 변조 등의 행위를 통하여 인증 로직(OTP 인증, 휴대폰 본인인증 등)의 우회 여부 확인



[인증 로직 우회 여부 확인]

- 조치 방법

1. 중요 정보를 다루는 페이지에 접근 시 본인인증을 재확인하는 로직을 구현하고, 사용자 승인 여부를 페이지마다 검증
2. 인증 과정을 처리하는 로직 구현 시 클라이언트 사이드 방식으로 구현할 경우 사용자가 임의로 인증 과정에 대한 우회가 가능하므로 서버 사이드 방식을 이용하여 구현
3. 접근 통제 코드는 구조화되고 모듈화되어야 하며, 모든 페이지에 로그인 및 권한 체크 기능을 구현하고 공통 모듈 사용 권장

▪ 재인증 로직 구현: 중요 정보(개인정보 변경 등)를 표시하거나 수정하는 페이지에 접근 시 사용자가 최근에 본인인증을 수행했는지 확인하는 로직을 구현하며, 사용자가 페이지에 접근할 때마다 세션을 통해 인증된 사용자임을 검증

재인증 로직 구현 예시

```
...
public String editProfile(HttpSession session, Model model) {
    User user = (User) session.getAttribute("user");
    Boolean isVerified = (Boolean) session.getAttribute("isVerified");    // 세션을 통해 인증 유무 검증

    if (user == null || isVerified == null || !isVerified) {
        return "redirect:/user/authenticate";
    }
    model.addAttribute("user", user);
    return "edit_profile";
}
...
@PostMapping("/verify_code")
public String verifyCode(@RequestParam String code, HttpSession session) {
    if (input.equals(code)) {
        session.setAttribute("isVerified", true);
        return "redirect:/user/edit_profile";
    } else {
        return "redirect:/user/authenticate?error=true";
    }
}
...
```

▪ 접근 제어 로직을 별도의 클래스로 분리 및 관리하여 접근 통제 로직을 공통 모듈로 구현하여 코드의 일관성 유지

접근 통제 공통 모듈 예시

```
...
public class AccessControl {
    public static boolean isAuthenticated(HttpSession session) {
        return session.getAttribute("user") != null;
    }
```

```
    ...
    public static boolean isVerified(HttpSession session) {
        return Boolean.TRUE.equals(session.getAttribute("isVerified"));
    }
}
...
```

▪ 서버 사이드 스크립트 사용: 인증과정을 처리할 때 클라이언트 사이드 스크립트(Javascript 등)를 사용하지 않고, 서버 사이드 스크립트(PHP, Java, ASP.NET 등)를 사용하여 인증 및 필터링 과정을 수행하며, 모든 인증 및 권한 검증 로직은 서버 사이드에서 수행함으로써 클라이언트 측에서는 단순히 UI 제공 및 요청 전송 역할만 담당하도록 유도

서버 사이드 인증 모듈 예시

```
@PostMapping("/login")
public String login(@RequestParam String username,
                    @RequestParam String password,
                    HttpSession session,
                    Model model) {
    User user = userService.findByUsername(username);
    if (user != null && user.getPassword().equals(password)) {
        session.setAttribute("user", user);
        session.setAttribute("isVerified", false);     // 인증 세션 초기값 설정
        return "redirect:/user/dashboard";
    } else {
        model.addAttribute("error", "Invalid username or password");
        return "login";
    }
}
```

IN (상)	Web Application(웹)
	11. 불충분한 권한 검증
개요	
점검 내용	타 사용자의 권한을 탈취하여 민감한 데이터 접근 및 수정 가능 여부 점검
점검 목적	사용자 검증 로직을 서버 사이드에서 구현하여 비인가자로부터 악의적인 접근을 차단하기 위함
보안 위협	패킷 변조, 클라이언트 측 로직 변조 등을 포함한 사용자 식별이 가능한 시퀀스 등의 데이터 변조를 통해 타 사용자의 권한을 탈취할 경우, 개인정보 유출 및 데이터 조작이 가능하므로 서버 사이드에서 권한 검증 로직을 구현해야 함
참고	※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드
판단 기준	**양호 :** 중요 페이지에 사용자 검증 로직이 구현되어 있어, 타 사용자의 권한 탈취가 제한된 경우
	취약 : 중요 페이지에 사용자 검증 로직이 미흡하여, 타 사용자의 권한 탈취가 가능한 경우
조치 방법	접근 제어가 필요한 모든 페이지에 서버 사이드 방식 사용자 권한 검증 로직 구현
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

- 점검 방법

Step 1) 타 사용자 접근이 제한된 페이지(비밀 게시글, 개인정보 변경 등)에서 사용되는 URL 구조와 파라미터를 분석하여 사용자 간 구분을 ID, 숫자, 일련번호 등 단순한 값의 사용 여부 확인

[타 사용자 권한 탈취 시도]

Step 2) 식별된 URL 구조와 파라미터를 변조하여 타 사용자의 비공개 정보나 권한 외 리소스에 대한 접근 가능 여부 확인

[타 사용자 권한 탈취 유무 확인]

- 조치 방법

1. 세션을 이용한 사용자 검증 로직을 서버 측에 구현하여, 인가된 사용자만 중요 페이지에 접근할 수 있도록 구현
2. 각 페이지의 접근 권한을 정의하는 권한 매트릭스를 작성하고, 모든 페이지에 대해 해당 매트릭스를 참조하여 서버 사이드에서 권한 체크를 일관되게 수행

서버 사이드 세션 검증 예시

```
@GetMapping("/inquiry/{id}")
public String viewInquiry(@PathVariable Long id, Model model, HttpSession session) {
    User currentUser = (User) session.getAttribute("currentUser"); // 세션에서 현재 사용자 정보를 가져옴

// 세션에 사용자 정보가 없으면 로그인되지 않은 상태이므로 401 Unauthorized 응답
if (currentUser == null) {
        throw new ResponseStatusException(HttpStatus.UNAUTHORIZED, "Not logged in");
    }
    Inquiry inquiry = inquiryService.findInquiryById(id); // ID에 해당하는 문의를 데이터베이스에서 가져옴

// 현재 사용자가 문의의 작성자가 아니면 403 Forbidden 응답을 보냄
    if (!inquiry.getUser().getUsername().equals(currentUser.getUsername())) {
        throw new ResponseStatusException(HttpStatus.FORBIDDEN, "permission denied");
    }
}
...
```

PR (상)	Web Application(웹)
	12. 취약한 비밀번호 복구 절차
개요	
점검 내용	비밀번호 복구 기능 사용 시, 단순 정보(이름, 사번, 아이디 등)만을 활용하거나 SMS나 이메일 인증 시 발급되는 임시 비밀번호가 동일하거나 유추 가능 여부 점검
점검 목적	비밀번호 복구 로직을 유추하기 어렵게 구현하고, 인증된 사용자 메일이나 SMS에서만 복구 비밀번호를 확인할 수 있도록 하여 비인가자가 사용자 비밀번호를 획득 및 변경하지 못하도록 방지하기 위함
보안 위협	취약한 비밀번호 복구 로직(비밀번호 찾기 등)으로 인하여 공격자가 불법적으로 다른 사용자의 비밀번호를 획득, 변경할 수 있음
참고	※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드
판단 기준	**양호 :** 비밀번호 재설정 시 난수를 이용하여, 인증된 사용자 메일이나 SMS로 임시 비밀번호 또는 비밀번호 재설정을 위한 링크가 전송될 경우
	취약 : 비밀번호 재설정 시 일정 패턴으로 재설정되고 웹 사이트 화면에 바로 출력될 경우
조치 방법	비밀번호 복구 로직을 강화하고, 인증된 사용자 메일이나 SMS에서만 재설정된 비밀번호를 확인할 수 있도록 하여 비인가자가 비밀번호를 획득하지 못하도록 조치
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

- 점검 방법

Step 1) 비밀번호 복구 기능 유무를 파악하고, 복구 과정에서 보안 질문이 추측 가능하거나 소셜 엔지니어링으로 쉽게 답을 찾을 수 있는 단순 정보를 요구하는지 확인

[비밀번호 복구 시 보안 질문 단순성 확인]

Step 2) 비밀번호 복구 시 재설정된 비밀번호에 대하여 추측가능한 일정 패턴으로 발급 유무 확인

[발급 비밀번호에 대한 복잡성 판단]

Step 3) 비밀번호 복구 과정에서 해당 계정에 등록된 이메일 및 전화번호가 아닌 공격자의 정보로 변조가 가능한 포인트의 점검 및 패킷 변조를 통해 공격자 측으로 인증 번호 및 임시 비밀번호 발급 여부 확인

[비밀번호 초기화 시 사용자 정보에 대한 무결성 검증]

- 조치 방법

1. 추측이 어렵고 공개적으로 알 수 없는 정보를 기반으로 한 보안 질문 사용
2. 비밀번호 재설정 과정에서 사용자의 메일이나 SMS로 인증 코드 전송 후 이를 확인하는 2단계 인증 도입
3. 사용자의 개인정보(연락처, 주소, 메일 주소 등)로 비밀번호의 생성을 제한하며, 불규칙적이고 최소 길이(6자 이상 권고) 이상의 복잡도를 만족하는 비밀번호를 발급 및 웹 사이트 화면에 바로 출력하지 않고 인증된 사용자의 메일이나 SMS로 전송되게 구현
4. 비밀번호 재발급 검증 실패에 대한 임계값을 설정하여 일정 횟수 이상 실패한 경우 다른 방식으로 비밀번호 찾기 기능을 제공하고, 검증 후 기존의 비밀번호가 아닌 임시 비밀번호를 발급하도록 설계하며, 사용자가 임시 비밀번호를 재사용하지 못하도록 발급받은 즉시 새로운 비밀번호로 재설정하도록 구현

※ Java

▪ SecureRandom : 난수를 생성하는 데 사용되는 클래스로, 일반 Random 클래스와 달리 시드를 명시적으로 설정할 수 없으므로 키 생성, 보안 토큰 등 보안이 필요한 애플리케이션의 경우 SecureRandom 사용

SecureRandom 클래스를 사용하여 안전한 임시 비밀번호를 생성하는 예시

```
private static final String CHARACTERS =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
//대소문자 영문, 숫자 조합으로 12자리의 난수 생성 규칙 정의
private static final int PASSWORD_LENGTH = 12;
    private String generateTemporaryPassword() {
        SecureRandom secureRandom = new SecureRandom();
        StringBuilder password = new StringBuilder(PASSWORD_LENGTH);

        for (int i = 0; i < PASSWORD_LENGTH; i++) {
            int randomIndex = secureRandom.nextInt(CHARACTERS.length());
            password.append(CHARACTERS.charAt(randomIndex));
        }
    return password.toString()
    } //임시 비밀번호로 사용될 난수 생성
```

※ ASP.NET

▪ RNGCryptoServiceProvider : 난수를 생성하는 데 사용되는 클래스로, 사전에 정의된 문자 집합에서 안전한 암호화 알고리즘을 기반으로 문자를 선택하여 복잡한 난수를 생성

RNGCryptoServiceProvider 클래스를 사용하여 안전한 임시 비밀번호를 생성하는 예시

```
protected void btnCheckRandom_Click(object sender, EventArgs e) {
    lblRandomNumber.Text = GenerateAlphanumericRandom(12); // 자리수 정의(12)
    }

    private string GenerateAlphanumericRandom(int length) {
        const string chars =
                "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
            //대소문자 영문, 숫자 조합으로 생성 규칙 정의
        StringBuilder result = new StringBuilder(length);
        byte[] randomBytes = new byte[4 * length];

        using (var rng = new RNGCryptoServiceProvider())
        {
```

```
        rng.GetBytes(randomBytes);
        for (int i = 0; i < length; i++)
        {
            uint randomInt = BitConverter.ToUInt32(randomBytes, i * 4);
            result.Append(chars[(int)(randomInt % (uint)chars.Length)]);
        }
    }
    return result.ToString();
}
...
```

※ PHP

▪ rand 메소드 보다 random_int 메소드가 암호학적으로 안전한 난수를 생성 (PHP 7.0 버전부터 사용 가능)

random_int() 메소드를 사용하여 안전한 임시 비밀번호를 생성하는 예시

```
...
function generateRandomPassword($length = 12) {
    $characters =
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!@#$%^&*()';
    $charactersLength = strlen($characters);
    $randomPassword = '';
    for ($i = 0; $i < $length; $i++) {
        $randomPassword .= $characters[random_int(0, $charactersLength - 1)];
    }
    return $randomPassword;
}
....
if ($_SERVER['REQUEST_METHOD'] == 'POST') { //세션 정보와 'userid' 값이 일치하는지 검증
    $userid = $_POST['userid'];
    $email = $_POST['email'];

    if (isset($_SESSION['userid']) && $_SESSION['userid'] === $userid) {
        $temporaryPassword = generateRandomPassword();
        updateUserPassword($userid, password_hash($temporaryPassword, PASSWORD_DEFAULT));
        if (sendPasswordEmail($email, $temporaryPassword)) {
            echo "귀하의 이메일로 임시 비밀번호가 발송되었습니다.";
....
```

PV (상)	Web Application(웹)
	13. 프로세스 검증 누락
개요	
점검 내용	서비스 제공에 필요한 사용자 입력 및 실행단계의 흐름에 대한 검증의 적절성 여부 점검
점검 목적	서버 사이드에서 적절한 검증을 통해 비정상적인 입력 및 프로세스 흐름으로 허용되지 않은 웹 애플리케이션 내 발생할 수 있는 논리적 오류를 차단하기 위함
보안 위협	웹 애플리케이션 내 프로세스 또는 기능에 대한 접근 제어 및 검증이 미흡할 경우, 비정상적인 논리 오류를 유발하여, 중요 페이지에 대한 URL 직접 접근, 가격 변조 등의 다양한 행위가 가능하며 서비스 제공에 불이익이 발생할 수 있음
참고	※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드
판단 기준	**양호** : 프로세스에 대한 검증이 존재하며, 악의적인 행위(URL 직접 접근, Javascript 로직 변조 등)를 통하여 논리 오류가 발생하지 않는 경우
	취약 : 프로세스에 대한 검증이 미흡하여, 논리 오류를 통한 의도된 기능이 왜곡되거나 보안 취약점이 노출되는 경우
조치 방법	프로세스 검증이 필요한 경우 각 프로세스에 대한 체크 로직 구현
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

- 점검 방법

Step 1) 웹 사이트 내 기능들의 권한 종류 및 프로세스의 흐름을 파악하고 각 프로세스의 통제를 우회 및 악용 가능성 여부 점검(권한 상승, 민감 정보 획득, 가격 변조 등)

▪ 비 로그인 상태로 URL 직접 접근을 통해 내부 사용자 관리 페이지로 접근할 수 있는지 확인하는 예시

[내부 페이지 URL 직접 접근 시도]

▪ 가격 정보에 대한 검증이 미흡하여, 파라미터 변조를 통해 변조된 가격으로 상품 구매 가능 여부 확인

[프록시 도구를 이용한 가격 변조 시도]

- 조치 방법

1. 모든 비즈니스 로직에서 예상된 프로세스 흐름을 검토하여, 중간 단계를 생략하거나 우회할 수 없도록 플로우 제어 로직을 추가하여, 사용자가 단계별로 진행했는지 검증하도록 구현
2. 인증이 필요한 주요 정보 페이지에 접근 요청자의 권한을 검증하는 로직을 구현하고 임의로 수정할 수 없도록 서버 사이드에서 구현된 프로세스를 사용(세션 기반 접근 통제, 토큰 기반 접근 통제, Referer 검증 등)

※ Java

▪ 세션을 확인하여 비인가자의 접근을 통제하는 로직의 추가 예시

세션을 통한 비인가자 접근 통제 로직

```
public class SessionConfig implements WebMvcConfigurer {
    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new SessionInterceptor())
                .addPathPatterns("/home", "/board/**")
                .excludePathPatterns("/login", "/register", "/error");
    } //비공개 페이지와 공개용 페이지 정의
}
...
public class SessionInterceptor implements HandlerInterceptor {
    public boolean preHandle(HttpServletRequest request,
```

```
                                HttpServletResponse response,
                                Object handler) throws Exception {
        if (request.getSession().getAttribute("user") == null) {
            response.sendRedirect("/login");
            return false;
        } //보호된 페이지 접근 시 요청을 가로채, preHandle 메서드를 호출하여 세션을 검증
        return true;
...
```

※ ASP.NET

- Page_Load : ASP.NET Web Form에서 제공하는 이벤트 핸들러. 페이지 생명주기의 일부로써, 페이지의 로드 시마다 자동으로 호출. ASP.NET 런타임은 페이지 클래스에서 Page_Load라는 이름의 메서드를 찾아 자동으로 이동하며, 페이지가 로드될 때 실행해야 하는 초기화 코드를 넣는데 주로 사용

Page_Load 메서드를 통한 세션 검증 로직

```
protected void Page_Load(object sender, EventArgs e) {
    if (Session["UserName"] == null)
        {
        //세션 내 UserName 키값이 존재하지 않으면 접근권한 없음
        Response.Redirect("~/Login.aspx");
    }
}
```

- 어트리뷰트 : 코드의 메타데이터를 나타내는데 사용되는 선언적 태그. 클래스, 메서드, 속성, 이벤트, 필드 등과 같은 다양한 프로그래밍 요소에 추가 정보를 제공. [Authorize] 어트리뷰트의 경우, 컨트롤러나 액션 메서드에 대한 접근을 인증된 사용자로 제한하는 기능을 담당함

[Authorize] 어트리뷰트를 통한 특정 페이지 접근 정책 설정 예시

```
//설정 파일(Program.cs) 내 사용자 정의 권한 정책 생성
...
builder.Services.AddAuthorization(options =>
{    options.AddPolicy("AdminOnly", policy => policy.RequireClaim("Auth", "admin"));
});
...

//로그인 컨트롤러에서 auth값 클레임 추가
```

[Authorize] 어트리뷰트를 통한 특정 페이지 접근 정책 설정 예시

```
...
[HttpPost]
public async Task<IActionResult> Login(string username, string password)
{    var user = _context.Users.FirstOrDefault(u => u.Username == username);
    if (user != null)
    {
        var result = _passwordHasher.VerifyHashedPassword(user, user.Password, password);
        if (result == PasswordVerificationResult.Success)
        {
            var claims = new List<Claim>
            {
                new Claim(ClaimTypes.Name, username),
                new Claim("FullName", user.Name),
                new Claim("Auth", user.Auth)
            };
        }
....

//권한 제어 페이지 컨트롤러 내 [Authorize] 어트리뷰 사용
...
[Authorize(Policy = "AdminOnly")]
    public class AdminController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
...
```

※ PHP

세션 변수 내 플로우 제어 값 설정 로직

```
...
session_start();
if ($step_completed) {
```

```
    $_SESSION['step1_completed'] = true;
    header('Location: step2.php'); // 이전 단계 완료 시 2단계로 접근
    exit;
}

// 2단계 직접 접근 시
if (!isset($_SESSION['step1_completed']) || $_SESSION['step1_completed'] !== true) {
    header('Location: step1.php'); // 1단계를 완료하지 않은 경우, 1단계 과정으로 리다이렉트
    exit;
}
...
```

FU (상)	Web Application(웹)
	14. 악성 파일 업로드
개요	
점검 내용	웹 애플리케이션 내 업로드 기능 이용 시 악성 파일의 업로드 및 실행 가능 여부 점검
점검 목적	업로드되는 파일의 확장자에 대한 적절성을 검증하는 로직을 구현하여 악성 파일(Server Side Script, exe, dll, bat 등)의 업로드를 방지하고, 서버에 저장된 파일 경로를 유추하여 해당 파일의 실행을 제한하기 위함
보안 위협	해당 취약점이 존재할 경우, 공격자는 악성 파일을 서버에 업로드 및 실행하여 시스템 관리자 권한을 획득하거나 인접 서버에 대한 침입을 시도할 수 있음
참고	※ **Server Side Script**: 웹에서 사용되는 스크립트 언어 중 서버 측에서 실행되는 스크립트 ※ **악성 콘텐츠**: Flash 파일이나 dll, bat, exe 실행 파일 등 악성코드가 포함될 수 있는 콘텐츠 ※ 기반시설 특성상 원칙적으로 업로드 기능을 제한해야 하나, 부득이하게 사용해야 하는 경우 특정 사용자만 허용된 확장자의 콘텐츠 파일을 업로드할 수 있도록 구현 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 애플리케이션 서버, 웹 방화벽
판단 기준	**양호 :** 업로드되는 파일에 대한 확장자 검증이 이루어지는 경우
	취약 : 업로드되는 파일에 대한 확장자 검증이 이루어지지 않고 업로드 경로 접근 시 정상적으로 실행이 가능한 경우
조치 방법	업로드되는 파일에 대한 확장자 검증 및 실행 권한 제거
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

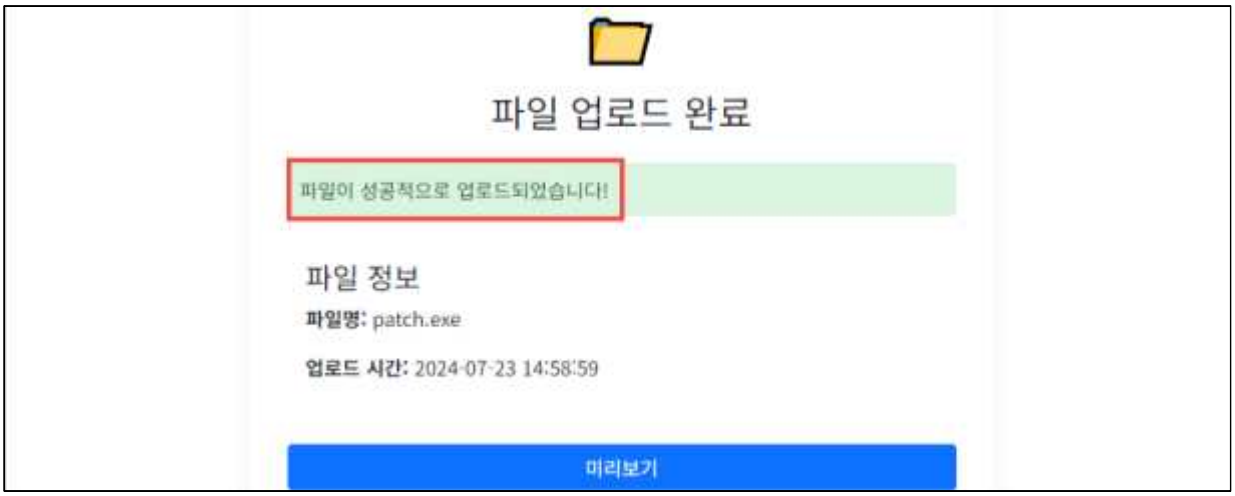
● 악성 콘텐츠 업로드

- 점검 방법

Step 1) 파일 업로드 기능 이용 시 파일 확장자(.exe, .bat, .sh, .dll 등) 검증 여부 확인

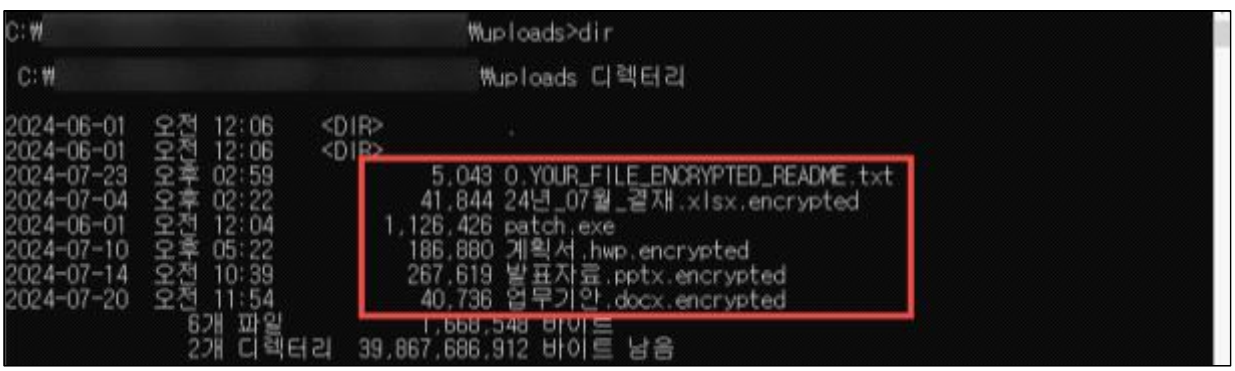
[악성 파일 업로드 시도]

Step 2) 임의 악성 파일에 대하여 정상적으로 업로드 가능 여부 확인



[악성 파일 업로드 유무 확인]

Step 3) 파일 다운로드 및 특정 서비스 간 실행 기능 제공 시 클라이언트/서버에 대하여 악성 코드 감염 가능성 여부 확인

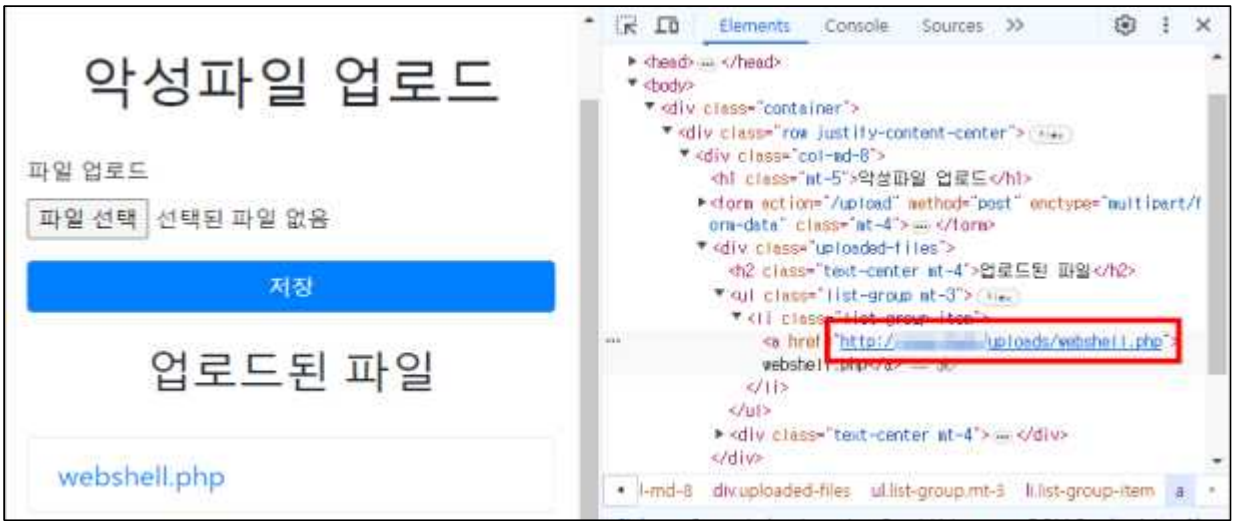


[악성 파일에 대한 익스플로잇 및 실행 가능성 여부 확인]

Server Side Script 업로드

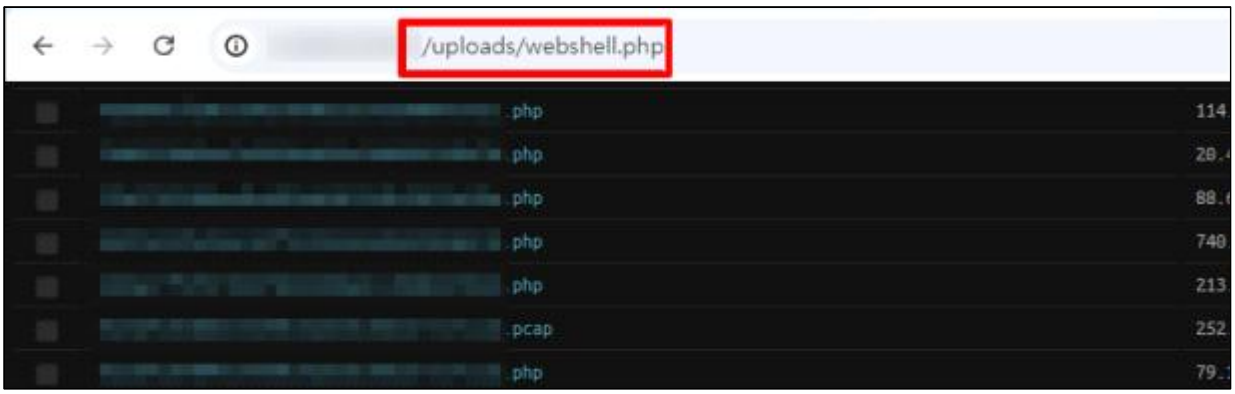
- 점검 방법

Step 1) Server Side Script 파일 업로드 및 파일 경로 확인



[업로드 된 악성 Server Side Script에 대한 경로 확인]

Step 2) 업로드된 Server Side Script 파일 경로 접근 시 파일 실행 여부 확인



[악성 Server Side Script 실행 유무 확인]

- 조치 방법

1. 업로드 파일명에 인코딩/디코딩, 널바이트, 태그 등을 제거할 수 있도록 정규화 및 필터링
2. 업로드 파일의 확장자 및 MIME 타입에 대해 화이트리스트 방식으로 검증 로직을 구현하여 서버 사이드로 하여금 허용된 파일 유형만 업로드를 허용하며 대용량 파일 업로드 시 용량 제한 구현
3. 업로드된 파일의 이름을 암호화 후 저장하여 파일 이름을 유추할 수 없도록 처리
4. 업로드된 파일의 실행 권한을 제한하여 해당 파일이 서버 사이드에서 실행되지 않도록 설정
5. 업로드 경로에 대하여 웹 디렉터리와 격리 조치
6. 주기적으로 업로드된 파일을 대상으로 바이러스 검사 실시

※ Java

▪ 화이트리스트 방식의 확장자 검증 및 MIME 타입 검증 로직을 구현하여 허용된 유형의 파일만 업로드 허용

파일 업로드 보안코드 예시

```
...
private static final String[] ALLOWED_EXTENSIONS = {"jpg", "png", "pdf", "txt"};
private static final Set<String> ALLOWED_MIME =  Set.of("image/jpeg","image/png","application/pdf",
"text/plain");
...
// 파일명 정규화
private static String normalizeFilename(String filename) {
        if (filename == null) return null;
        String name = java.net.URLDecoder.decode(filename, StandardCharsets.UTF_8);
        name = Normalizer.normalize(name, Normalizer.Form.NFC);
        name = name.replace("\0", "");
        name = name.replaceAll("[<>:\"/\\\\|?*]", "");
        name = name.replaceAll("^[.\\s]+|[.\\s]+$", "");
        return name;
    }

// 확장자 추출 + 이중 확장자 차단
private static String getExtension(String filename) {
        String safe = normalizeFilename(filename);
        int dotCount = safe.length() - safe.replace(".", "").length();
        if (dotCount != 1) return ""; // 이중 확장자 차단
        int idx = safe.lastIndexOf('.');
        if (idx == -1) return "";
        return safe.substring(idx+1).toLowerCase();
    }

public static String saveFile(MultipartFile file, String uploadDir) throws IOException {
        String original = file.getOriginalFilename();
        String ext = getExtension(original);

        if (!ALLOWED_EXTENSIONS.contains(ext)) {
            throw new IOException("허용되지 않은 확장자");
        }

// MIME 시그니처 검증
```

```
Tika tika = new Tika();
String mime = tika.detect(file.getInputStream());
if (!ALLOWED_MIME.contains(mime)) {
throw new IOException("허용되지 않은 파일 유형");
}

// 파일명 난수화
String newName = UUID.randomUUID().toString().replace("-", "") + "." + ext;
java.nio.file.Path savePath = java.nio.file.Paths.get(uploadDir, newName);
file.transferTo(savePath.toFile());

return newName;
    }
}

...
```

※ ASP.NET

▪ 화이트리스트 방식의 확장자 검증 로직을 통한 허용된 확장자 파일만 업로드

파일 업로드 보안 코드 예시

```
...
private static readonly string[] AllowedExtensions = { ".jpg", ".jpeg", ".png", ".gif", ".pdf", ".txt" };
    private static readonly string[] AllowedMime = { "image/jpeg","image/png","image/gif","application/pdf",
"text/plain" };
...
// 파일명 정규화
private static string NormalizeFilename(string filename)
    {
        if (string.IsNullOrWhiteSpace(filename)) return string.Empty;
        string name = System.Web.HttpUtility.UrlDecode(filename, Encoding.UTF8);
        name = name.Normalize(NormalizationForm.FormC);
        name = name.Replace("\0", "");
        name = Regex.Replace(name, "[<>:\"/\\\\|?*]", "");
        name = Regex.Replace(name, "^[.\\s]+|[.\\s]+$", "");
        return name;
    }


 // 확장자 검증 (이중 확장자 차단 포함)
    private static bool IsValidExtension(string filename)
    {
        string safeName = NormalizeFilename(filename);
        if (string.IsNullOrEmpty(safeName)) return false;

// 이중 확장자 차단
        if (safeName.Split('.').Length != 2) return false;
        string ext = Path.GetExtension(safeName).ToLowerInvariant();
        return Array.Exists(AllowedExtensions, e => e == ext);
    }

    protected void btnUpload_Click(object sender, EventArgs e)
    {
        if (FileUpload1.HasFile)
        {
            string safeName = NormalizeFilename(FileUpload1.FileName);
```

```
// 확장자 검증
if (!IsValidExtension(safeName))
{
    Response.Write("허용되지 않은 확장자입니다.");
    return;
}

// MIME 타입 검증 (주의: Web Forms의 ContentType은 신뢰성이 낮음)
string mime = FileUpload1.PostedFile.ContentType.ToLowerInvariant();
if (Array.IndexOf(AllowedMime, mime) < 0)
{
    Response.Write("허용되지 않은 MIME 타입입니다.");
    return;
}

// 난수화된 파일명 생성
string ext = Path.GetExtension(safeName).ToLowerInvariant();
string newName = Guid.NewGuid().ToString("N") + ext;

// 저장 경로 (웹 루트 외부 권장)
string uploadPath = Server.MapPath("~/Uploads/");
if (!Directory.Exists(uploadPath))
{
    Directory.CreateDirectory(uploadPath);
}

string savePath = Path.Combine(uploadPath, newName);

try
{
    FileUpload1.SaveAs(savePath);
    Response.Write("업로드 성공: " + HttpUtility.HtmlEncode(newName));
}
catch (Exception ex)
{
    Response.Write("업로드 실패: " + HttpUtility.HtmlEncode(ex.Message));
}
```

```
        }
        else
        {
            Response.Write("업로드할 파일을 선택하세요.");
        }
    }
}
...
```

※ PHP

- move_uploaded_file의 경우, php.ini 파일 내 'upload_tmp_dir' 속성에 정의된 경로에 'php***.tmp' 형식의 임시 파일로 업로드되며, 유효하지 않은 파일의 경우 삭제 처리됨
- 화이트리스트 방식의 확장자 검증 및 MIME 타입 검증 로직을 구현하여 허용된 유형의 파일만 업로드 허용

파일 업로드 보안 코드 예시

```
// 파일명 정규화
    function normalize_filename($filename) {
        $filename = urldecode($filename);
        $filename = normalizer_normalize($filename, Normalizer::FORM_C);
        $filename = str_replace("\0", "", $filename);
        $filename = preg_replace('/[<>:"\/\\\\|?*]/', '', $filename);
        $filename = preg_replace('/^[\.\s]+|[\.\s]+$/u', '', $filename);
        return $filename;
    }
    // 이중 확장자 차단
    function is_valid_extension($filename, $allowed_exts) {
        $safe = normalize_filename($filename);
        if (substr_count($safe, '.') !== 1) return false;
        $ext = strtolower(pathinfo($safe, PATHINFO_EXTENSION));
        return in_array($ext, $allowed_exts, true);
    }
    // 확장자 검증
    function save_upload($file, $uploadDir) {
```

```
    $allowed_exts = ['jpg','jpeg','png','pdf','txt'];
    $allowed_mime = ['image/jpeg','image/png','application/pdf','text/plain'];

    if (!is_valid_extension($file['name'], $allowed_exts)) {
        throw new Exception("허용되지 않은 확장자");
    }

    // MIME 시그니처 확인
    $mime = mime_content_type($file['tmp_name']);
    if (!in_array($mime, $allowed_mime, true)) {
        throw new Exception("허용되지 않은 파일 유형");
    }

    // 파일명 난수화
    $ext = strtolower(pathinfo($file['name'], PATHINFO_EXTENSION));
    $newName = bin2hex(random_bytes(16)) . '.' . $ext;
    $dest = rtrim($uploadDir, DIRECTORY_SEPARATOR) . DIRECTORY_SEPARATOR . $newName;

    if (!move_uploaded_file($file['tmp_name'], $dest)) {
        throw new Exception("파일 저장 실패");
    }

    return $newName;
}
```

FD (상)	Web Application(웹)
	15. 파일 다운로드
개요	
점검 내용	웹 사이트에서 허용된 경로 외 다른 경로의 파일 접근 및 다운로드 가능 여부 점검
점검 목적	허용된 경로 외 다른 경로의 비인가된 접근을 방지하여, 공격자가 임의의 경로에 존재하는 파일을 열람하거나 다운로드하는 것을 차단하기 위함
보안 위협	• 해당 취약점이 존재할 경우, 공격자는 파일 다운로드 시 애플리케이션의 파라미터 값을 조작하여 웹 사이트의 중요한 파일(DB 커넥션 파일, 애플리케이션 파일 등) 또는 웹 애플리케이션 서버 루트에 있는 중요한 설정 파일(/etc/passwd, /etc/shadow 등)을 다운로드할 수 있음 • CGI, JSP, PHP 등 파일 다운로드 기능을 제공하는 애플리케이션에서 입력 경로를 검증하지 않는 경우, 공격자는 임의의 문자(../.. 등)나 주요 파일명을 입력하여 웹 애플리케이션 서버의 홈 디렉터리를 벗어나 임의의 위치에 있는 파일을 열람하거나 다운로드할 수 있음
참고	※ **경로 추적:** 웹 서버와 웹 애플리케이션의 파일 또는 디렉터리에 대한 접근이 적절히 통제되지 않아, 중요한 파일과 데이터에 비인가된 접근을 허용하는 취약점 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 애플리케이션 서버, 웹 방화벽
판단 기준	**양호 :** 입력값이 검증되어 허용된 경로와 파일만 접근 가능하고, 경로 조작 및 임의 시스템 파일 다운로드가 불가능하며, 다운로드 디렉터리 외의 접근과 상위 디렉터리 접근이 차단된 경우
	취약 : 입력값이 검증없이 처리되어 임의의 경로로 접근이 가능하거나 비인가된 파일을 다운로드할 수 있는 경우
조치 방법	다운로드 시 허용된 경로 이외의 디렉터리와 파일에 접근할 수 없도록 구현하고, 서버 사이드에서 ../..와 같은 경로 이동 관련 문자열에 대해 입력값 검증을 수행하여 비인가된 접근을 차단함
조치 시 영향	일반적인 경우 영향 없음

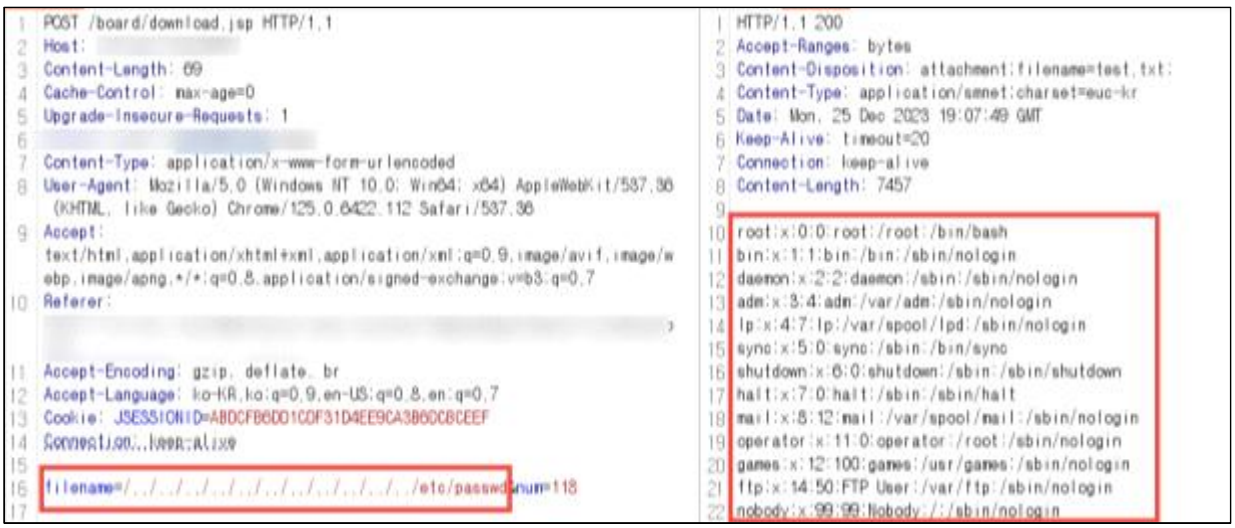
점검 및 조치 사례

- 점검 방법

Step 1) 웹 사이트 내 파일 다운로드 기능 식별 후 파일 다운로드 요청 및 응답 패킷 내 URL 파라미터, POST 데이터, 쿠키 등을 통해 파일 이름 및 경로 노출 여부 확인(또는 웹 사이트 내 이미지 렌더링 시 이미지 파일의 경로를 참조하는 경우)

[프록시 도구를 이용한 파일 다운로드 파라미터 확인]

Step 2) 파일 다운로드 시 요청 패킷 내 파일 경로를 상대 경로(../../ 또는 ..\..\)로 변조하여 요청할 경우 정상적으로 해당 경로 내 파일 다운로드 여부 확인



[프록시 도구를 이용한 경로 조작 및 임의 시스템 파일 다운로드 시도]

Step 3) "Step 2"에서 파일 다운로드가 불가능한 경우 변조한 파일 경로를 아래의 인코딩(또는 치환, 종단 문자 추가)을 적용하여 우회 가능 여부 확인

[입력값 인코딩을 통한 우회 공격 시도]

※ 참고: 운영체제별 중요 시스템 파일 예시

구분	파일 경로	설명
Linux	/etc/passwd	시스템유저 계정 리스트
	/etc/group	시스템그룹 정보 리스트
	/etc/hosts	호스트명과 IP주소 매핑 정보
	/var/log/message	시스템 메시지 로그
	~.bash_history	사용자 명령어 기록
	/etc/service	활성화된 서비스 정보
Windows	C:\Windows\System32\config\SAM	사용자 계정 및 암호의 해시 정보
	C:\Windows\System32\config\SYSTEM	시스템 설정과 관련된 레지스트리 파일
	C:\Windows\System32\config\AppEvent.evt	응용프로그램 이벤트 로그
	C:\Windows\System32\config\SecEvent.evt	보안 이벤트 로그
	C:\Windows\System32\drivers\etc\hosts	호스트명과 IP주소 매핑 정보

※ 참고: 우회 방안 예시

▪ 인코딩 적용 : 필터링 시스템이 특정 문자열 패턴을 차단하는 경우 문자열을 인코딩하여 우회 시도

인코딩 방식	사용 예시
16bit 유니코드 인코딩	.(%u002e), /(%u2215), \(%u2216)
URL 인코딩	.(%2e), /(%2f), \(%5c)
더블 URL 인코딩	.(%252e), /(%252f), \(%255c)
Base64인코딩	../../etc/passwd (Li4vLi4vZXRjL3Bhc3N3ZA==)
HTML 엔티티 인코딩	../../etc/passwd

▪ 특수문자 중첩 사용 : 필터링 시스템이 단일 패턴(../)을 감지하여 소거할 경우, 중복된 경로 문자를 사용하여 우회(..../// → ../)

▪ 종단 문자 추가 : 필터링 시스템이 특정 확장자만 허용할 때 Nullbyte, 개행 문자 등을 사용하여 우회([파일명]%00.jpg , [파일명]%0a.jpg 등)

▪ 문자열 패턴 검증 우회 : 시스템 파일 관련 단어(etc, passwd 등)들을 필터링할 때 대소문자 변환(EtC, PaSswD 등), URL인코딩과 조합(e%74c%2Fpa%73%73wd) 등의 방법을 사용하여 우회

- 조치 방법

1. 파일 다운로드의 취약점은 주로 파일 이름 조작으로 인해 발생되므로 파일의 이름을 데이터베이스에 저장하고 다운로드 수행 시 요청 파일 이름과 비교하여 적절한지 확인하여 사용자가 조작할 수 있는 공격 포인트를 제거
2. 파일 다운로드가 가능한 디렉터리를 별도의 파티션에서 관리하거나 특정 디렉터리로 한정하여 다른 디렉터리에서는 파일을 다운받을 수 없도록 설정
3. URL을 통해 파일 경로를 직접 노출시키지 않고, 파일 ID나 토큰을 사용하여 파일 경로를 은닉
4. 공격자가 입력값 변조를 통한 경로 추적이 불가하도록 파일 확장자 등의 일부 패턴을 화이트리스트로 적용하여 검증하고, 경로 추적 관련 특수문자(. / \ % 등) 필터링 로직 구현

※ Java

특수문자 필터링을 통한 입력값 검증 로직 예시

```
public class FileDownloadUtil {
    //파일 이름에 영문,숫자,일부 특수문자를 제외한 \,/ 문자 등이 탐지되었을 때 다운로드 차단
    public static boolean isValidFileName(String fileName) {
        return fileName != null && fileName.matches("^[a-zA-Z0-9._-]+$");
    }

    private static final Set<String> ALLOWED_EXTENSIONS;

    //허용 확장자 리스트
    static {
        Set<String> tempSet = new HashSet<>();
        tempSet.add("jpg");
        tempSet.add("png");
        ALLOWED_EXTENSIONS = Collections.unmodifiableSet(tempSet);
    }

    public static boolean isAllowedExtension(String filePath) {
        String extension = filePath.substring(filePath.lastIndexOf(".") + 1).toLowerCase();

        // 파일 확장자 검증
        return ALLOWED_EXTENSIONS.contains(extension);
    }
}
...
@GetMapping("/downloadFile")
    public ResponseEntity<Resource> downloadFile(@RequestParam String filename,
HttpServletResponse response) throws IOException {
        if(!FileDownloadUtil.isValidFileName(filename) || !FileDownloadUtil.isAllowedExtension(filename)) {

            // 다운로드 컨트롤러에 검증 로직 적용
            return ResponseEntity.badRequest().build();
        }
...
```

※ ASP.NET

- 경로 정규화: 다양한 형태의 파일 경로를 표준화된 형식으로 변환시키는 과정으로 ../ 와 같은 상대 경로 참조를 해석하고 제거하여 허가되지 않은 디렉터리 접근을 차단함

경로 정규화를 통한 경로 검증 로직 예시

```
filename = Path.GetFileName(filename);

var uploadsFolder = Path.Combine(_environment.WebRootPath, "uploads");
//지정된 업로드 폴더와 정규화된 파일 이름을 결합
var filePath = Path.GetFullPath(Path.Combine(uploadsFolder, filename));
...
if (!filePath.StartsWith(uploadsFolder, StringComparison.OrdinalIgnoreCase)) {
    //해당 파일이 업로드 폴더 내 존재하는지 검증
    return BadRequest("Invalid file path.");
}
...
```

※ PHP

- php.ini 에서 magic_quotes_gpc를 On으로 설정하여 .\./ 와 같은 역 슬러시 문자 입력 시 치환되도록 설정이 가능하나, PHP 5.4.0 이후로는 해당 기능이 제거됨
- filter_input() 함수를 이용해 사용자 입력 데이터를 안전하게 필터링할 수 있으며, realpath() 함수를 이용해 경로를 정규화하고 실제 경로를 반환하므로, 이를 통해 상위 경로 탐색을 방지함

경로 정규화를 통한 경로 검증 로직 예시

```
// 파일 다운로드 요청 처리
if (isset($_GET['file'])) {
    // file 파라미터 검증 및 처리
    $file = filter_input(INPUT_GET, 'file', FILTER_SANITIZE_STRING);
    $filePath = realpath('../uploads/' . $file); //지정된 업로드 디렉터리에 실제 존재하는지 확인

    // 파일 경로가 지정된 디렉토리 내에 있는지 확인
    if ($filePath && strpos($filePath, realpath('../uploads/')) === 0) {
        downloadFile($filePath);
    } else {
        echo "잘못된 파일 경로입니다.";
        exit;
    ...
```

IS (상)	Web Application(웹)
	16. 불충분한 세션 관리
개요	
점검 내용	세션 만료 기간 설정, 예측 가능한 세션 ID 생성, 고정된 세션 ID 발행 등 세션 관리 정책을 점검하고, 인증 토큰(JWT 등) 사용 시에도 안전한 서명 알고리즘 사용 여부, 안전한 비밀 키 사용 여부, 토큰 만료 등의 정책 점검
점검 목적	사용자의 세션 ID를 적절히 관리하여 공격자가 불법적으로 접근하거나 비인가적인 세션 탈취를 차단하기 위함
보안 위협	사용자에게 발급되는 세션 ID가 만료되지 않거나, 고정 및 예측 가능한 형태일 경우, 공격자는 해당 세션 ID를 탈취하여 타 사용자나 시스템에 무단 접근할 수 있으며, 이로 인해 중요 데이터의 무결성이 훼손될 수 있음
참고	※ **세션(Session):** 일정 시간 동안 같은 사용자(브라우저)로 부터 들어오는 일련의 요구를 하나의 상태로 보고 그 상태를 일정하게 유지시키는 기술 ※ **JSON Web Token (JWT):** 헤더, 페이로드, 서명으로 구성되는 JSON 객체 형식이며, 사용자 정보가 클라이언트에 저장되고 서버에 요청 시마다 전송하여 인증 상태를 유지시키는 기술 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 애플리케이션 서버
판단 기준	**양호 :** 추측 불가능한 세션 ID가 발급되고, 세션 종료 시간이 설정되어 있는 경우
	취약 : 세션 ID가 일정한 패턴으로 발급되거나 세션 종료 시간이 설정되지 않아 세션 재사용이 가능한 경우
조치 방법	추측 불가능한 세션 ID가 발급되도록 로직을 구현하고, 세션 종료 시간 설정 또는 자동 로그아웃 기능을 구현하여 사이트 특성에 맞게 적정 시간을 설정
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

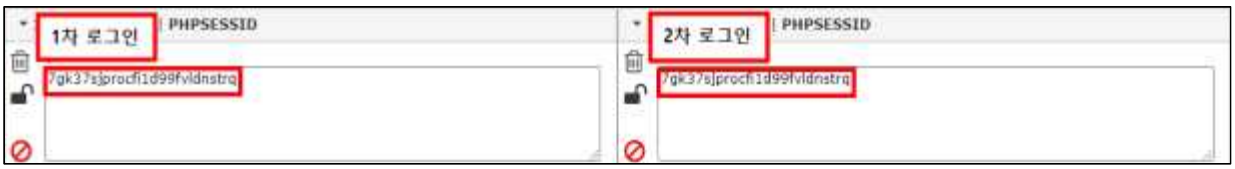
- 점검 방법

● 세션(Session)

Step 1) 로그인 과정에서 발급받은 세션 ID를 확인하고 로그아웃 후 재 로그인 시 각각 발급받은 세션 ID에 대해 일정한 패턴이 존재하는지 검증

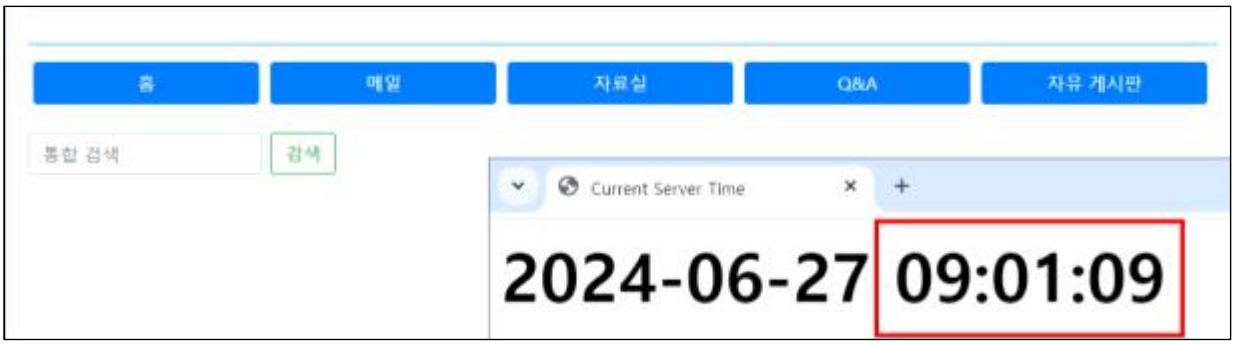
▪ 각각 발급받은 세션 ID를 확인하여 예측 가능한 패턴으로 생성되는지 검증

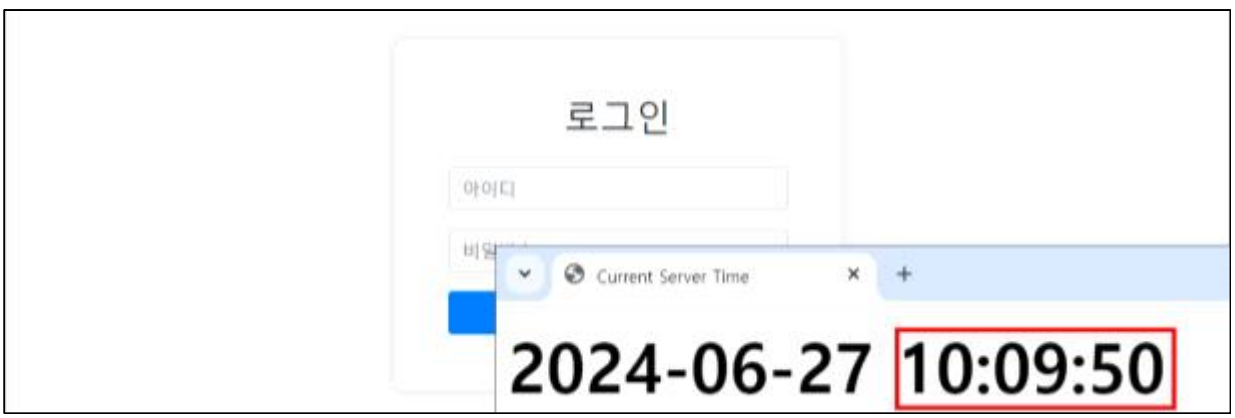
▪ 각각 발급받은 세션 ID를 확인하여 고정된 값으로 생성되는지 검증



[세션 발급 시 동일 패턴 유무 확인]

Step 2) 로그인 후 웹 페이지 사용을 중지한 상태로 일정 시간이 경과하였을 때 세션이 유지되는지 확인



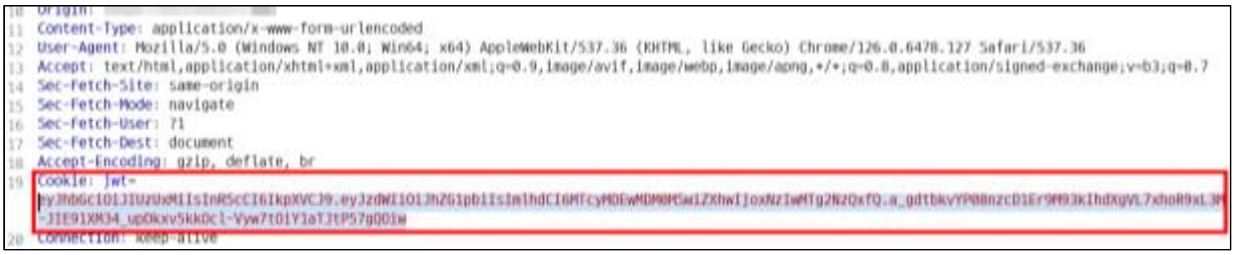


[일정 시간 경과 시 세션 만료 유무 확인]

JWT(JSON Web Token)

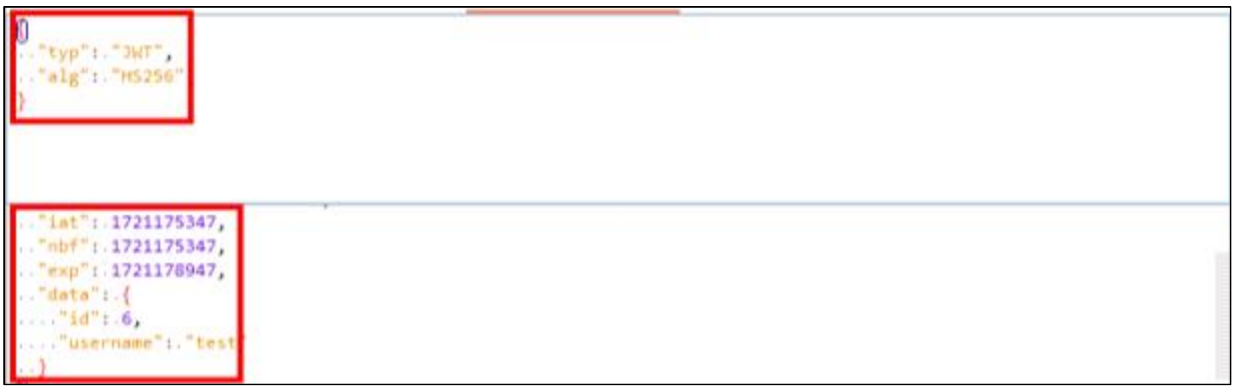
Step 1) 사용자 인증 및 상태 유지를 위해 JWT(JSON Web Token)가 사용되는지 확인

(JWT는 일반적으로 Authorization 헤더에 포함되고, 경우에 따라 Cookie 헤더에 포함, 또는 POST 데이터로 전송할 수 있음)



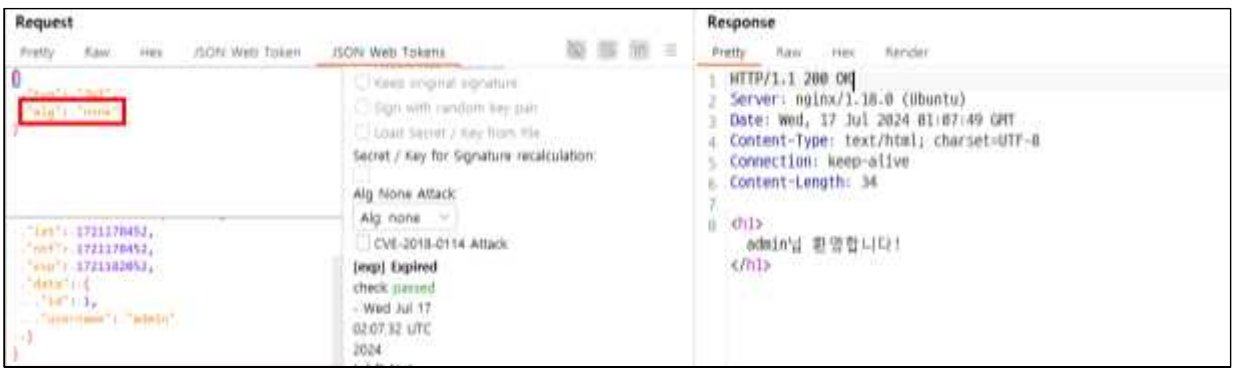
[JWT 토큰 사용 유무 확인]

Step 2) JWT(JSON Web Token)의 헤더와 페이로드 부분을 각각 Base64 디코딩하여 내용을 확인할 수 있으며, 안전한 알고리즘을 사용하는지, 토큰 만료 설정이 적절하게 되었는지, 민감정보가 노출되는지 등을 확인



[JWT 토큰 디코딩 시 민감정보 노출 유무 확인]

Step 3) 페이로드 변조 후 서명을 삭제하거나 None 알고리즘으로 변조하였을 때 토큰이 유효하여 타 계정의 권한 탈취가 가능한지 확인



[JWT 토큰 익스플로잇 가능성 유무 확인]

- 조치 방법

● **세션 고정 및 예측**

1. 공격자가 하나의 유효한 세션 ID를 추측하는 것이 불가능에 가깝도록 길고 복잡한 세션 ID를 생성하여야 하며, 새 로그인 시 기존 세션 ID를 폐기하고 새로운 세션 ID로 발급해야 하며, 이용 중인 계정에 대해 접속 IP와 디바이스 정보를 통해 동시 세션 로그인 여부를 검사해야함

※ Java

세션 생성 로직 예시

```
@RequestMapping("/login")
public String login(HttpServletRequest request) {
    request.changeSessionId(); // 신규 로그인 시 세션 ID 변경
    return "redirect:/home";
}

//Spring Security 설정에서 세션 고정 방지 및 세션 ID 생성
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.sessionManagement()
        .sessionFixation().migrateSession()
        .sessionCreationPolicy(SessionCreationPolicy.IF_REQUIRED)
        .maximumSessions(1).maxSessionsPreventsLogin(false)
        .expiredUrl("/login?expired");
}
```

※ ASP.NET

web.config 파일 설정을 통한 불규칙 세션 ID 발급 설정 예시

```
<system.web>
...
<!--세션 고정 방지 -->
<sessionState regenerateExpiredSessionId="true">
...
<!--세션 예측 방지 -->
<machineKey validationKey="AutoGenerate,IsolateApps"
    decryptionKey="AutoGenerate,IsolateApps"
    validation="HMACSHA512"
```

```
    decryption="AES"
/>
```

※ PHP

세션 생성 로직 예시

```
// 기존 세션을 삭제하고 새로운 세션 ID를 생성
session_start();
session_regenerate_id(true);

// 예측 불가한 안전한 세션 ID 생성
ini_set('session.entropy_length', '256');
ini_set('session.entropy_file', '/dev/urandom');
```

● 세션 만료

1. 세션 타임아웃이 미흡하게 설정되어있는 경우 사용자가 로그아웃하지 않고 시스템을 떠날 때 타 사용자가 재사용할 수 있는 여지가 존재하므로, 세션 타임아웃 기능을 구현하고 서비스에 따라 10~60분으로 설정할 것을 권고(아래 예시들은 60분으로 설정)

※ Java

web.xml 파일 설정을 통한 세션 만료 시간 설정 예시

```
<session-config>
    <session-timeout>60</session-timeout> <!-- 60분 -->
</session-config>
```

▪ Spring Boot의 경우 application.properties 및 소스코드를 통하여 세션 만료 제어 가능

application.properties 파일 설정을 통한 세션 만료 시간 설정 예시

```
server.servlet.session.timeout=60m
```

소스 코드를 통한 세션 만료 시간 설정 예시

```
public String login(HttpSession session) {
    session.setMaxInactiveInterval(3600); // 60분 (3600초)
    return "Session timeout set to 60 minutes";
...
```

※ ASP.NET

web.config 파일 설정을 통한 세션 만료 시간 설정 예시

```
<configuration>
    <system.web>
        <sessionState timeout="60" /> <!-- 60분 -->
    </system.web>
</configuration>
```

▪ Global.asax 파일에서 세션을 확인하고 접근을 제어하는 로직 추가 예시

(ASP.NET Core의 경우 Startup.cs 파일에서 ConfigureServices, Configure 메소드 수정)

Global.asax 파일을 통한 세션 만료 시간 설정 예시

```
...
public class Global : HttpApplication {
    void Session_Start(object sender, EventArgs e) {
        Session.Timeout = 60; // 60분
    }
}
...
```

※ PHP

php.ini 파일 설정을 통한 세션 만료 시간 설정 예시

```
session.gc_maxlifetime = 3600 ; 60분 (3600초)
session.cookie_lifetime = 3600 ; 60분 (3600초)
```

.htaccess 파일 설정을 통한 세션 만료 시간 설정 예시

```
php_value session.gc_maxlifetime 3600
php_value session.cookie_lifetime 3600
```

소스 코드를 통한 세션 만료 시간 설정 예시

```
...
ini_set('session.gc_maxlifetime', 3600); // 60분 (3600초)
ini_set('session.cookie_lifetime', 3600); // 60분 (3600초)
session_start();
...
```

JWT(JSON Web Token)

1. 서명이 없는 JWT(JSON Web Token)는 쉽게 조작할 수 있으므로, 서명이 포함된 JWT를 사용하고 해당 서명에 대한 검증 로직 구현
2. 취약한 서명 알고리즘을 사용하는 경우 공격자가 서명을 위조할 수 있으므로, 강력한 서명 알고리즘 사용 권고
3. 공격자가 JWT의 알고리즘을 none으로 설정하여 서명을 우회할 수 있으므로 토큰의 알고리즘을 명시적으로 설정

- 안전한 서명 알고리즘을 사용하더라도 약한 키(짧거나 예측 가능한 키)는 추측될 위험이 있으므로, 강력하고 랜덤한 비밀 키를 사용하고, 유출되지 않도록 키를 안전하게 관리
- 토큰이 재사용되지 않도록 짧은 만료 시간을 설정하고, 리프레시 토큰을 사용하여 주기적으로 토큰을 갱신

안전한 JWT 서명 알고리즘	
HS256~512	비밀 키를 사용하여 메시지에 해시를 적용 (숫자가 높을수록 긴 출력 길이를 가지며, 높은 수준의 보안을 제공)
RS256~512	비대칭 키 쌍을 사용하여 서명 생성. 서명은 개인 키로 생성되고 공개 키로 검증
ES256~512	타원 곡선 암호화를 사용하여 서명을 생성
취약한 JWT 서명 알고리즘	
HS1	취약한 암호화 기술인 SHA-1을 기반으로 함
RS1	취약한 암호화 기술인 SHA-1을 기반으로 함
none	서명을 생략하여 무결성 검증에 취약함
plaintext	서명을 평문으로 전달하여, 무결성 검증에 취약함

SN (상)	Web Application(웹)
	17. 데이터 평문 전송
개요	
점검 내용	서버와 클라이언트 간 통신 시 데이터의 암호화 여부 점검
점검 목적	서버와 클라이언트 간 통신 시 데이터의 암호화 전송 미흡으로 정보 유출의 위험을 방지하고자 함
보안 위협	웹 애플리케이션 통신은 주로 텍스트 기반으로 이루어지므로, 서버와 클라이언트 간 암호화 프로세스를 구현하지 않을 경우 악의적인 사용자가 네트워크 도청(Sniffing)을 통해 정보를 탈취 및 도용할 수 있음
참고	※ Sniffing : 스니퍼(sniff: 냄새를 맡다, 코를 킁킁거리다)를 이용하여 네트워크 상의 데이터를 도청하는 행위 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 애플리케이션 서버
판단 기준	**양호 :** 중요 정보 전송구간에 암호화 통신이 적용된 경우
	취약 : 중요 정보 전송구간에 암호화 통신이 이루어지지 않는 경우
조치 방법	사이트의 중요 정보 전송구간(로그인, 회원가입, 회원정보관리, 게시판 등)에 대하여 암호화 통신(https, 애플리케이션방식) 적용
조치 시 영향	일반적인 경우 영향 없음

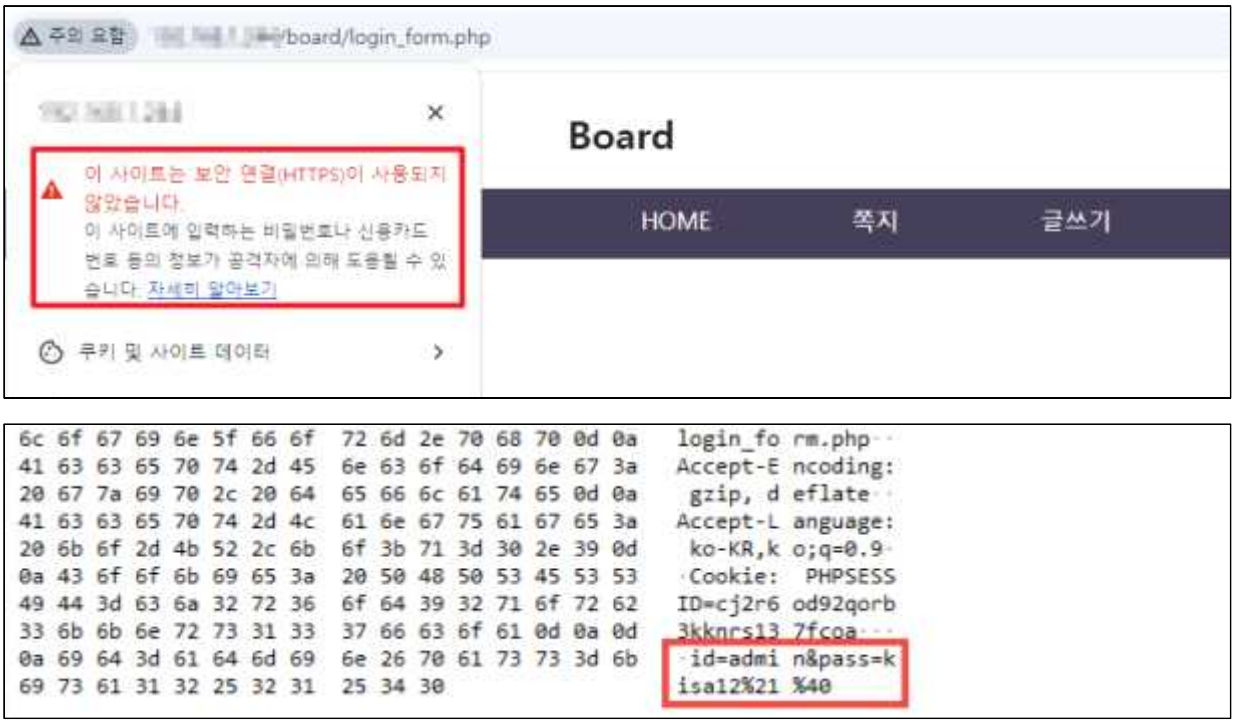
점검 및 조치 사례

- 점검 방법

Step 1) 중요 정보(인증정보, 개인정보, 로그인페이지 등)를 송수신하는 페이지 존재 여부 확인

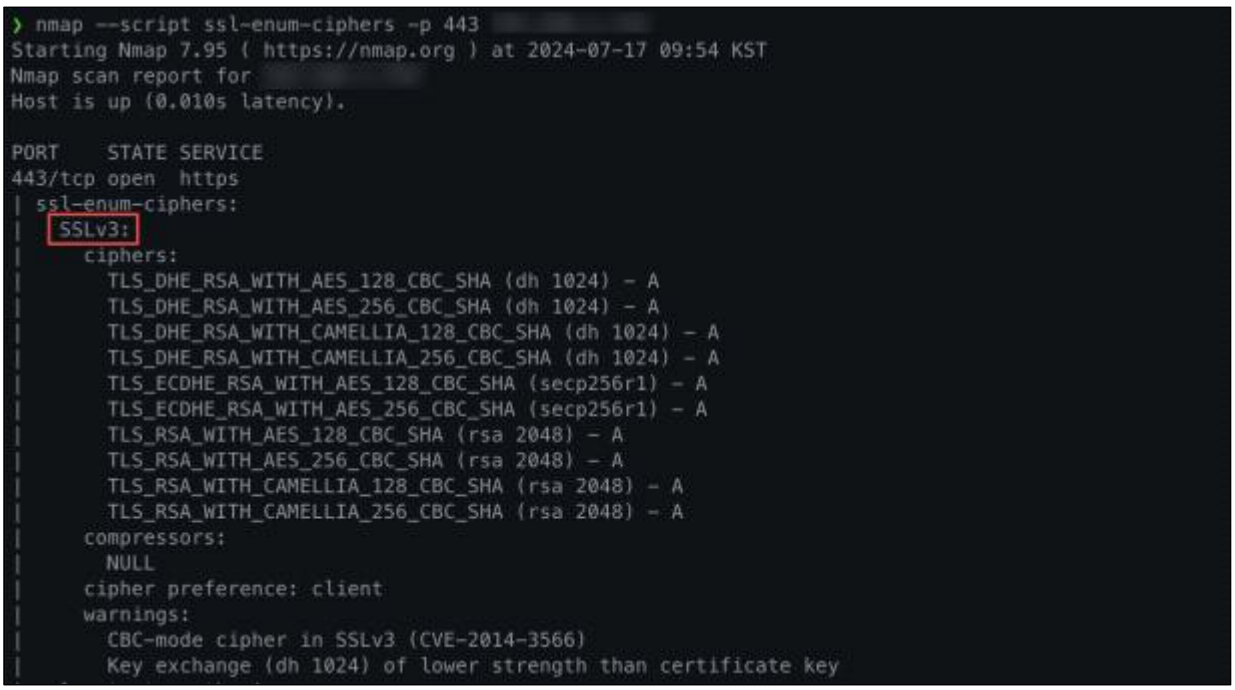
[중요 정보 송수신 페이지 확인]

Step 2) 중요 정보 송수신 페이지가 암호화 통신(https, 데이터 암호화 등)을 하는지 확인



[데이터 통신 시 패킷 내 중요 정보 평문 노출 유무 확인]

Step 3) 취약한 버전의 암호 프로토콜 사용 시 암호화된 통신 내용이 유출될 가능성이 존재하므로 취약한 버전의 SSL(SSL 2.0, 3.0) 사용 여부를 점검



[취약한 버전의 암호 프로토콜 사용 여부 확인]

- 조치 방법

1. 웹상에서의 전송 정보를 제한하여 불필요한 비밀번호, 주민등록번호, 계좌정보와 같은 중요 정보의 전송을 최소화하여야 하며, 중요 정보에 대해서는 SSL 등의 암호화 통신을 사용하여 도청으로부터의 위험을 제거함
2. 쿠키와 같이 클라이언트 사이드에서 노출되는 곳에 비밀번호, 인증인식 값, 개인정보 등의 중요 정보 제거
3. 암호화 전송 시 프로토콜 설계의 결함이 있는 SSLv2, SSLv3은 비활성화 필수, TLSv1.2 이상 사용을 권장함

※ Apache

Apache 서버 설정을 통한 프로토콜 제어 예시
httpd-ssl.conf 또는 ssl.conf의 SSL 관련 VirtualHost 설정에 아래를 추가 SSLProtocol all -SSLv2 -SSLv3 -TLSv1 -TLSv1.1

※ IIS

IIS 서버 설정을 통한 프로토콜 제어 예시
[SSL v2 사용 안 함] [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 2.0\Server] 하위에 '새로만들기' > 'DWord(32비트)' 값 선택 > 이름 부분에 'Enabled' 입력 > 데이터 부분에 '0' 입력 > 시스템 재부팅 **[SSL v3 사용 안함]** [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Protocols\SSL 3.0\Server] 하위에 '새로만들기' > 'DWord(32비트)' 값 선택 > 이름 부분에 'Enabled' 입력 > 데이터 부분에 '0' 입력 > 시스템 재부팅

CC (상)	Web Application(웹)
	18. 쿠키 변조
개요	
점검 내용	쿠키 변조를 통한 임의의 타 사용자 권한 탈취 여부 점검
점검 목적	쿠키를 사용하는 경우, 안전한 알고리즘으로 암호화하여 공격자가 쿠키 값 변조를 통해 다른 사용자로 위장하거나 권한을 변경하는 것을 방지하기 위함
보안 위협	클라이언트에 전달되는 쿠키에 사용자 식별 값이 평문으로 노출될 경우 쿠키 변조를 통해 타 사용자의 유효한 세션을 취득할 수 있으며, 기타 중요 정보의 유출 및 변조 가능함
참고	※ **쿠키(Cookie):** 서버가 사용자의 웹 브라우저에 전송하는 작은 데이터 조각. 브라우저는 그 데이터 조각들을 저장 후, 동일한 서버에 대하여 재요청 시 저장된 데이터를 함께 전송 ※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드
판단 기준	**양호 :** 쿠키를 사용하지 않고 서버 사이드 세션을 사용하고 있거나, 쿠키를 사용하는 경우 안전한 알고리즘(SEED, 3DES, AES)이 적용되어 있는 경우
	취약 : 안전한 알고리즘이 적용되지 않은 쿠키를 사용하거나, 쿠키로만 인증 및 권한 부여를 적용하는 경우
조치 방법	쿠키 대신 서버 사이드 세션 방식을 사용하거나, 쿠키를 통해 인증 등 중요한 기능을 구현해야 하는 경우 안전한 알고리즘(SEED, 3DES, AES 등)을 적용
조치 시 영향	일반적인 경우 영향 없음

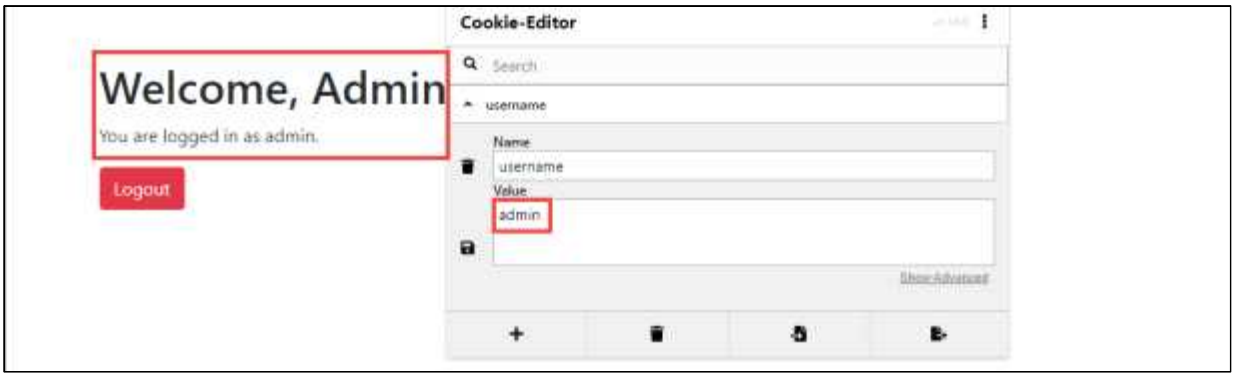
점검 및 조치 사례

- 점검 방법

Step 1) 일반 사용자 계정으로 로그인 한 뒤 쿠키 내용 및 발행되는 쿠키에 중요 정보(인증을 위한 ID, 권한을 위한 구분자 등)의 노출 여부 확인 후 변조 시도

[쿠키 내 사용자 및 주요 권한 검증값 존재 유무 확인]

Step 2) 쿠키 내 노출되는 중요 정보를 변조하여 다른 사용자 및 권한으로 정상 이용이 가능한지 확인



[쿠키 값 변조를 통한 타 사용자 권한 탈취 여부 확인]

- 조치 방법

1. 쿠키 대신 보안성이 강한 서버 사이드 세션 방식 사용. 클라이언트 사이드 방식인 쿠키는 구조상 다양한 취약점에 노출될 가능성이 존재
2. 쿠키를 사용해서 중요 정보나 인증을 구현해야 할 경우엔 안전한 알고리즘(SEED, 3DES, AES 등) 적용
3. 쿠키 서명(HMAC)을 통해 변조 여부 검증 및 HttpOnly, Secure, SameSite 속성을 적용하여 보안 강화

※ Java

AES + HMAC CookieUtil 예시

```
public class CookieUtil {
    private static final int IV_LEN = 16, HMAC_LEN = 32;
    private static final SecureRandom RNG = new SecureRandom();
    private final byte[] encKey;
    private final byte[] hmacKey;

    public CookieUtil(byte[] encKey, byte[] hmacKey) {
        if (encKey.length != 32) throw new IllegalArgumentException("AES key must be 32 bytes");
        this.encKey = encKey;
        this.hmacKey = hmacKey;
    }

    // 쿠키 생성 (HttpOnly, Secure, SameSite 설정 포함)
    public void addSecureCookie(HttpServletResponse resp, String name, String plaintext, int maxAgeSec)
throws Exception {
        byte[] iv = new byte[IV_LEN]; RNG.nextBytes(iv);
```

```
    // AES-256-CBC 암호화
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, new SecretKeySpec(encKey, "AES"), new IvParameterSpec(iv));
    byte[] ciphertext = cipher.doFinal(plaintext.getBytes(StandardCharsets.UTF_8));

    Mac mac = Mac.getInstance("HmacSHA256");
    mac.init(new SecretKeySpec(hmacKey, "HmacSHA256"));
    mac.update(iv); mac.update(ciphertext);
    byte[] hmac = mac.doFinal();

    byte[] payload = new byte[iv.length + hmac.length + ciphertext.length];
    System.arraycopy(iv, 0, payload, 0, iv.length);
    System.arraycopy(hmac, 0, payload, iv.length, hmac.length);
    System.arraycopy(ciphertext, 0, payload, iv.length + hmac.length, ciphertext.length);

    String encoded = Base64.getUrlEncoder().withoutPadding().encodeToString(payload);

    // HttpOnly, Secure 속성 설정
    Cookie cookie = new Cookie(name, encoded);
    cookie.setHttpOnly(true);
    cookie.setSecure(true);
    cookie.setPath("/");
    cookie.setMaxAge(maxAgeSec);

    // SameSite 속성은 Cookie API로 직접 지정 불가 → 헤더로 세팅
    String header = String.format(
        "%s=%s; Max-Age=%d; Path=/; HttpOnly; Secure; SameSite=Strict",
        name, encoded, maxAgeSec
    );
    resp.addHeader("Set-Cookie", header);
}

// 쿠키 읽기 (HMAC 검증 + 복호화)
public String readSecureCookie(String value) throws Exception {
    if (value == null) return null;
    byte[] data;
    try { data = Base64.getUrlDecoder().decode(value); } catch (IllegalArgumentException e) { return null; }
    if (data.length < IV_LEN + HMAC_LEN + 1) return null;
```

```
        byte[] iv = Arrays.copyOfRange(data, 0, IV_LEN);
        byte[] hmac = Arrays.copyOfRange(data, IV_LEN, IV_LEN + HMAC_LEN);
        byte[] ciphertext = Arrays.copyOfRange(data, IV_LEN + HMAC_LEN, data.length);
        // 복호화 전 HMAC 검증
        Mac mac = Mac.getInstance("HmacSHA256");
        mac.init(new SecretKeySpec(hmacKey, "HmacSHA256"));
        mac.update(iv); mac.update(ciphertext);
        byte[] calc = mac.doFinal();

        if (!MessageDigest.isEqual(hmac, calc)) return null; // 변조 탐지 후 복호화
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(encKey, "AES"), new IvParameterSpec(iv));
        return new String(cipher.doFinal(ciphertext), StandardCharsets.UTF_8);
    }
}
```

※ ASP.NET

MachineKey 적용 예시

```
<!-- web.config -->
<machineKey
  validationKey="AutoGenerate,IsolateApps"
  decryptionKey="AutoGenerate,IsolateApps"
  validation="HMACSHA256"
  decryption="AES" />

<!-- cookie.aspx.cs -->
// 쿠키 발급 (FormsAuthenticationTicket 활용)
var ticket = new FormsAuthenticationTicket(
    1, "username", DateTime.Now, DateTime.Now.AddMinutes(30), true, "role=admin");

string encrypted = FormsAuthentication.Encrypt(ticket); // 내부적으로 AES + HMAC 적용

var cookie = new HttpCookie("AuthCookie", encrypted) {
// 쿠키 보안 적용
    HttpOnly = true,
    Secure   = true,
```

```
    SameSite = SameSiteMode.Lax
};

Response.Cookies.Add(cookie);
```

※ PHP

OpenSSL + HMAC 적용 예시

```
<?php
function setSecureCookie($name, $value, $encKey, $hmacKey) {
    $iv = random_bytes(16); // AES-256-CBC IV
    $ciphertext = openssl_encrypt($value, "AES-256-CBC", $encKey, OPENSSL_RAW_DATA, $iv);

    // HMAC 생성 (무결성 검증용)
    $hmac = hash_hmac('sha256', $ciphertext, $hmacKey, true);

    // payload = IV + HMAC + 암호문
    $payload = base64_encode($iv . $hmac . $ciphertext);

    setcookie($name, $payload, [
        'expires' => time() + 3600,
        'httponly' => true,
        'secure'   => true,
        'samesite' => 'Lax'
    ]);
}

function getSecureCookie($name, $encKey, $hmacKey) {
    if (!isset($_COOKIE[$name])) return null;

    $data = base64_decode($_COOKIE[$name]);
    $iv = substr($data, 0, 16);
    $hmac = substr($data, 16, 32);
    $ciphertext = substr($data, 48);

    $calcHmac = hash_hmac('sha256', $ciphertext, $hmacKey, true);
    if (!hash_equals($hmac, $calcHmac)) {
        return null; // 무결성 검증 실패
```

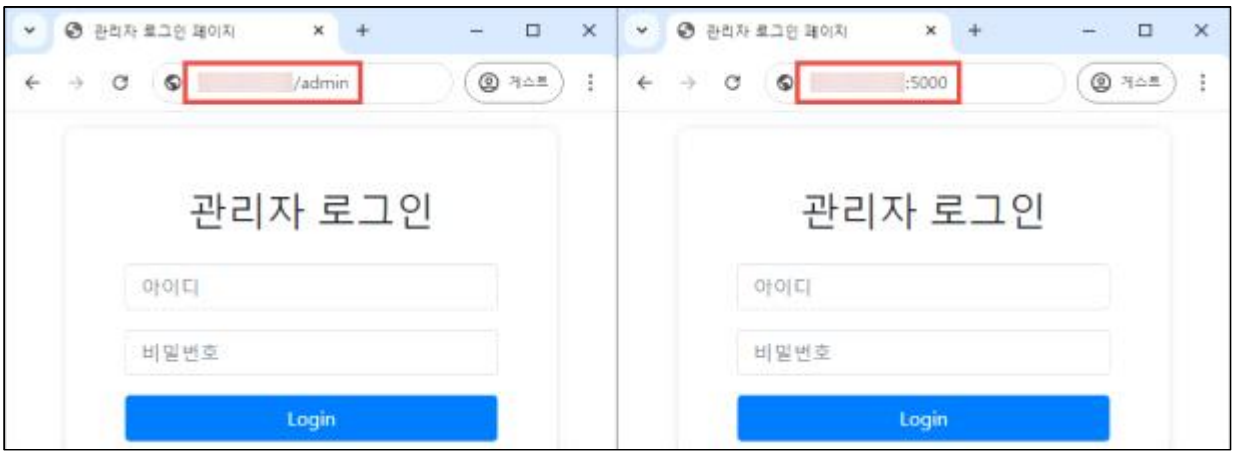
```
    }

    return openssl_decrypt($ciphertext, "AES-256-CBC", $encKey, OPENSSL_RAW_DATA, $iv);
}
?>
```

AE (상)	Web Application(웹)
	19. 관리자페이지 노출
개요	
점검 내용	유추 가능한 URL 또는 설계상의 오류로 인해 관리자 페이지 및 메뉴에 접근 가능 여부 점검
점검 목적	관리자 페이지의 URL을 추측하기 어렵게 설정하고, 웹 사이트 설계 오류를 수정하여 비인가자의 관리자 메뉴 접근을 방지하기 위함
보안 위협	웹 관리자의 권한이 노출될 경우, 웹 사이트 변조뿐만 아니라 취약성 정도에 따라 웹 애플리케이션 서버의 권한까지도 노출될 가능성이 있으므로 시스템 전체의 보안이 심각하게 위협받을 수 있음
참고	※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 애플리케이션 서버, 웹 방화벽
판단 기준	**양호 :** 유추하기 쉬운 URL로 관리자 페이지 접근이 불가능한 경우
	취약 : 유추하기 쉬운 URL로 관리자 페이지 접근 또는 계정 로그인이 가능한 경우
조치 방법	유추하기 어려운 이름(포트 번호 변경 포함)으로 관리자 페이지를 변경하여 비인가자가 접근할 수 없도록 하고, 근본적인 해결을 위해 지정된 IP만 관리자 페이지에 접근할 수 있도록 제한함. 단, 부득이하게 관리자 페이지를 외부에 노출해야 하는 경우, 관리자 페이지 로그인 시 2차 인증(OTP, VPN, 인증서 등)을 적용
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

- 점검 방법

Step 1) 유추하기 쉬운 URL, 포트 등 접속을 시도하여 관리자 페이지가 노출되는지 확인



[유추하기 쉬운 관리자 페이지 URL 접근 시도]

Step 2) 추측하기 쉬운 관리자 계정(admin, adm, administrator, manager 등) 및 비밀번호를 입력하여 로그인 가능한지 확인

[추측 가능한 관리자 계정 로그인 유무 확인]

- 조치 방법

1. 일반 사용자의 접근이 불필요한 관리자 로그인 페이지 주소를 유추하기 어려운 URL 및 포트로 변경
2. 관리자 페이지에 접근 가능한 IP를 지정하여 지정된 IP만 관리자 페이지에 접근 가능하도록 제한
3. 부득이하게 관리자 페이지를 외부에 노출해야 하는 경우, 관리자 페이지 로그인 시 2차 인증(OTP, VPN, 인증서 등)을 적용
4. 지정된 IP만 관리자 페이지에 접근 가능하도록 제한관리자 페이지의 하위 페이지 URL을 직접 입력하여 접근하지 못하도록 페이지 별 세션 검증 필요

CC (상)	Web Application(웹)
	20. 자동화 공격
개요	
점검 내용	웹 애플리케이션의 특정 프로세스(로그인 시도, 게시글 등록, SMS 발송 등)에 대한 반복적인 요청 시 통제 여부를 확인하여, 자동화 공격(봇 공격 등)을 방지 여부 점검
점검 목적	무차별 대입 공격 및 자동화 공격으로 인한 자원 고갈, 계정 탈취, 서비스 거부 상태를 방지하기 위함
보안 위협	웹 애플리케이션의 특정 프로세스에 대한 반복적인 요청을 통제하지 않을 경우, 무차별 대입 공격으로 사용자 계정을 탈취할 수 있으며, 자동화 공격을 통해 게시글 등록 또는 SMS 발송 요청을 반복하여 웹 애플리케이션 자원을 고갈시킬 수 있음
참고	※ 소스코드 및 취약점 점검 필요
점검 대상 및 판단 기준	
대상	웹 애플리케이션 소스코드, 웹 방화벽
판단 기준	**양호 :** 웹 애플리케이션의 특정 프로세스에 대한 반복적인 요청 시 통제가 적절한 경우
	취약 : 웹 애플리케이션의 특정 프로세스에 대한 반복적인 요청 시 통제가 미흡한 경우
조치 방법	웹 애플리케이션의 특정 프로세스에 대한 대량 사용을 통제하는 로직을 구현하고, 웹 방화벽의 룰셋을 설정하여 대량의 불특정 프로세스 요청을 차단
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

- 점검 방법

Step 1) 로그인 실패 일정 횟수 초과 시 반복적인 요청에 대한 통제가 미흡한지 확인

[자동화 공격을 통한 사용자 계정 로그인 유무 확인]

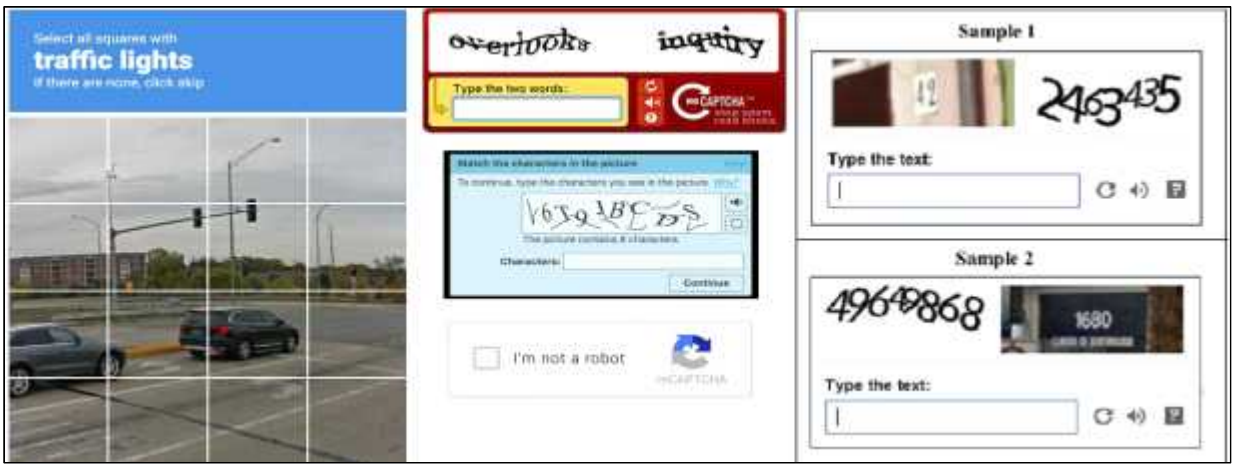
Step 2) 본인인증(계좌 인증, SMS인증 등)을 반복적으로 시도하여 반복적인 요청에 대한 통제가 미흡한지 확인

[자동화 공격을 통한 인증번호 탈취 유무 확인]

- 조치 방법

1. 로그인 시도, 게시글 등록, 본인인증(계좌 인증, SMS 발송 등)에 대한 사용자 요청에 대하여 횟수 제한을 설정 또는 *캡차 등 일회성 확인 로직을 구현해야 함
2. 자동화 공격을 시도하면 짧은 시간에 다량의 패킷이 전송되므로 이를 공격으로 감지하고 방어할 수 있는 IDS/IPS 시스템을 구축해야 함.

※ 캡차(CAPTCHA): 사람과 컴퓨터를 구분하기 위한 자동화된 테스트



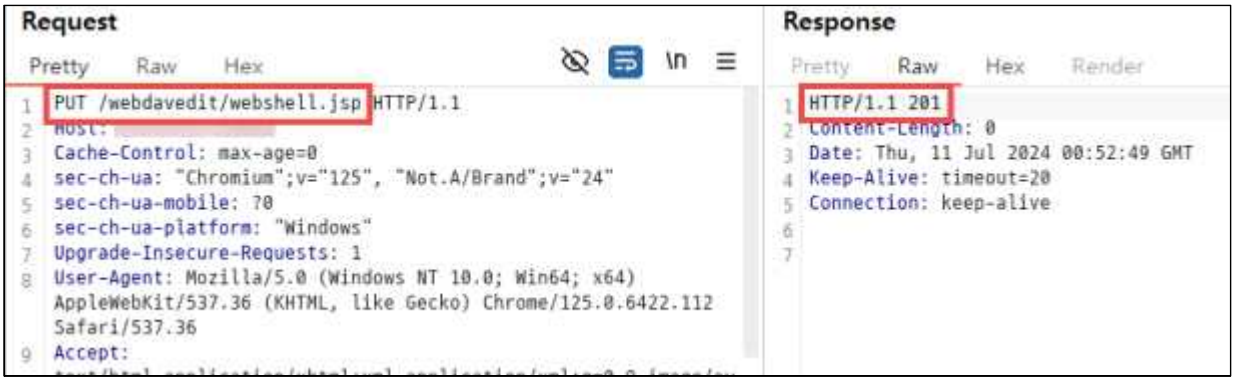
[캡차(CAPTCHA) 예시]

WM (상)	Web Application(웹)
	21. 불필요한 Method 악용
개요	
점검 내용	PUT, DELETE, TRACE 등 불필요한 HTTP 메소드의 악용 여부 확인
점검 목적	불필요한 HTTP 메소드(PUT, DELETE, TRACE 등) 요청을 제한하여 서버의 무단 접근 및 악의적인 행위(임의의 파일 생성, 데이터 삭제 등)를 방지하기 위함
보안 위협	PUT, DELETE, TRACE 등의 메소드가 활성화되어 있는 것만으로는 취약 여부를 판단하기 어려우나, 해당 메소드들이 악용될 경우, 조작된 Server Side Script 파일 업로드, 민감 데이터 삭제, 서버 정보 노출 등이 가능함
참고	※ Server Side Script: 웹에서 사용되는 스크립트 언어 중 서버 측에서 실행되는 스크립트 ※ XST(Cross-Site Tracing): TRACE 메소드를 악용하여 httpOnly 등으로 보호된 세션 및 쿠키를 탈취할 수 있는 공격 기법
점검 대상 및 판단 기준	
대상	웹 애플리케이션 서버
판단 기준	**양호 :** 웹 애플리케이션에 불필요한 메소드(PUT, DELETE, TRACE, CONNECT 등)로 변조한 요청 패킷 전송 시 해당 메소드를 통하여 임의의 행위로부터 악용이 제한되는 경우
	취약 : 웹 애플리케이션에 불필요한 메소드(PUT, DELETE, TRACE, CONNECT 등)로 변조한 요청 패킷 전송 시 해당 메소드를 통하여 임의의 행위에 대한 악용이 가능한 경우
조치 방법	WEB/WAS의 설정파일을 변경하여 불필요한 메소드의 사용을 제한
조치 시 영향	일반적인 경우 영향 없음

점검 및 조치 사례

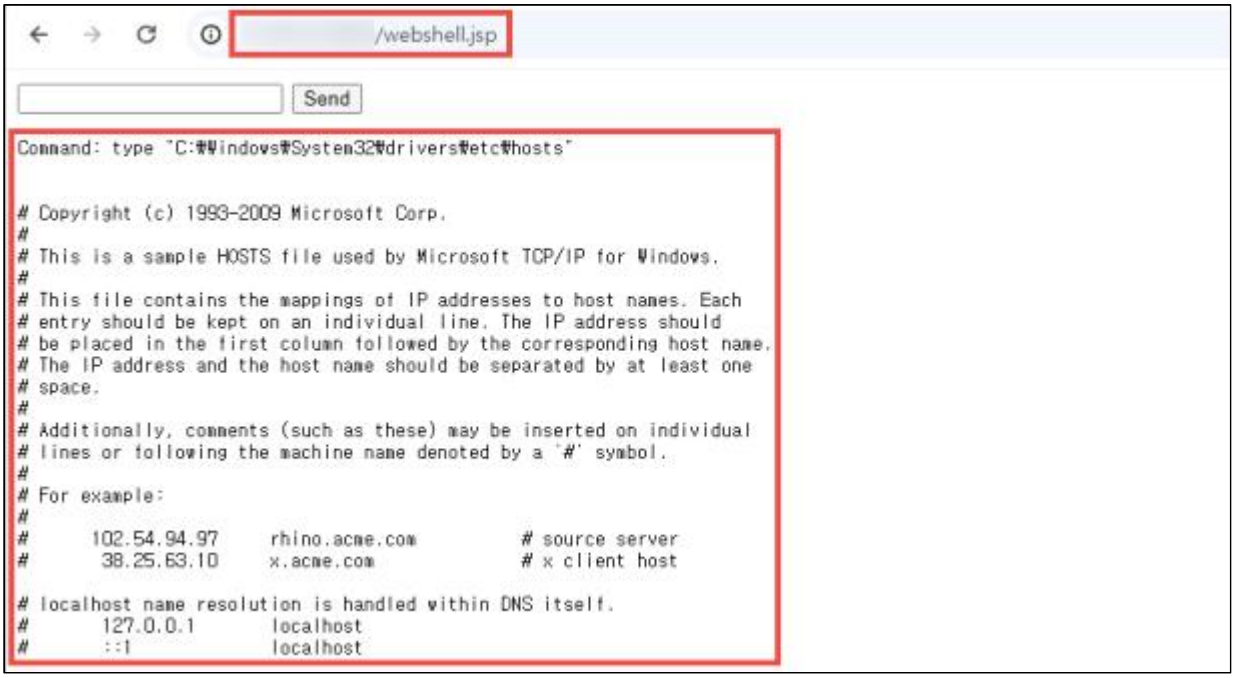
- 점검 방법

Step 1) PUT 메소드를 사용하여 악성 파일(Server Side Script, 악성 스크립트 등) 생성 시도



[프록시 도구를 이용한 임의 파일 생성 시도]

Step 2) 생성된 악성 파일을 이용하여 웹쉘(Webshell) 업로드, 크로스사이트 스크립팅 등의 공격 수행 가능



[생성 파일 실행 유무 확인]

Step 3) DELETE 메소드를 사용하여 서버에 저장된 임의의 데이터 삭제 가능

[프록시 도구를 이용한 임의 파일 삭제 시도]

- 조치 방법

1. PUT, DELETE 메소드를 이용해 파일 생성/삭제가 가능한 WebDAV(Web Distributed Authoring and Versioning) HTTP 확장 프로그램은 비활성화를 권장하며, 부득이하게 사용해야할 경우 망 분리 되어있는 내부망에서 적절한 인증 절차를 거쳐 운용
2. RESTful API 애플리케이션 상의 PUT, DELETE 메소드의 경우 기본적으로 취약하게 설계된 메소드가 아니며, POST 메소드를 통하여 구현하는 것과 동일하게 핵심 비즈니스 로직에 대한 시큐어 코딩을 구현
3. 불필요하게 활성화된 메소드는 비활성화하여 잠재적인 취약점의 최소화를 권장
   (웹 서버 설정 파일 구성은 리눅스 OS 내에서도 종류 및 버전에 따라 차이가 있으므로 개발 환경에 맞게 조치)

Apache

▪ 명령어를 통한 dav 및 dav_fs 모듈 비활성화(WebDAV)

WebDAV 기능 비활성화

```
sudo a2dismod dav
sudo a2dismod dav_fs
```

▪ apache2.conf 파일 수정을 통한 WebDAV 서비스 비활성화

PUT, DELETE 메소드 비활성화 (WebDAV 서비스 비활성화)

```
/etc/apache2/apache2.conf (예시 :Apache/2.4.52-ubuntu)
...
<Directory "/var/www/html">
        Dav On #On으로 활성화 되어있을 경우 Off 또는 지시어 삭제로 비활성 조치
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
    <LimitExcept GET POST PUT DELETE OPTIONS>
                #PUT, DELETE 메소드 사용이 허용되어 있을 경우,
                #PUT, DELETE 부분을 삭제하여 해당 메소드 사용을 금지
        Order Allow,Deny
        Deny from all
    </LimitExcept>
</Directory>
...
```

- Apache의 경우 TRACE 메소드가 기본적으로 활성화 되어있음
- 데비안 계열은 /etc/apache2/apache2.conf 파일 내 TraceEnable Off 지시어를 추가하는 것으로 비활성화 조치 (2.4.X 버전부터 /etc/apache2/conf-available/security.conf 파일에서 TRACE 사용을 별도로 관리하며, 기본적으로 비활성화 되어있음)

TRACE 메소드 비활성화

```
# /etc/apache2/apache2.conf (예시 :Apache/2.2.8-ubuntu)
...
TraceEnable Off # 파일 내용 하단에 추가
...
```

- 레드햇 계열의 경우 httpd.conf 파일 수정을 통해 TRACE 메소드 비활성화 조치

TRACE 메소드 비활성화

```
# /etc/httpd/conf/httpd.conf (예시: Apache/2.4.6-CentOS)
...
TraceEnable Off # 파일 내용 하단에 추가
...
```

- mod_rewrite 모듈을 이용한 CONNECT 메소드 비활성화

CONNECT 메소드 비활성화

```
# /etc/apache2/sites-available/000-default.conf (예시: Apache/2.4.52-ubuntu)
...
<VirtualHost *:80>
    ...
    <IfModule mod_rewrite.c>
        RewriteEngine On
        RewriteCond %{REQUEST_METHOD} ^CONNECT #CONNECT 비활성화
        RewriteRule .* - [F]
    </IfModule>
    ...
</VirtualHost>
```

Tomcat

▪ WebDAV 서비스 활성화 시 PUT, DELETE 메소드를 이용하여 임의의 파일 생성 및 삭제 가능

PUT, DELETE 메소드 비활성화 (WebDAV 서비스 비활성화)

```
// 아래 코드를 제거하여 WebDAV 서비스 비활성화 또는 readonly 옵션을 true로 설정하여 쓰기 권한 제거
<servlet>
   <servlet-name>webdav</servlet-name>
   <servlet-class>org.apache.catalina.servlets.WebdavServlet</servlet-class>
   <init-param>
      <param-name>debug</param-name>
      <param-value>1</param-value>
   </init-param>
   <init-param>
      <param-name>listings</param-name>
      <param-value>false</param-value>
   </init-param>
   <init-param>
      <param-name>readonly</param-name>
      <param-value>true</param-value> // true로 설정하여 쓰기 권한 제거
   </init-param>
   <load-on-startup>1</load-on-startup>
</servlet>
```

▪ server.xml 설정 파일 내 allowTrace="true" 설정 값을 제거하여 TRACE 메소드 비활성화

TRACE 메소드 비활성화

```
//
<Connector port="8080" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443"
           maxParameterCount="1000"
           allowTrace="true" // 설정 값 제거
           />
```

▪ Tomcat의 경우 설계 목적이 웹 애플리케이션 서버임에 따라, 프록시 서버에서 주로 사용되는 CONNECT 메소드를 기본적으로 지원하지 않음

Nginx

- WebDAV 서비스 활성화 시 PUT, DELETE 메소드를 이용하여 임의의 파일 생성 및 삭제 가능
- /etc/nginx/sites-available/default 설정 파일 수정(환경 : nginx/1.18)

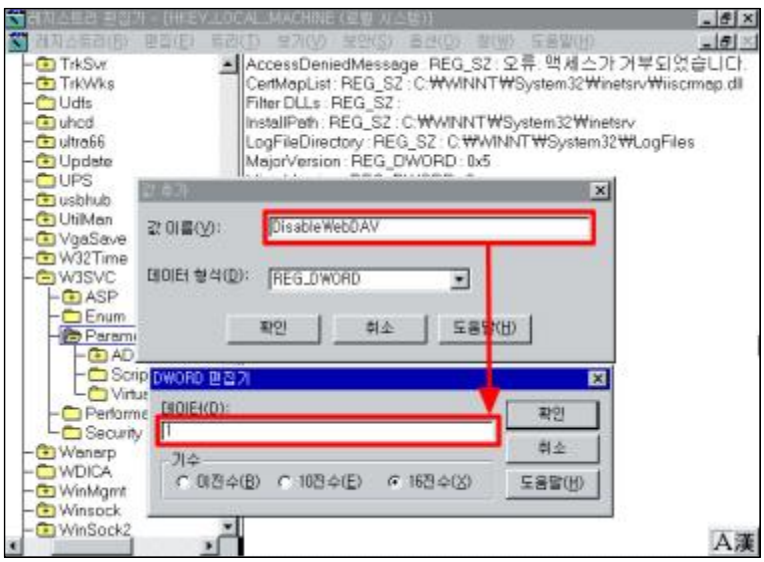
PUT, DELETE 메소드 비활성화 (WebDAV 서비스 비활성화)

```
...
location /dav/ {
...
dav_methods PUT DELETE MKCOL COPY MOVE; #dav_methods 지시어 부분 삭제
...
}
```

- Nginx는 보안상의 이유로 0.5.17버전 이후로 TRACE 메소드를 항상 405 응답 코드로 거부하도록 패치되었으며, 기본적으로 포워드 프록시 기능을 지원하지 않으므로 CONNECT 메소드를 허용하지 않음

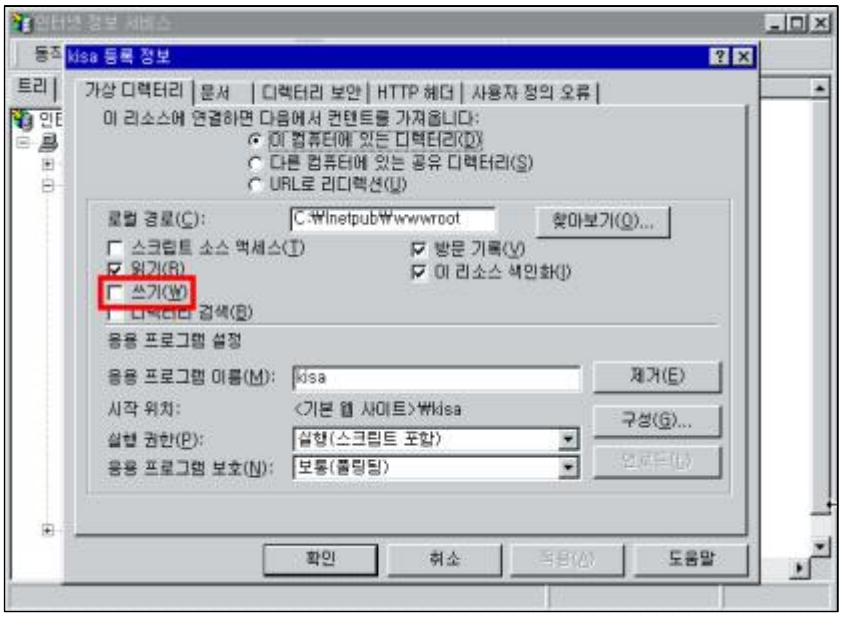
IIS 5.0 이하

- IIS 5.0의 경우 기본적으로 WebDAV 기능이 활성화 되어있어 PUT/DELETE 등의 메소드를 통하여 임의적인 파일 생성 삭제가 가능하므로 WebDAV 기능을 비활성화하거나 서비스 중인 애플리케이션의 디렉터리 내 '쓰기' 권한을 제거하여 PUT/DELETE를 통한 파일 생성 삭제 제한 가능
- 레지스트리 편집기(Regedt32.exe) → HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\W3SVC\Parameters → 편집 → 값 추가 → 값 이름: DisableWebDAV, 데이터형식: REG_DWORD, 데이터: 1 → IIS 재시작



[생성 파일 실행 유무 확인]

▪ 웹 애플리케이션 홈 디렉터리 또는 가상 디렉터리 내 쓰기 권한 제거



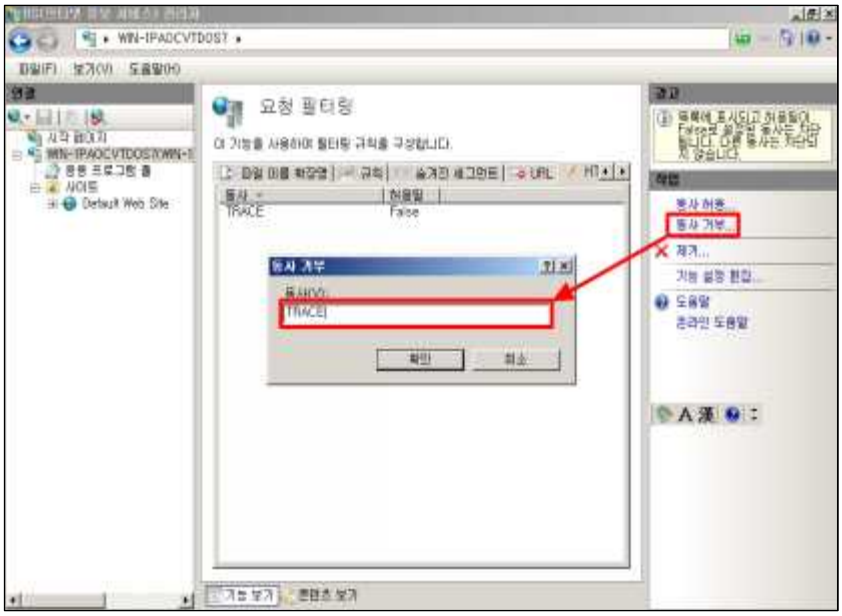
[디렉터리 쓰기 권한 제거]

IIS 6.0 이상

▪ IIS 6.0 이상의 버전의 경우 IIS 웹 서버 설치 시 WebDAV 기능이 비활성화 되어있으며, OPTIONS/TRACE/GET/HEAD/POST 메소드가 기본적으로 활성화되어있음

▪ TRACE 메소드 비활성화를 위하여 요청 필터링 → HTTP 동사 → 동사 거부 메뉴에 TRACE 메소드 추가

[IIS 관리자 요청 필터링 메뉴 접근]



[동사 거부 설정 내 TRACE 메소드 추가]