

# 침입탐지 기술의 패러다임 변화



- IPS의 공격 검출 패러다임 변화
- 예제로 살펴보는 문자열 시그니처의 한계와 PCRE 패턴의 강력함
- (특별기고) PCRE 탐색에서 종전 기술의 한계와  
문자열 시그니처 기반 탐색의 보안 취약성에 대한 이론적 고찰
- PCRE 검색 성능을 비약적으로 끌어 올린 혁신적 신기술의 소개
- PCRE 기능과 성능의 검증 방법론
- PCRE의 올바른 검증을 위한 상세 시험 절차서



INFNIS

(주)인프니스네트웍스







# 목차

- IPS의 공격 검출 패러다임 변화 1
- 예제로 살펴보는 문자열 시그니처의 한계와 PCRE 패턴의 강력함 9
- PCRE 탐색에서 종전 기술의 한계와 15  
문자열 시그니처 기반 탐색의 보안 취약성에 대한 이론적 고찰  
(워싱턴 주립대 김민식 교수 특별기고)
- PCRE 검색 성능을 비약적으로 끌어 올린 혁신적 신기술의 소개 21
- PCRE 기능과 성능의 검증 방법론 27
- PCRE의 올바른 검증을 위한 상세 시험 절차서 35



# IPS의 공격 검출 패러다임 변화

“유한개념의 문자열 시그너처, 무한개념의 PCRE 패턴”  
 “넓은 문자열 시그너처를 버리고 PCRE 패턴으로 패러다임을 전환해야”

(주)인프니스네트웍스 기술연구소 박동기 이사

## I. 네트워크 보안의 발전

### 네트워크 보안의 초기 기술 - 방화벽

방화벽(Firewall):  
 침입차단시스템, 내부  
 전산망을 보호하기 위  
 해 전송되는 패킷의  
 헤더를 검사하여 차단  
 여부를 결정하는 네트  
 워크 구성 요소

네트워크 보안은 방화벽 기술에서 시작되었다. 전통적인 방화벽은 패킷의 헤더(header)만 보고 네트워크 출입 여부를 결정한다. 즉, 패킷의 아이피 주소(IP address)와 서비스 포트(port)에 따라 차단하거나 통과시키는 것이다. 스팸(spam) 문자를 차단할 때 발신 번호를 지정해서 그 번호로부터 오는 모든 문자를 차단하는 것과 같은 방식이다.

### 네트워크 보안의 공격 검출 방법 확장

#### - 문자열 시그너처 기반 IPS (침입 탐지 및 방어)

IPS(Intrusion  
 Prevention System):  
 침입방지시스템, 방화  
 벽을 우회하는 공격을  
 패킷 내용 분석 등을  
 통해 판단하고 수신  
 거부나 세션 차단 등  
 으로 침해를 방지

방화벽을 우회하는 공격이 등장하면서 IPS 개념이 등장하였다. 방화벽이 패킷의 헤더 부분만 보는 것과 달리, IPS는 패킷의 데이터부, 즉 페이로드(payload)까지 확인한다. 이는 스팸 문자를 걸러내기 위해 “대출”, “캐피탈”, “대리운전” 등의 단어가 포함된 경우 그 문자를 스팸으로 간주하는 방식과 같다.

	1세대	2세대	3세대
종류	방화벽	문자열 시그너처 기반 IPS	?
공격 검출 비교 부분	패킷의 헤더	패킷의 페이로드 (데이터)	
공격 검출 비교 방법	아이피 주소, 포트 번호 식별	단순 문자열 비교	
스팸 문자 차단 방식과 유사성	발신 번호 지정 문자 차단	“대리운전”, “대출” 등 지정 단어 포함 하는 문자 차단	

표 1 : 네트워크 보안의 발전

## II. IPS의 한계와 진화

### 문자열 시그너처 기반 IPS의 한계

스팸을 문자의 내용으로 식별하는 것이 발신 번호만으로 식별하는 것보다 고도화된 기술인 것처럼, 방화벽 기술에서 IPS 기술로의 변화는 한 걸음 더 나아간 것이다. 그러나, 종래의 IPS는 공격을 식별하는 방식에서 근본적인 한계가 있었다. 그 한계는 문자열 시그너처(signature)로 공격을 판단하는 것에서 시작된다.

문자열 시그너처는 공격 패킷에서 반복적으로 사용되는 코드를 추출한 것으로서, IPS는 이 문자열 시그너처들을 별도의 목록에 등록해 놓고, 같은 문자열 시그너처를 포함한 패킷이 들어오면 공격으로 간주하여 차단한다. 이는 대리운전 스팸 문자를 막기 위해 “대리운전”이라는 문자를 스팸 문구 목록에 등록하는 것과 같다.

한 개의 문자열 시그너처가 표현할 수 있는 공격 패킷의 유형은 제한적

문제는 한 개의 문자열 시그너처가 표현할 수 있는 공격 패킷의 유형이 한정적이라는 것이다. 공격에 사용된 코드가 조금만 바뀌어도 공격으로 판정하지 못한다. “대♡리운전”처럼 변형된 형태를 전혀 다르다고 인식하는 것이다. 코드, 즉 문자열에 다양한 옵션을 추가하는 변형된 문자열 시그너처가 근래 많이 사용되고 있지만, 이 역시 한 문자열 시그너처와의 비교로 판정할 수 있는 공격 패킷의 변종 형태는 그 개수가 매우 제한적이다.

문자열 시그너처 기반 IPS는 변종 공격 방어의 경우 제약이 큼

문자열 시그너처를 이용하여 공격을 탐지하는 방식으로는 더 이상 네트워크 공격을 제대로 방어할 수 없다. 문자열 시그너처 기반 IPS 제품들은 보통 2-3천 개 정도의 문자열 시그너처를 활용하는데, 이 정도로는 수많은 변종 공격을 효과적으로 방어하기에 턱없이 부족하다. 악의적인 공격자, 즉 해커들은 이를 노리고 패킷의 형태를 계속 바꾸어 공격하므로, 기존의 문자열 시그너처 기반의 IPS는 해커의 다양한 공격 위협에 별다른 대책없이 노출되어 있다고 할 수 있다. 하나의 변종 공격에 대응 가능한 새로운 문자열 시그너처를 추가할 수는 있으나, 추가한 시그너처 역시 또 다른 변종 공격을 막을 수는 없으므로, 아무리 이를 계속 만든다고 해도 단순 문자열 비교 방식인 문자열 시그너처 기반 IPS는 그 한계가 너무나 자명하다.

정규표현식(regular expression):  
특정한 규칙을 가진 문자열의 집합을 표현하는 전산 이론의 언어

PCRE(Perl Compatible Regular Expressions):  
펄 호환 정규표현식

다양한 문자열 패턴은 정규표현식을 이용하여 표현 가능

PCRE 패턴을 사용하면 문법 구조로 변종 패킷을 쉽게 표현

## PCRE란? - 고정된 문자열 대신 문자열의 패턴을 표현

정규표현식은 전산 이론에서 깊이 있는 연구가 확립된 문자열 표현의 표준적 방식으로, 가장 잘 알려진 버전(version)은 PCRE이다. PCRE는 프로그래밍 언어인 펄(Perl)의 문법과 호환되는 형태이다. 펄이 문자열 연산을 위한 언어로 여러 분야에서 사용되고 있고, 공개 소스(open source) 형태로 PCRE 라이브러리(library)를 제공하므로, 정규표현식의 방식 중에서 PCRE가 가장 많이 쓰인다.

“대♡리운전”을 막기 위해서 “대리운전”과 “대♡리운전”의 두 개 문자열을 스팸 문구 목록에 등록해도 “대★리운전”을 차단할 수 없다. 이렇듯 “대”와 “리” 사이에 들어올 수 있는 코드나 문자는 무한하기 때문에 문자열을 개별적으로 나열할 수는 없다. 그러나, 전산 이론을 활용하면 PCRE와 같은 정규표현식을 이용하여 다양한 문자열을 쉽게 표현할 수 있다. “대”와 “리” 사이에 어떤 문자든 다 포괄하고 싶다면 “대.\*리운전”이라고 표현한다. 또 다른 예로, “대”와 “리” 사이에 한 자리의 영문 소문자를 모두 표현하는 PCRE 표현은 “대[a-z]리운전”이다.

## 문자열 시그니처의 표현력의 한계

문자열 시그니처는 고정된 문자열을 기반으로 한다. 추가적인 옵션 처리를 활용해도 문자열의 고정된 형태를 크게 벗어날 수 없다. 문자열 시그니처를 이용하여 공격을 차단하고자 한다면 유한 개의 공격만 방어가 가능하다. 반면에, PCRE 패턴, 즉 정규표현식을 사용하면 이론적으로 무한의 변종 패킷을 방어할 수 있다. “대.\*리운전”과 같은 PCRE 정규표현식은 “대1리운전”, “대11리운전”, “대김수한무거북이와두루미리운전” 등 무한 개의 문자열을 나타낸다.

언어와 비교하자면 문자열 시그니처는 단어 개념이고, PCRE 패턴은 문법을 갖춘 언어 개념이다. 공격 유형을 가장 잘 표현할 수 있는 것은 사람의 언어이다. 사람의 언어로 표현이 불가능한 공격 유형은 존재하지 않는다. 따라서, 공격을 표현하는 방식도 가능한 사람의 언어와 가까워야 한다. PCRE 패턴은 컴퓨터 언어의 일종이므로 문자열 시그니처에 비해 사람의 언어와 비교할 때 훨씬 가깝다.

문자열 시그니처는 IPS 제품마다 방식이 달라 이기종 간 일괄 업데이트 불가능

### 문자열 시그니처의 호환성 문제

문자열 시그니처는 공개 소스 기반 IPS인 스노트(Snort)가 정한 방식이 널리 사용되긴 하지만 IPS 제품 개발 회사마다 조금씩 달리 사용하고, 회사들은 그 공개를 꺼려한다. 문자열 시그니처의 형태가 공개되면 해당 회사의 IPS 제품이 공격 당하기 쉽기 때문이다.

네트워크 공격자들은 끊임없이 새로운 공격을 개발하고 실제로 사용한다. 이를 방어하기 위해서는 신속히 차단 목록을 만들어 업데이트해야 하지만, 문자열 시그니처는 이런 일련의 과정이 용이하지 않다.

우선, 문자열 시그니처를 만드는 작업이 만만치 않다. 이는 마치 긴 문장을 몇 개의 단어로 요약하는 것과 같아서 공격을 완벽히 분석한 후에 상당한 작업 시간을 거쳐야 한다. 완벽한 분석 과정 없이 짧은 시간 동안 만들어진 문자열 시그니처는 변종 공격에 취약하기 마련이므로, (근본적으로는 제대로 대응할 수는 없지만) 일정 수준에서라도 변종 공격에 대응하기 위해서는 만들기까지 많은 시간이 소모된다.

더불어서 문자열 시그니처는 표준이 정립되어 있지 않다. 만일 여러 회사로부터 각기 다른 IPS 제품을 도입하여 사용한다면, 새로운 공격을 막기 위해 각 공급 회사로부터 신규 문자열 시그니처를 따로따로 제공 받아야 한다. 때문에, 차단 목록을 제때에 지속적으로 업데이트하는 것이 문자열 시그니처 기반의 IPS 장비에서는 거의 불가능하다.

	문자열 시그니처	PCRE 패턴
공격 패킷 표현 범위	유한	무한
공격 패킷 표현 수준	단어	문법을 갖춘 언어
업데이트 관리	표준 없어 어려움	편리함
차단 목록 생성	IPS 제품을 개발한 회사에서만 지원가능	PCRE 패턴 문법을 이해하면 누구나 가능

표 2 : 문자열 시그니처 vs. PCRE 패턴

제3의 전문 기관이나 국가 기관으로부터 공격 목록을 PCRE 패턴 형태로 제공받을 수 있음

### PCRE 패턴의 장점 - 호환성으로 효율적 업데이트 관리

근래 보안 사고가 빈번해지자 새로운 공격을 분석하여 정보를 제공하는 회사나 기관이 많아졌다. 정부에서도 국가 기간망의 보호를 위해 공격을 추적하고 정보를 수집하는 일을 한다. IPS 개발 회사들은 모든 공격을 홀로 추적할 수는 없으므로, 전문 정보 제공 회사나 국가 기관으로부터 공격 정보를 공급 받는 경우가 늘고 있다.

문자열 시그니처 기반 IPS의 경우에는 제공받은 공격 정보를 IPS 개발사가 자신들의 형식에 맞도록 수정해야 하지만, PCRE 패턴이라면 공통된 문법이므로 제공받은 정보를 그대로 활용할 수 있다. PCRE 패턴을 사용하면 IPS 공급 회사에만 의존하지 않고 제3의 전문 기관이나 국가 기관으로부터 공격 목록을 PCRE 패턴 형태로 제공받을 수도 있다.

공격 차단 목록의 공유와 업데이트 효율성을 담보하는 유일한 방식은 PCRE 패턴 방식

PCRE 패턴을 사용하면 지속적인 공격 목록의 업데이트가 용이하다. 전산 이론의 중요 개념인 정규표현식의 일종이고 가장 널리 사용되므로 호환성에 문제가 없다. 도입하는 IPS 제품들이 모두 PCRE 패턴을 제대로 지원한다면, 각 제품의 개발사가 다르더라도 별도의 변환 작업 없이 업데이트를 빠르고 정확하게 모든 장비에 적용할 수 있다.

### 그동안 왜 문자열 시그니처를 사용했나?

전산이론에서 정규표현식이나 PCRE는 새로운 것이 아니다. 정규표현식은 수십 년 전부터 전산 이론의 중요한 개념이었다. 그런데, 왜 IPS가 등장할 당시부터 PCRE 패턴을 사용하지 않았을까? 사실, 하나의 공격을 분석해서 그 규칙을 알아낸 후에 이를 가장 쉽게 표현하는 것은 다름아닌 정규표현식이다. 이미 많은 공격 유형이 PCRE 패턴으로 표현이 되어 있다.

기존 IPS 제품들은 PCRE 연산 과정에서 심각한 성능 저하

엄밀하게 이야기하면 기존의 IPS 제품들이 PCRE 패턴을 사용하지 못해서 어쩔 수 없이 문자열 시그니처를 사용한 것이다. PCRE 패턴 검색을 위한 연산 과정에서 발생하는 부하로 인해 실제 네트워크에서 요구하는 성능을 감당할 수 없었기 때문이다. 그동안 IPS 개발 회사들은 이론과 활용의 모든 측면에서 너무도 당연한 정규표현식 기반의 검색을 적용하지 못하고, 문자열 시그니처와 같은 편법에 많은 시간과 노력을 쏟았다.

### III. IPS 패러다임의 전환

#### PCRE 패턴 기반의 새로운 패러다임

PCRE 패턴은 하나의 표현으로 수없이 많은 변종 공격을 나타낸다. 공격은 기본적인 규칙이 존재하기 마련이므로, 그 규칙을 잘 이해한다면 몇 개의 PCRE 패턴으로 같은 종류의 변종 공격을 모두 차단할 수 있다.

PCRE 패턴은 콘텐츠 차단 등 광범위하게 적용 가능

UTM (Unified Threat Management): VPN, IPS, 유해사이트 차단, 스팸 차단 등의 다양한 보안 기능을 통합한 제품

공격을 차단하는 목적뿐만 아니라 유해사이트 차단 등의 콘텐츠 필터에도 PCRE 패턴을 사용할 수 있다. “야한동영상”이라는 문구가 들어간 웹 페이지를 차단하고 싶다면 “야.{,10}한.\*(동)?영상”과 같은 PCRE 패턴을 유해사이트 차단 문구에 입력하면 된다. 스팸 메일 필터에도 동일한 방법을 사용할 수 있다. 그러나, IPS와 마찬가지로 기존의 네트워크 보안 제품들은 콘텐츠 필터, 스팸 메일 필터 등의 기능에도 문자열 시그너처 방식을 적용하는 수준에 머물러 있다. 이제 UTM 제품은 문자열 시그너처 검색 대신 PCRE 패턴 검색 방식을 전면적으로 도입해야 한다.

새로운 IPS는 낡은 문자열 시그너처 방식을 버리고 “PCRE 패턴” 방식으로 전환해야 함

PCRE 패턴으로 네트워크 공격을 막거나 원치 않는 트래픽을 차단하는 것은 새로운 이론이 아니다. 다만, 지금까지는 구현 기술의 한계로 단순 문자열이나 문자열 시그너처 방식을 사용하였고, 이는 편법에 불과하다. 이제 제대로 PCRE 패턴을 활용해야 한다. 낡은 문자열 시그너처를 버리고 PCRE 패턴으로 보안 제품의 패러다임을 전환할 때이다.

	1세대	2세대	3세대
종류	방화벽	문자열 시그니처 기반 IPS	PCRE 패턴 기반 IPS
공격 검출 비교 부분	패킷의 헤더	패킷의 페이로드 (데이터)	패킷의 페이로드 (데이터)
공격 검출 비교 방법	아이피 주소, 포트 번호 식별	단순 문자열 비교	문자열의 패턴(구조) 비교
스팸 문자 차단 방식과 유사성	발신 번호 지정 문자 차단	"대리운전", "대출" 등 지정 단어 포함하는 문자 차단	"대.*리운전"과 같은 표현으로 "대♡리운전" 등 변종도 함께 차단

표 3 : 네트워크 보안의 발전 - PCRE 패턴 기반 IPS



# 예제로 살펴보는 문자열 시그니처의 한계와 PCRE 패턴의 강력함

(주)인프니스네트웍스 기술연구소 이준호 팀장

문자열 시그니처는 단순 문자열 비교 방법으로 공격 여부를 판정하므로 공격의 구조를 문법적으로 표현하는 PCRE 패턴과 비교해서 심각한 단점을 갖고 있다. 여기서는 예를 들어 설명하기로 하고, 개념적으로는 앞의 “IPS의 공격 검출 패러다임 변화”에서, 이론적으로는 뒤의 “PCRE 탐색에서 종전 기술의 한계와 문자열 시그니처 기반 탐색의 보안 취약성에 대한 이론적 고찰”에서 상세히 기술하므로 해당 내용을 참고하면 되겠다.

## 1. 문자열 시그니처는 변종 공격에 대응하기 어려움 - PCRE 패턴으로 변종 공격도 막아야 한다.

공개 소스(open source)로 널리 알려진 IPS인 스노트(Snort)의 문자열 시그니처 중에 “cross-site scripting”이라는 공격을 막는 다음 두 개를 보자.

문자열 시그니처 1

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-MISC cross site scripting attempt";
flow:to_server,established; content:"<SCRIPT"; nocase;
classtype:web-application-attack; sid:1497; rev:6;)
```

문자열 시그니처 2

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"WEB-MISC cross site scripting HTML Image tag set to
javascript attempt"; flow:to_server,established;
content:"img src=javascript"; nocase;
reference:bugtraq,4858; reference:cve,2002-0902;
classtype:web-application-attack; sid:1667; rev:7;)
```

위의 문자열 시그니처 구문에서 빨간 색으로 표기된 부분이 공격 패킷에 포함된 문자열을 지정하는 것인데, 위의 시그니처는 공격 패킷의 데이터부, 즉 페이로드(payload)에 <SCRIPT가 들어가는 경우를, 아래의 것은 img src=javascript가 들어가는 경우를 의미한다.

두 문자열 시그니처에서 alert로 시작하는 앞부분과 nocase부터의 뒷부분은 공격 패킷의 다른 정보를 지정한 것으로 여기서는 설명을 생략한다.

문제는 <SCRIPT와 img src=javascript가 포함되었다고 해서 모두 공격이 아닐 뿐만 아니라, cross-site scripting은 이 두 문자열 외의 다른 문자열을 포함하는 경우도 많다는 점이다. 따라서, 이와 같은 방식으로는 오탐과 미탐의 가능성이 모두 존재하게 된다. Cross-site scripting을 PCRE 패턴으로 표현하면 오탐의 가능성을 줄이면서 미탐을 방지할 수 있는데, 그 PCRE 표현은 다음과 같다.

```
/(\x22|\x27|`)??(java|vb)?script>?/i
```

Cross-site scripting에 대한 보다 정확한 PCRE 패턴은 조금 더 복잡하나 여기서는 이해의 편리를 위해 단순화했음을 밝혀둔다. \x22와 \x27은 각각 작은 따옴표와 큰 따옴표를 나타내는 hex 값이고, (java|vb)는 java 또는 vb 중에 한 가지를 의미하고, ?는 바로 앞의 내용이 존재하거나 또는 그렇지 않거나 상관없음을 의미하며, 끝의 /i는 대소문자를 구별하지 않는다는 뜻이다. 따라서, 이 PCRE 패턴이 포함하는 문자열은 javascript, vbscript, ‘<javascript, “<vbscript> 등 대소문자 구별이 없는 점을 빼고 고려해도 48가지가 된다. 앞서 스노트의 문자열 시그니처가 하나의 시그니처 당 하나의 문자열에 대응되는 점과 비교하여 엄청난 표현력의 차이를 확인할 수 있다.

실제 공격 패킷의 검출 여부로 문자열 시그니처와 PCRE 패턴의 차이를 확인해 보자. 다음은 위 스노트의 문자열 시그니처가 막고자 하는 공격의 범주 내에서 다양한 변종 공격 코드에 대한 탐지 여부를 정리한 것이다.

문자 변종 공격이 포함하는 코드	탐지 결과		
	문자열 시그니처1	문자열 시그니처2	PCRE 패턴
<SCRIPT>alert!("XSS")</SCRIPT>"> (<SCRIPT>와 </SCRIPT> 태그 사이에 공격 코드)	O	X	O
<SCRIPT/XSS SRC="http://xxx/xss.js"> </SCRIPT> (<SCRIPT 태그 내부에서 공격 코드 삽입)	O	X	O
<IMG SRC=javascript:alert!("XSS")> (<IMG 태그에 파일 대신 공격 코드 삽입, 이하 같음)	X	O	O
<IMG SRC=JaVaScRiPt:alert!("XSS")>	X	O	O
<IMG SRC='javascript:alert!("RSnake says, 'XSS'")>	X	X	O
<IMG SRC='vbscript!:msgbox("XSS")>	X	X	O

표 4 : 문자열 시그니처가 문자 변종 공격을 탐지 못하는 예

두 문자열 시그니처가 마지막 두 개의 공격 패킷을 모두 탐지하지 못하는 것을 확인할 수 있다. PCRE 패턴으로는 당연히 탐지가 된다.

Cross-site scripting은 스노트의 위 두 개의 문자열 시그니처에서 규정하는 형태 외에도 다양한 형태의 공격 유형이 가능하다. 다음은 두 개의 문자열 시그니처가 규정하는 형태가 아닌 다른 종류의 cross-site scripting에 대한 탐지 여부이다

구조 변종 공격이 포함하는 코드	탐지 결과		
	문자열 시그니처 1	문자열 시그니처2	PCRE 패턴
<BASE HREF="javascript:alert('XSS');//">	X	X	O
<BGSOUND SRC="javascript:alert('XSS');">	X	X	O
<DIV STYLE="background-image: url(javascript:alert('XSS'))">	X	X	O
<FRAMESET><FRAME SRC="javascript:alert('XSS');"></FRAMESET>	X	X	O
<INPUT TYPE="IMAGE" SRC="javascript:alert('XSS');">	X	X	O
<IMG DYNSRC="javascript:alert('XSS');">	X	X	O
<IMG LOWSRC="javascript:alert('XSS');">	X	X	O
<IMG SRC='vbscript:msgbox("XSS")'>	X	X	O
<META HTTP-EQUIV="refresh" CONTENT="0:url=javascript:alert('XSS');">	X	X	O
</TITLE><SCRIPT>alert("XSS");</SCRIPT>	X	X	O
<TABLE BACKGROUND="javascript:alert('XSS')"></TABLE>	X	X	O

표 5 : 문자열 시그니처가 구조 변종 공격을 탐지 못하는 예

스노트의 문자열 시그니처는 단순 문자열 비교이므로 문자열의 구조를 반영하지 못한다. 따라서, 문자 자체의 변경은 물론이고 구조가 달라지는 공격에 대해서는 새로운 시그니처를 만들지 않는 이상 탐지가 불가능하다. 그러나, PCRE 패턴은 문자의 다양한 변화와 더불어 그 구조를 문법적으로 표현하므로 해당되는 여러 변종 공격을 손쉽게 표현하고 탐지가 가능하다.

앞에서 본 cross-site scripting 공격처럼 포함되는 문자열도 다양하고 구조의 특징이 변화하는 경우라면 문자열 시그니처 방식은 오탐과 미탐의 확률이 매우 높다.

## 2. 문자열 시그니처로 아예 표현이 불가능한 공격은 다양함 - PCRE 패턴으로 쉽게 표현 가능

공격 유형에 따라서 아예 문자열 시그니처로 표현이 불가능한 경우가 있다. 근래는 새로운 공격 형태가 개발되어서 문자열 시그니처로 표현이 불가능한 빈도가 높아지고 있는 추세이다.

### 2.1 반복 또는 무한의 개념이 필요한 경우

```
/(xmlns\x3A.*?){15}/smi
```

위의 PCRE 패턴은 프로토콜 위반 공격의 한 종류를 표현하는 것으로서 xmlns:으로 시작되는 문자열이 15개 이상 포함되는 경우를 의미한다. 14개까지는 정상이나 15개 이상 포함되면 DoS 공격으로 간주하는 것이다. 이런 식의 공격은 문자열 시그니처로 표현이 불가능하다. 15개부터 무한 개까지 문자열 시그니처를 만들 수는 없기 때문이다. 그러나 PCRE 패턴은 반복의 개념을 표현한다. PCRE를 포함하는 모든 정규표현식은 수학적 언어 모델이므로, 정규표현식에서 반복의 개념은 매우 중요한 요소로 다루어진다.

### 2.2 고정된 문자열이 없거나 있어도 의미가 없는 경우

```
/^sin\d+\x3A[^\r\n]*\x3A\d+\x3A\d+\x3A/smi
```

위의 PCRE 패턴은 backdoor 공격 중 하나를 표현한 것이다. Hex 코드가 있어 해독이 좀 불편하긴 하나, 상세한 문법 적용은 생략하고 대강 손쉽게 변환하면 다음과 같다.

```
sin 임의의수열 : 엔터 : 임의의수열 : 임의의수열 :
```

이 패턴에서는 문자열이 sin으로 한 개 드러난다. 이 공격 유형을 문자열 시그니처로 식별하려면 사용할 수 있는 문자열은 오로지 sin 하나인데, sin은 정상적인 트래픽에 얼마든지 포함될 수 있으므로 식별의 문자열로 사용하기 곤란하다. 스노트와 같은 많은 IPS 제품들이 활용하는 방식인 1단계 문자열 검색 후 2단계 PCRE 패턴 검색의 편법을 활용한다고 해도, sin으로 검출되는 트래픽이 매우 많기 때문에 PCRE 패턴 검색의 부하를 줄이는 효과는 미미하다. 아예 PCRE 패턴으로 처음부터 검색하고도 성능이 문제가 되지 않는다면 두 단계로 나누어 문자열 검색을 할 필요가 없는 것이다.

## 2.3 문법 구조를 표현해야 하는 경우

스노트의 문자열 시그니처 중에 다음은 backdoor 공격 중 한 형태를 표현하는 것이다.

```
content:"M1 ";
content:"C1 ";
```

그런데 실제로 이 backdoor 공격은 C2와 같이 1 외의 다른 숫자가 오기도 하고, 뒤에는 IP 주소가 이어 나오는 공격이다. content:"M1과 같은 표기가 혼란 것은 아니나 오탐을 방지하기 위해서는 뒤에 IP 주소가 오는지를 확인해야 한다. 이를 고려하여 PCRE 패턴으로 정확히 공격을 표현해 보면 다음과 같다.

```
/(C|M)\d\s\d+\x2E\d+\x2E\d+\x2E\d+/smi
```

여기서 (C|M)\d는 C나 M 뒤에 어떤 숫자가 와도 해당된다는 의미이고, 뒤의 내용은 IP 주소를 의미한다. IP 주소가 x.x.x.x의 구조로 .과 네 개의 숫자로 이루어져 있는 것을 PCRE 문법으로 표현한 것이다.

이와 같이 복잡한 구조의 공격 유형은 문자열 시그니처로 표현할 경우 표현 자체가 어려울 뿐만 아니라 오탐과 미탐의 가능성을 높이게 되므로, PCRE 패턴으로 표현하는 것이 바람직하다.



(특별기고)

# PCRE 탐색에서 종전 기술의 한계와 문자열 시그너처 기반 탐색의 보안 취약성에 대한 이론적 고찰

“기존 IPS 기술의 한계”

워싱턴 주립대 전기컴퓨터공학부 김민식 교수

유한 오토마타(finite automata): 컴퓨터의 기본적인 동작 방식을 설명하는 수학 모델

정규표현식을 이용한 문자열 비교는 유한 오토마타 이론으로 정립되어 있음

## 정규표현식 또는 PCRE 탐색의 이론 - 유한 오토마타

정규표현식(regular expression)은 단순히 문자열의 패턴을 나타내는 기호가 아니라, 이론적으로 증명된 계산 모델에 기반을 둔 언어이다. 계산 이론에서는 언어를 그 표현력에 따라 몇 단계로 구분하는데, 그 중에서 비교적 간단하고 배우기 쉬우면서도 다양한 패턴을 기술할 수 있는 언어가 정규표현식이다. 이러한 배경 덕분에 정규표현식을 이용한 문자열 비교 방법 또한 이론적으로 정립되어 있으며, 유한 오토마타(finite automata)라 불리는 방식으로 구현할 수 있다.

유한 오토마타는 연속된 문자열을 읽어들이며 처리하는 장치로서 매 순간 하나의 상태를 가지며, 현재 상태와 입력 문자에 따라 다른 상태로 이동할 수 있다. 입력의 길이에는 제한이 없지만, 가질 수 있는 상태의 수는 유한하기 때문에 “유한” 오토마타라고 불린다. 문자열을 모두 읽어들이는 결과, 유한 오토마타의 상태가 미리 정해 놓은 “최종 상태” 중의 하나가 되면 해당 문자열은 이 오토마타에 의해 정의되는 언어에 속하는 것이다. 정규표현식을 구현해 놓은 오토마타라면, 해당 문자열이 주어진 정규표현식의 패턴과 일치하는 경우에 해당한다.

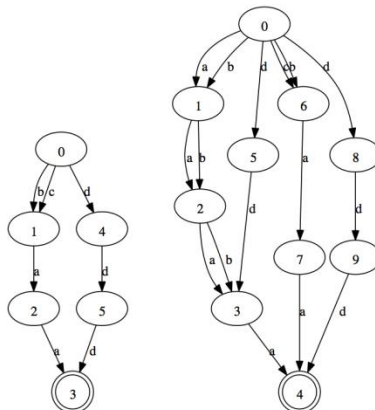


그림 1 : 유한 오토마타의 예

## 결정적 유한 오토마타 이론과 IPS 적용 시 한계

### - 메모리 사용량의 기하급수적 증가

결정적 유한 오토마타 (DFA):

현재 상태와 입력값에 따라 다음 상태가 유일하게 결정되는 유한 오토마타

유한 오토마타 중에서 현재 상태와 입력 문자에 따라 이동할 수 있는 다음 상태가 언제나 한 가지뿐인 오토마타를 결정적 유한 오토마타(DFA, deterministic finite automata)라고 한다. 기호로 표시하면  $M = (K, \Sigma, \delta, s, F)$ 와 같이 정의할 수 있으며,  $K$ 는 유한 오토마타가 가질 수 있는 모든 상태의 (유한) 집합,  $\Sigma$ 는 입력 문자열에 쓰일 수 있는 문자들의 집합,  $s$ 는 초기 상태( $s \in K$ ),  $F$ 는 최종 상태( $F \subseteq K$ ),  $\delta$ 는 상태 천이 함수, 즉 현재 상태와 입력 문자가 주어지면 다음 상태를 알려주는 함수이다. 천이 함수  $\delta$ 를 현재 상태와 입력 문자 조합을 한 축으로, 다음 상태를 또 다른 한 축으로 하는 이차원 배열 형태로 만들면, 입력 한 글자당 배열 참조를 한 번씩 하는 방식으로 DFA를 구현할 수 있다.

DFA로 정규표현식(또는 PCRE) 비교는 메모리 사용에서 한계가 있음

이처럼 DFA는 구현이 용이하고 실행 방법이 단순하여, 간단한 정규표현식 패턴을 비교할 때 널리 사용된다. 그러나 침입 탐지 목적으로 사용하기에는 결정적인 문제점을 안고 있다. 정규표현식이 복잡해지고 수가 늘어남에 따라 천이 함수를 저장하기 위한 배열의 크기가 기하급수적으로 증가하기 때문이다. 침입 탐지 시스템에서 널리 쓰이는 패턴들만 포함하더라도 DFA 구현 시 요구되는 메모리의 양은 중대형 장비들의 메모리 크기를 훨씬 초과한다. 그렇기 때문에 어쩔 수 없이 DFA보다 복잡한 비결정적 유한 오토마타(NFA, nondeterministic finite automata)로 정규표현식을 구현하는 것이 일반적이다.

## 비결정적 유한 오토마타 이론과 IPS 적용 시 한계

### - 활성 상태 수의 기하급수적 증가

비결정적 유한 오토마타 (NFA):

현재 상태와 입력값이 주어졌을 때, 복수 개의 다음 상태를 허용하는 유한 오토마타

NFA는 현존하는 많은 정규표현식 처리 라이브러리들이 택하고 있는 구현 방식이다. 그 중에서도 PCRE는 많은 엔지니어들에게 익숙한 펄 스크립트 언어와 호환되는 방식의 정규표현식 확장 문법을 제공하고 있어서, 널리 쓰이는 공개 소스 침입 탐지 시스템인 스노트(Snort)에 채택되어 사용되고 있다.

NFA는 기호로  $M = (K, \Sigma, \Delta, s, F)$ 와 같이 정의하는데, DFA와의 차이는 천이 함수  $\Delta$ 가 하나 이상의 상태를 돌려줄 수 있다는 점이다. 즉, 현재 상태와 입력 문자가 주어졌을 때, 다음 상태가 하나로 주어지지 않고, 복수 개의 상태 중 하나로 임의로 이동할 수 있다. 물론 다음 상태가 하나도 없는

NFA의 성능은 활성 상태의 수에 반비례

경우도 가능하며, 이런 유연성 때문에 DFA에 비해 적은 양의 메모리로도 구현이 가능하다. 일례로, NFA의 상태의 수가  $|K|$ 일 때, DFA의 상태는 최대  $O(2^{|K|})$ 까지 늘어날 수 있다. 문제는, NFA가 여러 개의 다음 상태 중 임의로 하나를 택한다고는 하나, 정규표현식의 매치는 여러 상태 중 하나라도 최종 상태에 도달할 수 있으면 매치가 되는 것으로 간주하기 때문에, 실제 구현에서는 결국 모든 경우의 수를 다 고려해야 한다는 점이다. 즉, 언제나 하나의 상태만을 고려하면 되는 DFA와는 달리 NFA에서는 가능한 모든 상태를 고려해야만 하는 것이다. 이렇게 NFA에서 가능한 모든 상태들을 활성 상태라고 하며, NFA의 실행 성능은 활성 상태의 수에 반비례한다.

NFA로 PCRE 비교 시 활성 상태 수 증가로 성능 저하

PCRE 라이브러리를 비롯한 대다수의 NFA 기반 정규표현식 처리 라이브러리에서는 복수의 활성 상태를 다루기 위해 트리 검색의 깊이 우선 탐색에 해당하는 방법을 제공한다. 주어진 정규표현식에 해당하는 NFA를 만들고, 입력 문자열을 한 글자씩 읽어가면서, 활성 상태가 증가하면 모든 활성 상태를 스택에 저장해 놓고 하나씩만 꺼내어 탐색하는 방식이다. 이 방법을 따르면, 입력 문자열의 끝까지 모두 읽어들이고 나서도 스택에 다른 활성 상태가 남아 있으면 다시 그 상태와 그에 해당하는 문자열 위치로 돌아가서 남아 있는 활성 상태를 탐색하게 된다. 결국 늘어나는 활성 상태 수만큼 입력 문자열을 반복해서 처리하게 되는 것이다. 이는 침입 탐지 시스템에서 같은 패킷을 여러 번 검사하는 셈이기에, 심각한 성능 저하로 이어질 수 밖에 없다.

기존 IPS는 성능 저하로 PCRE 기반 검색을 최대한 기피하고 문자열 시그너처 사용 - 궁여지책에 지나지 않음

### 기존 IPS의 PCRE 검색 회피와 이에 따른 보안 취약성 - 문자열 시그너처 방식의 근본적 결함

NFA 기반 정규표현식 검색의 이런 성능상의 문제를 잘 알고 있기에, 스노트에서는 가급적이면 정규표현식 검색을 피하고 단순 문자열 검색으로 최대한 대체하는 문자열 시그너처 방식을 사용하고 있다. 그러나 문자열 시그너처 방식은 문제에 대한 근본적인 해결책이 아니라 궁여지책으로 나온 만큼, 근본적으로 많은 결함을 내포하고 있다.

### 문자열 시그너처 방식의 결함 - 문자열 비교 전처리의 한계

스노트에서 사용하는 문자열 시그너처 비교는 정규표현식을 비교하기에 앞서, 그보다 빨리 처리할 수 있는 단순 문자

정규표현식 비교의 편법 방식인 단순 문자열 비교 전처리는 효과 없음

열 검색을 먼저 하고, 다음 단계로 선별적인 정규표현식 검색을 행하는 방식이다. 예를 들어, `Cookie\s+Monster`와 같은 정규표현식이라면(`\s+`는 공백 문자가 하나 이상 있음을 나타냄), 먼저 `Cookie`와 `Monster`로 단순 문자열 검색을 한 다음, 두 문자열 모두가 발견되는 패킷에 대해서만 `Cookie\s+Monster`로 정규표현식 검색을 행하는 것이다. 두 문자열 모두를 포함하는 패킷이 드문 경우에는 정규표현식 검색 횟수를 효과적으로 줄일 수 있다.

그러나 대부분의 침입 탐지 패턴에는 이런 방식이 효과를 거두기 어렵다. 예를 들면, 웹 트래픽에서 특정 클라이언트를 골라내기 위한 `^User-Agent\x3a[^\r\n]*`로 시작하는 정규표현식의 경우다. 클라이언트는 HTTP 헤더의 `User-Agent:` 뒤에 나오기에 이런 방식으로 해당 헤더를 뽑아내는 것이다. 그러나 `User-Agent:`로 단순 문자열 검색을 먼저 거치더라도 정규표현식을 줄이는 데에는 크게 도움이 되지 못한다. `User-Agent:`는 거의 모든 웹 요청 패킷에 들어 있기 때문이다. 이처럼 정규표현식에서 단순 문자열 검색으로 바꿀 수 있는 부분은 침입 탐지 대상인 패킷 보다는 평범한 패킷에서 늘상 보이는 부분인 경우가 많다. 정작 침입에 해당하는 문자열은 공격자들이 이런 검색 방식에 걸리지 않기 위해 수시로 변형을 가하기 때문이다. 이런 변형된 문자열을 모두 걸러내려는 것이 표현력이 풍부한 정규표현식을 사용하는 주된 이유이기도 하다.

## 문자열 시그니처 방식의 결함

### - 공격 패턴 표현의 심대한 제약

단순 문자열 검색으로 정규표현식 검색을 피하려는 시도가 효과적이지 못한 또 다른 이유는 애초에 단순 문자열을 추출하기가 애매한 정규표현식이 많다는 것이다. 공격자가 변형할 수 있는 모든 형태를 고려하여 패턴을 만들다 보면 패턴이 점점 일반화되기 때문이다. 일례로 영문자가 여덟 번 나온 후 네 자리 숫자가 뒤따르는 패턴은 `[a-zA-Z]{8}[0-9]{4}`인데, 이를 단순 문자열 검색으로 대체하려면  $52^8 \cdot 10^4$ 개의 단순 문자열이 필요하다. 이 경우는 워낙 많은 문자열 수로 말미암아 정규표현식 검색을 행하는 것보다 훨씬 못한 결과를 가져오게 된다.

단순 문자열 검색으로 표현 가능한 공격 형태가 점점 줄고 있음

## 문자열 시그너처 방식의 결합

### - 이종 오류에 무대책, 일종 오류에 취약

최악의 경우는, 저렇게 많은 문자열이 필요한 정규표현식을 몇몇 문자열만 사용하여 대체하는 것이다. 단순 문자열 10개가 있어야 정규표현식 하나를 대체할 수 있다고 할 때, (주관적인 판단으로) 가능성이 높아 보이는 두세 개만을 골라서 비교 결과 맞을 때만 정규표현식 검색을 행하는 식이다. 앞에서 예로 든 `User-Agent`: 같은 경우는 단순 문자열이 정규표현식 `^User-Agent \x3a[^\r\n]*`에 해당하는 모든 패킷에 포함되어 있기 때문에 단순 문자열 검색에서는 통계적으로 일종 오류(Type I error, false positive)만 있을 뿐 이종 오류(Type II error, false negative)는 절대 발생하지 않는다. 게다가 일종 오류는 추후에 이어지는 정규표현식 검색에서 모두 거를 수 있다. 그러나 필요한 개수보다 적은 수의 단순 문자열을 사용하는 경우에는 일종 오류와 이종 오류가 모두 발생할 수 있으며, 이종 오류의 경우 걸러낼 방법이 전무하여 공격 미탐의 소지를 열어두는 것이다.

문자열 시그너처는 이종 오류에 무방비가 되어 공격 미탐 가능성 높아짐

## 기존 시그너처 방식의 보안 취약성 해결

### - 이종 오류의 가능성 방지

침입 탐지 시스템의 최고의 가치는 이종 오류를 없애서 침입 가능성을 0으로 만드는 것에 있다. “보안”이 최우선인 것이다. 속도 향상은 그 다음의 부차적인 목표이며, 보안을 훼손하지 않는다는 전제 하에 기술력으로 해결해야 하는 문제이다. 앞에서 본 예에서처럼 침입 탐지 속도 향상을 위해 침입 가능성을 높이는 행위야말로 침입 탐지 시스템의 본분을 망각한, 주객이 전도된 행위라 하겠다. 불행히도 이러한 행태가 국내 보안 업계에 만연해 있는 것이 현실이다.

침입 탐지의 목표는 침입 가능성을 낮추는 것이므로 문자열 시그너처는 보안의 근본을 훼손하는 방식



# PCRE 검색 성능을 비약적으로 끌어 올린 혁신적 신기술의 소개

(주) 인프니스네트웍스 기술연구소 정지호 부장

PCRE(Perl Compatible Regular Expressions): 컴퓨터 언어인 펄(Perl)과 호환되는 정규표현식으로 관련 라이브러리가 공개됨

인프니스네트웍스의 Soligate UTM은 고도로 진화된 알고리즘(algorithm)으로 PCRE의 처리 성능을 비약적으로 향상시켰다. 그동안 관련 업계는 소프트웨어만으로 PCRE 수행 시 패턴의 개수 증가에 따른 성능 저하를 막기가 불가능한 것으로 인식하고 있었다. 불가능을 가능하게 만든 혁신적 기술의 개요를 소개한다.

PCRE는 가장 널리 사용되는 정규표현식이고 소개하고자 하는 이 기술은 모든 정규표현식에 적용이 가능하므로, 용어로서 “PCRE” 와 “정규표현식”을 구분없이 사용하고 보다 일반적인 문맥에서는 “정규표현식”을 우선 사용한 점을 밝혀둔다.

## I. 정규표현식과 유한 오토마타

### 정규표현식

정규표현식은 줄여서 정규식이라고도 하고 컴퓨터를 비롯한 전자장치에서 문자열 비교나 검색 시에 찾고자 하는 패턴(pattern)을 나타내는 용도로 가장 많이 사용된다. 정규표현식은 단순한 기호의 조합이 아니고 수학 이론으로 증명된 계산 모델을 기반으로 한다. 전산학의 분야인 형식 언어 이론에 따르면 정규표현식은 아무 내용도 없는 문자열을 의미하는  $\epsilon$  과, 한 글자로만 이루어진 정규식, 예를 들면 a, b, c 등을 기본으로 하며, 이들을 다음의 세 방식으로 조합하여 다양한 패턴을 만들 수 있다.

- 이어 붙이기 : abc, bbbb, baba 등
- 선택 :  $ab|c$  (ab 또는 c),  $ab|ba$  (ab 또는 ba)
- 반복 :  $c^*$  (c를 0번이나 그 이상 반복, 즉  $\epsilon, c, cc, ccc, \dots$ )

조합할 때 괄호를 이용하면 더욱 풍부한 표현이 가능하는데, 예를 들어  $a(b|d)c$ 는 “abc 또는 adc”를 의미한다.

PCRE는 사용의 편의를 위해 여러 확장 문법을 덧붙인 것인데, 대표적으로, 알파벳 소문자 중 한 글자를 뜻하는  $a|b|c|\dots|y|z$ 를  $[a-z]$ 로 줄여 쓰거나,  $a$ 가 세 번 이상 반복되는 것을 뜻하는  $aaaa^*$  ( $a$ 가 세 번 나온 후  $a$ 를 0번이나 그 이상 반복)를  $a\{3,\}$ 처럼 알아보기 쉽게 쓴다.

### 유한 오토마타

정규표현식의 검색을 효율적으로 행하기 위해 전산학의 이론에서는 유한 오토마타를 만들어서 비교하는데, 유한 오토마타는 체계화된 전산 이론의 한 분야이고 내용이 복잡하므로 이 글에는 설명을 생략하기로 한다. 정규표현식을 유한 오토마타로 바꾸는 방법은 톰슨 오토마타, 글루시코프 오토마타, 팔로우 오토마타, 안티미로프 오토마타 등 여러 가지가 있지만, 다수의 정규표현식을 검색하는 경우처럼 정규표현식의 개수가 중요한 경우라면 어떤 오토마타 방식을 취하느냐가 결정적인 것은 아니므로, 개별 오토마타 변환 방식은 따로 설명하지 않기로 한다.

정규표현식 검색은 유한 오토마타로 수행

## II. 다수의 정규표현식을 검색하는 기존 방법들

### 순차적 검색

다수의 정규표현식을 검색하는 가장 단순한 방법은 정규표현식마다 하나씩 유한 오토마타를 만들어서 순차적으로 검색하는 방법이다. 당연히 정규표현식의 개수가 늘어나면 비례해서 검색 시간이 증가하게 된다. 실제 보안 제품에 적용하면 PCRE 패턴의 개수가 수십 개 이상만 되어도 엄청나게 처리 성능이 떨어지므로 실질적으로 적용할 수 있는 방법이 아니다.

순차적 검색은 PCRE 패턴 개수 증가에 따라 비례하여 검색 속도가 증가하므로 사용 불가

NFA로 PCRE 검색 시  
활성 상태의 증가에  
따라 급격한 성능 저  
하로 실제 적용 불가

### 비결정적 유한 오토마타(NFA)로 통합

순차적 검색을 조금 발전시킨 것이 주어진 다수의 정규표현식을 모두 합쳐서 하나의 정규표현식으로 만들고, 최종 정규표현식에 해당하는 비결정적 유한 오토마타(NFA: Nondeterministic Finite Automata)를 만들어 검색하는 방법이다. 순차적 검색 방법보다 개선점이 있기는 하지만, 입력 문자열의 한 글자마다 검사해야 하는 오토마타의 상태, 즉 활성 상태(active states)의 수가 늘어나서 검색 시간이 증가하는 단점이 있다. 정규표현식의 개수가 수십 개 이상 되거나 또는 많은 상태를 만드는 표현이 포함된 경우에는 급격히 처리 속도가 증가하므로 역시 실질적으로 제품에 적용할 수 없다.

DFA로 PCRE 검색 시  
메모리 사용량의 급격  
한 증가로 실제 적용  
불가

### 결정적 유한 오토마타(DFA)로 변환

비결정적 유한 오토마타로 합친 경우 활성 상태의 수가 증가하는 문제를 극복하기 위해서 결정적 유한 오토마타(DFA: Deterministic Finite Automata)로 변환하는 방법도 있다. 비결정적 오토마타를 결정적 오토마타로 변환하게 되면 활성 상태 수가 한 개로 유지가 되므로 검색 속도가 많이 향상이 된다. 그러나, 정규표현식의 개수가 증가함에 따라 오토마타가 차지하는 메모리 사용량이 기하급수적으로 증가하게 되어서 실제 제품 적용이 또한 불가능하다.

## Ⅲ. 기존의 한계를 극복한 새로운 정규표현식 처리 기술

앞서 언급한 방식의 단점을 보완하기 위해 NFA와 DFA를 혼합하기도 하나, 근본적으로 다수의 정규표현식을 통합한 NFA로부터 출발하기 때문에, 해당 NFA의 복잡도가 성능을 결정한다. 결국 NFA의 활성 상태 수를 최소화해야 메모리 사용량도 줄이고 검색 성능 저하도 막을 수 있다. 아래 소개하는 새로운 기술은 다수의 정규표현식을 NFA로 병합하는 과정에서 활성 상태의 수를 줄이는 혁신적인 알고리즘을 기반으로 하는데, 두 가지 방식이 그 핵심이다.

**NFA의 상태를 비교하여 동일한 상태는 하나로 합쳐 NFA의 복잡도를 낮춤**

### 상태 공유 병합 (State-Sharing Merge)

상태 공유 병합은 합치고자 하는 두 NFA의 상태를 서로 비교하여, 동일한 상태는 하나로 합치는 것이다. 서로 다른 NFA의 상태가 동일한지 여부는 각 NFA에서 시작 상태에서 출발하여 해당 상태에 도달하기까지 어떤 문자열이 필요한가를 조사함으로써 알 수 있다. 시작 상태에서 해당 상태에 이르는 가능한 모든 경로를 고려했을 때 사용되는 문자열이 같은 상태가 양쪽 NFA에 모두 있다면, 이 두 상태는 하나로 합칠 수 있다. 두 상태 사이의 가능한 모든 경로를 고려하는 것은 알고리즘적으로 결코 쉬운 일이 아니나, 인프니스네트웍스가 채택한 알고리즘은 이런 병합 과정을 정규표현식으로부터 NFA를 생성한 후에 별도의 단계로 거치는 대신, NFA의 각 상태를 생성하면서 바로 병합 여부를 판정함으로써 복잡도를 현저하게 낮추었다. 이런 식으로 상태를 합치게 되면, 그 상태에 이르는 문자열이 입력으로 주어진 경우 활성 상태 수를 둘에서 하나로 줄일 수 있다. 합치고자 하는 NFA의 수가 늘어나면 늘어날수록 상태 공유 병합의 이점은 더욱 커진다.

**동일하지 않은 상태라도 천이가 같을 경우 공유 가능한 상태를 만들어 검색 성능 향상**

### 천이 공유 병합 (Transition-Sharing Merge)

상태 공유 병합을 통해 활성 상태를 상당수 줄일 수 있지만, 복잡한 정규표현식의 경우에는 한계가 있다. 보안 제품에서 사용하는 정규표현식에는 복잡한 정규표현식이 다수 포함되어 있기 때문에, 보다 세밀하게 NFA를 조사하여 상태 단위보다 더 작은 단위로 병합을 시도할 필요가 있다. 천이 공유 병합은 바로 이런 경우를 위해서 고안된 방법이다. 각 상태별로 비교하여 동일 여부를 판정하는 상태 공유 병합에서 한걸음 더 나아가, 상태가 동일하지 않더라도 해당 상태로 들어오는 천이들을 하나씩 비교하여, 동일한 천이가 있으면 그 천이에 대해서는 공유 가능한 상태를 생성한다. 새로운 상태를 만들어내는 과정이 포함되기 때문에 경우에 따라서는 합친 NFA의 상태 수를 더 늘릴 수도 있다. 그러나 상태들을 보다 세밀한 단위로 분류하기 때문에, 입력 문자열 처리 시에 활성 상태가 늘어나는 것을 대부분 방지한다. 천이 공유 병합 방식은 복잡한 정규표현식이 많은 경우에 더욱 진가를 발휘한다.

#### IV. PCRE 기반 IPS 시대를 개척한 쾌거

인프니스네트웍스의 Soligate UTM은 정규표현식 처리를 위해 전에 없었던 획기적인 기술을 기반으로 PCRE 패턴을 검색한다. 다수의 정규표현식을 동시에 검색하는 경우에 비결정적 오토마타로 병합하되, 병합 과정에서 개별 NFA를 공유와 천이 상태로 구별하여 합치는 과정을 수학 모델에 바탕하여 반복함으로써, 동시에 검색하고자 하는 정규표현식의 개수가 늘어나도 성능 저하를 최소화하는 것이다. 수년간 연구의 성과로서 초보적 방식인 문자열 시그너처 기반 보안 제품의 시대를 마감시키고 정규표현식, 즉 PCRE 기반 보안 제품 시대의 막을 열게 한 쾌거라고 자부한다.



# PCRE 기능과 성능의 검증 방법론

“PCRE 검증, 속일 수 없도록 제대로 해야”

(주)인프니스네트웍스 솔루션지원2팀 오종주 팀장

최근에 일부 IPS 개발 회사가 자신의 제품이 PCRE를 지원한다고 주장하지만 실제로는 PCRE 운용 성능을 충족하지 못한다. 올바르게 검증해야 성능 충족 여부를 확인할 수 있다. 잘못된 검증 방식의 허점을 이용해서 성능을 속일 수 있기 때문이다.

PCRE 엔진의 정확한 성능 측정을 위해서는 부하 트래픽과 공격 트래픽이 “정확히 PCRE 엔진을 통과”하여 우회가 불가능하도록 해야 한다. 특히 부하 트래픽의 엔진 통과를 강제하여 PCRE 엔진이 모든 트래픽을 검사하도록 해야 한다.

## 1. PCRE 기능 및 성능 측정의 기본 개념

첫번째로, 패킷 생성기로 정상 트래픽을 만들어 PCRE 엔진을 통과하도록 흘려 보낸다. 정상 트래픽의 부하를 점점 높여서 장비가 감당 가능한 최대 수준으로 설정한다.

두번째로, 이렇게 최대량의 정상 트래픽이 흐르는 상태에서, 장비에 설정된 PCRE 패턴에 해당하는 공격 트래픽을 추가로 보내고, 공격 트래픽이 검출되는지 확인한다. 제대로 공격 트래픽이 검출되는지 확인되었다면 이 최대량의 정상 트래픽이 장비의 성능이다.

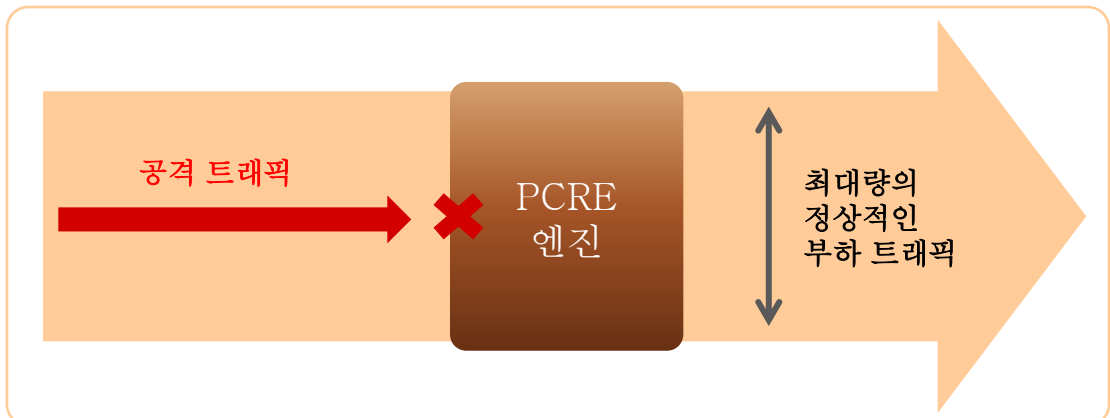


그림 2 : PCRE 엔진의 성능 측정 개념도

## 2. PCRE 패턴별 공격 탐지와 PCRE 엔진 성능 측정의 검증 개요

검증 항목	검증 내용 요약
1단계, PCRE 엔진의 공격 탐지 기능 테스트	다수의 PCRE 패턴에 대한 정확한 탐지 및 차단 여부 확인 ➔ 공격 트래픽 생성기를 이용하여 다양한 공격 트래픽을 발생시키고 각 패턴에 대한 정확한 탐지 여부, 즉 미탐과 오탐 여부를 확인
2단계, 우회 편법 배제된, 패턴 개수별 PCRE 엔진의 성능 측정	편법이 배제된 정확한 PCRE 엔진의 성능 측정 ➔ 우회 편법 방지를 위해 부하 트래픽과 공격 트래픽을 한 계측기에서 발생시키고, 프로토콜/포트는 트래픽과 패턴이 서로 무작위 또는 같도록 한 후, 패턴 개수에 따라 부하 트래픽을 증가시키면서 부하 트래픽의 손실이 없고 공격 트래픽을 정확히 탐지했을 때의 부하 트래픽량을 성능으로 인정

표 6 : PCRE 검증 방법 요약

## 3. 잘못된 검증 방식의 예

### 3.1 잘못된 검증 방식의 예 1

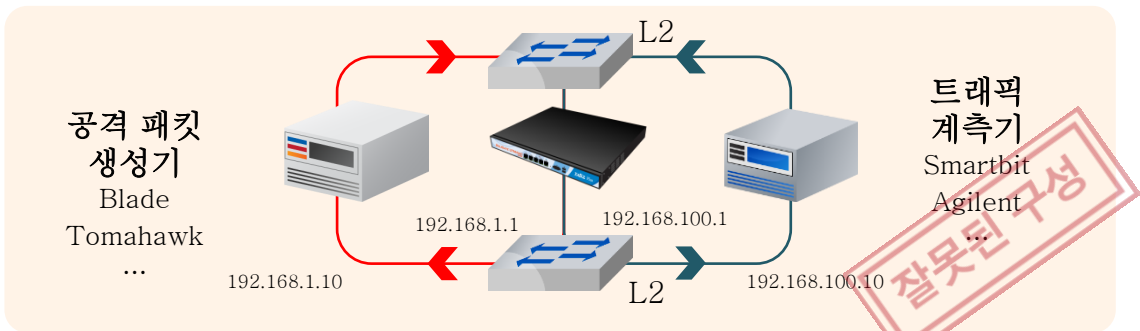


그림 3 : 부하와 공격을 분리하는 잘못된 검증 방식의 구성도

첫번째 잘못된 검증 방식의 예는 그림처럼 부하 측정 트래픽과 공격 트래픽의 경로가 다른 경우이다.

이런 방식의 문제점은 부하 측정 트래픽과 공격 트래픽이 라우팅 경로로 확실히 구분이 가능하다는 것이다. 라우팅 경로만 보아도 부하 측정 트래픽을 분리하므로, 부하 트래픽이 PCRE 엔진을 거치지 않도록 할 수 있다. 공격 트래픽 쪽의 라우팅 경로에 대해서만 PCRE 엔진을 가동하는 것이다.

이렇게 부하 측정 트래픽이 PCRE 엔진을 통과하지 않도록, 즉 우회하도록 해서, 실제 장비 성능보다 훨씬 높은 성능이 나오는 것처럼 속일 수 있다.

### 3.2 잘못된 검증 방식의 예 2

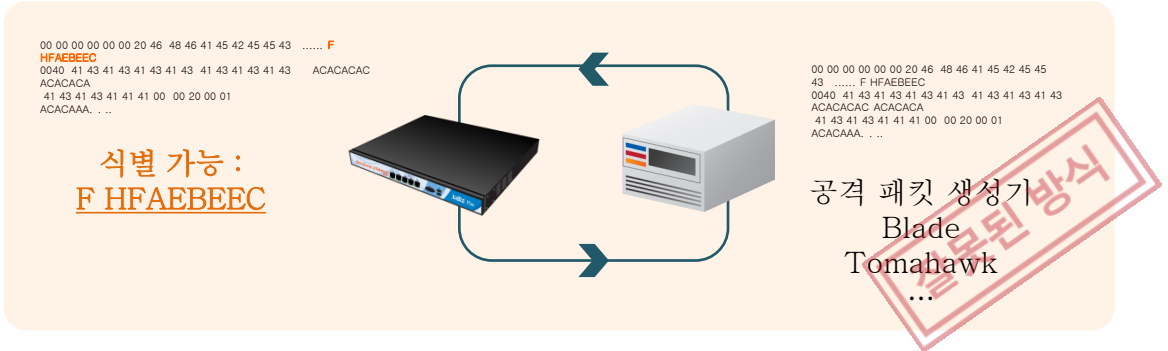


그림 4 : 공격 패킷을 분석할 수 있도록 하는 잘못된 검증 방식의 구성도

두번째 잘못된 예는 공격 패킷을 미리 알려주거나, 여러 번 장비를 통과시키는 경우이다.

이렇게 하면 공격 패킷의 해쉬(hash) 값이나 포함된 문자열을 확인할 수 있으므로, PCRE 엔진을 통과시키지 않고도 공격 검출이 가능하다. 실제로 BMT 현장에서 시험관이 공격 패킷을 교부한 후에 시간을 얼마간만 주어도, 금새 공격 패킷의 해쉬 값이나 포함된 문자열을 추출해 낼 수 있다. PCRE 엔진을 통과시키지 않고 단지 추출한 값을 보고 그 패킷을 골라내므로 얼마든지 PCRE 엔진의 성능을 속일 수 있는 것이다.

이는 어떤 패킷이 공격 패킷인지 미리 알려주는 것과 다를 바 없다. 실제 공격은 자신이 공격인지 스스로 알려주지 않는다. 공격인지를 식별하기 위해서 PCRE 엔진으로 확인해야 하는데, 이렇게 미리 공격 패킷을 알려주고 내용을 볼 시간을 주면 마치 PCRE 엔진으로 검출한 것처럼 시험관을 속일 수 있게 된다.

### 3.3 잘못된 검증 방식의 예 3

세번째 또 다른 잘못된 검증 방식은 프로토콜과 포트별로 부하 트래픽의 우회가 가능하도록 하는 경우이다.

예를 들면, 검증하고자 하는 PCRE 패턴을 TCP 프로토콜의 80번으로 하고 부하 측정 트래픽은 UDP로 설정하는 경우이다. PCRE 패턴이 TCP로 한정되므로, 부하 측정 트래픽을 PCRE 엔진으로 보내지 않아도 공격 탐지에 문제가 없다. 따라서, 이런 방식을 쓰면 부하 측정 트래픽의 전부 또는 상당 부분을 PCRE 엔진을 거치지 않도록 해서, 마치 성능이 잘 나오는 것처럼 속일 수 있게 된다.

PCRE 패턴과 부하 측정 트래픽의 프로토콜을 똑같이 TCP로 하더라도, 포트로 식별이 가능하면 마찬가지로 성능 측정이 정확히 이루어지지 않는다. 예를 들어, PCRE 패턴의 포트를 80번으로 하고, 부하 측정 트래픽의 포트를 무작위로 하는 경우이다. 이렇게 되면 부하 측정 트래픽 중에서 80번 포트인 것만 골라서 PCRE 엔진으로 보낼 수 있다.

포트의 수가 6만 개 정도이므로 부하 트래픽 6만 개 중에서 한 개 정도의 비율만 PCRE 엔진을 거치게 되어, 성능이 굉장히 높은 것처럼 보일 수 있다. 당연히 PCRE 엔진의 성능 측정을 제대로 할 수 없게 된다.

PCRE 패턴의 트로토콜과 포트	부하 측정 트래픽의 프로토콜과 포트	편법 시 부하 트래픽의 PCRE 엔진 통과 여부	정확한 성능 측정 여부
TCP 80	UDP Random	부하 트래픽은 PCRE 엔진을 전혀 통과하지 않음	X
TCP 80	TCP Random	부하 트래픽의 대부분이 PCRE 엔진을 통과하지 않음	X

표 7 : PCRE 검증 시 잘못된 프로토콜과 포트 설정

실제 알려진 PCRE 패턴만 살펴보다라도 TCP 프로토콜의 80번 포트에 대한 공격이 전체 패턴의 80%를 넘고, 일반적인 기업에서도 TCP 프로토콜의 80번 포트에 대한 트래픽이 대부분이기 때문에, 시험 환경에서는 부하 트래픽이 우회하지 못하도록 PCRE 엔진이 모든 트래픽을 검사해야 한다.

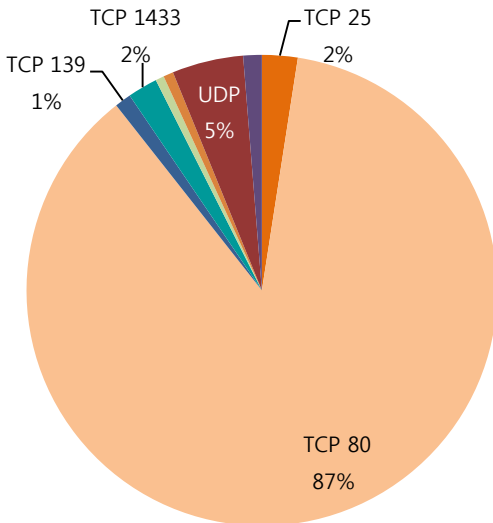


그림 5 : PCRE 패턴의 프로토콜과 포트 분포

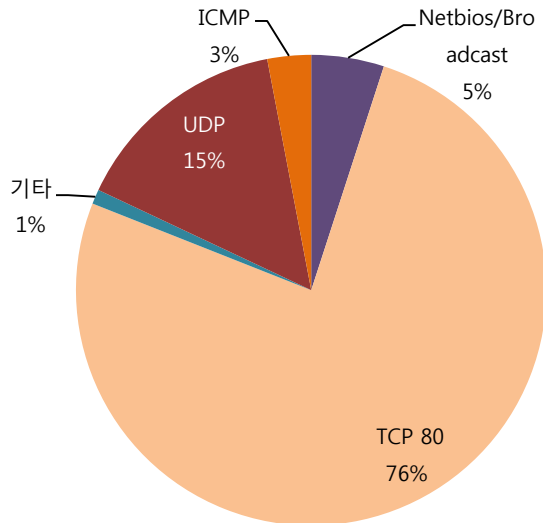


그림 6 : 기업 트래픽의 프로토콜과 포트 분포

정확한 PCRE 엔진 성능 측정을 위해서 부하 트래픽을 프로토콜과 포트로 우회하는 일이 없도록 해야 한다. 다음의 두 가지 중 한 가지로 설정하여 “부하 트래픽이 엔진을 통과” 하도록 강제해야 한다.

PCRE 패턴의 프로토콜과 포트	부하 측정 트래픽의 프로토콜과 포트	부하 트래픽의 PCRE 엔진 통과 여부	정확한 성능 측정 여부
Any	Random	PCRE 엔진 통과	O
TCP 80	TCP 80	PCRE 엔진 통과	O

표 8 : PCRE 검증 시 올바른 프로토콜과 포트 설정

성능 측정을 위해서는 공격 트래픽과 부하 트래픽을 모두 PCRE 엔진이 검사하도록 **확실히 강제**해야 한다. 하나의 방법으로는 PCRE 패턴의 포트를 특정 포트로 한정하지 않고 any로 설정한 상태에서 부하 트래픽의 포트를 무작위(random)가 되도록 하는 것이다. 이렇게 하면 공격과 부하 트래픽을 프로토콜이나 포트로 구별할 수 없으므로, 자연스럽게 모든 트래픽을 검사해야만 공격 패킷을 검출할 수 있게 된다. 또 다른 방법으로는 PCRE 패턴과 부하 측정 트래픽의 프로토콜 및 포트를 같게 하는 것이다. 마찬가지로 부하 트래픽을 구별하여 우회할 수 없게 된다.

#### 4. PCRE 검증 시 올바른 시험 방법의 요건

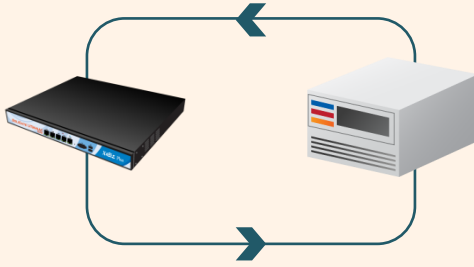
##### 편법 방지를 위한 PCRE 시험 요건

- ✓ 요건 1. 공격 트래픽과 정상 부하 트래픽의 라우팅 경로를 일치 (공격 패킷 경로 = 부하 트래픽 경로)
- ✓ 요건 2. 공격 패킷의 내용을 미리 보여주지 않음
- ✓ 요건 3. 공격 탐지 확인은 PCRE 패턴에 대응되는 매칭문자열을 부하 트래픽의 데이터 페이로드(payload)에 예고 없이 입력하여 검증
- ✓ 요건 4. 부하 트래픽의 데이터 페이로드는 무작위(random) 값을 사용
- ✓ 요건 5. PCRE 패턴과 부하 트래픽의 프로토콜/포트를 임의로 하거나 같게 함 (패턴 포트 = any, 부하 트래픽의 포트 = random 또는 모두 TCP 80)

정확한 PCRE 기능과 성능 측정을 위해서는 **시험 요건을 강화**해야 한다.

위의 요건을 모두 충족하는 것이 가장 바람직하다.

우회 등의 편법이 가능한 방식으로 검증하면 자격 미달인 장비를 실제와 달리 성능이 만족스러운 것으로 오판할 수 있다.



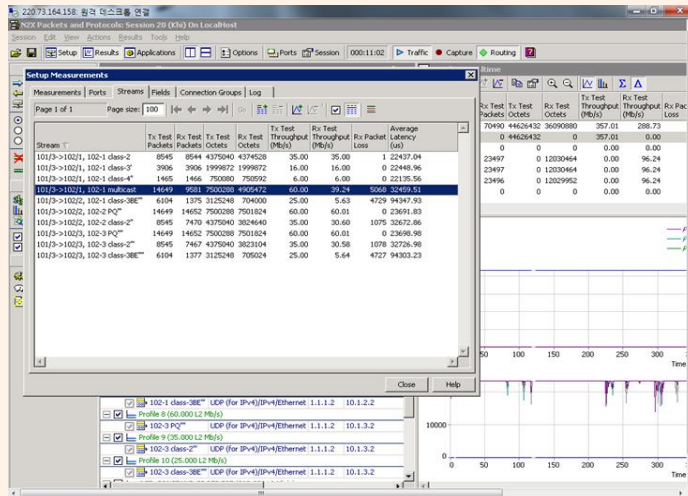
공격 패킷 & 정상 부하 트래픽  
생성기 & 생성 계측기

공격 트래픽과 부하 트래픽을  
한 장비에서 발생

그림 7 : PCRE 검증의 올바른 구성도

## 5. PCRE 기능과 성능 측정의 올바른 예

### 5.1 N2X를 이용한 PCRE 시험의 단계



- Step 1. TCP/UDP/Multicast 등 여러 가지 트래픽으로 장비에 부하 인가
- Step 2. PCRE 패턴의 개수를 추가하면서 트래픽 손실 여부 확인
- Step 3. 패킷의 데이터 payload 값에 PCRE 패턴에 인식되는 hex 값을 입력하여 검출 여부 확인

그림 8 : N2X를 이용한 올바른 PCRE 검증 방법의 단계

부하 트래픽 생성기(계측기)로 Agilent 사의 N2X장비를 사용하여 PCRE 기능과 성능을 검증하는 올바른 예를 간략히 소개한다.

먼저, 다양한 프로토콜의 정상 트래픽을 구성해서 장비에 부하를 주고, PCRE 패턴의 개수를 늘리면서 트래픽 손실 여부를 확인한다. 마지막으로 정상 트래픽의 최대 부하에서 PCRE 패턴에 대응되는 공격 패킷의 검출 기능이 제대로 작동하는지 확인한다.

이때 공격 패킷은 PCAP이나 DLL 형식 대신에, N2X를 사용하여 패킷의 페이로드(payload)에 매칭문자열의 hex 값을 입력해서 만드는 것이 핵심이다.

## 5.2 N2X를 이용하여 페이로드 내에 PCRE 패턴에 대응되는 매칭문자열의 hex 값 입력 방법

페이로드 내에 PCRE 패턴에 대응되는 매칭문자열의 hex 값을 입력하는 방법을 소개한다. 다음처럼 PCRE 패턴별로 검출 가능한 매칭문자열을 여러 개 작성한다. 이 매칭문자열을 hex 값으로 변환한 후 N2X가 생성하는 부하 트래픽 중 임의로 선택한 일부의 페이로드에 붙여 넣는다. PCRE 엔진이 검출한 패턴 번호가 해당 패턴과 같은지를 확인한다.

패턴 번호	PCRE 패턴	매칭문자열1	매칭문자열2	매칭문자열2
1	A.*BC[0-9]	AxxxxBC1	ABBBBBC2	AYYYYYBC3
2	...	...	...	...



패턴 번호	매칭문자열	Hex 변환
1	AxxxxBC1	41 78 78 78 78 42 43 31
1	ABBBBBC2	41 42 42 42 42 42 43 32
1	AYYYYYBC3	41 59 59 59 59 59 42 43 33

Step 1. 패턴별로 대응되는 매칭문자열 복수 개 생성

Step 2. 매칭문자열을 hex 값으로 변환

Step 3. N2X의 기능을 사용하여 부하 트래픽의 데이터 페이로드에 hex 값 입력

Step 4. 시험 장비의 PCRE 엔진이 공격으로 검출한 패턴 번호 확인

표 9 : 부하 트래픽의 payload에 패턴 대응 매칭문자열의 hex 값을 입력하여 공격 검출 여부를 확인하는 방법



# PCRE의 올바른 검증을 위한 상세 시험 절차서

(주)인프니스네트웍스 솔루션지원2팀 오종주 팀장

## 1. 시험 개요

### 1.1 시험 환경

PCRE 검증 시 공정하고 정확한 성능 측정을 위하여 아래와 같이 PCRE 성능 시험 방법을 제시한다. 시험은 1단계인 “PCRE 패턴별 공격 검출 시험”과 2단계인 “PCRE 엔진의 성능 측정”으로 나누어 진행한다.

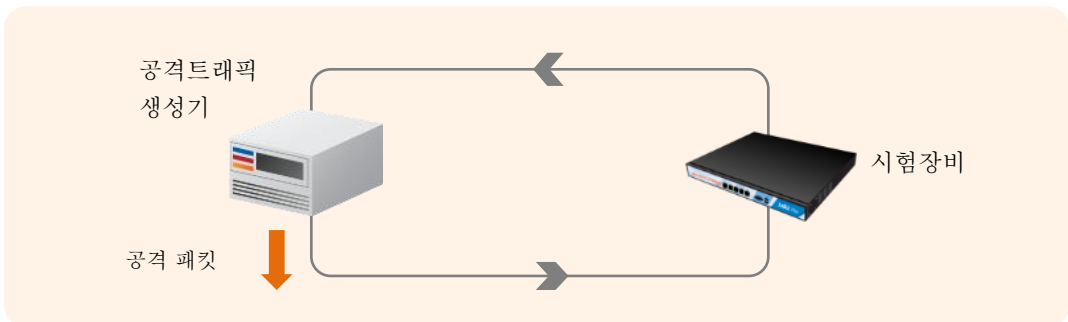


그림 9 : 공격 검출 시험 구성도 (1단계)

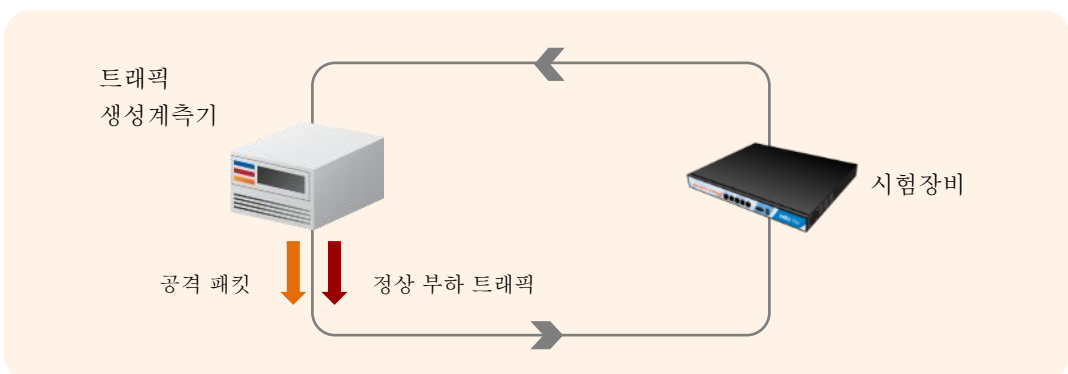


그림 10 : 성능 측정 시험 구성도 (2단계)

장비 표기	장비의 역할	비고
시험장비	검증 대상 IPS 또는 UTM 장비	
트래픽생성계측기	정상 부하 트래픽 생성기와 공격 트래픽 발생기를 결합 (예: N2X)	공격 시험은 PCRE 패턴 중 무작위 선택하여 매칭 문자열의 hex 값 입력
공격트래픽생성기	PCAP, DLL 등으로 공격 트래픽을 생성함 (예: Blade Informer)	전체 PCRE 패턴별 공격 검출 검증

표 10 : 장비별 표기와 역할

## 1.2 시험 절차 개요

### 1) 사전 준비 단계

시험평가자는 각 평가대상업체로부터 PCRE 패턴과 패턴별로 공격 검출을 위한 공격 트래픽의 PCAP 또는 DLL 파일 및 PCRE 성능 측정 용도의 검출 가능한 매칭문자열을 복수 개 취합한다.

### 2) 시험 준비 단계

시험평가자는 PCRE 시험장에 입회한 평가대상업체 담당자가 시험장비를 설치하고 PCRE 검증 시험 환경을 구축할 수 있도록 한다.

### 3) 환경 구축 검증 단계

시험평가자는 평가대상업체별로 구축한 시험 환경을 검사한다.

### 4) 시험 단계

#### - 1단계 : PCRE 패턴별 공격 검출 시험

시험평가자는 공격트래픽생성기로 공격 패킷을 발생시키고 침입 탐지 및 차단 시험을 수행한 후 각 패턴에 대한 탐지 및 차단 결과를 확인한다.

#### - 2단계 : PCRE 엔진의 성능 측정

- 성능 측정을 위한 부하 트래픽과 PCRE 패턴별 공격 검출을 위한 공격 트래픽을 모두 하나의 트래픽생성계측기에서 생성하여 시험한다. 시험평가자는 성능 측정을 위한 트래픽을 성능 단계별로 발생시키고, 각 단계별로 임의의 검출 가능 매칭문자열을 삽입하여 PCRE 엔진의 성능과 매칭문자열의 검출 여부를 확인한다.

- PCRE 패턴의 개수를 50개부터 500개까지, 또는 그 이상으로 추가하여 패턴의 개수에 따른 시험장비의 성능을 측정한다.

## 2. 시험 절차 중 준비 단계의 세부 사항

### 2.1 사전 준비 단계

- 1) 시험평가자는 각 평가대상업체로부터 공격 트래픽을 생성하기 위해 공격에 해당하는 PCAP 또는 DLL 파일과, 각 PCRE 패턴에 검출 가능한 매칭문자열의 정보(문자열과 hex 변환값)를 각 평가대상업체로부터 PCRE 패턴별 복수 개씩 제공받는다.

PCRE 패턴 번호	PCRE표현	매칭문자열1 (hex값)	매칭문자열2 (hex값)	매칭문자열3 (hex값)
n	A.*BC[0-9]	AxxxxBC1	ABBBBBC2	AYYYYYBC3
		41 78 78 78 78 42 43 31	41 42 42 42 42 42 43 32	41 59 59 59 59 59 42 43 33

표 11 : PCRE 패턴별 매칭문자열과 해당 hex 값의 예

- 2) 시험평가자는 각 평가대상업체로부터 받은 PCRE 패턴과 그에 대응하는 공격 패킷의 PCAP 또는 DLL 파일 및 검출 가능 매칭문자열 복수 개를 취합한다.

### 2.2 시험 준비 단계

- 1) 시험평가자는 사전 준비 단계에서 취합한 PCAP(또는 DLL) 파일과 PCRE 패턴 목록을 평가대상업체에게 미리 교부하지 않는다.  
※ 미리 교부하였을 경우 공격 패킷의 해쉬 값이나 포함된 문자열을 확인할 수 있으므로, PCRE 엔진을 통과시키지 않고도 탐지가 가능
- 2) 시험평가자는 평가대상업체들이 시험 환경 구성을 할 수 있도록 한다.
- 3) 평가대상업체는 PCRE 검증 시험의 환경 구성을 완료한 후에 시험평가자에게 통보하고, 시험평가자는 이를 확인한다.

## 3. 시험 절차 중 검증 단계의 세부 사항

### 3.1 검증 단계별 세부 사항

- 1) 1단계 : 공격트래픽생성기로 공격 검출 시험  
시험평가자는 사전에 제출된 PCAP 또는 DLL 파일을 선별하여 공격트래픽생성기를 이용하여 공격 트래픽을 발생시키고, 공격 트래픽을 시험장비에서 검출하는지 확인한다.

2) 2단계 : 트래픽생성계측기로 성능 측정

- 성능 측정용 부하 트래픽의 프로토콜과 포트는 무작위(random)로 하고, PCRE 패턴의 프로토콜과 포트는 any로 설정한다. 프로토콜과 포트를 무작위로 하지 않는다면 TCP 80과 같이 부하 트래픽 및 PCRE 패턴의 프로토콜과 포트를 하나로 고정한다.
- 정상 부하 트래픽의 페이로드는 무작위(random) 값으로 채운다. 페이로드의 값이 하나로 정해지면 쉽게 식별하여 PCRE 엔진을 우회시킬 수 있다.
- 시험장비가 감당 가능한 최고 부하의 트래픽을 흘리는 도중 임의의 순간에, 성능 측정 부하 트래픽 중 임의의 패킷 페이로드에 시험평가자가 무작위로 선별한 매칭문자열의 hex 값을 입력하여 해당 매칭문자열의 PCRE 엔진에서의 검출 여부를 확인한다. 정상 부하 트래픽과 공격 트래픽이 모두 하나의 트래픽생성계측기로부터 발생하는 것에 유의한다. 부하 트래픽이 손실없이 시험장비를 통과하고, 동시에 매칭문자열이 검출되어야만 당시의 트래픽량을 시험장비의 성능으로 인정한다.

3.2 평가대상업체의 편법 여부 검증

1) 부하 트래픽의 PCRE 엔진 우회 여부 확인

- 부하 트래픽이 PCRE 엔진을 정확히 통과하였는지 파악

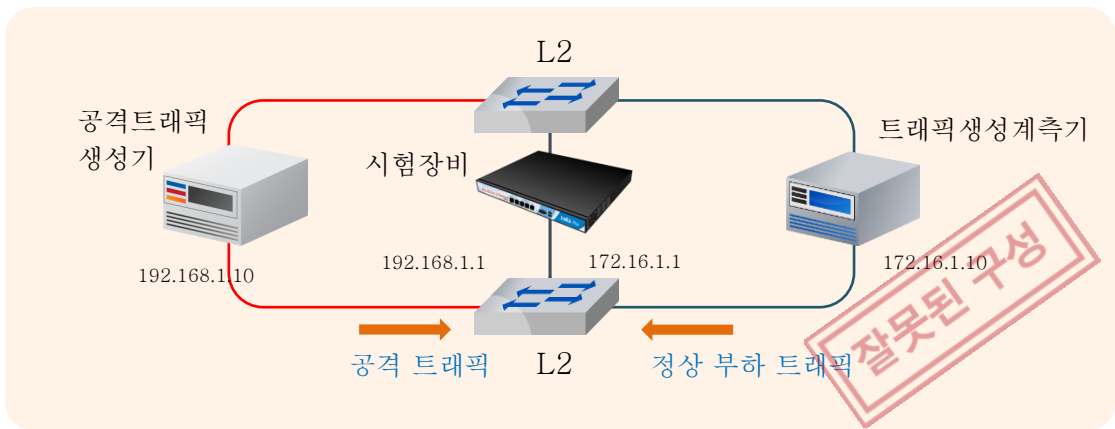


그림 11 : 공격 트래픽과 정상 부하 트래픽이 분리된 잘못된 측정 구성의 예

- 위 그림과 같이 네트워크 환경을 구축했을 경우 성능 측정을 위한 부하 트래픽이 PCRE 엔진을 통과하지 않도록 우회할 수 있다.
- 성능 측정 트래픽과 공격 트래픽이 라우팅 경로로 확실히 구분되면, 공격 트래픽의 라우팅 경로에 대해서만 PCRE 엔진을 통과하도록 하는 편법이 가능하다.

## 2) 시험장비에 설정된 네트워크와 보안 정책 정보 검증

## - 시험장비에 설정된 PCRE 패턴의 프로토콜과 포트를 확인

- 정확한 PCRE 엔진 성능 측정을 위해서는 정상 부하 트래픽이 PCRE 엔진을 모두 통과하도록 해야 한다.
- PCRE 패턴을 TCP 프로토콜의 80번 포트로 하고 정상 부하 트래픽은 UDP로 설정한 경우 PCRE 패턴이 TCP에 한정되므로, 정상 부하 트래픽을 PCRE 엔진으로 보내지 않아도 공격 탐지가 가능하다. 따라서, 이런 방식을 쓰면 정상 부하 트래픽을 PCRE 엔진으로 보내지 않고, 마치 성능이 잘 나오는 것처럼 시험평가자를 속일 수 있다.
- 따라서, PCRE 패턴과 부하 트래픽의 프로토콜/포트를 서로 같게 하거나, PCRE 패턴은 any로 부하 트래픽은 random으로 설정해야 한다.

## 3.3 다량의 PCRE 패턴으로 PCRE 패턴 개수에 따른 성능 측정

## - 다량의 PCRE 패턴 적용 시 성능 저하 여부를 검증

- 주요 PCRE 패턴의 개수가 300개가 넘고 새로운 공격에 따라 그 수가 계속 증가하므로, 최소 500개 이상에서도 성능이 담보되어야 실제 운용이 가능하다.
- PCRE 패턴의 개수가 많아도 편법을 방지하는 상기 절차가 준용된다면, 검증 1단계인 공격 검출에서 모든 패턴에 대해 검출 여부를 확인하고, 검증 2단계인 성능 시험에서는 부하 트래픽과 공격 트래픽을 한 계측기에서 생성하여 부하 트래픽이 PCRE 엔진을 우회하지 않도록 하면 올바른 검증이 가능하다.











# INFNIS

**(주)인프니스네트웍스**

서울시 강남구 논현동 130-29 (논현로 653) 3층

<http://www.infnis.com>

TEL 02-3443-3456 (代) FAX 02-3443-6060

E-mail [sales@infnis.com](mailto:sales@infnis.com)

Copyright © Infnis Networks, Inc. All rights reserved.





INFNIS

**(주)인프니스네트웍스**

서울시 강남구 논현동 130-29 (논현로 653) 3층

<http://www.infnis.com>

TEL 02.3443.3456(代)

FAX 02.3443.6060

E-mail [sales@infnis.com](mailto:sales@infnis.com)