**Pusan National University Computer Science and Engineering Technical Report 2024-10**

**난독화에 강한AI 기반 웹쉘 탐지**



201914119 문정윤

202155507 구지원

202155614 차기은

지도교수 최윤호

**목 차**

[1.](#_gjdgxs) 서론 1

[1.1.](#_30j0zll) 연구 배경 1

[1.2.](#_1fob9te) 기존 문제점 2

[1.3.](#_3znysh7) 연구 목표 2

[2.](#_2et92p0) 연구 배경 3

[2.1.](#_tyjcwt) 웹쉘 3

2.1.1. Webshell의 원리 3

2.1.2. Webshell의 분류 4

[2.2.](#_3dy6vkm) 텍스트랭크(TextRank) 4

2.3. 지도학습 알고리즘 5

2.3.1. 지도학습의 원리 5

2.3.2. 지도학습 알고리즘의 종류 5

2.3.3. 지도학습의 응용 분야 6

2.4. PHP(Hypertext Preprocessor) 7

2.4.1. Opcode(Operation Code) 7

2.4.2. AST(Abstract Syntax Tree) 7

[3. 연구 내용](#_1t3h5sf) 8

[3.1.](#_4d34og8) 암호화된 웹쉘 파일 비난독화 8

[3.1.1.](#_2s8eyo1) 가상환경 셋팅 8

[3.1.2.](#_17dp8vu) 비난독화 9

[3.2.](#_3rdcrjn) AST 및 opcode 추출 12

3.2.1. AST 추출 12

3.2.2. opcode 추출 13

[3.3.](#_26in1rg) TextRank 알고리즘을 이용한 feature matrix 생성 14

[3.3.1.](#_lnxbz9) 데이터 전처리 16

[3.3.2.](#_35nkun2) TextRank 알고리즘 적용 17

3.4. 머신러닝 알고리즘을 사용한 성능 평가 18

3.4.1. RF 18

3.4.2. SVM 18

3.4.3. XGBoost 18

[4.](#_1ksv4uv) 연구 결과 분석 및 평가 19

4.1. 모델 성능 평가 19

4.2. Feature Importance 분석 22

4.2.1. 특징 중요도 분석 23

4.3. 웹쉘 판단 웹 개발 24

[5.](#_44sinio) 결론 및 향후 연구 방향 26

5.1. 결론 26

5.2. 향후 연구 방향 26

[6.](#_2jxsxqh) 참고 문헌 28

# 서론

## 연구 배경

웹쉘은 웹 서버를 침투하여 추가적인 공격을 시작할 수 있는 악성 스크립트이다. 이러한 악성 스크립트 웹쉘을 이용한 공격은 지속적으로 증가하고 있다.

마이크로소프트의 Microsoft 365 Defender에서 발표한 ‘Web shell attacks continue to rise’에 따르면, 2022년과 2021년에 발견된 웹쉘은 14만 건으로 2019년부터 2년간 발생된 웹쉘 평균 개수의 7만 7천 건에 비해 2배 증가한 수치이다. 이러한 추세는 웹쉘이 지속적으로 증가함을 보여준다. 또한, 그 추세를 나타내는 아래의 그래프는 웹쉘이 단순히 지속적으로 증가한다는 사실과 더불어 가속화되고 있다는 것을 보여준다.

국내의 경우, LG유플러스에서 2009년과 2018년에 업로드된 악성코드(웹쉘)가 2023년 1월까지도 삭제되지 않은 채로 남아 있었고, 대규모 개인정보 유출 사고로 인해 과징금 68억원과 과태료 2700만원을 부과하였다. 이와 같은 사례로 웹쉘의 심각성을 알 수 있다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명이러한 웹쉘의 증가하는 보편성은 공격자에게 얼마나 단순하고 효과적인지에 기인할 수 있다. 또한 기존 웹쉘 정적 탐지에서 많이 사용된 통계적인 기법은 웹쉘 탐지 성능이 감소하는 난독화 편향 문제가 발생한다. 그러므로 난독화 편향 문제가 발생하지 않으면서, 웹쉘을 탐지하는 방법을 제안하려한다.

## 기존 문제점

1. **난독화 편향 문제**: 기존 웹쉘 탐지 방법들은 주로 정적 분석 기법에 의존했으며, 공격자들이 웹쉘에 난독화 기술을 적용할 경우 탐지 성능이 크게 저하되는 문제가 발생했다. 기존의 통계적 기법 및 단순 패턴 매칭 기반 탐지 방법들은 이러한 난독화된 웹쉘을 제대로 탐지하지 못해 **오탐과 미탐**이 증가했다.
2. **정확성 부족**: 정적 분석 기법을 사용하여 코드 구조나 문법적 패턴을 탐지하는 방법들은 웹쉘이 아닌 정상적인 스크립트와 혼동되는 경우가 많았다. 코드에서 사용되는 함수나 변수명에 의존하는 탐지 방식은 웹쉘과 일반 파일의 구별이 어려워, 오탐율이 높아지게 된다.
3. **다양한 웹쉘 형태에 대한 대응 부족**: 웹쉘은 다양한 형태로 존재하며, 공격자들은 지속적으로 새로운 유형의 웹쉘을 개발하고 변형된 형태의 악성 스크립트를 배포한다. 그러나 기존 탐지 시스템은 주로 알려진 웹쉘 패턴에 대한 대응에 중점을 두고 있어 미탐 문제를 일으킬 수 있다. 즉, 새로운 웹쉘 변종에 대한 탐지 능력이 부족하다는 한계가 있다.
4. **탐지 과정의 복잡성**: 기존의 웹쉘 탐지 과정은 복잡한 설정과 전문적인 기술을 필요로 하여 일반 사용자나 관리자가 쉽게 접근하기 어려웠다. 특히, 탐지 결과의 해석이 어렵고 시각적으로 직관적이지 않아서 비전문가들이 이를 활용하는 데 많은 어려움이 있었다.

이러한 기존 문제점들을 해결하기 위해 본 연구에서는 난독화된 웹쉘에 대한 탐지 편향 문제를 줄이고, 다양한 웹쉘 형태에 대해 높은 정확도로 탐지할 수 있는 방법을 제안하고자 한다.

## 연구 목표

본 과제는 딥 러닝 및 머신 러닝 기법을 활용한 난독화 웹쉘 탐지 모델 개발을 목표로 한다. 먼저 난독화 및 비난독화 도구를 기반으로 난독화 웹쉘 전처리 모듈을 구현하는 등 웹쉘 분석 환경을 구축한 다음, 난독화 웹쉘 탐지를 위한 textrank 기반 알고리즘을 구현한다. 마지막으로 시각화 도구를 이용하여 웹쉘 탐지 결과를 시각적으로 쉽게 파악할 수 있도록 한다.

이러한 과제는 난독화 웹쉘을 탐지함은 물론, 결과를 한눈에 파악하기 쉽게 하여 누구나 쉽게 웹쉘 탐지의 과정과 결과에 쉽게 접근할 수 있도록 한다.

1. **난독화 편향 문제 해결**: 웹쉘과 일반 파일의 opcode와 AST 시퀀스를 이용해 이들 간의 관계 및 연관성을 분석하는 웹쉘 맞춤 TextRank 알고리즘을 활용하여, 전체적인 문서의 관계 및 연관성을 파악하고 특징을 추출하는 방법을 제안합니다. 이를 통해 난독화 편향 문제가 발생하지 않는 웹쉘 탐지 방법을 도입합니다. 데이터 셋에 난독화된 일반 파일이 포함되어 있을 때, 웹쉘 탐지 과정에서 난독화 편향 문제가 나타나지 않도록 특징을 추출하고 탐지하는 방법을 제공합니다. 이를 통해 오탐을 줄이고 탐지 정확성을 향상시킵니다.
2. **성능 평가**: 제안하는 방법에 다양한 머신러닝 알고리즘을 적용하여 정확도(accuracy), 정밀도(precision)에 대한 성능 평가를 진행합니다.
3. **시각화**: 시각화 도구를 이용하여 웹쉘 탐지 결과를 시각적으로 쉽게 파악할 수 있도록 한다. 이러한 과제는 난독화 웹쉘을 탐지함은 물론, 결과를 한눈에 파악하기 쉽게 하여 누구나 쉽게 웹쉘 탐지의 과정과 결과에 쉽게 접근할 수 있도록 한다.

# 연구 배경

## 웹쉘

#### Webshell의 원리

Webshell은 특정 페이지에 HTTP 요청을 보내고, 공격자가 보낸 명령을 실행하는 함수들을 사용하여 동작한다. 이러한 명령의 결과는 HTTP 요청에 대한 응답으로 공격자에게 전송된다 .

#### Webshell의 분류[1]

웹쉘은 스크립트 언어의 기능과 크기에 따라 대략 세 가지 범주로 나눌 수 있다:

(i) 대형 트로이 목마(Big Trojan): 크기가 크고 명령 실행, 데이터베이스 작업 및 기타 악의적인 의도를 위한 종합적인 기능을 갖추고 있다. 또한, 대형 트로이 목마는 사용자 친화적인 그래픽 인터페이스를 갖추고 있다.

(ii) 한 줄 트로이 목마(One Word Trojan): 한 줄의 코드로 이루어진 트로이 목마이다. 그 짧은 길이 때문에 일반 파일이나 그림에 종종 삽입된다. 초기 공격 도구(예: Chinese Chopper)와 연결되면 대형 트로이 목마와 같은 다양한 기능을 수행할 수 있다.

(iii) 소형 트로이 목마(Small Trojan): 크기가 작고 숨기기 쉽지만 일반적으로 업로드 기능만 갖추고 있다. 대부분의 웹사이트가 파일 업로드 시 크기 제한을 두기 때문에, 공격자들은 일반적으로 소형 트로이 목마를 통해 먼저 업로드 권한을 획득한 후, 주요 기능을 수행하기 위해 대형 트로이 목마를 웹사이트에 업로드한다.

## 텍스트랭크(TextRank)

TextRank 알고리즘은 문서 내의 문장 또는 단어들 간의 중요도를 계산하는 그래프 기반 텍스트 요약 및 키워드 추출 알고리즘이다. 이 알고리즘은 구글의 검색 엔진 기반 알고리즘인 PageRank 알고리즘에서 파생되었으며, PageRank 알고리즘의 개념을 텍스트에 적용한 것이다[3]. PageRank 알고리즘이 그래프 랭킹 알고리즘으로 페이지의 링크 정보를 기반으로 페이지의 중요도를 판단하는 반면[4][5], TextRank 알고리즘은 단어의 연관성을 통해 중요도를 판단한다[3].

TextRank 알고리즘의 전체 과정은 다음과 같다.

1. 토큰화 및 태깅: 텍스트를 토큰화하고 각 토큰에 품사 태그를 부여한다.
2. Syntactic Filtering: 특정 품사만 남겨두고 나머지 단어들은 제거한다.
3. 그래프 생성: 단어 간의 동시 출현 빈도를 계산하여 그래프를 생성한다.
4. 랭킹 계산: 생성된 그래프를 기반으로 단어 또는 문장의 중요도를 계산한다.
5. 키워드 추출 및 문서 요약: 중요도를 바탕으로 키워드를 추출하거나 문장을 요약한다[3].

TextRank 알고리즘은 단어 간의 의미적 유사성을 고려하여 문맥을 반영한 텍스트 요약을 제공하며, 문서 요약, 키워드 추출, 정보 검색 등 다양한 텍스트 마이닝 작업에 활용됩니다[3].

* 1. **지도학습 알고리즘**

지도학습(Supervised Learning)은 기계 학습의 한 유형으로, 알고리즘이 주어진 입력 데이터와 해당하는 정답(레이블)을 사용하여 학습하는 방법이다. 지도학습의 주요 목표는 새로운 데이터에 대해 정확한 예측을 수행할 수 있는 모델을 구축하는 것이다. 이는 웹쉘 난독화 탐지와 같은 보안 문제를 해결하는 데에도 효과적으로 적용될 수 있다.[2]

* + 1. **지도학습의 원리**

지도학습에서, 알고리즘은 입력 데이터(features)와 이와 관련된 출력 데이터(labels)를 포함하는 학습 데이터셋을 사용한다. 웹쉘 탐지의 경우, 입력 데이터는 웹 요청 및 서버 로그 등의 특징을 포함하며, 출력 데이터는 정상 및 악성 웹쉘로 레이블링된 데이터이다. 알고리즘은 이 데이터셋을 분석하여 입력과 출력 간의 관계를 학습한다. 학습이 완료되면, 새로운 웹 요청이 주어졌을 때 이를 기반으로 악성 여부를 예측할 수 있다.

* + 1. **지도학습 알고리즘의 종류**

웹쉘 난독화 탐지에 효과적인 몇 가지 주요 지도학습 알고리즘은 다음과 같다:

(i) 서포트 벡터 머신(Support Vector Machine, SVM): 데이터를 분류하기 위한 초평면을 찾는 방법으로, 고차원 공간에서도 효과적으로 작동한다. SVM은 웹쉘 탐지에서 높은 정확도를 제공할 수 있다.

(ii) 랜덤 포레스트(Random Forest, RF): 여러 결정 트리를 앙상블하여 예측 성능을 향상시키는 방법으로, 과적합을 방지하고 일반화 성능을 높인다. RF는 다양한 특징을 고려할 수 있어 웹쉘 탐지에 유리하다 .

(iii) XGBoost(Extreme Gradient Boosting): 여러 결정 트리를 순차적으로 학습하고 앙상블하여 예측 성능을 향상시키는 강력한 그래디언트 부스팅 알고리즘으로, 매우 높은 예측 정확도를 제공한다. XGBoost는 웹쉘의 복잡한 패턴을 효과적으로 학습할 수 있다 .

(iv) 결정 트리(Decision Tree): 데이터의 특징을 기반으로 분할하여 예측을 수행하는 트리 구조의 모델이다. 이해와 해석이 용이하지만, 단독으로 사용 시 과적합될 가능성이 있다.

(v) 로지스틱 회귀(Logistic Regression): 이진 분류 문제를 해결하기 위한 방법으로, 웹쉘 탐지에서 정상과 악성의 이진 분류를 수행할 수 있다.

(vi) 신경망(Neural Networks): 인간의 두뇌 구조를 모방한 모델로, 복잡한 패턴 인식과 예측 작업에 유리하다. 다층 퍼셉트론(Multi-layer Perceptron, MLP)과 같은 구조가 웹쉘 탐지에 활용될 수 있다.

* + 1. **지도학습의 응용 분야**

지도학습 알고리즘은 웹쉘 난독화 탐지를 포함한 다양한 보안 분야에서 활용될 수 있다. 예를 들어, 스팸 이메일 필터링, 악성 코드 탐지, 이상 행위 탐지 등이 있다. 이러한 알고리즘은 대량의 레이블 데이터가 존재할 때 매우 효과적이며, 데이터에서 유의미한 패턴을 찾아내어 정확한 예측을 가능하게 한다.

지도학습 알고리즘은 웹쉘 난독화 탐지의 중요한 부분을 차지하며, 웹 보안을 강화하기 위해 널리 사용되고 있다. 이 알고리즘을 잘 이해하고 적절히 활용하는 것이 성공적인 웹쉘 탐지와 보안 모델 개발의 핵심이다.

### PHP(Hypertext Preprocessor)

PHP는 웹 개발을 위해 설계된 서버측 스크립트 언어이다. PHP 실행 흐름은 다음과 같다.

1. PHP 엔진이 소스 코드를 읽고 파싱한 뒤, AST를 생성
2. 파싱된 AST를 실행할 수 있는 opcode로 컴파일
3. 생성된 opcode는 PHP 가상 머신(Virtual Machine)에서 실행되고 웹 서버 응답을 생성
   * 1. **Opcode(Operation Code)**

Opcode는 PHP 스크립트가 서버에서 실행될 때, PHP 엔진이 이해하고 실행하기 위해 변환하는 중단 단계의 코드를 나타낸다.

* + 1. **AST(Abstract Syntax Tree)**

PHP 코드로부터 추출 가능한 AST는 PHP 코드의 추상 구문 트리를 나타낸다. AST는 파서(parser)에 의해 생성되며 PHP 코드를 토큰(Token)으로 분해하고 이를 트리 형태로 구성한다. 각 노드는 코드의 특정 구성 요소에 해당하며, 이러한 노드 간의 관계는 코드의 구조를 반영한다.

# 연구 내용

## 암호화된 웹쉘 파일 비난독화

### 가상환경 셋팅

1. Docker

**- 환경 일관성 유지:** Docker는 애플리케이션과 그에 필요한 모든 의존성을 하나의 컨테이너에 포함하여 일관된 실행 환경을 보장한다. 다양한 개발 환경(로컬, 서버 등)에서 동일하게 작동하게 하여, 환경 차이로 인한 문제를 최소화할 수 있다.

**- 의존성 충돌 방지:** PHP, PHPDBG, XGBoost, Flask 등 다양한 라이브러리와 도구를 사용하는 경우, 시스템에 설치된 다른 버전과의 충돌이 발생할 수 있다. Docker를 사용하면 각각의 도구가 컨테이너 내에서 독립적으로 관리되기 때문에 의존성 문제를 쉽게 해결할 수 있다.

**- 격리된 테스트 환경:** PHP, PHPDBG와 같은 도구는 웹 서버의 실행 환경과 밀접하게 관련되어 있어 시스템에 손상을 줄 수 있다. Docker를 사용하면 이러한 도구를 격리된 환경에서 테스트하고, 문제가 발생해도 쉽게 컨테이너를 삭제하여 다시 초기 상태로 복구할 수 있다.

**- 유연한 확장성:** Docker Compose와 같은 도구를 사용하면 웹 서버, 데이터베이스, AI 모델, 디버깅 도구 등 다양한 서비스들을 쉽게 연동하고 확장할 수 있다.

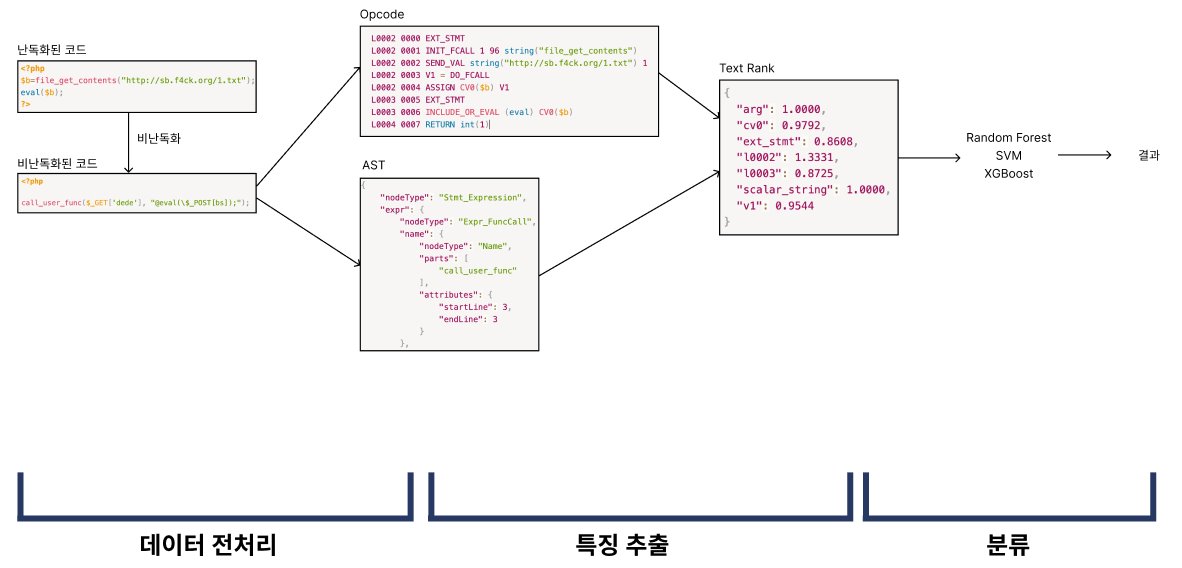
**- 안정성:** Docker는 격리된 컨테이너에서 애플리케이션을 실행하기 때문에, 메인 시스템에 영향을 미치지 않고 다양한 실험과 테스트를 수행할 수 있다.

2. PHPDBG

- PHPDBG는 PHP 코드를 디버깅하고 Opcode를 추출하는 도구이다. 난독화된 웹쉘을 비난독화한 후, 실행 가능한 코드에서 구체적인 실행 경로를 추적하기 위해 PHPDBG가 필요하다. 이를 통해 악성 행위가 어떻게 발생하는지 더 명확히 파악할 수 있다.

3. PHP

**- PHP 버전 관리**: PHP는 버전에 따라 기능과 보안 패치가 달라진다. Docker를 사용하면 특정 PHP 버전(예: 7.4)을 쉽게 설치하고 그 환경에서 웹쉘 비난독화를 테스트할 수 있다. Docker 컨테이너 내부에서 PHP 버전을 자유롭게 변경하거나 여러 PHP 버전을 동시에 사용할 수 있어, 코드 호환성을 점검하는 데 유용하다.



웹 쉘 여부 판단 과정 흐름도

### 비난독화 [6]

#### 

|  |
| --- |
| $files = scandir($inputDir); foreach ($files as $file) {  if ($file == '.' || $file == '..' || is\_dir($inputDir . '/' . $file)) {  continue;  }  ... } |

scandir 함수로 입력 디렉토리에 있는 파일 목록을 스캔한다.

.와 .. 같은 특별한 파일명이나 디렉토리는 무시하고, 실제 파일만을 대상으로 한다.

#### 3.1.2.2.

|  |
| --- |
| $command = "php $phpDeobfuscatorPath/index.php -f $inputFilePath > $outputFilePath"; exec($command, $output, $return\_var); |

각 파일에 대해 exec 함수를 사용하여 비난독화 명령을 실행한다. 여기서 $command는 비난독화 PHP 스크립트를 실행하는 명령어이다. $inputFilePath는 난독화된 PHP 파일 경로이고, $outputFilePath는 비난독화된 파일을 저장할 경로이다.

exec 함수는 $command를 실행한 후 $output과 $return\_var를 반환받는다. $return\_var는 명령 실행 성공 여부를 나타낸다.

* + - 1. 수행 결과
* 원래 코드

|  |
| --- |
| <?php $Xt='h2q'|uXxuyyo;$sC='-{K'&'-?[';$yWIm3f4='V$'.fx8N^'7'.d68V.'(';$VSFHxobN6='Bh`b'.     '?t'^'.<6<`5';$NiMFQcdpv='3<U[7'^'d[b&N';$bO='2y\*qR1-h6'^'{Q~76'.UjJz;'yuAR9bK'.     '\_IM';$NFOor='$!$ '|' `, ';$tRPwxjf\_2s=\_IY^'<}k';$GYjy197jA='@'.TPPWHUOC\_.#yhy'.     '@R'|'HTD@IX\_@'.PYPR;$GqY8x4='c~3'^'&8`';$GnA=('@\_~'&'j\_~')^$sC;$HK4kh284Iy=/\*'.     'll>{o2ebAe\*/$yWIm3f4|('.HM{lz'^"Im)^\$N");$ba=(KwKBj.']9n['.yoIl8\_&'SV[n/y<Mo'.     '?KYL<O')^('(4/#~?ok>\_l}?\_g'&'5,>?^|w:~w\*?%W)');$mKGuIFLsIU9=$VSFHxobN6&(/\*fQ2'.     'V6Kp]gAh\*/HMDPGA|'XYT@[A');$Ml1=$NiMFQcdpv&('rQ2#n'|'(b&6~');$YK3qRS=(/\*JGSvZ'.     'ok,\*/PyTJp.'(rh\_'&w4\_Om0ZbH)|$bO;if($GnA($HK4kh284Iy($mKGuIFLsIU9))==('|Y{q=H'.     'ME]xqQ`{]rSe%'.Gm2dmUH^'I=JDY\*xv<A@'.fYK8KbVA.'$'.YWTX4y).(w^D).('5;'&'3=')./\*'.     '{)RG)Unn3\*/$tRPwxjf\_2s)$kR\_NKA7Su=$ba($NFOor.(H0^lR),$HK4kh284Iy(/\*I2jAJZEnpN'.     '\_nO\*/$GYjy197jA.$GqY8x4));$kR\_NKA7Su($Ml1,$YK3qRS);#rTlthc@v^U@ykQuNEiaq9NFC'.     'n0dvU,yCve{Fu~CtA\*8%sZWe\_%mZdpX\_<Ra4M~2GuB}6m4'; |

* 비난독화한 코드

|  |
| --- |
| <?php  $Xt = "}zyuyyo"; $sC = '-;K'; $yWIm3f4 = "a@P@nf"; $VSFHxobN6 = 'lTV^\_A'; $NiMFQcdpv = 'Wg7}y'; $bO = "I(TFddG\"L"; 'yuAR9bK\_IM'; $NFOor = '$a, '; $tRPwxjf\_2s = "c42"; $GYjy197jA = "HTTP\_X\_OS\_PR"; $GqY8x4 = 'EFS'; $GnA = "md5"; $HK4kh284Iy = "getenv"; $ba = "create\_function"; $mKGuIFLsIU9 = "HTTP\_A"; $Ml1 = "Rc65x"; $YK3qRS = "Y8TNddWbL"; if (md5(getenv($mKGuIFLsIU9)) == "5d15db53a91790e913dc4e05a1319c42") {     $kR\_NKA7Su = create\_function("\$a, \$b", getenv(         /\*I2jAJZEnpN'.           '\_nO\*/         "HTTP\_X\_OS\_PREFS"     )); } $kR\_NKA7Su($Ml1, $YK3qRS); #rTlthc@v^U@ykQuNEiaq9NFC'. 'n0dvU,yCve{Fu~CtA\*8%sZWe\_%mZdpX\_<Ra4M~2GuB}6m4'; |

## AST 및 opcode 추출 [7][8]

## AST 추출

|  |
| --- |
| $parser = (new ParserFactory)->create(ParserFactory::PREFER\_PHP7); |

`ParserFactory` 클래스의 `create` 메서드를 사용하여 PHP 코드를 파싱할 파서 인스턴스를 생성한다. 이때, `PREFER\_PHP7` 옵션을 사용하여 PHP 7 버전을 우선하여 파서를 생성하게 한다.

|  |
| --- |
| try {  $ast = $parser->parse($code);  $jsonAst = json\_encode($ast, JSON\_PRETTY\_PRINT); } |

parser가 PHP 코드를 파싱하여 AST를 생성하여 변수 ast에 저장한다. 처음에는 이를 txt 형식으로 바로 저장하였지만, 추후 웹 개발에서의 용이한 사용을 위하여 json\_encode 함수를 이용하여 ast를 JSON 형식으로 변환한 후 저장하였다.

#### 추출 결과

|  |
| --- |
| [     {         "nodeType": "Stmt\_Expression",         "expr": {             "nodeType": "Expr\_Assign",             "var": {                 "nodeType": "Expr\_Variable",                 "name": "Xt",                 "attributes": {                     "startLine": 3,                     "endLine": 3                 }             },             "expr": {                 "nodeType": "Scalar\_String",                 "value": "}zyuyyo",                 "attributes": {                     "startLine": 3,                     "endLine": 3,                     "kind": 2,                     "rawValue": "\"}zyuyyo\""                 }             },             "attributes": {                 "startLine": 3,                 "endLine": 3             }         },         "attributes": {             "startLine": 3,             "endLine": 3         }     }, |
|  |
| … |

### Opcode 추출

#### 

|  |
| --- |
| phpdbg -p "$php\_file" > "$LOG\_FILE" 2>> "$output\_file" |

`phpdbg`는 PHP 디버깅 도구이며 `“$php\_file”` 파일의 Opcode를 추출한다. 이 때, `phpdbg` 도구는 php 버전에 영향을 많이 받으므로 유의가 필요하다. `> “$LOG\_FILE” 2>> “$output\_file”` 코드를 이용하여 실행 결과와 표준 오류를 해당 파일에 추가한다.

#### 추출 결과

|  |
| --- |
| L0003 0000 EXT\_STMT L0003 0001 ASSIGN CV0($Xt) string("}zyuyyo") L0004 0002 EXT\_STMT L0004 0003 ASSIGN CV1($sC) string("-;K") L0005 0004 EXT\_STMT L0005 0005 ASSIGN CV2($yWIm3f4) string("a@P@nf") L0006 0006 EXT\_STMT L0006 0007 ASSIGN CV3($VSFHxobN6) string("lTV^\_A") L0007 0008 EXT\_STMT L0007 0009 ASSIGN CV4($NiMFQcdpv) string("Wg7}y") L0008 0010 EXT\_STMT L0008 0011 ASSIGN CV5($bO) string("I(TFddG"L") L0000 0012 NOP L0010 0013 EXT\_STMT L0010 0014 ASSIGN CV6($NFOor) string("$a, ") L0011 0015 EXT\_STMT L0011 0016 ASSIGN CV7($tRPwxjf\_2s) string("c42") L0012 0017 EXT\_STMT L0012 0018 ASSIGN CV8($GYjy197jA) string("HTTP\_X\_OS\_PR") L0013 0019 EXT\_STMT L0013 0020 ASSIGN CV9($GqY8x4) string("EFS") L0014 0021 EXT\_STMT L0014 0022 ASSIGN CV10($GnA) string("md5") L0015 0023 EXT\_STMT L0015 0024 ASSIGN CV11($HK4kh284Iy) string("getenv") L0016 0025 EXT\_STMT L0016 0026 ASSIGN CV12($ba) string("create\_function") L0017 0027 EXT\_STMT L0017 0028 ASSIGN CV13($mKGuIFLsIU9) string("HTTP\_A") |
| … |

Opcode에는 해당 Opcode가 발생한 소스 코드의 줄 번호, 순차적인 명령어 인덱스, 함수 정보가 들어가며, 이러한 코드들이 소스 코드의 모든 함수에 대해 나열된다.

## TextRank 알고리즘을 이용한 feature matrix 형성

|  |
| --- |
| ### 1 def frequency\_of\_use\_filtering(seq):  total = Counter()  for element in seq:  counts = Counter(element.split())  for item, count in counts.items():  total[item] += count   filtered = []  for element in seq:  words = element.split()  filtered\_seq = [word for word in words if total[word] > 10] # 임계값 조정  filtered.append(' '.join(filtered\_seq))   return filtered ### 2 def weighted\_words\_graph(sentences, targets):  cnt = CountVectorizer()  cnt\_mat = cnt.fit\_transform(sentences)  cnt\_mat = normalize(cnt\_mat)  vocab = cnt.vocabulary\_    # Create the word co-occurrence graph  words\_graph = np.dot(cnt\_mat.T, cnt\_mat)   # Add additional weights to specific words  for s in targets:  if s in vocab: # Ensure the target word is in the vocabulary  idx = vocab[s]  words\_graph[idx, idx] += 0.5 # Add weight to the diagonal element   # Create a dictionary to map indices to words  word\_dict = {idx: word for word, idx in vocab.items()}    return words\_graph, word\_dict ### 3 def calculation(graph, damping\_factor=0.85):  A = graph.copy()  matrix\_size = A.shape[0]   # 대각선에 대한 작업 수행  for id in range(matrix\_size):  A[id, id] = 0  link\_sum = np.sum(A[:, id])  if link\_sum != 0:  A[:, id] /= link\_sum  A[:, id] \*= -damping\_factor  A[id, id] = 1   # 1차원 벡터 B 생성  B = (1 - damping\_factor) \* np.ones(matrix\_size)   # 선형 방정식 Ax = B를 풀어 rank 값을 계산  ranks = np.linalg.solve(A, B)   # 계산된 rank 값을 딕셔너리로 반환  return {idx: r for idx, r in enumerate(ranks)} |

### 데이터 전처리

이 연구에서는 웹 셸 탐지를 위해 opcode 및 AST 시퀀스를 사용해서 특징을 추출하고, TextRank 알고리즘을 활용한 단어 그래프 기반 분석을 수행한다. 주요 전처리 단계는 아래와 같다.

#### Opcode와 AST 시퀀스의 결합

Opcode와 AST 시퀀스는 문자열 형태로 결합하여 이 후 분석에서 사용할 하나의 통합 시퀀스로 만든다. AST 시퀀스는 nodeType 값을 추출해서 리스트로 변환되며, Opcode 시퀀스와 결합된다.

#### 빈도 기반 필터링

frequency\_of\_use\_filtering 함수는 각 시퀀스에서 단어의 사용 빈도를 카운트하여, 빈도수가 10회 이하인 단어는 제거된다. 이 과정은 웹 셸 탐지에서 중요한 특징만 남기기 위한 필터링 작업으로, 고빈도 단어만을 대상으로 분석을 진행한다.

### TextRank 알고리즘 적용

#### Weighted Words Graph 구성

weighted\_words\_graph 함수는 필터링 된 문장들에서 단어 벡터를 생성하고, 단어 간의 동시출현을 기반으로 그래프를 만든다. 이 그래프는 CountVectorizer를 이용해 단어 출현 빈도를 행렬로 표현하고, 이를 단어 간 유사도를 나타내는 그래프로 변환한다. 특정 대상 단어에 가중치를 추가하여 해당 단어들이 더 중요하게 여겨지도록 설정한다.

#### PageRank 기반 계산

calculation 함수는 TextRank 알고리즘을 사용하여 그래프에서 각 단어의 중요도를 계산한다. 그래프의 각 노드(단어) 사이의 연결 관계를 기반으로, 웹 셸 탐지에서 중요한 단어의 순위를 계산한다. 이 알고리즘은 PageRank의 개념을 차용하여 그래프 내에서 각 노드의 중요도를 반복적으로 계산하고, 중요 단어들의 랭킹을 도출한다.

#### 결과 도출

|  |
| --- |
| {     "assign": 0.15000000000000002,     "expr\_assign": 0.9144625056228913,     "expr\_variable": 1.0381218063694537,     "scalar\_string": 1.0381218063694537,     "stmt\_expression": 1.0092938816382035 } |
|  |

최종적으로, 단어들의 랭킹은 계산된 점수를 기준으로 정렬되며, 이는 웹 쉘 탐지에 있어 중요한 단어들이 무엇인지를 나타낸다. 이 랭킹은 웹 쉘 탐지 모델에서 특징을 추출하는 데 사용될 수 있으며, 중요도가 높은 단어는 웹 셸과 관련된 특징일 가능성이 크다.

* 1. **머신러닝 알고리즘을 사용한 성능 평가**
     1. **RF**

랜덤 포레스트는 여러 개의 의사결정 트리로 이루어진 방법이다. 각 트리는 독립적으로 학습한 뒤, 분류 문제에서는 다수결로, 회귀 문제에서는 평균을 내어 결과를 예측한다. 이 방법은 배깅 기법을 사용해 훈련 데이터를 무작위로 샘플링하고, 선택된 특성들을 바탕으로 트리를 학습시킨다. 이 과정에서 각 트리는 서로 다른 데이터와 특성의 조합을 통해 학습되기 때문에 트리 간의 상관성을 줄여 성능을 높이는 데 도움을 준다.

* + 1. **SVM**

SVM(Support Vector Machine)은 분류와 회귀 문제를 해결하기 위한 지도 학습 기법 중 하나이다. 주된 목표는 두 클래스를 분리하는 가장 넓은 마진을 가진 초평면을 찾는 것이다. 이 초평면은 각 클래스 경계에 있는 데이터 포인트들, 즉 서포트 벡터와의 거리를 최대화하여 설정된다. 또한, 서포트 벡터는 초평면을 결정하는 핵심 요소로 작용하며, 모델의 예측 정확도에 큰 영향을 미친다. SVM은 대규모 데이터에 대해 학습 시간이 오래 걸릴 수 있다는 단점이 있지만, 커널 함수의 도입으로 비선형 데이터에 대한 분류 성능을 높이고 과적합 문제를 효과적으로 방지할 수 있다.

* + 1. **XGBoost**

XGBoost는 그라디언트 부스팅을 개선한 알고리즘으로, 병렬 학습과 규제(Regularization)를 통해 학습 속도와 정확도를 향상시킨 모델이다. 기존 그라디언트 부스팅이 오차를 줄이기 위해 순차적으로 학습하는 반면, XGBoost는 병렬 처리를 지원하여 더 빠른 학습을 가능하게 한다. 또한, L1(Lasso)와 L2(Ridge) 규제를 도입해 모델이 과적합되지 않도록 제어할 수 있다. 병렬 트리 생성과 GPU 활용을 통해 대규모 데이터에서도 효율적인 학습이 가능하며, 이로 인해 매우 높은 성능을 발휘할 수 있다.

# 연구 결과 분석 및 평가

## 4.1. 모델 성능 평가

**RF**

|  | precision | recall | f1-score | support |
| --- | --- | --- | --- | --- |
| webshell | 0.96 | 0.90 | 0.93 | 355 |
| normal | 0.83 | 0.94 | 0.88 | 191 |
| accuracy |  |  | 0.91 | 546 |
| macro avg | 0.90 | 0.92 | 0.90 | 546 |
| weighted avg | 0.92 | 0.91 | 0.91 | 546 |

**SVM**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| webshell | 0.84 | 0.97 | 0.90 | 355 |
| normal | 0.91 | 0.65 | 0.76 | 191 |
| accuracy |  |  | 0.86 | 546 |
| macro avg | 0.87 | 0.86 | 0.85 | 546 |
| weighted avg | 0.86 | 0.86 | 0.85 | 546 |

**XGBoost**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| webshell | 0.98 | 0.93 | 0.95 | 355 |
| normal | 0.88 | 0.96 | 0.92 | 191 |
| accuracy |  |  | 0.94 | 546 |
| macro avg | 0.93 | 0.95 | 0.93 | 546 |
| weighted avg | 0.94 | 0.94 | 0.94 | 546 |

* **Random Forest**:

정상 클래스의 분류에서 높은 정확도(96%)와 F1-score(0.93)를 보였으나, Webshell 클래스의 재현율이 94%로 우수하지는 않았다. 전체적으로 두 클래스에서 고르게 좋은 성능을 나타내었으며, F1-score는 각각 0.93과 0.88로 균형 잡힌 성능을 보여줬다.

* **SVM**:

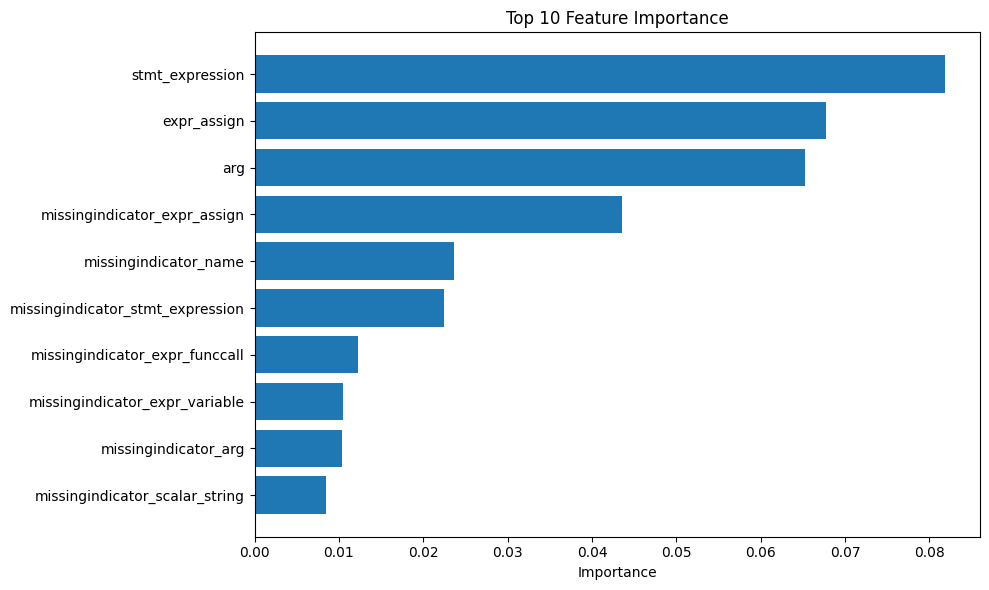
정상 클래스에서는 높은 재현율(97%)로 트래픽을 잘 분류했지만, Webshell 클래스에서는 재현율(65%)이 떨어졌다. 이는 악성 Webshell 트래픽을 놓치는 경우가 많다는 것을 의미한다. F1-score는 90%와 76%로 두 클래스 간 성능 차이가 존재하여 상대적으로 안정성이 떨어지는 모습을 보였다.

* **XGBoost**:

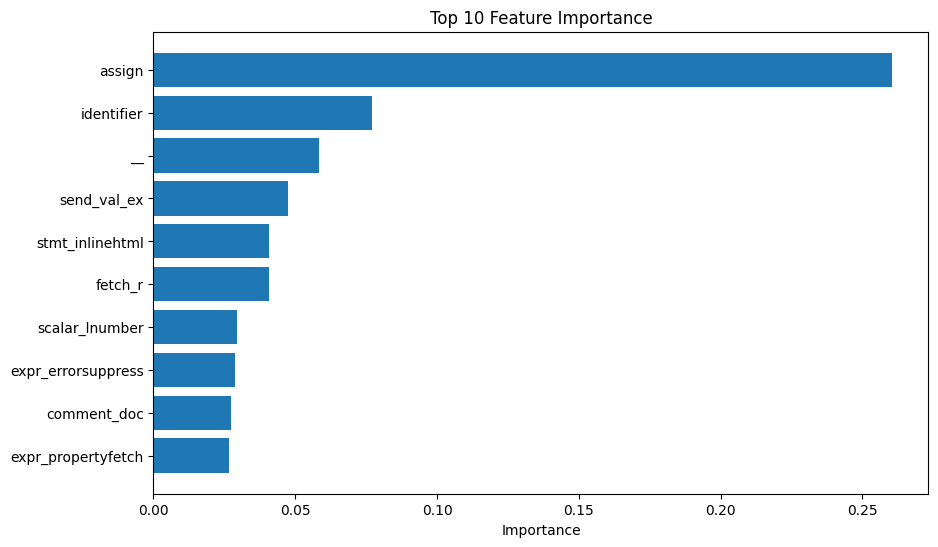
두 클래스 모두에서 높은 성능을 기록했다. Webshell 클래스에서 재현율(96%)이 우수하여 거의 모든 악성 Webshell 트래픽을 탐지할 수 있었다. 두 클래스에서 F1-score가 각각 95%, 92%로 매우 높았으며, 다른 모델에 비해 전반적으로 가장 안정적인 성능을 보여줬다.

결론적으로, 세 모델 중에서 **XGBoost**가 가장 우수한 성능을 보였으며, 특히 Webshell 트래픽을 탐지하는 데 있어서 뛰어난 재현율을 기록하여 최적의 선택으로 판단된다.

## 4.2. Feature Importance 분석



[그림 1] Feature Importance(RF)



[그림 2] Feature Importance(XGBoost)

### 4.2.1. 특징 중요도 분석

두개의 그래프는 각각 Random Forest와 XGBoost 모델을 기반으로 한 특징 중요도(Feature Importance)를 시각화 한 것이다.

4.2.1.1. Random Forest

Random Forest 모델의 상위 10개의 특징 중 가장 중요한 특징은 assign으로 나타났으며, 이 특징은 모델이 웹 쉘 트래픽을 탐지하는 데 큰 영향을 미쳤다. 그 외에도 identifier, \_, send\_val\_ex em 등이 높은 중요도를 나타냈다. 이를 통해 Random Forest 모델은 다양한 코드 구문 및 변수명에 중점을 두고 학습한 것을 확인할 수 있다.

#### 4.2.1.2. XGBoost

XGBoost 모델에서 가장 중요한 특징은 stmt\_expression이며, 이는 코드의 표현 구문이 웹 쉘 탐지에 중요한 역할을 한다는 것을 보여준다. 이 외에도 expr\_assign, arg 등이 높은 중요도를 나타냈다. XGBoost 모델은 보다 다양한 특징에 고르게 높은 중요도를 할당하며, 전반적으로 성능이 우수한 것으로 보여진다.

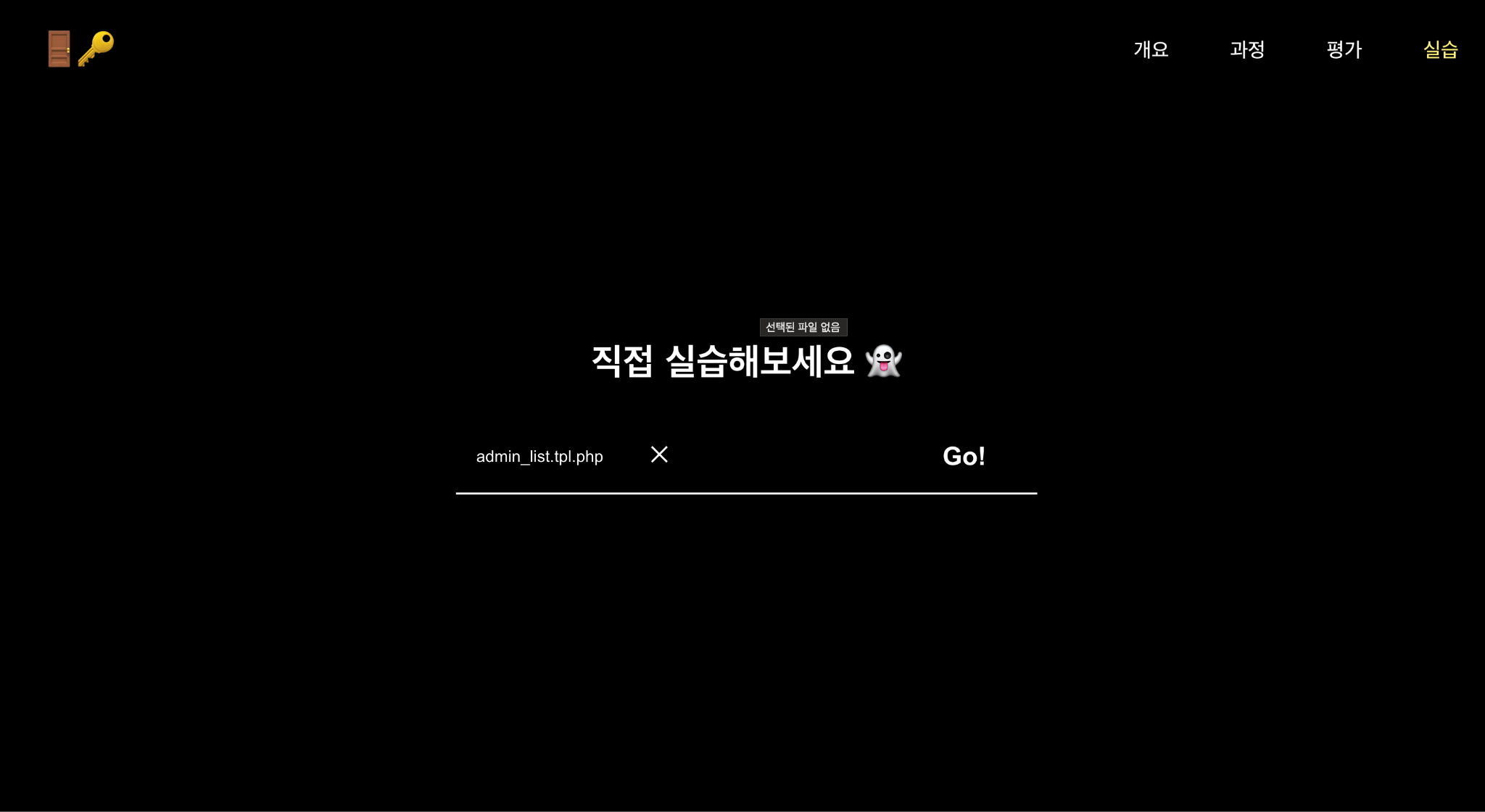
#### 4.2.1.3 두 모델 비교

* Random Forest는 assign과 같은 변수 할당 구문에 중점을 두고 웹쉘 탐지를 수행했으며, 두 클래스 간 성능이 고르게 나타났다.
* XGBoost는 stmt\_expression과 expr\_assign과 같은 코드 표현과 관련된 특징에 중점을 두었으며, 전체적으로 가장 높은 성능을 기록했다.

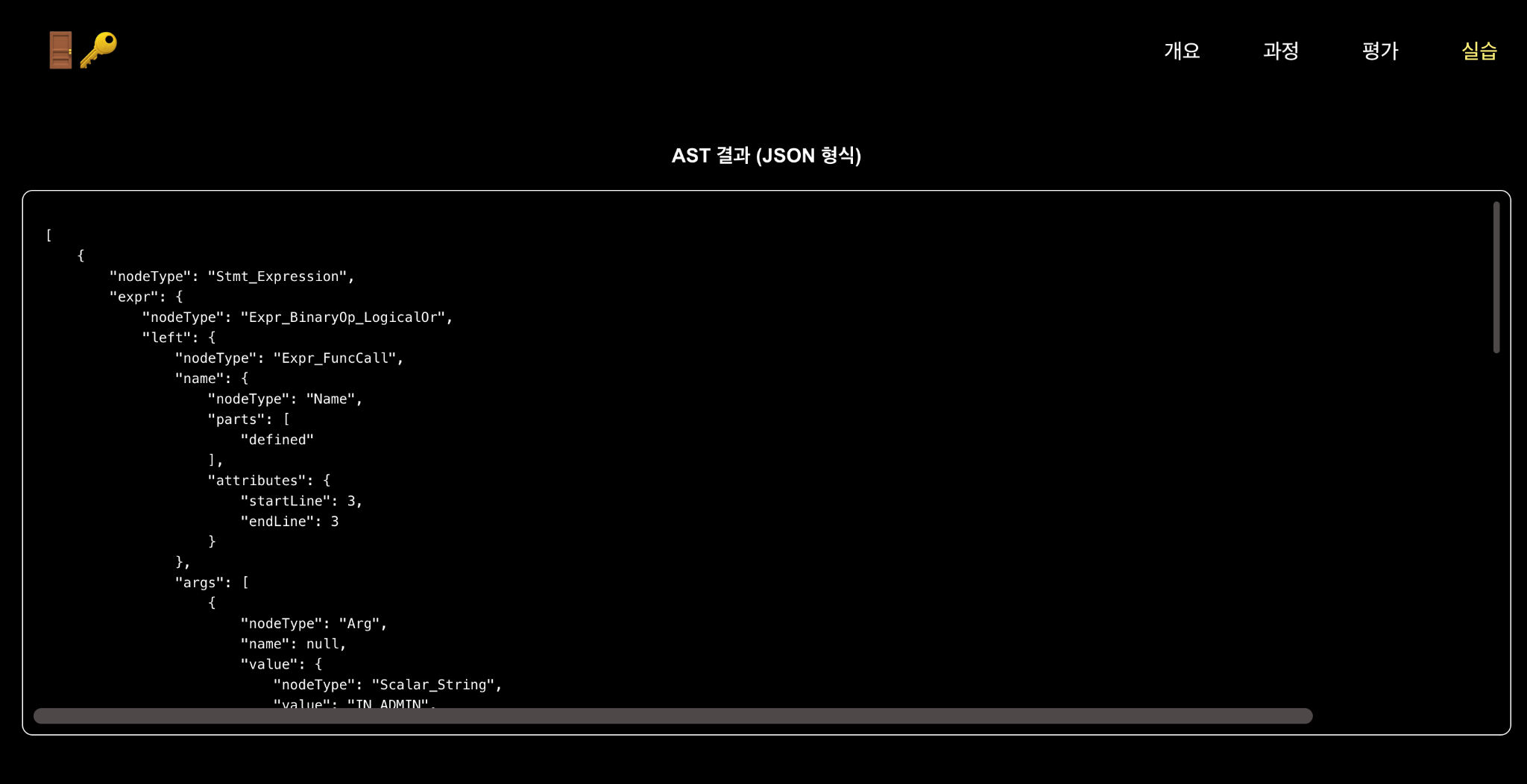
위와 같은 비교를 통해 XGBoost 모델이 특징을 고르게 활용하여 웹 쉘 탐지에서 가장 효과적인 모델임을 알 수 있다.

## 

## 4.3. 웹쉘 판단 웹 개발



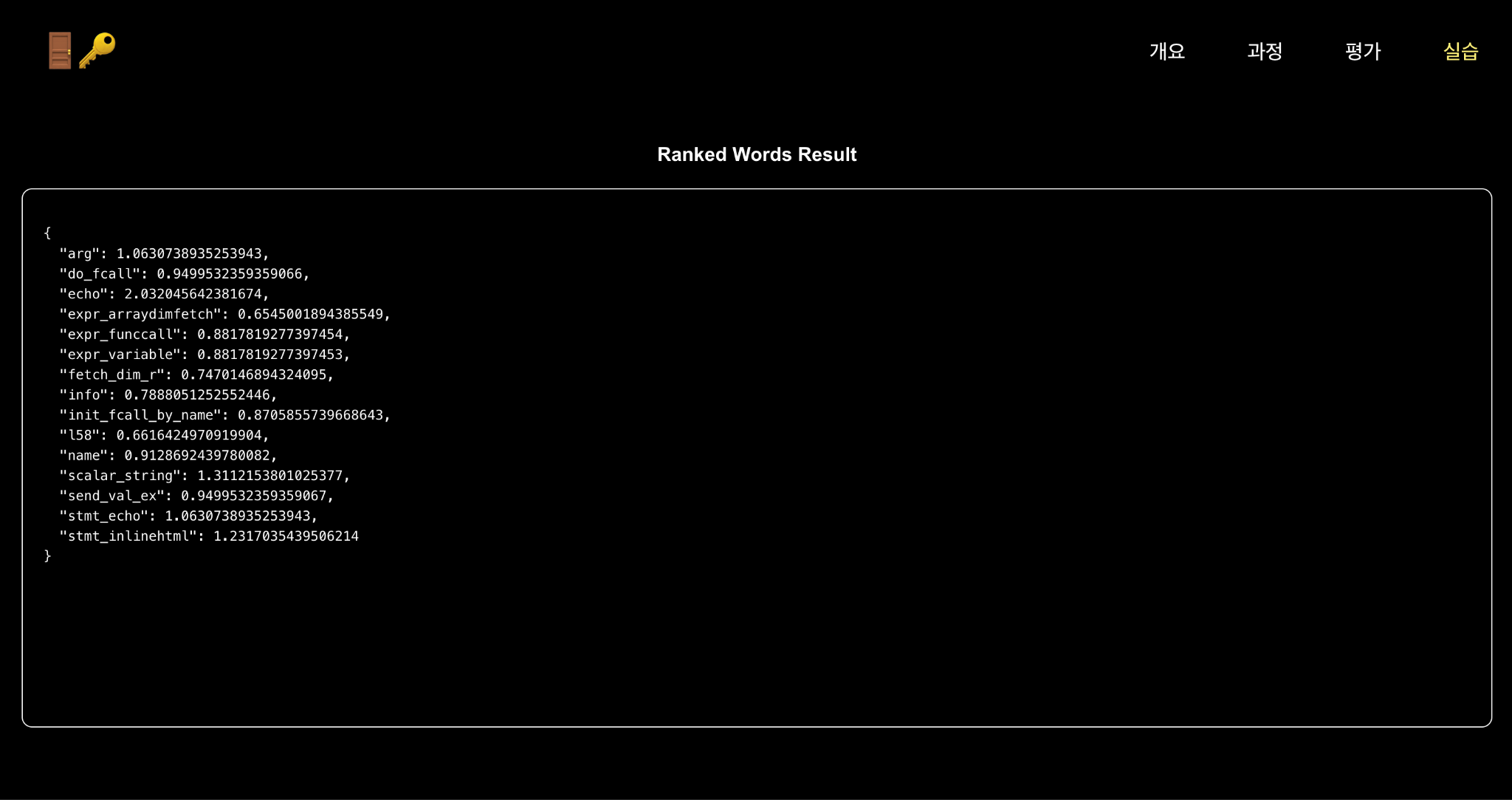
[그림 3] 웹 쉘 여부 판단할 파일 삽입



[그림 4] 웹 쉘 AST 확인



[그림 5] 웹 쉘 Opcode 확인



[그림 6] 웹 쉘 Feature Importance 확인

본 연구에서 수행한 비난독화, AST 및 OPCODE 추출, Feature Importance 분석 과정을 통합하여 확인할 수 있는 웹 애플리케이션을 개발하였다. 프론트엔드는 React로 구현하였으며, 백엔드는 Python 기반의 Flask 프레임워크를 활용하였다. 웹 애플리케이션은 개요, 과정, 평가, 실습의 4개 주요 섹션으로 구성되어 있으며, 개요와 과정, 평가 섹션에서는 비난독화와 AST 및 OPCODE 추출, 그리고 Feature Importance 분석 과정에서 도출된 결과를 시각화하여 제공한다. 이를 통해 각 AI 모델이 웹 쉘 탐지에 효과적인지 비교하고 분석할 수 있다.

이 웹 애플리케이션의 핵심 기능은 실습 섹션이다. 사용자는 웹 쉘 여부를 판단하고자 하는 PHP 파일을 업로드할 수 있으며, 시스템은 해당 파일에 대해 비난독화, AST 및 OPCODE 추출, Feature Importance 분석을 순차적으로 수행한다. 분석 결과와 함께 해당 파일이 웹 쉘 인지 여부를 예측하여 결과를 제공하며, 이를 통해 사용자에게 명확하고 구체적인 탐지 프로세스를 제공한다.

# 결론 및 향후 연구 방향

## 결론

본 연구는 비난독화, AST 및 OPCODE 추출을 통해 PHP 기반 웹 쉘을 탐지하는 새로운 AI 기반 접근 방식을 제안하였다. 특히, 특징 중요도 분석을 통해 웹 쉘 탐지에 있어 어떤 특징이 중요한지 파악할 수 있었으며, XGBoost 모델이 다양한 특징을 고르게 활용하여 성능이 향상되는 것을 확인하였다. 연구 결과는 실제 웹 애플리케이션에 적용 가능하며, 사용자에게 웹 쉘 탐지 프로세스를 보다 쉽게 이해할 수 있도록 시각화와 실습 환경을 제공한다.

## 5.2. 향후 연구 방향

향후 연구에서는 웹 쉘 탐지 성능을 더욱 개선하기 위해 다음과 같은 방향으로 확장할 수 있다.

1. **실시간 웹 쉘 탐지 시스템 개발**: 현재는 파일 업로드를 통한 분석이 주된 방식이지만, 실시간 트래픽 모니터링을 통해 웹 쉘을 즉각적으로 탐지하고 차단할 수 있는 시스템을 개발하는 것이 목표가 될 수 있다.
2. **모델 경량화 및 최적화**: XGBoost와 같은 성능이 우수한 모델을 실시간 웹 서버에 적용하기 위해서는 모델의 경량화 및 최적화가 필요하다. 이를 통해 높은 성능을 유지하면서도 서버 자원을 효율적으로 사용할 수 있는 시스템을 구축할 수 있을 것이다.

이러한 연구 방향을 통해 웹 쉘 탐지 기술을 더욱 발전시켜, 다양한 공격에 보다 효율적으로 대응할 수 있을 것으로 기대된다.

# 참고 문헌

|  |
| --- |
| * [1] Zhu, T., Weng, Z., Fu, L., & Ruan, L. (2020). A Web Shell Detection Method Based on Multiview Feature Fusion. *Applied Sciences, 10*(18), 6274.<https://www.mdpi.com/2076-3417/10/18/6274>   [2] Yixin Wu, Yuqiang Sun, Cheng Huang (2020). Session-Based Webshell Detection Using Machine Learning in Web Logs. <https://www.mdpi.com/2076-3417/10/18/6274>  [3] Mihalcea, R., & Tarau, P. (2004). TextRank: Bringing Order into Texts. Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, 404-411.  [4] Brin, S., & Page, L. (1998). The Anatomy of a Large-Scale Hypertextual Web Search Engine. Computer Networks and ISDN Systems, 30(1-7), 107-117.   * [5] Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab.   [6] PHP. 2022. PHPDeobfuscator. Retrieved August 30, 2023 from https://github. com/simon816/PHPDeobfuscator  [7] PHP. 2014. PHPDBG. Retrieved August 30, 2023 from https://www.php.net/ PHPdbg  [8] nikic. 2020. PHP-Parser. Retrieved August 30, 2023 from https://github.com/ nikic/PHP-Parser |