INTERNAL Information System Journal
INTERNAL (Information System Journal)
Volume 8 No. 1 | Juni 2025 : 1-9
DOI: 10.32627
http://jurnal.masoemuniveristy.ac.id/index.php/internal
p-ISSN: 2621-9433
e-ISSN: 2656-0259

Deteksi Kerentanan Kode PHP Menggunakan TF-IDF, AST Parsing, dan Random Forest

Mohamad Erdda Habiby
Teknik Informatika, Universitas Bina Nusantara, Indonesia
moh.erdda@binus.ac.id

Info Artikel

Sejarah artikel:
Diterima Mei 2025
Direvisi Juni 2025
Disetujui Juni 2025
Diterbitkan Juni 2025

ABSTRACT

Vulnerabilities in PHP programming code remain one of the primary threats to the security of web applications, especially in the absence of automated analysis mechanisms during development. This study presents a hybrid system designed to detect vulnerabilities in PHP code in real time, utilizing a combination of Term Frequency-Inverse Document Frequency (TF-IDF) and Abstract Syntax Tree (AST) parsing techniques, along with the Random Forest classification algorithm. The process begins with preprocessing, cleaning, and tokenization, followed by feature extraction using TF-IDF. The next stage involves AST-based parsing to identify potentially dangerous syntax, such as the use of eval() or include() functions with parameters derived from user input. Both feature sets are combined and used to train a Random Forest model on a labeled dataset that distinguishes between vulnerable and secure code. Reliability testing was conducted on several PHP files from existing applications. The results show that the system is capable of identifying vulnerabilities such as File Inclusion and SQL Injection with confidence scores ranging from 52% to 61%, regardless of the structure or complexity of the code analyzed. Although the confidence level is not yet optimal, the system has demonstrated effectiveness in providing early warnings of potential threats. In the future, this system can be further developed as a simulation tool for static code analysis to support improved secure coding practices in software development.

Keywords **:** *Abstract Syntax Tree; Frequency-Inverse Document Frequency; Random Forest; Static Code Analysis.*

ABSTRAK

Kerentanan dalam kode pemrograman PHP merupakan salah satu bahaya utama bagi keamanan aplikasi *web*, terutama tanpa adanya mekanisme analisis otomatis dalam tahap pengembangannya. Penelitian ini mengembangkan sistem hibrida yang berfungsi untuk mendeteksi kerentanan pada kode PHP secara real-time, menggunakan pendekatan *Frequency-Inverse Document Frequency* (TF-IDF) dan teknik Parsing *Abstract Syntax Tree* (AST) dengan algoritma klasifikasi *Random Forest*. Proses dimulai dengan pre-processing, pembersihan, dan tokenisasi, lalu dilanjutkan dengan ekstraksi data menggunakan fitur TF-IDF. Proses berikutnya adalah parsing dengan metode AST untuk mendapatkan sintaks yang berpotensi berbahaya, seperti penggunaan fungsi eval() atau *include*() dengan parameter yang berasal dari input. Kedua metode ini digabungkan dan digunakan untuk melatih model *Random Forest* berdasarkan dataset yang telah diklasifikasikan menjadi kerentanan yang aman dan yang tidak aman. Pengujian keandalan dilakukan terhadap beberapa file PHP dari aplikasi yang ada. Hasil pengujian menunjukkan bahwa sistem mampu mengidentifikasi kerentanan *File* Inclusion dan SQL Injection dengan skor kepercayaan yang cukup baik, yakni antara 52% hingga 61%, tanpa terpengaruh oleh struktur dan kompleksitas kode yang dianalisis. Meskipun tingkat kepercayaan masih belum sepenuhnya memuaskan, sistem telah terbukti efektif dalam memberikan deteksi awal terhadap

potensi ancaman. Selanjutnya, sistem ini dapat dikembangkan untuk berfungsi sebagai simulasi dalam analisis statis yang berkaitan dengan pembaruan praktik pengkodean yang dilakukan dalam perangkat lunak.

Kata Kunci : *Abstract Syntax Tree; Frequency-Inverse Document Frequency; Random Forest; Static Code Analysis.*

PENDAHULUAN

Aplikasi *web* saat ini telah menjadi tulang punggung utama dalam banyak aspek kehidupan digital, mulai dari layanan publik, transaksi *e-commerce*, hingga sistem pendidikan. Dibalik kenyamanan dan kemudahan akses tersebut, ada persoalan mendasar yang kerap diabaikan, yaitu aspek keamanan kode sumber, khususnya dalam pengembangan aplikasi berbasis PHP. Bahasa ini memang populer karena fleksibilitas dan kemudahan penggunaannya, namun juga dikenal memiliki banyak potensi kerentanan apabila tidak ditulis dengan hati-hati [1].

Beberapa jenis kerentanan yang sering ditemukan dalam aplikasi PHP di antaranya adalah *SQL Injection, Cross-Site Scripting* (XSS), *Remote Code Execution*, dan *File Inclusion*. Jenis-jenis serangan ini sering dimanfaatkan oleh pihak tidak bertanggung jawab untuk mengakses data sensitif, merusak sistem, bahkan mengambil alih kendali server [2]. Dalam banyak kasus, kerentanan baru terdeteksi setelah sistem sudah berjalan atau lebih buruk lagi, setelah terjadi insiden keamanan. Selama ini, pendekatan tradisional untuk mengidentifikasi kerentanan dilakukan melalui pemeriksaan manual oleh pengembang/programer atau penggunaan alat analisis statis berbasis aturan. Meskipun berguna, pendekatan ini bersifat reaktif, tidak adaptif terhadap pola baru, dan kurang efisien untuk skala kode yang besar [3]. Oleh karena itu, dibutuhkan pendekatan yang lebih cerdas dan otomatis, seperti yang ditawarkan oleh kombinasi *Natural Language Processing* (NLP) dan *Machine Learning* (ML) dalam konteks deteksi kerentanan.

Penelitian-penelitian sebelumnya telah menunjukkan efektivitas teknik TF-IDF (*Term Frequency-Inverse Document Frequency*) dalam mengekstrak fitur dari teks kode program untuk mendeteksi pola berbahaya [4]. Di sisi lain, *Abstract Syntax Tree* (AST) Parsing memungkinkan sistem untuk menganalisis struktur dan logika dari kode secara lebih kontekstual, seperti mendeteksi penggunaan fungsi eval() atau include() yang melibatkan input dari pengguna [5].

Berdasarkan hal tersebut, penelitian ini mengembangkan sistem deteksi otomatis kerentanan pada kode PHP berbasis *web* dengan menggabungkan pendekatan TF-IDF dan AST Parsing, serta menggunakan algoritma *Random Forest* sebagai metode klasifikasi. Sistem ini dirancang untuk melakukan deteksi secara *realtime*, sekaligus memberikan informasi berupa *confidence score* untuk membantu pengembang dalam memahami tingkat risiko yang ada. Dengan adanya sistem ini, diharapkan proses *secure coding* dapat dilakukan lebih awal dan menyatu dalam siklus pengembangan perangkat lunak.

METODE

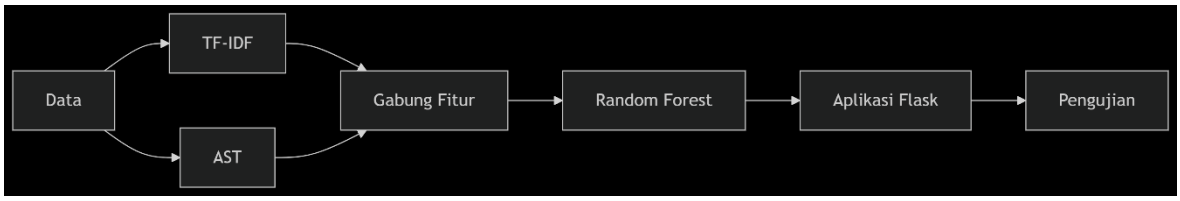
Penelitian ini menggunakan pendekatan eksperimental dengan model pengembangan sistem berbasis *prototyping*. Metode prototyping telah banyak

p-ISSN: 2621-9433
e-ISSN: 2656-0259

digunakan dalam pengembangan sistem berbasis PHP untuk menghasilkan solusi yang lebih efisien dan mudah diadaptasi [9]. Selain itu tujuan dari pendekatan ini adalah untuk mengembangkan dan menguji sebuah sistem deteksi kerentanan kode PHP yang mampu bekerja secara otomatis dan *real-time*. Fokus utama terletak pada penerapan kombinasi metode analisis teks (TF-IDF) dan analisis struktur sintaks (AST Parsing) untuk menghasilkan fitur yang dapat digunakan dalam klasifikasi kerentanan menggunakan algoritma *Random Forest*. Pendekatan ini dipilih karena mampu menangani kebutuhan sistem yang dinamis, serta dapat diuji dan disesuaikan secara iteratif berdasarkan hasil eksperimen pada data nyata [1].

Tahapan Penelitian

Penelitian ini dilakukan melalui beberapa tahapan sebagai berikut:



Gambar 1. Tahapan penelitian

1. Pengumpulan dan Persiapan Data
   Data yang digunakan berasal dari dua sumber utama, yaitu:
   a. Dataset publik dari GitHub: vulnerable-php-code-examples, yang berisi file PHP rentan.
   b. Dataset tambahan yang disusun secara manual (dummy dataset), dirancang agar mencerminkan struktur kode berisiko tinggi seperti eval($_POST), system($_GET), dan include($_GET).
2. *Preprocessing* dan Ekstraksi Fitur TF-IDF
   File PHP yang telah dikumpulkan akan dibersihkan, di-tokenisasi, dan diproses menggunakan metode TF-IDF (*Term Frequency-Inverse Document Frequency*) untuk menghasilkan fitur berbasis kata. Teknik ini banyak digunakan dalam pengolahan teks dan terbukti efektif dalam mendeteksi pola anomali pada kode program [4].
3. Parsing *Abstract Syntax Tree* (AST)
   Selain fitur berbasis teks, sistem juga mengekstrak fitur berbasis struktur menggunakan metode AST Parsing. AST digunakan untuk mengenali struktur kode yang berisiko, seperti pemanggilan fungsi-fungsi berbahaya (*eval, system, include*) yang terkait dengan input. Penelitian Alzahrani dan Cordy [5] menunjukkan bahwa AST sangat efektif untuk deteksi kerentanan yang tidak terlihat secara eksplisit dari token kata. AST Parsing dilakukan untuk mengekstrak struktur sintaksis dari kode PHP yang mengandung pola risiko, seperti pemanggilan fungsi eval() atau include() dengan parameter dari input eksternal seperti `$_GET` atau `$_POST`. Karena PHP tidak memiliki parser AST bawaan untuk Python, maka proses dilakukan dengan pendekatan manual berbasis pemrosesan string dan pencocokan ekspresi reguler menggunakan library Python seperti `re`. Setiap file PHP dipindai untuk mendeteksi:

Fungsi berisiko: `eval, system, include, require`

Parameter input tidak tervalidasi: `$_GET, $_POST, $_REQUEST`

Contohnya, baris `eval($_GET['cmd']);` akan dipetakan ke struktur AST sederhana:

```
{
  "function": "eval",
  "argument": "input",
  "source": "$_GET['cmd']"
}
```

Hasil parsing ini dikonversi ke fitur biner (misal: ast_eval_input = 1) dan digabungkan dengan fitur TF-IDF untuk membentuk vektor akhir sebelum diklasifikasikan oleh algoritma Random Forest.

4. Penggabungan Fitur dan Pelatihan Model Fitur dari TF-IDF dan AST Parsing kemudian digabung menjadi satu vektor fitur. Gabungan ini dimasukkan ke dalam algoritma *Random Forest*, yang dipilih karena kemampuannya dalam menangani data berdimensi tinggi dan memberikan klasifikasi yang stabil serta mudah diinterpretasi [2].
5. Pembuatan Aplikasi
   Untuk pembuatan aplikasi menggunakan bahasa pemrograman *Python* dengan *framework Flask* sebagai *backend*. Antarmuka pengguna dibuat sederhana agar pengguna dapat dengan mudah mengunggah file PHP, melihat hasil prediksi jenis kerentanan, serta *confidence score*-nya. Hasil prediksi juga secara otomatis dicatat ke dalam file log.
6. Pengujian Sistem
   Pengujian dilakukan terhadap sejumlah file .php dari aplikasi nyata untuk menilai kemampuan sistem dalam mengenali jenis kerentanan seperti SQL *Injection* dan *File Inclusion*. Evaluasi difokuskan pada jenis label yang terdeteksi serta nilai confidence score yang diberikan oleh model.

Alat dan Teknologi

Dalam pengembangan sistem pendeteksi kerentanan ini, digunakan berbagai alat dan teknologi yang saling mendukung, baik pada sisi pengolahan data, pemrosesan fitur, pengembangan antarmuka, maupun implementasi algoritma pembelajaran mesin. Berikut ini adalah penjelasan dari alat dan teknologi yang digunakan:

1. Bahasa Pemrograman: *Python*
   Python dipilih karena memiliki ekosistem pustaka yang luas, khususnya dalam bidang *machine learning*, *natural language processing*, dan *web development*. Bahasa ini juga memiliki sintaks yang mudah dipahami dan mendukung pengembangan cepat (*rapid prototyping*) [9].
2. *Framework Web* : Flask
   Flask digunakan sebagai *framework backend* ringan untuk membangun antarmuka web sistem. Dengan *Flask*, pengguna dapat mengunggah file .php melalui browser dan langsung memperoleh hasil prediksi secara real-time. Flask sangat sesuai untuk aplikasi skala kecil hingga menengah karena fleksibilitas dan kemudahannya dalam integrasi dengan pustaka *Python* lainnya [8].

p-ISSN: 2621-9433
e-ISSN: 2656-0259

3. Ekstraksi Teks: TF-IDF
   TF-IDF (*Term Frequency-Inverse Document Frequency*) digunakan untuk mengekstrak fitur dari kode PHP dalam bentuk vektor numerik. Pendekatan ini telah banyak digunakan dalam deteksi konten berbahaya di halaman web [10].
4. Analisis Struktur Kode: AST Parsing
   AST (*Abstract Syntax Tree*) Parsing. Teknik ini terbukti efektif untuk mendeteksi kerentanan yang tidak terlihat secara eksplisit melalui analisis teks biasa [5].
5. Algoritma Pembelajaran Mesin: Random Forest
   Algoritma klasifikasi Random Forest memiliki interpretabilitas yang cukup baik dan telah digunakan secara luas dalam sistem pendeteksi anomali dan keamanan siber [2].

HASIL DAN PEMBAHASAN

Untuk implementasi dan hasil evaluasi dari deteksi kerentanan CVE dalam kode sumber file program PHP yang dikembangkan dengan algoritma *TF-IDF*, *Parsing AST*, dan *Random Forest*, akan dijelaskan melalui beberapa tahapan dibawah ini. Penjelasan mencakup hasil pelatihan model, performa evaluasi, serta interpretasi terhadap hasil prediksi.

Gambaran Umum Sistem

Pada tahap ini, peneliti telah berhasil membangun sebuah aplikasi *web* sederhana yang bisa digunakan untuk mendeteksi kerentanan keamanan pada file PHP secara otomatis. Aplikasi ini dibangun menggunakan *Flask*, dan memanfaatkan algoritma Random Forest untuk memprediksi apakah suatu file mengandung jenis kerentanan tertentu, seperti SQL Injection, XSS, *Remote Code Execution*, dan lainnya. Hal yang menarik dari sistem ini adalah kemampuannya untuk memberikan hasil prediksi beserta tingkat keyakinan model (*confidence score*), serta melakukan analisis berbasis struktur kode menggunakan fitur AST (*Abstract Syntax Tree*). Ini menjadikan sistem tidak hanya sekadar menebak dari kata kunci, tapi juga memahami pola logika kode PHP itu sendiri.

Struktur Aplikasi

Untuk mempermudah pengelolaan dan pemisahan tanggung jawab antar komponen, aplikasi disusun dengan struktur direktori sebagai berikut:

```
php_vuln_detector/
├── app/
│   ├── app.py                ← Logika utama sistem
│   ├── uploads/
│   │   ├── .gitkeep
│   │   └── log.txt           ← Laporan otomatis
│   └── templates/
│       └── index.html        ← Tampilan web upload
```

Gambar 3. Struktur Aplikasi

Penjelasan tiap folder:

1. app.py: Merupakan inti dari aplikasi yang menangani logika pemrosesan data, pelatihan model, dan prediksi jenis kerentanan.
2. uploads/: Folder sementara tempat menyimpan file PHP yang diunggah oleh pengguna untuk dianalisis.
3. .gitkeep: File kosong untuk memastikan folder uploads/ tetap terdeteksi oleh sistem version control seperti Git.
4. templates/index.html: File antarmuka pengguna berbasis HTML yang menampilkan form upload dan hasil prediksi.
5. Log.txt : File ini berisi laporan hasil yang bisa di *download*.

Struktur yang sederhana dan modular ini memudahkan proses pengembangan dan pemeliharaan aplikasi.

Dataset yang Digunakan

Dataset yang digunakan terdiri dari dua sumber utama:

1. Dataset *Real*: Diambil dari proyek open-source di GitHub bernama vulnerable-php-code-examples. Dataset ini berisi potongan kode PHP yang mewakili beberapa jenis kerentanan paling umum, seperti SQLi dan XSS. Walau jumlahnya terbatas, keberadaan kode nyata memberi bobot realistis pada pelatihan model.
2. Dataset Tambahan (*Dummy*): Untuk melengkapi kekurangan variasi dan kuantitas dari data real, peneliti menambahkan data buatan sendiri yang menyerupai pola kerentanan sebenarnya. Misalnya, penggunaan *eval()*, *include()* dengan variabel user input, atau *move_uploaded_file()* tanpa validasi. Semua contoh ini diberi label sesuai jenis serangan yang disimulasikan.

Dengan strategi ini, total dataset terdiri dari sekitar 120 baris kode, yang sudah cukup merepresentasikan pola-pola utama untuk tahap awal pengujian sistem. Lalu untuk Jenis kerentanan yang dicakup antara lain: *SQL Injection*, *Cross Site Scripting (XSS)*, *Command Injection*, *Remote Code Execution (RCE)*, *File Inclusion*, *Unsafe File Upload*, dan kode yang dianggap Aman (*Safe)*

Cara Kerja Aplikasi

Pengguna Pengguna cukup membuka aplikasi lewat browser, mengunggah file .php, lalu sistem akan menganalisis isi file tersebut dan menampilkan:

1. Jenis kerentanan yang terdeteksi
2. Confidence score dalam bentuk persen
3. Opsi untuk mengunduh log hasil prediksi

Secara teknis, prosesnya terdiri dari tiga tahapan utama:

1. File diubah menjadi representasi numerik menggunakan TF-IDF
2. AST parser mengekstrak struktur penting dari kode (seperti pemanggilan eval(), penggunaan $_POST, atau include($_GET))
3. Kedua hasil itu digabung, lalu diproses oleh model Random Forest untuk melakukan prediksi

p-ISSN: 2621-9433
e-ISSN: 2656-0259

Mekanisme *Confidence Score* dan AST *Parsing*

Agar sistem tidak sekadar memberi hasil "Rentan" atau "Aman", maka ditambahkan dua fitur penting:

1. *Confidence score* diperoleh dari metode *predict_proba*() milik model *Random Forest*. Fungsi ini mengembalikan nilai probabilitas dari setiap kemungkinan kelas, dan sistem menampilkan:
   a. Kelas dengan probabilitas tertinggi sebagai hasil prediksi
   b. Nilai probabilitas tersebut dalam bentuk persen sebagai *confidence score*

   Hasil ini kemudian dikirim ke index.html dan ditampilkan secara *user-friendly.*
2. AST Parsing

   AST digunakan untuk memahami struktur kode, misalnya:
   a. Apakah $_GET digunakan di dalam fungsi eval()
   b. Apakah include() menggunakan input dari pengguna
   c. Apakah ada fungsi upload file yang tidak divalidasi

   Dengan fitur ini, prediksi model menjadi lebih berbasis konteks, bukan hanya token teks.

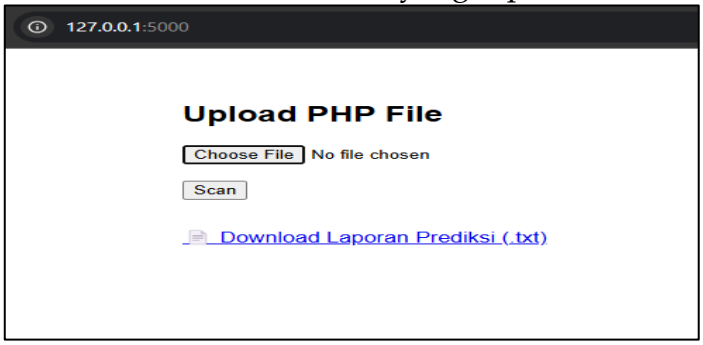
Hasil Pengujian

Pengujian dilakukan dengan cara mengunggah beberapa file .php yang memiliki variasi konten, baik yang mengandung kerentanan maupun yang aman. Berikut beberapa contoh hasil:

```
[2025-04-22 15:22:41] File: index.php → Prediction: File Inclusion, Confidence: 53.00%
[2025-06-16 11:22:02] File: index.php → Prediction: File Inclusion, Confidence: 61.00%
[2025-06-16 11:24:03] File: index.php → Prediction: File Inclusion, Confidence: 52.00%
[2025-06-16 11:24:59] File: admin.php → Prediction: XSS, Confidence: 56.00%
[2025-06-16 11:25:16] File: jurusan.php → Prediction: XSS, Confidence: 54.00%
[2025-06-16 11:25:30] File: tahunajar.php → Prediction: XSS, Confidence: 56.00%
[2025-06-16 11:25:45] File: pengumuman.php → Prediction: XSS, Confidence: 54.00%
```

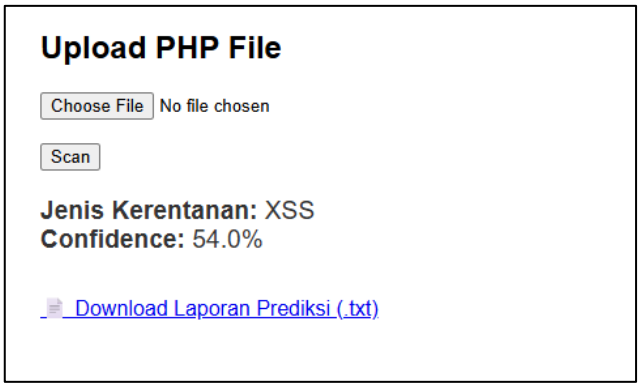
Gambar 4. Hasil Pengujian

Secara umum, sistem mampu memberikan prediksi yang sesuai dengan isi kode yang diunggah. Meskipun dapat dilihat juga bahwa beberapa hasil memiliki *confidence score* yang rendah, yang menandakan ketidakpastian model. Hal ini sangat mungkin disebabkan oleh struktur kode yang kompleks, netral, atau kurang representatif dibandingkan data pelatihan. Berikut ini adalah tampilan sistem aplikasi yang sudah dibuat dan contoh hasil yang diperoleh :

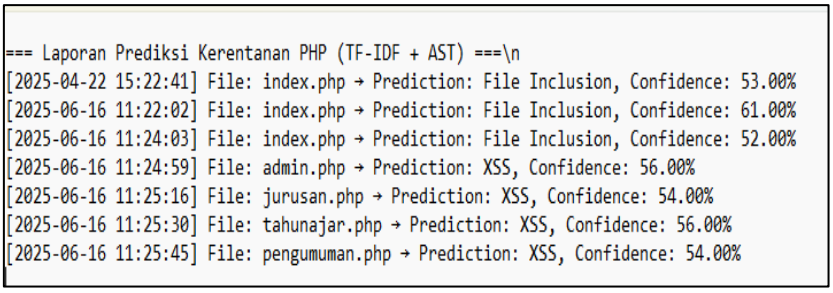


Gambar 5. Tampilan Aplikasi

p-ISSN: 2621-9433
e-ISSN: 2656-0259



Gambar 6. Tampilan Hasil Prediksi



Gambar 7. Tampilan Hasil Log Laporan

Kelebihan Sistem

Beberapa keunggulan dari aplikasi yang dibangun antara lain:

1. Dapat mendeteksi jenis kerentanan secara spesifik, bukan hanya "rentan" atau "aman".
2. Terdapat *Confidence score* memberi transparansi pada hasil prediksi
3. Deteksi lebih dalam berkat adanya fitur AST
4. Terdapat log otomatis yang bisa diunduh sebagai laporan
5. Ringan dan portabel, dapat dijalankan di komputer lokal tanpa server khusus.

Keterbatasan Sistem

Meski berfungsi dengan baik pada tahap awal, sistem ini masih memiliki sejumlah keterbatasan:

1. Dataset masih terbatas dan belum mencakup banyak gaya penulisan kode.
2. *Confidence score* masih rendah di banyak kasus.
3. Belum mendeteksi lokasi baris kerentanan (hanya prediksi satu label untuk seluruh file).
4. Sistem belum mendukung *multi-label detection.*

PENUTUP

Penelitian ini berhasil menghasilkan sebuah sistem deteksi kerentanan keamanan pada kode PHP yang ringan, praktis, dan dapat dijalankan secara lokal. Sistem ini dibangun dengan pendekatan *machine learning* menggunakan algoritma *Random Forest*, dan ditingkatkan melalui kombinasi dua pendekatan fitur utama:

1. TF-IDF, yang membaca pola kata dan token dalam kode
2. AST Parsing, yang melihat struktur dan konteks logika kode PHP

Pengujian menunjukkan bahwa sistem mampu mengklasifikasikan jenis kerentanan dengan cukup baik dan memberikan informasi tambahan berupa confidence score untuk memperlihatkan tingkat keyakinan terhadap prediksi. Meski sebagian confidence score masih berada di bawah 70% dengan maksimal hasil di 61%, hal ini menjadi sinyal bahwa sistem masih dalam tahap awal, dan belum sepenuhnya mengenali keragaman struktur kode di dunia nyata.

beberapa saran yang dapat dipertimbangkan untuk pengembangan lebih lanjut adalah adanya perluasan dataset, peningkatan parser PHP yang lebih canggih, menggunakan metode evaluasi seperti *confusion matrix, precision, recall*, dan F1-*score* untuk mengukur performa model secara objektif, lalu bisa juga Membandingkan performa model TF-IDF+AST dengan pendekatan NLP atau *embedding* seperti Word2Vec atau CodeBER.

DAFTAR PUSTAKA

[1] Hossain, M. S., et al. (2020). Security vulnerabilities in PHP applications: A systematic literature review. Computer Standards & Interfaces, 71, 103427.

[2] Karbab, E. B., et al. (2018). Malware detection with deep learning: An empirical study with PHP web applications. Computers & Security, 77, 612–628.

[3] Islam, S., & Falcarin, P. (2011). Using rule-based and statistical analysis for vulnerability detection. Journal of Computer Virology and Hacking Techniques, 7(4), 295–305.

[4] Sharma, S., Singh, A., & Chauhan, D. S. (2020). TF-IDF and Machine Learning Based Detection of Malicious Code in Web Pages. International Journal of Computer Applications, 975(8887).

[5] Alzahrani, A., & Cordy, J. R. (2019). Structural detection of security vulnerabilities in PHP code using AST and graph-based techniques. Journal of Systems and Software, 153, 190–203.

[6] T. Ying and S. Liu, "Security Analysis of PHP Web Applications Based on Machine Learning," in 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference, IEEE, 2019.

[7] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8] Wijana, M., Juliansyah, G., & Budiman, D. A. (2022). Sistem Pendukung Keputusan Penilaian Kinerja Guru Menggunakan Metode Weighted Product. Jurnal Dimamu, 2(1), 21–28.

[9] Permana, A. A., & Wijana, M. (2023). Rancang Bangun Sistem Informasi Penjualan Barang Berbasis Web di Toko Kelontong Haji Agus. INTERNAL (Information System Journal), 6(1), 46–54.

[10] Ferdiansyah, R., & Ramadhani, R. (2021). Analisis Kode Berbahaya pada File Website Menggunakan TF-IDF dan Random Forest. Jurnal Teknologi dan Sistem Informasi, 2(2), 83–90.