Terbit pada laman web jurnal : **http://ojsamik.amikmitragama.ac.id**

JURNAL JARINGAN SISTEM INFORMASI ROBOTIK (JSR)
Vol. 7 No. 1 TAHUN 2023 E - ISSN : 2579-373X

PENDETEKSI MALWARE JAVASCRIPT MENGGUNAKAN FITUR SEMANTIK PROGRAM

Edwin Lesmana Tjiong[1]
[1]Program Studi Informatika - Kalbis Institute
edwin.tjiong@kalbis.ac.id[1]

Abstrak

Di dalam karya ilmiah ini, kami mengembangkan sebuah program yang berfungsi untuk mendeteksi piranti lunak jahat (*malware*). Program kami menerima masukan dari sebuah file *Javascript* dan menggunakan parser untuk membangun pohon sintaksis abstrak (*Abstract Syntax Tree*). Pohon ini mengambil informasi sintaksis dari kode yang diberikan. Dari pohon ini, kami menambahkan sisi alur program (*control-flow edge*) dan sisi alur data (*data-flow edge*). Program kami mempunyai dua alur: alur normal dan alur khusus. Di dalam alur normal, kami mengambil fitur N-Gram saat menjelajah pohon sintaks dengan algoritma DFS. Di dalam alur khusus, kami merakit 28 fitur statis baru untuk mendeteksi *malware* dengan lebih spesifik. Kami telah menguji program kami dengan *dataset* sejumlah 203,117 file Javascript dan memperoleh *F1-score* sebesar 99 persen.

Kata Kunci: Keamanan Siber, Analisa Malware, Javascript, Pembelajaran Mesin, Pohon Sintaksis Abstrak

Abstract
In this paper, we presented a Javascript malware detector based on static analysis. Our program takes input from Javascript files to construct Abstract Syntax Trees (AST), a data structure which capture syntactic information of the source code. From AST graph, our program created more edges such as control-flow edges to capture semantic information of the given Javascript file. Our program is comprised of two workflows: normal workflow and custom workflow. For normal workflow, our program will extract N-Gram feature as it traverses AST graph in depth-first search manner. For custom workflow, our program will extract 28 novel static features which can detect Javascript malware in more specific manner. We have tested these features against dataset of 203,117 Javascript files and obtained F1-score of 99 percents.

Keywords: Cybersecurity, Malware Analysis, Javascript, Machine Learning, Abstract Syntax Tree

1. PENDAHULUAN

Di era industri 4.0, piranti lunak telah membumi ke segala aspek kehidupan manusia modern. Segala jenis perangkat elektronik yang tadinya bersifat analog telah atau akan dikomputerisasi dan diotomasi. Beberapa jenis perangkat elektronik yang dikomputerisasi dan/atau diotomasi misalnya jam pintar (*smart watch*), mobil otonom (*autonomous vehicle*), dan pesawat tanpa awak (*unmanned aerial vehicle*).

Dengan cara tersebut, maka perangkat elektronik yang menyediakan jaringan internet untuk pihak penyedia perangkat keras (*hardware*) dapat melakukan update pada piranti lunak tertanam (*firmware*) dengan teknik yang disebut *over-the-air* (OTA) update. Selain itu, banyak juga vendor memberikan keleluasaan kepada pengguna untuk menginstal segala jenis piranti lunak ke dalam perangkat elektronik mereka.

Tren ini berakibat pada melunjaknya jenis dan jumlah piranti lunak yang beredar di jaringan internet. Akan tetapi, jumlah piranti lunak yang beredar tidak diimbangi dengan pengujian apakah piranti lunak tersebut aman digunakan. Tren meningkatnya jumlah piranti lunak ini juga diiringi oleh meningkatnya kejahatan siber yang dilakukan melalui piranti lunak jahat (*malware*). Maka, teknologi keamanan siber seperti anti-virus dan pemindai situs (*website scanner*) yang merupakan benteng untuk menjaga pengguna wajib diperlengkapi dengan informasi *malware* terkini yang dapat menyusup dan mengambil alih sistem mereka.

Javascript sangat umum digunakan dalam proses mengakses layanan online melalui *browser* dan sering digunakan sebagai komponen *malware*. Di dalam karya ilmiah ini, kami mengembangkan sebuah program untuk mendeteksi *malware* berbasis *Javascript* dengan memanfaatkan fitur statis. Program ini menerima masukan (*input*) dari sebuah file *Javascript* dan menggunakan pengurai (*parser*) untuk membangun pohon sintaksis abstrak (*Abstract syntax tree*). Selanjutnya sisi lain juga ditambahkan seperti sisi alur program (*control-flow edge*) dan sisi alur data (*data-flow edge*).

Dengan memanfaatkan data struktur tersebut, program ini dibagi menjadi dua alur untuk menganalisa file yang diberikan: alur normal dan alur khusus. Di dalam alur normal, kami mengambil fitur N-Gram (N=4) saat menjelajah pohon sintaks dengan algoritma DFS (*depth-first search*). Di dalam alur khusus, kami merancang 28 fitur statis baru supaya bisa mendeteksi malware dengan lebih spesifik. Kami telah menguji program ini dengan dataset *malware Javascript* sejumlah 203,117 file dan memperoleh F1-score sebesar 99 persen.

2. METODOLOGI PENELITIAN

Di dalam penelitian ini, kami mengumpulkan *dataset* baik file *Javascript* normal maupun *malware* yang tersedia secara gratis. Sesudah mengumpulkan dataset yang tersedia, kami merancang sebuah program untuk membuat dataset sintesis. Dataset ini ditujukan untuk menguji apakah fitur yang kami pilih dapat membedakan antara *malware* dengan file *Javascript* normal yang telah melalui teknik kebingungan (*obfuscation*).

Selain itu, kami melakukan observasi dengan memeriksa banyak contoh *malware Javascript* dan mengamati bagaimana cara mereka melakukan serangan ke dalam sistem. Sesudah itu, kami merangkum mekanisme mereka dan merancang fitur-fitur statis yang dapat membantu kami mendeteksi *malware* dengan lebi detil.

Kemudian, kami merancang alur program berdasarkan kode yang telah dilakukan oleh karya sebelumnya [1]. Kami menambahkan alur bagaimana program tersebut menerima masukan dan kami mengembangkan program untuk mengambil fitur yang kami rancang berdasar contoh-contoh *malware* yang kami amati. Kami menganalisa dan memberikan alasan kami mengambil fitur-fitur tersebut untuk mendeteksi *malware*.

Sesudah itu, kami melakukan eksperimen standar pembelajaran mesin (*machine learning*) dari mulai pemilihan dan pengambilan fitur, pelatihan model (*model training*), dan pengujian (*testing*). Kasus ini kami anggap sebagai sebuah *binary classification* (Mengkategorikan apakah sebuah *Javascript* file yang diberikan adalah file normal atau *malware*). Hasil eksperimen menunjukan bahwa fitur yang kami rancang dapat memberikan F1-score sebesar 99 persen yang setara dengan karya sebelumnya menggunakan dataset asli dan sintesis.

Yang terakhir, untuk membandingkan karya kami dengan karya sebelumnya, kami melakukan tinjauan pistaka karya ilmiah yang dipublikasikan di level nasional maupun internasional di bidang keamanan siber.

2.1. Pengumpulan Dataset

Untuk evaluasi, kami mengumpulkan dataset file *Javascript* normal dan malware, baik bentuk teks biasa maupun tersamar (*obfuscated*). *Javascript* normal sejumlah 142,382 file tersedia melalui situs ini [2]. *Javascript* file ini dikumpulkan dari akun Github, file yang duplikat telah dibuang dan hanya file yang bisa terurai yang akan disimpan. Hampir semua *Javascript* file dalam *dataset* ini adalah teks biasa. Untuk file normal yang tersamar, kami mengambil dari dua sumber yaitu 2,924 file dari situs ini [3] dan dataset sintesis.

Untuk membuat dataset sintesis, kami menggunakan tiga program yang tersedia gratis yaitu: Beautify-converter, obfuscator.io dan DaftLogic untuk menyamarkan file asal. Kami membuat 9,996 file menggunakan program-program tersebut.

Untuk dataset malware, kami mengambil sampel-sampel dari arsip online seperti HynekPetrak (39,451 file) dan VirusTotal (8,545 file). Dari sampel-sampel ini, kami juga membuat dataset sintesis untuk *malware* tersamar. Meskipun banyak *malware* tersebut sudah disamarkan, kami tetap menggunakan program-program di atas untuk melakukan *obfuscation*.

Tabel 1. Daftar sumber dataset Javascript

Sumber	Jumlah	Jenis	Asal
SRILAB [2]	142,389	Biasa	Asli
SOLA software lab [3]	2,736	Biasa	Asli
SRILAB (Beautify-converter)	4,999	Biasa	Sintesis
SRILAB (DaftLogic)	4,997	Biasa	Sintesis
HynekPetrak	39,451	Malware	Asli
VirusTotal	8,545	Malware	Asli
Total	**203,117**		

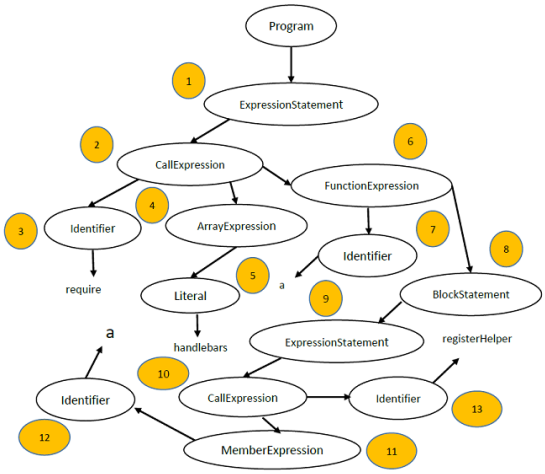
2.2. Pemrosesan Data

Program kami pertama-tama akan mengurai kode *Javascript* dan mengubahnya menjadi pohon sintaksis abstrak. Setelah itu, *graph* akan dijelajah untuk mengambil fitur n-gram yang akan digunakan untuk klasifikasi *malware Javascript*. Untuk ilustrasi yang lebih jelas, perhatikan contoh *Javascript* di bawah ini:

```
require(["handlebars"], function(a) {
      a.registerHelper("if_eq", function(d, c, e) {
            if (d === c) {
                  return e.fn(this);
            } else {
                  return e.inverse(this);
            }
      });
});
```

Gambar 1. Contoh kode Javascript

Setiap silabel dari kode tersebut akan diubah menjadi token (tipe token akan tergantung dari semantic mereka). Dengan parser Esprima, hubungan antar token akan dijadikan sebuah *graph* yang disebut dengan pohon sintaksis abstrak (*Abstract syntax tree*), sebagaimana ditunjukan oleh gambar di bawah ini:



Gambar 2. Pohon sintaksis abstrak dari kode Javascript

Program kami kemudian akan mulai menjelajah simpul *graph* dari akar 'Program' dengan urutan sebagaimana ditunjukan oleh lingkaran oranye bernomor di atas. Kami lalu menghitung frekuensi n-gram dari simpul-simpul yang kami jelajahi dengan teknik DFS (*depth-first search*). Misalnya 4-gram pertama yang kami jelajah adalah 1 → 2 → 3 → 4. Urutan ini diterjemahkan menjadi tipe simpul (**ExpressionStatement**, **CallStatement**, **Identifier**, dan **ArrayExpression**).

Di akhir pemrosesan data, kami menyimpan frekuensi dari setiap urutan tipe simpul tersebut yang terjadi dalam penjelajahan *graph*. Dengan fitur 4-gram tersebut, alur normal dapat mencapai akurasi 99 persen.

2.3. Pemilihan Fitur

Kami merancang dan memilih 28 fitur dari file *Javascript*, *graph* AST, dan matriks keterhubungan (*adjacency matrix*) dari *graph* tersebut. Fitur-fitur ini dipilih karena berdasarkan analisa kami mereka berkaitan erat dengan perilaku (*behavior*) program saat dieksekusi. Berikut daftar dari 28 fitur tersebut beserta alasan mereka dipilih:

1) **Fitur pola string (13 fitur)**: Fitur ini dipilih berdasar analisa manual sampel *malware*. Penggunaan keyword `eval` terdapat hampir di setiap sampel dikarenakan fungsi untuk menciptakan kode secara dinamis. Fungsi lain untuk menciptakan kode secara dinamis termasuk `setTimeout()` dan `setInterval()`. Metode untuk memanipulasi DOM (*Document Object Model*) yaitu `document.write`, `document.writeIn`, `innerHTML`, dan `outerHTML` juga dipilih. Dengan menggunakan data tersebut, data bisa dieksekusi bila ditulis dalam sebuah webpage. Kami juga menandai pemanggilan kepada objek tak lazim seperti `WScript` (digunakan ketika kode *Javascript* membuka terminal Windows), `RegRead` (digunakan ketika membaca *registry*), `RegExp` (penggunaan *regular expression*, sering digunakan dalam teknik pembingungan untuk *string encoding* dan *decoding*), dan `ActiveXObject` (karena *malware* sering mencoba mengakses `ActiveXObject` yang memiliki kerentanan (*vulnerability*)).

2) **Tiga tipe simpul terpopuler beserta frekuensinya (6 fitur)**: Tipe simpul di dalam AST mengekspresikan peran semantik mereka, baik itu operasi aritmatika, penugasan variabel, dan deklarasi fungsi. Sehingga, tipe simpul yang paling populer dan frekuensinya merangkum keseluruhan fungsi kode *Javascript* yang diberikan.

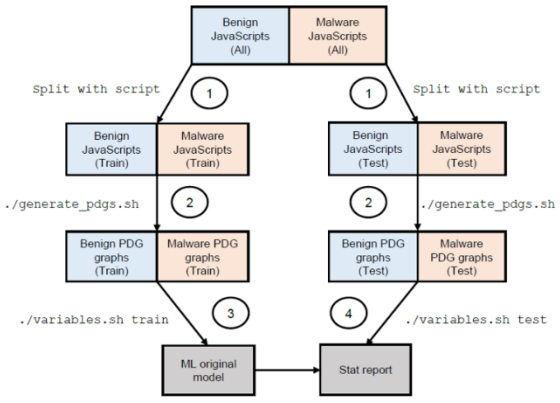
3) **Nilai rata-rata (*means*), nilai tengah (*median*), dan jarak maksimum dari simpul akar (*root node*) ke simpul daun (*leaf node*) – 3 fitur**: Fitur-fitur ini mengekspresikan kedalaman dari pemanggilan fungsi dalam kode *Javascript*. Intuisinya bahwa program *malware* cenderung memiliki pemanggilan fungsi yang lebih dalam dengan metode `eval()` untuk membuat peneliti keamanan siber lebih sulit memahami cara kerja mereka.

4) **Nilai rata-rata (*means*), nilai tengah (*median*), dan jumlah cabang simpul (*node degree*) maksimum (3 fitur)**: Fitur-fitur ini mengindikasikan kompleksitas program *Javascript*. Dari analisa manual, kami mengamati *malware* cenderung memiliki kode yang lebih berurut dan tidak banyak cabang. Fungsinya hanya mengeksekusi muatan (*payload*) yang ada di bagian kode tersamar sementara kode *Javascript* normal mempunyai banyak cabang untuk menangani banyak kasus dalam kasus yang berbeda-beda.

5) **Persentase jumlah sisi AST (*Abstract syntax tree*), CFG (*Control flow graph*), dan DFG (*Data flow graph*) – 3 fitur**: Dari analisa manual dari matriks keterhubungan didapatkan bahwa kode *Javascript* normal memiliki persentase sisi CFG dan DFG yang lebih banyak dari *malware*.

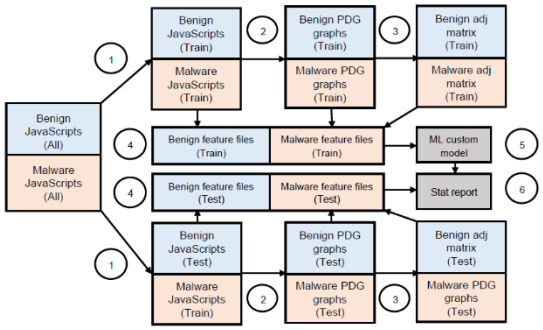
2.4. Perancangan Alur Program

Alur kerja dalam program kami terbagi menjadi dua tipe. Tipe pertama adalah alur yang memakai fitur N-Gram dan pasangan key-pair. Pertama-tama, file *Javascript* dipecah menjadi dataset pelatihan (*training*) dan pengujian (*testing*). Setelah itu, program kami membuat *graph* gabungan dari AST, CFG dan DFG (disebut dengan PDG atau *Program Dependence Graph*) yang prosesnya sebagaimana diceritakan pada bab pemrosesan data. Lalu, *graph* ini akan diambil dan dilakukan pengambilan fitur dengan dua tipe (N-Gram atau pasangan *key-pair*) dan empat level (token, AST, CFG dan DFG). Setelah pelatihan, program kami menghasilkan model pembelajaran. Lalu pada saat pengujian, sistem akan membuat model tersebut, mengambil fitur dengan tipe dan level yang sama dan menggunakannya untuk tugas klasifikasi kode *Javascript* yang diberikan. Hasil klasifikasi akan dipresentasikan di dalam sebuah laporan. Bagan di bawah ini merangkum seluruh proses alur kerja normal:



Gambar 3. Alur Kerja Normal Dalam Pendeteksi Malware Javascript

Selain itu, alur kerja khusus bila fitur-fitur baru perlu digunakan untuk mendeteksi *malware* (sebagaimana dijelaskan di bab pemilihan fitur). 28 fitur tersebut diambil dari kode *Javascript* mula-mula, *graph* PDG, dan matriks keterhubungan. Berbeda dengan alur normal, di sini kami perlu merakit matriks keterhubungan. Matriks ini diturunkan dari *graph* PDG dan perlu dibuat sebelum pengambilan fitur. Setelah itu, fitur-fitur akan digabung dalam sebuah file matriks. Dari file matriks tersebut, pelatihan dan pengujian akan dilakukan. Bagan di bawah ini merangkum keseluruhan alur kerja khusus:



Gambar 4. Alur Kerja Khusus Dalam Pendeteksi Malware Javascript

3. HASIL DAN PEMBAHASAN

Pada bagian ini kami melaporkan hasil eksperimen dari penelitian kami dan menganalisa hasil temuan kami untuk penggunaan secara praktis.

4.1. Hasil Eksperimen

Di dalam eksperimen, kami menjalankan alur kerja normal dan khusus dengan dataset yang sama untuk membandingkan hasil klasifikasi. Pertama-tama kami melakukan tahap 1 sampai 4 dalam alur kerja khusus untuk membuat matriks keterhubungan dan file daftar fitur. Dalam setiap tahap, terdapat file-file yang tidak dapat diproses karena alasan-alasan tertentu (*parser* tidak dapat membaca *malware* dengan sintaks Javascript terbaru, ukuran file terlalu besar, dst). Tabel di bawah ini menunjukan jumlah file *Javascript* biasa (*benign*) dan jahat (*malware*) dan jumlah struktur yang dihasilkan:

Tabel 2. Jumlah file dalam tahap pemrosesan data

Javascript awal (Tahap 1)	
Biasa	**Malware**
155,121	47,996
File graph PDG (Tahap 2)	
Biasa	**Malware**
150,272	40,087
Matriks keterhubungan (Tahap 3)	
Biasa	**Malware**
149,149	32,541
Daftar Fitur (Tahap 4)	
Biasa	**Malware**
149,154	32,543

Kami menemukan 5 file normal dan 2 *malware* yang mempunyai daftar fitur tapi tidak mempunyai struktur PDG. File tersebut dibuang dari eksperimen kami. Maka, untuk eksperimen akhir, kami mempunyai 149,154 – 5 = **149,149 file biasa** dan 32,543 – 2 = **32,541 malware**. Dataset tersebut dipecah menjadi dataset pelatihan dan pengujian dengan pembagian sebagai berikut:

Tabel 3. Dataset Javascript biasa dan malware

Tipe Sampel	Pelatihan (80%)	Pengujian (20%)
Biasa	119,320	26,033
Malware	29,829	6,508
Total	**149,149**	**32,541**

Kami melakukan proses alur kerja normal dan alur kerja dan alur kerja khusus pada dataset ini (yang mempunyai 3 struktur: *graph* PDG, matriks keterhubungan, dan daftar fitur). Hasil untuk alur kerja normal dan khusus ditunjukan dalam tabel di bawah ini.

Tabel 4. Hasil klasifikasi alur normal dan khusus

Tipe Alur Kerja	Normal	Khusus
True Positive Rate (TPR)	98.01%	98.30%
True Negative Rate (TNR)	99.98%	99.96%
Precision	99.95% (6379/6382)	99.82% (6398/6409)
Recall	98.01% (6379/6508)	98.30% (6398/6508)
F1-score	98.97%	99.05%

Alur kerja khusus telah mencapai F1-score yang sedikit lebih baik dengan dataset yang sama. Ini menunjukan bahwa 28 fitur baru dapat mengambil semantik program *Javascript* karena dataset tersebut juga mencakup *Javascript* normal yang tersamar (*obfuscated*).

4.2. Penggunaan Program

Alur kerja biasa cocok digunakan untuk kasus di mana dataset tersedia dalam jumlah besar tetapi pengguna tidak terlalu mementingkan tentang fitur yang digunakan untuk mengidentifikasi *malware*. Ini dikarenakan alur kerja normal lebih cepat dibandingkan alur kerja khusus. Alur kerja normal tidak merakit data struktur tambahan (matriks keterhubungan). Akan tetapi, pengguna tidak dapat menganalisa fitur apa yang berkontribusi terhadap proses klasifikasi *Javascript* di dalam dataset mereka.

Sebaliknya. pada alur kerja khusus kami membuat data struktur tambahan untuk menampung fitur dari berbagai tipe dan level (string, statistik *graph*, dan statistik matriks keterhubungan). Jumlah fitur yang sangat sedikit dan nilai dari fitur-fitur tersebut tersedia dalam file daftar fitur. Ini memungkinkan pengguna melakukan analisa lebih detil mengenai klasifikasi malware dalam program mereka. Pengguna dapat menghubungkan cara kerja *malware* dengan fitur yang ada dan membuat *signature* untuk piranti lunak pendeteksi malware seperti Yara.

4. KESIMPULAN

Di dalam karya ilmiah ini, kami telah mengembangkan sebuah pendeteksi *malware Javascript* dan menambah 28 fitur statis baru. Fitur statis ini diperoleh dengan analisa manual *malware Javascript* yang ada, dari struktur pohon sintaksis abstrak, dan dari matriks keterhubungan. Selain itu, kami mengumpulkan dan membuat dataset yang lebih bervariasi termasuk *Javascript* normal dan *malware* yang telah disamarkan. Kami telah mengevaluasi fitur statis baru kami dan membandingkannya dengan fitur N-Gram dan pasangan *key-value* dalam *graph* PDG. Kami berhasil memperoleh F1-score sebesar 99 persen, sedikit lebih baik dari fitur N-Gram yang digunakan oleh karya sebelumnya.

Untuk mengembangkan karya ini dengan lebih jauh, para peneliti dapat berfokus pada *malware* yang memiliki tipe file yang ada di platform yang lebih baru, misalnya *blockchain* atau alat-alat *smart device*. Pendeteksi *malware* ini juga hanya menggunakan fitur statis, sehingga akan mengalami kesulitan dalam mendeteksi *malware* yang berada dalam keadaan dikemas (*packed malware*). Untuk mendeteksi malware jenis tersebut, perlu dilakukan analisa secara dinamis di mana *malware* tersebut dieksekusi di dalam sebuah proses terisolir (*sandbox*) dan dimonitor artifaknya selama eksekusi.

5. DAFTAR PUSTAKA

[1] A. Fass, M. Backes, and B. Stock, "JSTAP: A static pre-filter for malicious Javascript detection," in *ACM International Conference Proceeding Series*, Dec. 2019, pp. 257–269. doi: 10.1145/3359789.3359813.

[2] ETH Zurich, "150k Javascript Dataset, available from https://www.sri.inf.ethz.ch/js150," Feb. 18, 2023.

[3] University of Sttutgart, "Javascript and Web Applications, available from https://www.software-lab.org/," Nov. 01, 2020.

[4] K. Rieck, T. Krueger, and A. Dewald, "Cujo: Efficient Detection and Prevention of Drive-by-Download Attacks," in *Proceedings of the 26th Annual Computer Security Applications Conference*, Dec. 2010, pp. 31–39. doi: 10.1145/1920261.1920267.

[5] P. Laskov and N. Šrndić, "Static detection of malicious JavaScript-bearing PDF documents," in *Proceedings of the 27th Annual Computer Security Applications Conference*, Dec. 2011, pp. 373–382. doi: 10.1145/2076732.2076785.

[6] B. Stock, B. Livshits, and B. Zorn, "Kizzle: A Signature Compiler for Detecting Exploit Kits," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, Jun. 2016, pp. 455–466. doi: 10.1109/DSN.2016.48.

[7] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert, "ZOZZLE: Fast and Precise In-Browser Javascript Malware Detection," in *USENIX Security Symposium*, Aug. 2011.

[8] C. Kolbitsch, B. Livshits, B. Zorn, and C. Seifert, "Rozzle: De-cloaking Internet Malware," in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 443–457. doi: 10.1109/SP.2012.48.

[9] K. Kim *et al.*, "J-Force: Forced Execution on JavaScript," in *Proceedings of the 26th International Conference on World Wide Web*, Apr. 2017, pp. 897–906. doi: 10.1145/3038912.3052674.

[10] E. L. Tjiong, S. Mechtaev, and H. B. Dirgantara, "Use of General Repair Tool for Fixing Security Vulnerabilities," in *2022 International Conference on Information Technology Research and Innovation (ICITRI)*, Nov. 2022, pp. 135–140. doi: 10.1109/ICITRI56423.2022.9970223.