BAB 1

PENDAHULUAN

1.1 Latar Belakang

keamanan komputer menjadi fokus utama bagi banyak organisasi dan individu. Ancaman dari *virus*, termasuk *virus* komputer, *worm*, dan *trojan horse*, semakin kompleks dan merugikan. Penelitian dalam bidang keamanan informasi menjadi semakin penting untuk menghadapi tantangan ini. Salah satu aspek krusial dalam perlindungan terhadap *virus* adalah pengembangan sistem *antivirus* yang efektif dan adaptif. [1]

Pendeteksian *virus* melibatkan proses pemindaian *registry, program startup*, layanan sistem, proses aktif, dan elemen lainnya. Berkas *Portable Executable* (*PE*) memainkan peran penting dalam mendeteksi *virus*, karena merujuk ke titik masuk (*entry point*) dari *virus*. Oleh karena itu, informasi seperti *Size Of Code* dan *Address Of Entry Point* dari *PE* menjadi kunci dalam upaya identifikasi dan deteksi *virus* komputer. [2]

Dalam konteks ini, pendekatan yang diusulkan adalah menggunakan Metode Pencarian *Header* pada *file* berekstensi *EXE*. Metode ini bekerja dengan mencari *header* pada *file* tersebut dan mengekstrak informasi tentang *Size of Code* dan *Address of Entry Point* sebagai pola untuk mengidentifikasi serta mengatasi serangan *virus*.

Masalah utama adalah bahwa metode deteksi virus tradisional sering kali tidak mendeteksi teknik penyamaran yang digunakan oleh virus *modern*, seperti *polymorphic* dan pengacakan kode. Oleh karena itu, pengembangan sistem antivirus yang menggunakan metode pencarian *header* pada *file* berekstensi *EXE* dengan menganalisis informasi seperti *Size of Code* dan *Address of Entry Point* menjadi penting. Metode ini diusulkan karena dapat memberikan deteksi yang lebih akurat dengan mengidentifikasi pola-pola yang mencurigakan dalam *file* eksekusi, bahkan ketika virus menggunakan teknik-teknik pengacakan yang lebih canggih.

Pendekatan ini didasarkan pada prinsip bahwa, meskipun virus mungkin memodifikasi bagian lain dari *file* untuk menghindari deteksi, data kunci seperti *Size of Code* dan *Address of Entry* tetap tidak berubah. Dengan memanfaatkan informasi ini, sistem *antivirus* dapat membedakan *virus* dari *file* yang aman, bahkan jika *virus* telah menggunakan teknik *polymorphic* atau pengacakan kode.

Menurut Chafid dan Braselino (2018), keunggulan utama dari metode ini adalah kemampuannya untuk tetap efektif dalam mengidentifikasi *virus* bahkan setelah mereka memodifikasi *dos header* atau menerapkan teknik *polymorphic* yang rumit. Berbeda dengan pendekatan seperti *CRC-32* yang mengandalkan perhitungan *checksum* dari seluruh *file*, metode ini lebih fokus pada karakteristik kunci yang cenderung tidak berubah bahkan setelah *virus* mengubah struktur *file*. Hal ini membuatnya lebih tangguh terhadap modifikasi yang dilakukan oleh *virus*, dan menghasilkan deteksi yang lebih andal dalam kasus-kasus di mana *virus* mengubah strukturnya secara signifikan [13].

Penelitian terdahulu yang membahas *antivirus* berbasis *header file* juga telah dilakukan. Sebagai contoh, dalam penelitian "Implementasi Metode Pencarian (*HSC*) *Header Size Of Code* Dan Titik Masukan Pengalamatan Pada Rancangan *Software Anti Virus*" oleh Nurul Chafid dan Rizky Tsaunauval Braselino pada tahun 2018, ditemukan bahwa terdapat kekurangan dimana jika ada *virus* baru, maka *database* harus dilakukan *update* secara manual [13]. Selain itu, dalam penelitian "Teknik Pembuatan *Anti Virus* Dengan Metode Pencarian Data *Header File* Menggunakan *Visual Basic* 6.0" oleh Hari Purwanto pada tahun 2015, ditemukan bahwa antarmuka pengguna (*user interface*) dari *antivirus* tersebut tidak ramah pengguna dan kurang *modern*. Selain itu, sensitivitas *antivirus* tersebut juga perlu ditingkatkan, khususnya dalam mendeteksi adanya *section dummy* pada urutan *section* kedua atau jika tidak ada pada *section*, serta pembelokan pada *entry point* [14].

Implementasi sistem *antivirus* yang memanfaatkan pendekatan ini diharapkan dapat memberikan perlindungan yang lebih efektif dan adaptif terhadap ancaman *virus* yang terus berkembang. Dengan kemampuannya untuk mengenali pola-pola yang khas dari *virus* komputer, sistem ini dapat memberikan respons yang cepat dan tepat dalam mendeteksi dan menangani serangan. Kelebihan dari *antivirus* ini adalah ukurannya kecil sehingga tidak membebani sistem, ringan saat dijalankan, serta memiliki antarmuka pengguna (*UI*) yang baik sehingga mudah digunakan oleh pengguna.

Selain itu, *antivirus* ini juga memungkinkan pengguna untuk melakukan pemindaian terhadap satu *file* atau satu direktori saja, serta dapat memperbarui *database virus* secara manual atau otomatis ketika dibuka. Kemampuannya sebagai aplikasi *portable* membuatnya tidak perlu melakukan instalasi terlebih dahulu sehingga dapat digunakan dengan mudah di berbagai perangkat. Selain itu, *antivirus* ini juga tersedia secara gratis dan bersifat *open source*, memberikan keleluasaan bagi pengguna untuk mengakses dan memodifikasi kode sumber sesuai kebutuhan.

Berdasarkan uraian tersebut maka peneliti tertarik untuk mengangkat judul ”Sistem *Anti Virus* Dengan Metode Pencarian *Header File* Data *Size Of Code* Dan *Address Of Entry* Dengan Perhitungan *Jaccard Similarity*”

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dikemukakan, yang menjadi pokok permasalahan adalah bagaimana merancang dan membangun Sistem *Anti virus* dengan metode pencarian *header file size of code* dan *address of entry point* dengan perhitungan *jaccard similarity*?.

1.3 Tujuan Penelitian

Adapun tujuan dari penelitian ini adalah merancang dan membangun Sistem *Anti virus* dengan metode pencarian *header file size of code* dan *address of entry point* dengan perhitungan *jaccard similarity.*

1.4 Batasan Masalah

Adapun batasan masalah dalam penelitian ini:

1. *File* yang akan di jadikan sampel *virus* adalah *file* yang berekstensi *.*exe.* dan beberapa sampel *virus sality, sality* merupakan *virus* yang berjenis *PE infector (Polymorphic)* yang menginfeksi *file-file Executabe "exe"*
2. Bahasa pemrograman yang digunakan adalah *Python,* dengan *library pe file* untuk analisis dan membaca *header file.*
3. Penelitian ini fokus pada pola *virus* yang diidentifikasi dalam bentuk *hexadecimal*, khususnya data *Address of Entry Point* dan *Size of Code* dari *file PE*. Pola-pola ini digunakan untuk mengidentifikasi dan mendeteksi *virus* dengan menganalisis struktur *internal file executable.*

1.5 Manfaat Penelitian

Manfaat penelitian adalah membantu dalam penemuan dan analisis pola serangan yang digunakan oleh *virus*. Hal ini dapat memberikan pemahaman yang lebih baik tentang cara kerja *virus* dan memungkinkan untuk pengembangan strategi pertahanan terhadap *virus* yang lebih efektif.

1.6 Sistematika Penulisan

Sistematika penulisan dari skripsi ini terdiri dari enam bagian utama, sebagai berikut:

BAB 1 PENDAHULUAN

Bab ini berisi latar belakang, rumusan masalah, tujuan penelitian, batasan masalah, dan sistematika penulisan.

BAB 2 LANDASAN TEORI

Bab ini berisi teori-teori yang digunakan pada penelitian ini. Teori-teori yang berhubungan dengan Sistem *Anti Virus* Dengan Metode Pencarian *Header File Data Size Of Code* Dan *Address Of Entry* Dengan Perhitungan *Jaccard Similarity.*

BAB 3 METODOLOGI PENELITIAN

Bab ini berisi tahapan-tahapan dalam pencarian sampel, perancangan sistem perumusan masalah, dan analisa.

BAB 4 ANALISA DAN PERANCANGAN

Bab ini berisi analisa dan perancangan Sistem *Anti Virus* Dengan Metode Pencarian *Header File Data Size Of Code* Dan *Address Of Entry* Dengan Perhitungan *Jaccard Similarity*.

BAB 5 IMPLEMENTASI DAN PENGUJIAN

Dalam bab ini, akan dijelaskan mengenai proses implementasi aplikasi yang telah dibangun dan hasil pengujian yang dilakukan untuk memastikan kinerja dan fungsionalitas aplikasi.

BAB 6 PENUTUP

Bab ini berisi kesimpulan dari hasil penelitian yang telah dilakukan dan saran-saran untuk pengembangan aplikasi atau penelitian selanjutnya.

BAB 2

LANDASAN TEORI

2.1 Sistem

Sistem adalah jaringan kerja prosedur-prosedur yang saling terkait, bertujuan untuk menjalankan kegiatan atau mencapai sasaran tertentu [3]. Sistem merupakan sekumpulan hal, kegiatan, elemen, atau subsistem yang saling bekerja sama atau terhubung dengan cara tertentu sehingga membentuk satu kesatuan untuk melaksanakan suatu fungsi demi mencapai tujuan [17]. Dalam konteks ini, sistem diartikan sebagai kumpulan elemen yang berinteraksi untuk mencapai tujuan yang ditentukan [4]. Di bawah ini adalah beberapa karakteristik penting dari suatu sistem:

1. Komponen Sistem: Setiap sistem terdiri dari beragam komponen atau *subsistem* yang saling terkait dan berinteraksi untuk mencapai tujuan tertentu. Komponen-komponen ini seringkali memiliki fungsi yang spesifik dan berperan dalam menjalankan operasi sistem secara keseluruhan.
2. Batas Sistem: Setiap sistem memiliki batas yang jelas yang memisahkan antara apa yang termasuk dalam sistem tersebut dan apa yang berada di luar sistem. Batas ini menentukan ruang lingkup operasi sistem dan membedakannya dari lingkungan eksternal.
3. Lingkungan Luar: Lingkungan luar sistem merujuk pada semua elemen, variabel, dan kondisi yang berada di luar batas sistem namun memiliki potensi untuk mempengaruhi atau dipengaruhi oleh sistem tersebut.
4. Penghubung: *Interface* atau penghubung merupakan titik-titik kontak antara sistem dan lingkungan luar serta antara komponen-komponen dalam sistem

itu sendiri. Penghubung ini memfasilitasi aliran informasi, energi, atau materi antara sistem dan entitas lainnya.

5. Tujuan: Setiap sistem memiliki tujuan yang ingin dicapai. Tujuan ini menjadi pedoman utama bagi operasi dan pengembangan sistem, serta menjadi parameter untuk mengevaluasi keberhasilan atau kegagalan sistem dalam mencapai hasil yang diinginkan [21].

2.2 *Polymorphic*

Polymorphic dalam konteks keamanan komputer mengacu pada kemampuan suatu *virus* untuk mengubah bentuk atau kodenya secara otomatis dan terus-menerus setiap kali ia menulari sistem baru. Tujuan utama dari teknik ini adalah untuk menghindari deteksi oleh program antivirus dan alat keamanan lainnya yang bergantung pada tanda tangan (*signature-based detection*) [34].

Cara Kerja *Polymorphic Virus*:

1. *Virus polymorphic* sering kali mengenkripsi tubuh aslinya menggunakan algoritma enkripsi yang bervariasi.
2. Setiap kali *virus* menulari sistem baru, ia mendekripsi dirinya untuk mengeksekusi kode berbahaya, kemudian mengenkripsi kembali dengan kunci atau metode yang berbeda sebelum menyebar lebih lanjut.
3. *Polymorphic virus* tidak hanya mengubah kunci enkripsinya tetapi juga dapat memodifikasi instruksi-instruksi dalam kodenya secara acak [35].

2.3 *Virus* komputer

Virus komputer adalah program yang dapat menginfeksi program lain dengan memodifikasinya untuk menyertakan salinan yang mungkin telah berevolusi dari dirinya sendiri [5]. Dengan sifat infeksi ini, sebuah *virus* dapat menyebar melalui

seluruh sistem komputer atau jaringan menggunakan izin dari setiap pengguna yang menggunakannya untuk menginfeksi program mereka. Setiap program yang terinfeksi juga dapat bertindak sebagai *virus* dan dengan demikian infeksi tersebut akan terus berkembang [6]. *Virus* komputer seringkali dirancang untuk merusak, mengganggu, atau mencuri data dari sistem yang terinfeksi. Mereka dapat menyebabkan kerusakan pada sistem operasi, merusak atau menghapus *file* penting, mencuri informasi sensitif, atau bahkan mengendalikan sistem secara keseluruhan [18]. Adapun Sebuah *malware* yang dapat dikategorikan sebagai *virus* adalah sebagai berikut:

1. Dapat melakukan infeksi terhadap *file* yang dapat dijadikan inang seperti *exe, scr, com, dll*. Sehingga ketika *file* tersebut dijalankan, akan mengaktifkan *virus* tersebut.
2. Manipulasi yang dilakukan lebih tinggi dari *worm*, sehingga lebih bertahan dalam sistem dibandingkan dengan *worm*.
3. Memakai ikon standar *executable* [12].

2.3.1 Teknik Penyebaran *Virus* Komputer

Teknik penyebaran *virus* komputer dapat bervariasi, dan beberapa di antaranya termasuk:

1. Melalui lampiran *email*: *Virus* sering kali disebarkan melalui lampiran *email* yang tampaknya berasal dari sumber tepercaya atau yang dikenal [18].
2. Situs *web* berbahaya: *Virus* dapat disematkan dalam situs *web* yang merugikan dan dapat menginfeksi komputer pengunjung [22].
3. Jaringan dan perangkat *removable*: *Virus* dapat menyebar melalui jaringan komputer atau perangkat penyimpanan berbagi seperti *USB drive*.

4. Eksploitasi kelemahan sistem: Beberapa *virus* menggunakan kelemahan atau kerentanan dalam sistem operasi atau perangkat lunak untuk menyebar dan menginfeksi perangkat.
5. *Sosial engineering*: Penyebaran *virus* juga dapat melibatkan teknik *sosial engineering*, seperti memanfaatkan ketidaktahuan atau kecurigaan pengguna untuk mengunduh atau menjalankan *file* yang terinfeksi [15].

2.4 Metode

Dalam penelitian ini, metode dijelaskan sebagai suatu pendekatan teratur yang digunakan untuk melakukan pekerjaan sesuai dengan tujuan yang dikehendaki, seperti yang didefinisikan dalam Kamus Besar Bahasa Indonesia (KBBI) [23]. Secara etimologis, kata "metode" berasal dari bahasa Yunani "*methodos*", yang terdiri dari kata "*meta*" yang berarti menuju, melalui, mengikuti, atau sesudah, dan "*hodos*" yang berarti jalan, cara, atau arah [24]. Metode merupakan suatu proses sistematis yang digunakan untuk mencapai tujuan tertentu. Dengan kata lain, metode berperan sebagai alat untuk mencapai suatu tujuan atau untuk menjelaskan cara melakukan atau membuat sesuatu [25]. Dalam konteks penelitian ini, metode yang digunakan adalah metode pencarian *header file size of code* dan *address of entry point* dari *executable* yang dicurigai sebagai *virus*.

2.5 *Antivirus*

Antivirus adalah program komputer dengan basis data yang berisi informasi yang digunakan untuk mengidentifikasi *virus*. Mesin pemindaian *antivirus* dirancang untuk mengidentifikasi *virus* tertentu menggunakan definisi tersebut dan dengan mengenali perilaku *virus* yang ditandai [7]. Perangkat lunak *antivirus* berperan dalam memberikan perlindungan dan keamanan bagi data komputer dari serangan *virus* [26]. Cara kerja *antivirus* adalah dengan memindai perangkat untuk mencari dan mencegah *virus* yang sudah dikenal maupun varian malware baru yang muncul [27].

Fungsi utama dari perangkat lunak *antivirus* meliputi:

1. Pemindaian: *Antivirus* melakukan pemindaian terhadap *file* dan sistem komputer untuk mendeteksi keberadaan *virus*.
2. Identifikasi: Berdasarkan definisi dan pola yang telah diketahui, *antivirus* mengidentifikasi *virus* yang ditemukan dalam sistem.
3. Karantina dan penghapusan: Setelah *virus* terdeteksi, *antivirus* dapat menempatkannya dalam karantina untuk mencegah penyebaran lebih lanjut atau menghapusnya dari sistem secara permanen.
4. Pembaruan definisi: Untuk tetap efektif dalam mendeteksi virus yang berkembang, *antivirus* memerlukan pembaruan berkala terhadap basis data definisi virusnya.

2.6 *Portable Executable (PE)*

Portable Executable (*PE*) adalah *format file* yang digunakan oleh sistem operasi *Windows* untuk menyimpan program eksekusi, *DLL* (*Dynamic Link Libraries*), dan berbagai jenis data lainnya [2]. *Format* ini adalah standar untuk aplikasi *Windows* dan biasanya ditemukan dalam *file* dengan ekstensi *.exe*, *.dll*, *.sys*, dan lainnya. *File PE* terdiri dari beberapa bagian yang mencakup *header*, bagian kode (*code section*), bagian data (*data section*), tabel eksternal (*import/export table*), dan informasi tambahan lainnya yang diperlukan untuk eksekusi program [8]. Nama "*Portable Executable*" merujuk pada kenyataan bahwa *format* tersebut tidak spesifik terhadap arsitektur [19].

Komponen-komponen utama dalam *file Portable Executable* (*PE*) meliputi:

1. *Header*: *Header file PE* berisi informasi dasar tentang *file*, seperti ukuran dan lokasi dari bagian-bagian utama dalam *file*.

2. Bagian Kode (*Code Section*): Bagian ini berisi kode eksekusi yang sebenarnya dari program.
3. Bagian Data (*Data Section*): Bagian ini berisi data yang digunakan oleh program.
4. Tabel Eksternal (*Import/Export Table*): Tabel ini berisi daftar fungsi dan modul eksternal yang digunakan atau diekspor oleh program.
5. Informasi Tambahan: *File* PE juga dapat menyertakan informasi tambahan seperti tabel eksepsi, tanda tangan digital, dan lainnya.

2.6.1 *Header File Size of Code* (Ukuran Kode)

Header file Size of Code merupakan bagian dari struktur *Portable Executable* (*PE*) yang menyimpan informasi tentang ukuran total kode yang dieksekusi oleh program. Ukuran ini mencakup semua instruksi dan data yang diperlukan untuk menjalankan program. Informasi ukuran kode ini biasanya terletak dalam *header file PE* dan dapat digunakan untuk memahami struktur dan ukuran program secara keseluruhan [2].

Size of Code dapat diartikan sebagai "Ukuran dari bagian kode, atau jumlah dari semua bagian kode jika terdapat beberapa bagian". Dalam konteks ini, "*size of code*" merujuk pada ukuran total dari bagian-bagian kode program yang disusun untuk dieksekusi oleh komputer [19].

2.6.2 *Address of Entry Point* (Alamat Titik Masukan Pengalamatan)

Address of Entry Point adalah alamat memori di mana eksekusi program dimulai setelah dimuat ke dalam memori. Ini adalah titik masukan utama di mana proses eksekusi program dimulai oleh sistem operasi. Informasi alamat titik masukan pengalamatan ini juga disimpan dalam *header file PE* dan digunakan oleh sistem

operasi untuk menentukan lokasi awal eksekusi program [2]. *Address of Entry Point* relatif terhadap *image base* saat *file executable* dimuat ke dalam memori. Untuk *driver* perangkat, ini adalah alamat dari fungsi inisialisasi. *Entry point* opsional untuk *Dynamic Library Link* (*DLL*). Ketika tidak ada *entry point*, bidang ini harus nol [19].

2.7 *Python*

Python adalah bahasa pemrograman tingkat tinggi yang serbaguna dan mudah dipelajari. Bahasa ini menonjol karena sintaksis yang sederhana dan mudah dimengerti, serta dukungan yang kuat untuk pemrograman berorientasi objek, pemrograman fungsional, dan struktur data yang kaya. *Python* dikembangkan oleh Guido van Rossum dan pertama kali dirilis pada tahun 1991 [9]. Salah satu fitur yang tersedia pada *python* adalah sebagai bahasa pemrograman dinamis yang dilengkapi dengan manajemen memori otomatis [20]. *Python* adalah bahasa pemrograman yang populer yang memberikan banyak manfaat untuk mendukung pemrograman berbasis objek dan dapat dijalankan di berbagai *platform* sistem operasi, termasuk *PC*, *Macintosh*, dan *UNIX* [28].

2.8 *Pefile*

Pefile merupakan sebuah pustaka (*library*) yang ditulis dalam bahasa pemrograman *Python* yang digunakan untuk mengekstrak fitur statis dari *file* eksekusi. Dengan menggunakan *pefile*, peneliti dapat mengakses informasi yang bersifat tetap (statis) dari *file* eksekusi, seperti *header file*, serta struktur data lainnya yang ada dalam *file Portable Executable* (*PE*) yang digunakan dalam sistem operasi *Windows* [11]. *Pefile* merupakan modul *multi-platform* untuk membaca dan bekerja dengan *file PE* serta mengekstrak berbagai informasi dari *file* tersebut [29]. Sebagai sebuah *toolkit* berbasis *Python* untuk menganalisis *file PE*, *Pefile* dapat diandalkan dalam menganalisis *file malware* di *virtual machine* [30].

Beberapa manfaat penggunaan *pefile* dalam analisis *virus* meliputi:

1. Akses Informasi Struktural: *Pefile* memungkinkan analis *virus* untuk mengakses informasi struktural yang tersembunyi dalam *file executable*,

seperti *header file*, tabel eksternal, dan bagian-bagian utama lainnya. Ini membantu dalam memahami struktur dasar dan karakteristik dari *virus* yang sedang diselidiki.

2. Ekstraksi *Metadata*: Dengan menggunakan *Pefile*, analis *virus* dapat mengekstrak *metadata* yang terkait dengan *file executable*, seperti versi, deskripsi, dan informasi pembuat. *Metadata* ini dapat memberikan wawasan tambahan tentang sumber dan tujuan potensial dari *virus*.
3. Analisis *Code Section* dan Data: *Pefile* memungkinkan pengguna untuk menganalisis secara mendalam sebagian besar bagian dari *file Portable Executable*, termasuk sejumlah besar data yang terkandung di dalamnya. Ini membantu dalam mengidentifikasi dan memahami perilaku *virus*, termasuk kode berbahaya dan manipulasi data yang mungkin dilakukan oleh *virus*.
4. Baca *Header File Size of Code* dan *Address of Entry Point*: *Pefile* memungkinkan analis untuk membaca informasi kunci seperti ukuran kode (*Size of Code*) dan alamat titik masukan pengalamatan (*Address of Entry Point*) dari *file Portable Executable*. Informasi ini memberikan gambaran tentang struktur eksekusi program dan dapat digunakan untuk memvalidasi integritas *file* serta mendeteksi modifikasi yang mencurigakan oleh *virus*.

2.9 *Streamlit*

Streamlit adalah sebuah *platform* pengembangan *web* yang dirancang untuk memfasilitasi penyebaran model dan visualisasi dengan menggunakan bahasa *Python* secara cepat dan sederhana, tetapi tetap memberikan tampilan yang menarik dan

mudah digunakan bagi pengguna [16]. Dengan menyediakan berbagai *widget* bawaan, seperti pengunggahan gambar, penggeser, masukan teks, dan elemen *HTML* lainnya, *Streamlit* memungkinkan interaksi yang lancar antara pengguna dan aplikasi. Penting untuk diingat bahwa setiap interaksi pengguna akan menjalankan ulang skrip *Python* dari awal, sehingga manajemen kondisi aplikasi menjadi lebih mudah. *Streamlit* tersedia secara gratis dan dapat digunakan tanpa keahlian khusus dalam pengembangan antarmuka pengguna. *Framework* ini dapat dijalankan di *editor Anaconda* dan membutuhkan bahasa *Python* versi 3.7 ke atas, namun tidak kompatibel dengan *Jupyter Notebook* sehingga memerlukan konversi ke *editor* lain seperti *PyCharm* atau *Visual Studio Code. Streamlit* merupakan *framework* yang tidak berbayar dan pengguna tidak perlu memiliki pengetahuan pengembangan *front-end* yang mahir untuk mengoperasikannya [29]. *Streamlit* memudahkan pengguna untuk mengubah data *script* menjadi aplikasi berbasis *web* yang interaktif. Selain bersifat *open source, streamlit* juga bersifat *open sharing* sehingga mudah untuk dibagikan dan interaktif [30].

2.10 *Jaccard Similarity*

Jaccard similarity adalah metode perbandingan yang berguna dalam membandingkan kesamaan antara dua *set data* [32]. Dalam konteks pengembangan *antivirus*, konsep ini diterapkan dengan membandingkan *metadata* dari *file* yang mencurigakan dengan *database virus*. *Metadata* tersebut dapat mencakup informasi seperti ukuran *file*, struktur kode, dan atribut lainnya. Dengan menggunakan J*accard similarity*, kita dapat mengukur seberapa miripnya *metadata file* yang mencurigakan dengan entri dalam *database virus*. Pendekatan ini berguna karena tidak hanya bergantung pada pola atau tanda tangan *virus* yang sudah diketahui, tetapi juga memperhitungkan kesamaan struktural antara *file* yang baru dan yang sudah teridentifikasi sebagai *virus*. Dengan demikian, penerapan *Jaccard similarity* dalam penelitian ini memungkinkan untuk deteksi dini dan penanganan efisien terhadap ancaman *virus* yang belum dikenali sebelumnya.

Rumus *Jaccard Similarity*:

$$Jaccard\ Similarity(A,B) = \frac{|A \cap B|}{|A \cup B|} * 2$$

Dimana A dan B adalah himpunan, $|A \cap B|$ adalah irisan dari himpunan A dan B, kemudian $|A \cup B|$ adalah gabungan dari himpunan A dan B [31].

Penelitian yang dilakukan oleh Jayesh Soni et al. menunjukkan penggunaan *Jaccard similarity* dalam deteksi anomali sistem pada mesin *virtual Windows*. Hasil penelitian tersebut mengindikasikan bahwa algoritma deteksi anomali dengan menggunakan *Jaccard similarity* dapat memberikan hasil yang efektif dalam mengidentifikasi pola anomali dalam urutan panggilan sistem, tanpa terpengaruh oleh ukuran urutan panggilan tersebut. Adapun aturan yang digunakan dalam penelitian ini adalah jika nilai *similarity* antara kedua urutan panggilan sistem tersebut kurang dari 0.99, maka diputuskan sebagai anomali [33].

Dalam pengembangan *antivirus* menggunakan metode pencarian *header file* berdasarkan *size of code* dan *address of entry* pada tugas akhir ini, aturan *similarity index* yang digunakan adalah 0,8. Tujuan penggunaan nilai ini adalah untuk mengurangi sensitivitas *antivirus* terhadap *false positive*, yaitu deteksi kesalahan di mana suatu *file* dianggap sebagai *virus* padahal sebenarnya tidak. Nilai 0,8 dianggap ideal untuk penggunaan *Jaccard similarity* sebagai algoritma pencocokan *antivirus*. Dalam algoritma ini, *file* yang di-*scan* akan diekstrak *header PE*-nya, kemudian dilakukan pencocokan dengan *Jaccard similarity* terhadap *data header* yang ada di *database antivirus*. Dengan demikian, deteksi *virus* menjadi lebih akurat dan efisien.

2.11 Penelitian Terkait

Berikut adalah beberapa penelitian terkait dengan Skripsi mengenai Sistem *Anti virus* dengan metode pencarian *Header file* data *size of code* dan *address of entry*.

Tabel 2.1 Penelitian Terkait

No	Nama	Tahun	Judul	Hasil
1	Nurul Chafid, Rizky Tsaunauval Braselino	2018	Implementasi Metode Pencarian (*HSC*) *Header Size Of Code* Dan Titik Masukan Pengalamatan Pada Rancangan *Software* Anti *Virus*	Dari hasil teknik uji coba dari sistem *anti virus* ini dapat di ambil kesimpulan bahwa: a) Teknik penggunaan *header file address of entry point* dan *size of code* cukup akurat dalam mengenal *virus* walau sudah merubah *header* filenya tapi datanya masih sama. b) Sebagaimana *file executable Windows* biasanya, program *Worms* komputer memiliki struktur *PE file*, tetapi tidak tertutup kemungkinan terjadi reduksi pada *DOS header*. Kunci utama pendeteksian *virus* komputer adalah ketersediaan *signature* unik yang digunakan sebagai pola pendeteksian. Pengambilan beberapa *byte* awal *executable virus* pada posisi *entry point* dapat dijadikan sebagai *signature* untuk

No	Nama	Tahun	Judul	Hasil
				mendeteksi *worms* tersebut. c) Telah berhasil membuat anti *virus* dengan metode pencarian *virus signature* (*pattern virus*) menggunakan pendekatan *size of code* dan *address of entry point* (titik masukan pengalamatan)
2.	Hari Purwanto	2015	Teknik Pembuatan Anti *Virus* Dengan Metode Pencarian Data *Header File* Menggunakan *Visual Basic* 6.0	Dari hasil teknik uji coba dari sistem *anti virus* ini dapat di ambil kesimpulan bahwa: 1.Teknik penggunaan *header file addres Of Entry point* dan *size of code*, Sangat akurat dalam mengenal *virus*, walaupun *virus* telah merubah *header* filenya tapi datanya tetap sama. 2.*Engine scanner* nya juga cepat dan ringan tidak terlalu memberatkan memori. *Anti virus* ini juga perlu di tingkatkan sensitifitasnya, karena bila terdapat *section* *dummy* pada urutan *section* kedua atau tidak ada pada section, atau pembelokan pada *entrypoint* nya maka data yang di teliti bisa saja

No	Nama	Tahun	Judul	Hasil
				salah, Fungsi ini juga masih rawan dari kesalahan analisa. Maka dari itulah butuh penelitian lebih lanjut, dengan menambahkan algoritma baru fungsi ataupun prosedur, agar dapat mengembangkan dan memajukan kualitas aplikasi anti *virus*.
3.	Rohmad Sufono, Bedri Purba Alam, Andrian Febri Yudi Pratama	2011	Pembuatan Aplikasi *Antivirus* Menggunakan Bahasa Pemrograman *Microsoft Visual Basic 6.0*	Kami sebagai penulis menyadari bahwa masih banyak kekurangan dalam pembuatan penelitian ini, maka untuk ke depan pembuatan *antivirus* kami harapkan akan ada beberapa pengembangan dan juga tambahan beberapa fitur yaitu: 1. Akan dibuat dengan *Microsoft Visual Basic* 2010 untuk *SO* terbaru 2. Akan tersedia untuk versi *linux* dan versi *mobile* dan *smartphone*. 3. Ada tambahan fitur Browser *security toolbar*, *anti spyware, Anti Phishing, , Heuristics engine, Smart virus definition updates, repaire, quarantine.*

No	Nama	Tahun	Judul	Hasil
				4. Akan ada manual update untuk *PC* yang belum memiliki koneksi internet.
4	Sugeng Santoso, Lusyani Sunarya	2008	Rekayasa Teknik Pemrograman Pencegahan Dan Perlindungan Dari *Virus* Lokal Menggunakan *API Visual Basic*	Dilihat dari teknik infeksi sistem, penggandaan ke sistem dan infeksi *registry*, *virus* lokal yang dibuat dengan bahasa pemrograman *visual basic* mudah untuk diciptakan bagi yang sedikit mengerti dan mudah pula untuk dimusnahkan bagi yang sedikit mengerti, hal ini sangat tergantung dari tingkat pemahaman. *Virus* lokal yang dibuat dengan bahasa *visual basic*, hampir memiliki kemiripan teknik, hal ini dapat dilihat dari *registry* yang diakses. Pada *registry* inilah *virus* akan melakukan beberapa pengaturan, dengan tujuan melancarkan aktivitasnya tanpa harus diketahui pengguna komputer. Program rekayasa *virus* yang dibuat dalam pembahasan ini, dimaksudkan saling memberi informasi tentang bagaimana *virus* dibuat

No	Nama	Tahun	Judul	Hasil
				dengan *visual basic*, dan diantara kemampuannya adalah dalam hal pengaturan *registry* agar pada saat *start virus* tersebut secara otomatis berjalan. Dalam memblokir beberapa fasilitas penting, seperti *regedit*, *msconfig*, *task manager*, *cmd* dan lain sebagainya merupakan bagian dari pertahanan.
5.	Richki Hardi	2015	Pengembangan *Antivirus* Berbasis *Client Server*	Proses pemakaian *antivirus* tersebut masih terbilang sederhana maksudnya adalah dalam proses tersebut server memilih *drive* atau *folder* mana yang perlu diperiksa atau dihapus virusnya pada komputer klien. Harapan untuk menjadi saran agar dapat ditingkatkan menjadi *antivirus* yang otomatis mendeteksi jenis *virus* dan menghapusnya lewat jaringan tanpa harus memilih secara manual.
6.	Ardiansyah, Cucu Suhery, Ilhamsyah	2014	Pengembangan *Antivirus* Menggunakan Metode *Heuristic* Ganda Dan Sistem *Realtime*	Setelah dilakukan pengujian dan analisis terhadap antivirus yang dikembangkan (*Spartan Antivirus*), maka dapat ditarik kesimpulan sebagai berikut:

No	Nama	Tahun	Judul	Hasil
			Protector Serta Perbandingannya Dengan *Antivirus* lokal.	1. Cara untuk mendeteksi, melumpuhkan (mematikan kinerja *virus*) dan menghapus *virus* komputer dengan metode *heuristic* ganda adalah dengan menguji dan menganalisa sampel *virus*, kemudian membuat metode heuristicnya berdasarkan ciri-ciri maupun tingkah laku dari virus tersebut. 2. Sistem *realtime protector* sangat efektif dalam melindungi sistem komputer dari serangan *virus*, karena
				sistem ini mampu mendeteksi keberadaan *virus* walaupun *antivirus* tidak sedang melakukan proses *scanning*. Sistem *realtime* protector telah diuji sebanyak 10 kali dengan sampel *malware* (*virus*, *worms* dan *trojan*) dan didapat rasio 100% dalam pengujian tersebut. 3. Pada perbandingan fitur beberapa *antivirus* lokal, *antivirus* yang dikembangkan memiliki kelebihan dari *antivirus*

No	Nama	Tahun	Judul	Hasil
				lokal lain (*Smadav*, *PC* Media *Antivirus* dan *Morphost Anti-virus*) yaitu memiliki sebuah fitur dimana *user* dapat menambahkan secara manual *signature* sebuah *file* yang dicurigai sebagai *virus* ke database *antivirus*. Kekurangan dari *antivirus* ini adalah belum memiliki *website* resmi sebagai sarana untuk publikasinya. 4. Pada perbandingan *scanning* beberapa *antivirus* lokal, *antivirus* yang dikembangkan memiliki
				rasio atau akurasi pendeteksian sebesar 94%. Kelebihan dari *antivirus* ini adalah akurasi pendeteksiannya sudah cukup tinggi, bahkan untuk *virus* yang sebelumnya masih luput dari pendeteksiannya, akan bisa dideteksi dengan cara menambahkan *signature virus* tersebut secara manual ke *database antivirus* menggunakan fitur *add sign* *virus manually*. Kekurangan antivirus ini adalah waktu yang diperlukan dalam

No	Nama	Tahun	Judul	Hasil
				proses *scanning* belum secepat *antivirus* lainnya.
7.	Fauzi Adi Rafrastara, Ricardus Anggi Pramunendar, dwi Puji Prabowo, Etika Kartika Darma, Usman Sudibyo	2023	Optimasi Algoritma Random *Forest* Menggunakan *Principal Component* Analysis Untuk Deteksi *Malware*	Penelitian tentang peningkatan performa algoritma Random *Forest* menggunakan *PCA* telah selesai dilakukan. Random *Forest* dipilih karena memiliki performa Akurasi dari *Recall* terbaik dibandingkan 4 algoritma lain, seperti: *Adaboost*, *Neural Network*, *Support Vector Machine* dan *k-Nearest Neighbor*. Performa yang diperoleh Random *Forest* pada Dataset *malware* yang telah disiapkan adalah 98.3%, baik untuk Akurasi maupun Recall. Selanjutnya eksperimen dilakukan dengan melakukan fitur reduksi pada dataset mengingat dataset yang digunakan tersebut memiliki fitur berjumlah 1084. Setelah fitur direduksi menjadi 32 *PC* dan memperhatikan jumlah *Variance* pada angka 84%, performa algoritma Random *Forest* meningkat menjadi 8.44%. Berdasarkan data pada tabel

No	Nama	Tahun	Judul	Hasil
				confusion matrix, dari total 1190 *file* dimana 595 diantaranya adalah *file malware*, algoritma Random *Forest* yang semula memiliki *False Negative* berjumlah 10, berhasil dikurangi menjadi 6 setelah dilakukannya reduksi fitur dengan *PCA*. Hal ini menunjukkan adanya peningkatan pada algoritma Random *Forest*, karena semakin sedikit jumlah kesalahan deteksi, khususnya mendeteksi *file* Pada penelitian selanjutnya, penurunan angka *false negative* dan *false positive* perlu diperhatikan untuk mencapai prinsip *zero tolerance* dalam deteksi *malware*. Selain itu, dataset malware juga perlu di *update*, mengingat dataset yang digunakan ini dibuat pada tahun 2019 sementara perkembangan *malware* terus meningkat dengan pesat.
8.	Susi Yanti Manalu, Jamaluddin	2022	Analisis Keamanan Anti *Virus* Berbasis *Web*	Berdasarkan teori dan Analisa percobaan,maka dapat disimpulkan bahwa kemampuan penambahan

No	Nama	Tahun	Judul	Hasil
				database pada sistem *antivirus* dapat memudahkan pengguna *antivirus* dalam mengupdate saat mengoperasikan komputer dan pencarian *file antivirus* dapat dilakukan dengan menghitung setiap *file* pada suatu pustaka, jika tidak ditemukan database maka *file* itu tidak disebut sebagai *virus*
9.	Muhammad Ilhamdi Rusydi, Meza Silvana, Aidil Akbar	2011	Perancangan Perangkat Lunak *Antivirus* Dengan Metode *Scanning Cyclic Redundancy Cheksum-32*	Berdasarkan analisa terhadap hasil percobaan yang didapat, maka dapat diambil beberapa simpulan sebagai berikut: 1. Kemampuan penambahan *database* secara manual oleh pengguna pada sistem *antivirus*, mampu memudahkan pengguna untuk mengupdate database *antivirus* setiap saat berbasis objek *virus* yang ada pada sebuah sistem operasi. Dan pengguna bisa menghemat pemakaian media penyimpanan dengan mengisi database sesuai *file virus* yang akan menjadi target pencarian pada sebuah sistem operasi komputer. Nilai *crc32* yang dihitung

No	Nama	Tahun	Judul	Hasil
				dari sebuah *file virus* digunakan sebagai *signature* atau *fingerprint* (sidik jari) dari *virus* tersebut dalam mendeteksi keberadaan *virus* tersebut dalam *directory* sistem operasi komputer. 2. Pencarian *file virus* oleh sistem *antivirus* dilakukan dengan menghitung nilai *crc32* dari setiap *file* yang ada pada suatu directory, kemudian nilai *crc32* yang diperoleh akan dibandingkan dengan nilai *crc32* yang terdapat pada database, jika tidak terdapat dalam *database* maka *file* tidak dapat dianggap sebagai *virus*.
10.	Rita Rahmawati, Agus Nursikuwagus	2012	Perancangan *Antivirus* Dengan Menggunakan Metode *MD5* dan *Heuristik ARRS*	Berdasarkan hasil yang didapat dalam penelitian ini, maka diperoleh beberapa kesimpulan sebagai berikut : 1. Metode *MD5* dapat digunakan dalam proses identifikasi *virus file* yang berextensi *EXE*, *DLL*, *VBS*, *VMX*, *DB*, *COM*, *SCR* dan *BAT* dengan membandingkan *checksum* yang didapat dengan

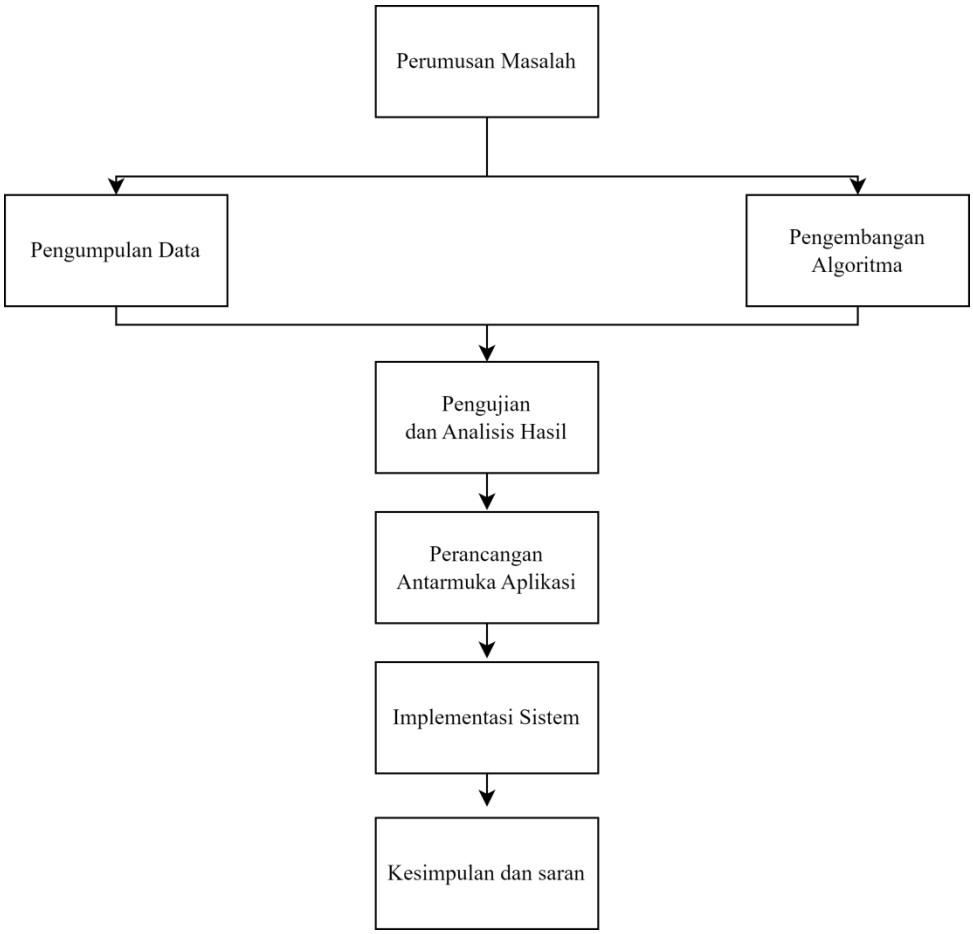
No	Nama	Tahun	Judul	Hasil
				checksum di database *Virus*. 2. *Heuristik Arrs* dapat digunakan sebagai pendekatan untuk identifikasi *virus file* dengan memanfaatkan *file Autorun.inf*. *Antivirus* yang dibuat dapat melakukan fungsi standar sebuah *antivirus*.

BAB 3

METODOLOGI PENELITIAN

3.1 Tahapan Penelitian

Berikut adalah tahapan-tahapan penelitian pada gambar 3.1:



Gambar 3. 1 Tahapan Penelitian

Penjelasan dari tahapan-tahapan penelitian pada gambar 3.1 dapat dilihat pada penjelasan berikut:

3.2 Perumusan Masalah

Pada tahap ini, dilakukan identifikasi dan perumusan masalah yang akan dipecahkan dalam penelitian ini. Penulis akan menjelaskan dengan jelas apa yang menjadi fokus penelitian dan mengapa masalah tersebut penting untuk diteliti. Dalam penelitian ini, dapat dijelaskan bahwa masalah yang ingin dipecahkan adalah bagaimana mengembangkan sistem *antivirus* yang efektif untuk mendeteksi *virus* dengan metode pencarian *Header File Data Size Of Code* dan *Address Of Entry*.

3.3 Pengumpulan Data

Langkah selanjutnya adalah mengumpulkan data yang diperlukan untuk penelitian. Ini termasuk mengumpulkan sampel *virus* yang akan digunakan untuk pengujian dan evaluasi sistem *antivirus*. Disini dijelaskan sumber data yang digunakan dan proses pengumpulannya untuk memastikan keandalan dan representativitas data yang digunakan dalam penelitian.

3.4 Pengembangan Algoritma

Pada tahap pengembangan algoritma untuk sistem *antivirus*, dilakukan penelaahan dan adaptasi terhadap algoritma yang dipaparkan dalam karya ilmiah berjudul “Teknik pembuatan *antivirus* dengan metode pencarian data *header file* menggunakan *visual basic* 6.0” [14]. Langkah awal melibatkan pemahaman mendalam terhadap algoritma pencarian yang diusulkan, terutama fokus pada teknik pencocokan pola pada *header file PE*.

Setelah memahami prinsip dasar algoritma tersebut, langkah berikutnya adalah menerapkannya menggunakan bahasa pemrograman *Python*. Pemilihan bahasa

pemrograman ini didasarkan pada fleksibilitas dan kemudahan implementasi yang ditawarkannya.

Selain itu, dalam pengembangan algoritma ini, dipertimbangkan juga pemilihan pustaka atau *library* yang relevan untuk mendukung implementasi algoritma secara efektif. Langkah ini penting untuk memastikan keefektifan dan efisiensi algoritma yang dikembangkan dalam konteks sistem *antivirus* yang akan dibangun.

3.5 Pengujian dan Analisis Hasil

Setelah pengembangan algoritma berhasil, langkah selanjutnya adalah melakukan pengujian sistem dan menganalisis hasilnya. Dalam pengujian, akurasi pendeteksian sistem *antivirus* dievaluasi menggunakan metrik yang sesuai, seperti tingkat deteksi (*detection rate*) atau tingkat kesalahan positif (*false positive rate*). Tingkat deteksi mengukur seberapa baik sistem mampu mengidentifikasi *virus*, sedangkan tingkat kesalahan positif mengukur seberapa sering sistem memberikan peringatan palsu tentang keberadaan *virus* ketika sebenarnya tidak ada. Pengujian diperkirakan akan memakan waktu sekitar 2 jam dimana akan diuji seberapa banyak *virus* yang benar-benar terdeteksi sebagai *virus*. Untuk pengujian ini, akan disiapkan sebuah *folder* yang berisi 10 *file* "*.exe*" yang tidak mengandung *virus*, 10 *file virus* dengan ekstensi "*.exe*", dan 30 *file* acak bukan *virus* yang tidak memiliki ekstensi "*.exe*". Pengujian dilakukan dalam lingkungan *virtual machine Windows* 10 yang dapat terkoneksi ke *internet* dengan bantuan *software virtualbox*. Hal ini dilakukan

untuk memastikan bahwa komputer yang digunakan untuk pengujian tidak terinfeksi oleh *virus* secara langsung.

3.6 Perancangan Antarmuka Aplikasi

Pada tahap ini, penulis merancang antarmuka aplikasi untuk sistem *antivirus* yang dibangun. Antarmuka aplikasi ini memainkan peran penting dalam penggunaan sistem oleh pengguna.

3.7 Implementasi Sistem

Tahap ini melibatkan implementasi sistem *antivirus* berdasarkan rancangan dan algoritma yang telah dikembangkan sebelumnya.

3.8 Kesimpulan dan Saran

Dalam bagian ini, dijelaskan kesimpulan dari penelitian yang telah dilakukan terkait implementasi dan pengembangan sistem *antivirus* dengan metode pencarian *Header File Data Size Of Code* dan *Address Of Entry*. Selain itu, penulis memberikan rekomendasi bagi pembaca untuk melanjutkan pengembangan penelitian ini di masa mendatang.