Perkembangan Teknik Penyembunyian Malware

Christopher Calvin

Program Studi Sistem dan Teknologi Informasi

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

18217039@std.stei.itb.ac.id

Abstract*—*Malware* adalah nama lain dari *Malicious Software* yang merupakan perangkat lunak yang didesain untuk menyebabkan kerusakan pada data atau sistem. Kamuflase dari malware merupakan sebuah tantangan besar bagi *cyber security*. Kamuflase digunakan *malware* agar *malware* tidak mudah dideteksi dan siklus hidup dari *malware* tersebut bisa menjadi lebih panjang. Salah satu cara menangani ancaman tersebut adalah dengan menggunakan *anti-virus*. *Anti*-virus bekerja dengan mendeteksi keberadaan dari *malware* dan menghapusnya. Namun, teknologi *anti-virus* tersebut harus terus berkembang untuk dapat mengatasi ancaman *malware* yang seringkali tersembunyi sehingga tidak bisa dideteksi. ujuan dari makalah ini adalah mengamati perkembangan teknik penyembunyian *malware* untuk dapat mengantisipasi perkembangan selanjutnya agar dapat meningkatkan efektivitas dari teknologi *anti-virus

***Keywords*—malware, anti-virus, penyembunyian, deteksi**

1. PENDAHULUAN

1.1. Latar Belakang

Akhir-akhir ini, teknologi menjadi sebuah bagian penting dalam kehidupan masyarakat. Teknologi mulai digunakan dalam setiap aspek kehidupan. Teknologi ada dalam kehidupan sehari hari dalam bentuk gawai, ATM, *website,* layanan internet, dll. Kepentingan teknologi dapat dibuktikan dengan ramainya penggunaan pembelajaran online, *work from home*, meeting online, dan lain-lain yang diakibatkan dengan pandemik *Covid-19 Virus* yang sedang menjadi fokus dunia.

Dengan adanya teknologi, keamanan informasi secara tidak langsung menjadi hal yang sangat penting. Teknologi memudahkan akses informasi namun tidak semua informasi dapat diberikan pada publik. Tidak ada sistem informasi yang tidak memiliki celah. Karena itu, mulai muncul ancaman-ancaman baru yang

menyalahkgunakan celah dari teknologi-teknologi tersebut. Salah satu bentuk dari ancaman tersebut adalah *malware*.

Malware (*Malicious Software*) adalah sebuah perangkat lunak yang dirancang sedemikian rupa untuk dapat menyebabkan kerusakan pada suatu infrastruktur atau sistem infrastruktur. *Malware* sudah ada sejak lama. Pada tahun 1970, muncul sesuatu yang dikenali sebagai *malware* pertama yaitu *creeper virus*. *Creeper virus* merupakan program yang dibuat oleh Bob Thomas sebagai eksperimen. *Creeper virus* pada dasarnya adalah sebuah program yang dapat menduplikasikan dirinya yang dibuat tidak untuk merusak namun hanya untuk mendemonstrasikan potensi bahaya dari virus. Eksperimen ini membuktikan bahwa *malware* merupakan sebuah ancaman yang besar bagi sistem.

Malware dapat ditangani jika dapat dideteksi oleh program *anti-virus.* Karena itu, pembuat *malware* berusaha membuat programnya agar tidak mudah dideteksi dengan teknik kamuflase. Kamuflase pada *malware* tidak membuat *malware* menjadi tidak bisa dideteksi. Tujuan dari kamuflase adalah untuk membuat *malware* menjadi tidak mudah terdeteksi guna memperpanjang siklus hidupnya. Kamuflase *malware* terus berkembang untuk dapat menghindari deteksi oleh *anti-virus* yang ada. Dengan adanya kamuflase *malware,* teknik-teknik deteksi baru harus diciptakan untuk dapat mengatasi ancaman *malware* tersebut.

1.2. *Rumusan Masalah*

Berikut merupakan rumusan masalah yang akan dibahas:

1) Apa itu *malware*?
2) Bagaimana perkembangan teknik kamuflase *malware*?
3) Bagaimana cara *anti-virus* mendeteksi adanya *malware*?

1.3. *Rumusan Masalah*

Berikut merupakan rumusan masalah yang akan dibahas:

1) Mengetahui definisi itu *malware*?
2) Mengetahui perkembangan teknik kamuflase *malware*?
3) Mengetahui cara *anti-virus* mendeteksi adanya *malware*?

1.4. *Metodologi*

Pada makalah ini, penulis menggunakan metode analisis terhadap data dari kajian pustaka. Makalah akan dibagi menjadi tiga bagian utama, yaitu pendahuluan, pembahasan, dan penutup. Berikut ini adalah deskripsinya.

Tabel 1 Bagian Makalah

No.	Bagian	Deskripsi	Isi Bagian
1.	Pendahuluan	Pendahuluan berisi penjelasan singkat mengenai topik dan isi dari makalah	Latar Belakang
			Rumusan Masalah
			Tujuan
			Metodologi
2.	Pembahasan	Pembahasan berisi inti dari makalah yang mengkaji masalah yang sudah didefinisikan pada rumusan masalah	Definisi *malware*
			Perkembangan teknik kamuflase *malware*
			Teknik deteksi *malware*
3.	Penutup	Penutup berisi kesimpulan dari pembahasan yang telah dilakukan di bagian sebelumnya.	Kesimpulan
			Saran

2. PEMBAHASAN

2.1. *Definisi malware*

A malware instance is a program that has malicious intent. Examples of such programs include viruses, trojans, and worms [1]. Menurut kutipan tersebut, *Malware* (*Malicious Software)* adalah program yang dibuat dengan tujuan jahat. Jahat dalam konteks ini berarti bertujuan untuk mengganggu keberjalanan sistem informasi yang ada. *Malware* biasanya diklasifikasikan menjadi berbagai bentuk berdasarkan cara *malware* tersebut menyebarkan dirinya kedalam sistem dan aksi yang dilakukan kedalam sistem menggunakan program yang telah didesain sebelumnya. Malware juga dapat diklasifikasikan sebagai *network based malware* (*malware* yang menginfeksi host menggunakan bantuan jaringan) dan *ordinary malware* (*malware* yang dapat menyebabkan kerusakan tanpa membutuhkan koneksi internet) [3]. Contoh bentuk program tersebut adalah *virus*, *trojan*, *ransomware*, dan *worm*.

Network based malware pada umumnya tidak ditujukan untuk merusak sistem, melainkan ditujukan untuk mengambil informasi. Salah satu contoh program ini adalah *spyware*. *Spyware* memasuki komputer user tanpa sepengetahuan user dan bertujuan untuk mengambil informasi. Perangkat lunak ini memonitor dan mengumpulkan informasi dari computer user dan mengirimkan informasi tersebut kembali ke

penyerang yang mengirimkan *spyware* tersebut. Contoh lainnya adalah *backdoor*. Seperti Namanya, *backdoor* memberikan akses kepada penyerang untuk dapat memanipulasi aplikasi atau OS dari user tanpa sepengetahuan user tersebut. Hal ini dapat dilakukan dengan membuka sebuah *port* dari komputer target.

Ordinary malware adalah *malware* yang tidak membutuhkan koneksi internet untuk melakukan aksinya. *Malware* ini biasanya bertujuan untuk merusak sistem. Program ini tidak mengirimkan informasi kembali ke penyerang sistem. Contoh dari program ini adalah *virus.* Pada umumnya, semua *malware* seringkali dianggap sebagai *virus.* Definisi *computer virus* adalah program yang mereplikasi dirinya sebagai bagian dari program lain dan bekerja dengan instruksi *carrier program* yang merupakan program lain yang diinfeksi tersebut [3]. *Virus* tidak bisa mengeksekusi dirinya sendiri. Contoh lain dari *ordinary malware* adalah *worm*. *Worm* pada dasarnya sama dengan *virus* namun *worm* dapat mengeksekusi dirinya sendiri sehingga lebih berbahaya dari *virus*. *Worm* dan *virus* sama-sama dapat menginfeksi komputer lain mengikuti persebarannya.

Malware dapat ditangani dengan mudah jika sudah terdeteksi *malware.* Karena itu, para pembuat *malware* berusaha untuk menyembunyikan programnya dengan berbagai cara. Teknik kamuflase ini tidak memastikan bahwa *malware* tidak akan terdeteksi. Hal itu tidak mungkin karena teknik deteksi *malware* terus berkembang seiring berkembangnya teknik kamuflase *malware*. Teknik kamuflase ini bertujuan untuk meningkatkan waktu yang dibutuhkan untuk mendeteksi dan menganalisa *malware* [1]. Hal ini dilakukan untuk memperpanjang siklus hidup dari *malware* tersebut sehingga dalam waktu yang dibutuhkan untuk mengatasi ancaman *malware* tersebut, program itu dapat menyebar lebih luas dan menyebabkan kerusakan yang sebesar mungkin.

2.2. Perkembangan teknik kamuflase malware

Dari pertama kali munculnya *malware*, terdapat semacam kompetisi antara *attacker* (pembuat *malware*) dan *defender* (umumnya spesialis *anti-virus*). Kedua belah pihak berkompetisi untuk mengalahkan pihak lainnya. *Attacker* membuat *malware* yang sulit dideteksi sedangkan *defender* membuat program untuk mendeteksi *malware* tersebut. Kompetisi tersebut mulai dari saat pertama kali muncul *malware* yaitu dari tahun 1970. Hingga sekarang, terdapat 4 macam teknik kamuflase yang umumnya digunakan oleh pembuat program *malware* mulai dari teknik enkripsi, Oligomorfisme, Polimorfisme, dan Metamorfisme [1]. Namun, teknik-teknik tersebut bukan merupakan akhir dari perkembangan penyembunyian *malware*. Masih mungkin ada perkembangan lainnya seiring dengan berkembangnya teknologi

2.2.1. Primitive Malware

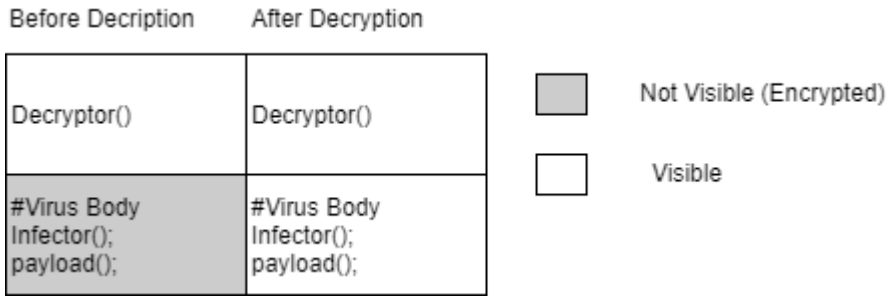
Pada awalnya, pembuat *malware* tidak perlu memikirkan tentang penyembunyian *malware*. Hal ini dikarenakan *malware* baru saja muncul dan belum ada sarana atau program yang digunakan untuk

menghentikan kerja *malware*. Teknologi yang digunakan untuk mengidentifikasi, menganalisa, dan menghapus program itu belum ada.

2.2.2. Encrypted Malware

Malware menggunakan teknik ini dengan cara mengenkripsi dirinya dan aktivitas yang dilakukannya. *Malware* yang terenkripsi terdiri dari dirinya sendiri, algoritma enkripsi, kunci enkripsi, algoritma dekripsi, dan kode perusaknya. Kunci dan algoritma dekripsinya digunakan untuk mendekripsi kode perusak dari dirinya sendiri saat *malware* tersebut beraktivitas. Setelah program tersebut berjalan, bagian perusak tersebut dienkripsi kembali sehingga tidak bisa terdeteksi. Kunci enkripsi tersebut terus berubah saat dilakukan dekripsi dan enkripsi [3].

Struktur virus terenkripsi terdiri dari 2 bagian yaitu *encrypted body* dan *decryption body* [2]. Sebelum dilakukan dekripsi, bagian *encryption body* dari program tidak dianggap sebagai kode berbahaya oleh pemindai virus. Saat program yang terinfeksi di-run, yang pertama kali dilakukan adalah mengeksekusi *decryption body*. Dengan itu, bagian *encrypted body* yang berisi kode perusak dapat digunakan. Setelah itu, program melakukan *jump* untuk mengeksekusi kode perusak yang telah didekripsi di tahap sebelumnya. Gambar 1 menunjukkan struktur dari *malware* dengan teknik kamuflase enkripsi.



Gambar 1 Struktur dari virus terenkripsi

Menurut Skulason, 4 motivasi dilakukannya enkripsi adalah [6]:

1) Menghindari analisa kode static

   Beberapa program menganalisa kode secara otomatis dan memberikan peringatan jika kode berpotensi berbahaya.

2) Menunda proses inspeksi

   Proses analisa kode dapat dibuat lebih sulit dan memakan waktu meskipun tidak begitu lama.

3) Menghindari perubahan dari virus

   Mempersulit pengubahan kode virus agar tetap sesuai dengan aksi yang telah dirancang.

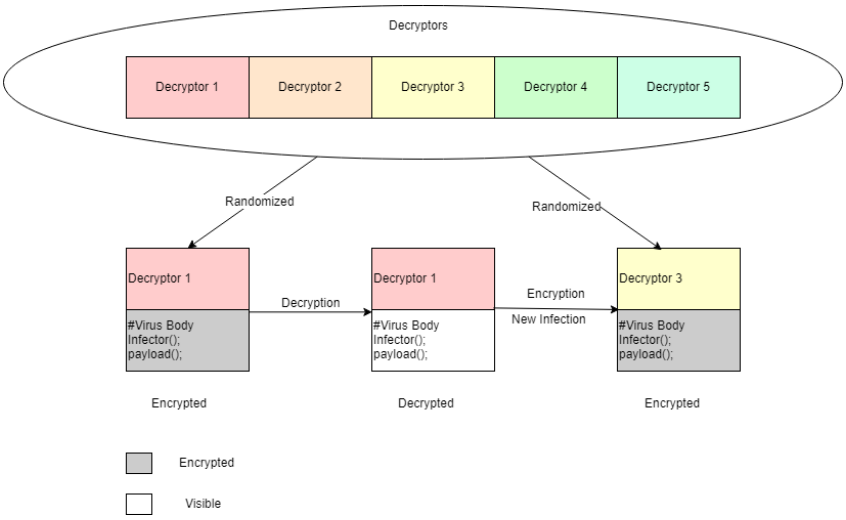
4) Menghindari deteksi

   Virus terenkripsi tidak dapat dideteksi hanya dengan program pengevaluasi *string* karena hanya decryptor loop yang terlihat.

2.2.3. Oligomorfisme

Dalam mendeteksi *malware* terenkripsi, yang tidak bisa dilihat hanyalah body yang berisi kode perusak. Hal ini disebabkan oleh bagian tersebut terenkripsi. Namun, kelemahannya adalah bagian dekripsi tidak disembunyikan. Hal ini membuat deteksi *malware* terenkripsi menjadi lebih mudah dengan melihat decryptor loop yang tidak berubah. Untuk melalui kelemahan ini, Pembuat *malware* mencoba untuk mengubah juga bagian *decryptor*nya sehingga tidak mudah dikenali oleh program *anti-virus* tersebut. Akibatnya, muncul teknik kamuflase baru yaitu teknik oligomorfisme [2].

Dalam penyembunian *malware*, konsep dari oligomorfisme adalah membuat bagian *decryptor* dari *malware* berubah ubah sehingga tidak dapat dengan mudah dikenali. Dalam setiap infeksi baru, bentuk dari *decryptor* berbeda beda. Hal ini dapat dilakukan dengan membuat beberapa bentuk *decryptor* dan memilih (secara acak) *decryptor* yang akan digunakan dalam setiap infeksi baru. Dengan itu, bentuk *malware* tidak sama seperti infeksi sebelumnya. Gambar 2 menjelaskan bentuk dan cara kerja dari *oligomorphic virus*.



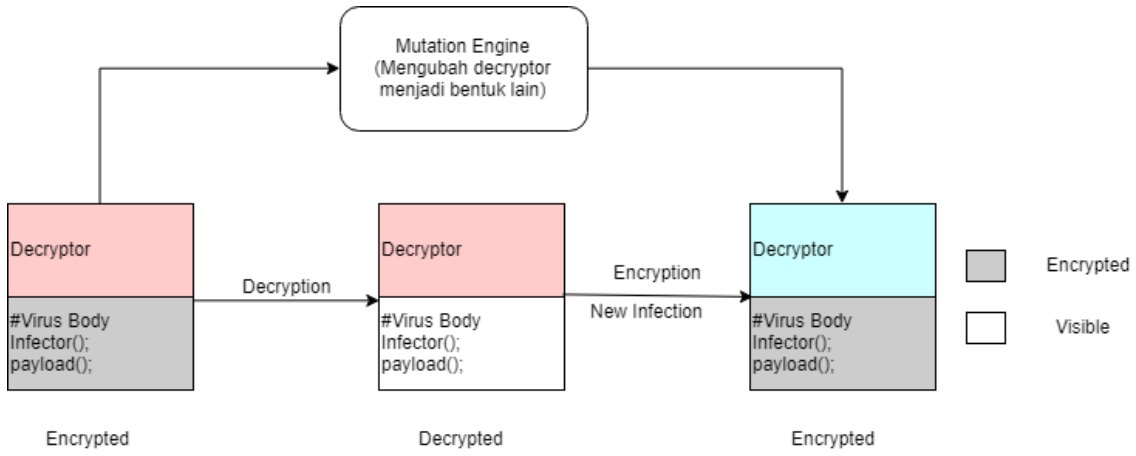
Gambar 2 Struktur dan mekanisme oligomorfisme

Bagi proses deteksi, oligomorfisme bukan sebuah masalah besar karena konsep oligomorfisme hanya menyebabkan *malware* sedikit sulit untuk dianalisa. Jika *anti-virus* sudah memiliki gambaran mengenai *decryptor* yang mungkin digunakan, maka mendeteksi *malware* dengan oligomorfisme tidak sulit. Namun, proses pengecekan tersebut akan tetap lebih lama karena *anti-virus* harus melakukan pengecekan terhadap semua *decryptor* yang mungkin digunakan. Virus pertama dengan kapabilitas oligomorfisme merupakan *Whale Virus* yang adalah virus DOS (Denial of Service) yang muncul pada tahun 1990 [1].

2.2.4. Polimorfisme

Melihat kelemahan dari konsep oligomorfisme, virus oligomorfisme tidak sulit dikenali karena badan *decryptor*nya terbatas. Virus oligomorfisme hanya bisa berubah menjadi beberapa bentuk sesuai dengan jumlah badan *decryptor* yang telah dideskripsikan sebelumnya. Konsep polimorfisme sangat mirip dengan konsep oligomorfisme. Namun, polimorfisme menghilangkan kelemahan dari oligomorfisme tersebut. Struktur dari poliformisme sama dengan *malware* terenkripsi biasa yaitu terdiri dari *decryptor* dan badan yang terenkripsi. Perbedaan utama antara polimorfisme dengan oligomorfisme adalah *malware* dengan kapabilitas poliformisme dapat merubah dirinya menjadi bentuk yang jumlah variasinya tidak terbatas.

Untuk melakukan mutasi (perubahan bentuk virus, pembuat *malware* memanfaatkan konsep *code obfuscation*. *Code obfuscation* adalah konsep untuk menghindari *signature based detection* dengan menambahkan "sampah", *jump* yang tidak dibutuhkan, mengganti nama register, mengubah urutan kode, dll [3]. Hal itu dilakukan di setiap infeksi baru yang ada. Dengan menggunakan itu, tidak ada bagian kode yang permanen (menjadi *signature* dari kode) sehingga sulit untuk dideteksi oleh *anti-virus*. Bagian yang bertanggung jawab untuk mengubah kode disebut dengan *mutation engine* atau *obsfucation engine*. Pada umumnya, hal yang dilakukan dalam bagian tersebut adalah pengubahan instruksi, permutasi instruksi, pengubahan variabel/register, penambahan kode "sampah" (kode yang tidak mengubah apapun), transposisi kode. Proses dari dekripsi, eksekusi, dan enkripsi kembali (infeksi baru) mirip dengan konsep enkapsulasi pesan pada jaringan komputer. Gambar 3 menggambarkan struktur dan cara kerja dari kode dengan kapabilitas poliformisme.



Gambar 3 Struktur dan cara kerja poliformisme

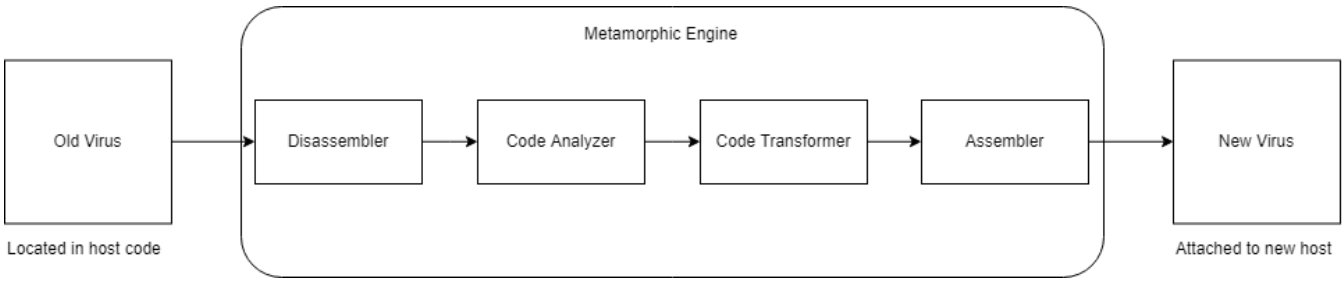
Tidak peduli sebaik apapun kode poliformisme tersebut didesain, setelah program anti-virus menemukan bentuk kode tersebut secukupnya, bagian terenkripsi yang tersembunyi akan terdeteksi dan dapat dilakukan pengecekan dengan mengecek string tersebut [1]. Hal ini disebabkan karena kode polimorfisme hanya mengubah bagian *decryptor*nya.

2.2.5. Metamorfis

Melihat dari kelemahan-kelemahan sebelumnya, kelemahan utama *malware* polimorfisme adalah setelah beberapa kali ditemukan, tanda utama (*signature*) dari virus tersebut dapat diketahui. Dengan itu, pengecekan virus dapat dilakukan dengan mudah. Metamorfisme, sebagai teknik terbaru dalam kamuflase virus, bertujuan untuk menghilangkan kelemahan dari teknik-teknik sebelumnya. Karena itu, konsep utama dari teknik metamorfisme adalah mengubah seluruh kodenya tanpa mengubah fungsionalitas dari kode tersebut. Hal ini dilakukan untuk dapat menghindari deteksi dengan mengamati *signature* dari kode tersebut. Menurut Peter Szor, definisi tersingkat dari virus dengan kapabilitas metamorfisme adalah "methamorphics are body-polymorphics." [7]. Artinya, Metamorfisme adalah *malware* dengan kemampuan untuk mengubah isinya menjadi berbagai bentuk hingga jumlah yang tak terbatas.

Struktur kode metamorf berbeda dengan teknik-teknik sebelumnya. Pada teknik-teknik sebelumnya, kode biasanya terdiri dari *decryptor* dan *encrypted body*. Pada teknik ini, karena seluruh bagian kode diubah, maka tidak diperlukan lagi teknik enkripsi. Akibatnya, kode hanya terdiri dari bagian utamanya. Bagian utama tersebut akan diubah ubah bentuknya tanpa mengubah fungsionalitasnya dengan menggunakan *metamorphic engine*. Untuk melakukan perubahan, hal yang dibutuhkan adalah *metamorphic engine*. *Metamorphic engine* pada umumnya terdiri dari *disassembler*, *code analyzer*, *code transformer*, dan *assembler*. Secara singkatnya, kode baru akan diproduksi setelah kode lama melalui *metamorphic engine*. Kode baru tersebut merupakan kode yang digunakan untuk menginfeksi *host* lain.

Cara kerja *metamorphic engine* adalah kode lama akan dipecah menjadi beberapa instruksi Bahasa *assembly* melalui *disassembler*. Kemudian, instruksi tersebut dianalisa oleh *code analyzer* untuk dapat direproduksi oleh *code transformer*. Setelah itu, *code transformer* berlaku sebagai *obfuscator* untuk membuat kode baru dengan teknik-teknik *obfuscator* seperti *jumps*, pemasukan kode "sampah", dll. Hal ini membuat kode baru yang dihasilkan berbeda dengan kode sebelumnya. Setelah kode baru dibuat, kode masih dalam bentuk *assembly* dan harus diubah ke bentuk biner dengan *assembler* [1]. Gambar 4 menjelaskan struktur dan cara kerja dari kode metamorfis.



Gambar 4 Struktur dan cara kerja metamorfisme

Jika dilakukan dengan baik, kode metamorfis sulit untuk dideteksi. Kode dapat mereplikasi dirinya menjadi berbagai macam bentuk hingga jumlah yang tidak terbatas. Kode tersebut tidak dapat terdeteksi karena tidak memiliki *signature string*. Karena itu, *anti-virus* harus menggunakan analisa yang lebih

kompleks berdasarkan perilaku kode [1]. Kode metamorfis pertama yang diidentifikasi adalah W32/Sdbot-ACG pada tahun 1998. Kode itu merupakan *worm* yang berjalan secara terus menerus di *background* dan membuka server *backdoor* yang dapat digunakan untuk mengambil informasi, serangan DDoS, memasukan software lain tanpa diketahui pengguna, dll.

2.3. Teknik deteksi malware

Ada berbagai macam teknik deteksi *malware* yang digunakan sejak pertama kali *malware* muncul. Teknik deteksi *malware* ini terus berkembang seiring berkembangnya teknik kamuflase malware. Deteksi ini dapat dilakukan dengan menggunakan *scanning, virus-specific detection, code-emulation*. Umumnya, deteksi *malware* dilakukan oleh spesialis *anti-virus* dan *signature* dari *malware* tersebut didistribusikan melalui program *anti-virus.*

2.3.1. Scanner

Pada awalnya, *malware* masih jarang yang menggunakan teknik kamuflase. Karenanya, *scanner* awal mendeteksi *malware* dengan mencari pola tertentu yang disebut *string signatures* [2]. Saat *malware* terdeteksi, *malware* tersebut akan dianalisa untuk dikenali *signature*-nya dan disimpan kedalam database *scanner*.

2.3.1.1. Scanner generasi pertama

Scanner generasi pertama menggunakan teknik yang tidak rumit untuk melakukan pencarian *malware.* Kasus yang dihadapi scanner ini juga masih tidak rumit sehingga scanner ini berfungsi secara efektif. Sesuai konsepnya, *scanner* mencari pola *string signatures* dari *malware* yang sudah teridentifikasi dan melakukan pencarian *signature* tersebut pada kode. Dalam kode biner, *anti-virus* melakukan pemindaian pola yang pada umumnya terdiri dari 16 bit. Namun, dalam kasus 32 bit, *string* lebih Panjang dibutuhkan untuk melakukan pengecekan terutama pada *malware* yang dibuat menggunakan bahasa pemrograman tingkat tinggi [2]. *Scanner* generasi pertama ini memiliki beberapa kapabilitas khusus. Salah satunya adalah pemindaian dengan kasus khusus dalam string. Kasus khusus itu dapat ditangani dengan adanya:

- *Wildcard* (pengecualian sebuah *byte* atau kumpulan *byte* yang konstan dari perbandingan)
- *Mismatches* (memperbolehkan adanya perbedaan dari perbandingan)
- *Generic degree* (Mendeteksi *signature* dari virus yang memiliki berbagai variasi)

Kapabilitas lainnya adalah *bookmark*. *Bookmark* digunakan untuk melakukan pengecekan bagian spesifik dari sebuah kode untuk mendeteksi virus spesifik. *Scanner* generasi pertama ini juga memiliki kapabilitas untuk mempercepat eksekusi pengecekan. Hal ini dilakukan dengan tujuan untuk mengatasi *malware* yang menyebar dengan kecepatan tinggi [2]. Hal ini dilakukan dengan teknik *hashing*

(menggunakan *hash function* untuk mempercepat pencarian), pengecekan awal dan akhir (umumnya virus berada di awal dan akhir dari kode), pengecekan *entry-point* dan *fixed point*.

2.3.1.2. Scanner generasi kedua

Scanner generasi kedua berkembang karena munculnya kebutuhan baru untuk mengatasi ancaman *malware*. *Scanner* ini muncul karena pengecekan dengan menggunakan pola saja mulai tidak efektif terhadap *malware-malware* baru yang juga berkembang menjadi lebih rumit. Generasi baru dari *scanner* ini memiliki kapabilitas-kapabilitas baru selain yang sudah ada sebelumnya. Kapabilitas tersebut adalah [2]:

- *Smart scanning*
  Kapabilitas ini efektif dalam melakukan pengecekan terhadap *malware* yang menggunakan teknik penyembunyan seperti polimorfisme dan metamorfisme yang menggunakan *code obfuscation*. Teknik ini melakukan pengecekan tanpa melakukan pengecekan instruksi dan kode "sampah".
- *Skeleton detection*
  Melakukan pencarian dengan hanya mengecek instruksi yang merupakan bagian dari kode *malware*. Hal ini dilakukan dengan membuang instruksi yang tidak merupakan kode virus. Teknik ini dikembangkan oleh Eugene Kaspersky, peneliti virus dari Russia [8].
- *Nearly exact identification*
  Menggunakan beberapa *string* sebagai *signature* dari virus yang akan dicek. Jika *signature* tersebut ada dalam kode yang dicek, maka virus dapat disimpulkan terdeteksi
- *Exact identification*
  Melakukan pengecekan *checksum* dari semua *byte* yang ada di dalam virus terhadap kode targetnya. Karena dilakukan pengecekan terhadap semua *byte*, maka pengecekan dilakukan secara lambat.
- *Heuristics analysis*
  Deteksi dengan menggunakan heuristik ini dilakukan dengan memonitor perilaku sistem untuk menemukan aktivitas yang dianggap tidak normal. Deteksi menggunakan metode ini tidak memerlukan *signature* sehingga dapat dilakukan terhadap *malware* yang belum pernah ditemukan sebelumnya. Namun, kekurangan dari metode ini adalah dengan pengecekan tanpa referensi ke *malware* yang sudah ada, banyak *false positive* yang akan muncul dalam sistem. Walaupun begitu, keuntungan dari deteksi dengan menggunakan heuristik adalah pengecekan ini dapat dilakukan tanpa adanya pengetahuan tentang *malware* [9]. Meskipun tingkat kepercayaan heuristik masih rendah, banyak spesialis *anti-virus* yang mengembangkan teknik ini. Heuristik baru mulai dikembangkan seperti contohnya yang dikembangkan oleh Ninyesiga Allan. Heuristik baru ini

merupakan perpaduan dari HRS (*hash based, rule based, SVM-based model*) dan behavioral model berdasarkan klasifikasi Bayesian untuk mendeteksi *malware* [3].

2.3.2. Virus specific Detection

Pada kasus tertentu, pendeteksi *malware* umum tidak dapat mendeteksi sebuah *malware* khusus. Hal ini dapat terjadi karena bentuk *signature* dari *malware* tersebut berbeda dengan *malware* lain secara umum, perlu dilakukan pengecekan kasus khusus yang banyak, dll. Karenanya, agar *anti-virus* dapat mendeteksi *malware* tersebut, dibutuhkan algoritma pengecekan secara khusus yang melakukan deteksi terhadap sebuah *malware* spesifik. Untuk melakukan pengecekan khusus tersebut, teknik ini menggunakan beberapa kapabilitas seperti *Filtering*, deteksi *decryptor* statis, *x-ray scanning* [2].

2.3.2.1. Filtering

Teknik filtering digunakan dengan melihat target dari *malware* spesifik tersebut. Teknik ini menggunakan prinsip deduksi untuk mengabaikan hal-hal yang tidak penting bagi *malware* spesifik tersebut. Sebagai contohnya adalah *boot virus*. Virus tersebut hanya ada di area boot dari disk. Dengan mengetahu perilaku khusus seperti itu, dapat disimpulkan bahwa untuk mendeteksi *boot virus*, bagian yang hanya perlu diamati hanya area boot dari disk. Hal ini berlaku juga bagi *malware-malware* lainnya.

Teknik ini digunakan untuk mengoptimasi performansi dari *anti-virus engine* tanpa memperhatikan kecepatan pemindaian. Hal ini sangat berguna di deteksi virus spesifik karena algoritma deteksi virus spesifik umumnya memakan waktu yang sangat lama dan sangat rumit untuk dilakukan [2]. Dengan ini, waktu yang dibutuhkan untuk mendeteksi *malware* spesifik dapat berkurang secara signifikan.

2.3.2.2. Static decryptor detection

Beberapa macam *malware* menggunakan enkripsi unuk menyembunyikan *body* dari *malware* tersebut. Dengan begitu, jumlah *bytes* yang ada dalam kode menjadi lebih sedikit. Karenanya, mendeteksi dengan *string matching* menjadi lebih sulit. Teknik ini digunakan dengan mendeteksi beberapa *decryptor* spesifik. Karenanya, teknik ini sangat berguna untuk mendeteksi virus-virus terenkripsi, oligomorfis, dan polimorfis. Namun, teknik ini masih tidak bisa memastikan bahwa *malware* dihapus seluruhnya karena masih ada bagian *body* yang terenkripsi.

2.3.2.3. X-Ray scanning

Teknik ini memanfaatkan data yang sudah ada dari *malware* yang akan dideteksi. Teknik ini menyerang tidak menyerang bagian *decryptor* dari *malware* melainkan bagian enkripsi dari *malware.* Cara bekerjanya adalah menggunakan *plaintext* dari *malware* yang telah diidentifikasi tersebut, teknik ini mengaplikasikan semua metode enkripsi ke masing-masing bagian dari *plaintext* tersebut. Bagian yang dienkripsi tersebut hanyalah bagian spesifik seperti *top* dan *tail*, titik masuk, dll [2].

Dengan menggunakan cara tersebut, *x-ray scanning* dapat mengidentifikasi *malware-malware* terenkripsi dengan tingkat kepercayaan yang lumayan tinggi. Teknik ini cukup efektif terhadap *malware* dengan kapabilitas enkripsi, oligomorfisme, dan polimorfisme. Namun, kelemahan dari teknik ini adalah penggunaannya sangat memakan waktu. Hal ini terjadi jika virus tidak bertempat pada lokasi spesifik sehingga perlu dilakukan pengecekan terhadap semua lokasi yang ada.

2.3.3. Code emulation

Teknik ini merupakan salah satu teknik yang memiliki tingkat efektivitas tinggi. Prinsip dari teknik ini adalah membuat mesin virtual (*virtual machine*) yang terdiri dari prosesor (*processor*), memori utama (*main memory*), penyimpanan (*storage*), dan hal lain yang merupakan bagian komputer. Setelah itu, *malware* dijalankan di *virtual machine* tersebut dan dianalisa perilakunya, performansinya, dll. Karena *malware* dijalankan di mesin virtual tersebut, maka siklus hidup dari *malware* tersebut dikontrol oleh mesin virtual tersebut. Karenanya, tidak ada resiko *malware* tersebut berjalan di mesin aslinya [2]. Pada dasarnya teknik ini melakukan simulasi perilaku program untuk menentukan apakah program tersebut merupakan *malware* atau bukan.

Untuk *malware* terenkripsi seperti polimorfis, oligomorfis, dll, dilakukan beberapa iterasi untuk dianalisa. Setelah beberapa iterasi tersebut, *scanner* melakukan pengecekan terhadap konten memori dari mesin virtual tersebut. Hal ini dilakukan untuk mengungkapkan bagian *body* yang sudah terdekripsi di dalam memori virtual. Setelah simulasinya diberhentikan, *source code* dari virus telah diidentifikasi dan *malware* tersebut dapat dihapuskan menggunakan pengecekan *string* atau teknik-teknik lainnya.

Salah satu aplikasi dari teknik *code emulation* adalah kasus untuk mendeteksi *decryptor* secara dinamis. Dalam kasus ini, yang dilakukan adalah mencari *decryptor* dari *malware* menggunakan simulasi dari perilaku kode. Hal ini dilakukan dengan mengidentifikasi *entry point* yang mungkin dari *malware* tersebut, lalu menjalankan sambal menganalisa perubahan yang terjadi pada register. Jika perubahan signifikan yang dianggap sebagai *body* dari *malware* yang sudah didekripsi ditemukan, maka *decryptor* tersebut telah diidentifikasi. Hal ini berguna jika isi *decryptor malware* panjang dan sangat memakan waktu untuk dijalankan [2].

2.3.1. Deteksi malware metamorfis

Malware dengan kemampuan metamorfis merupakan ancaman terbesar di dunia *malware* saat ini. Hal ini disebabkan oleh kemampuannya untuk berubah bentuk menjadi berbagai macam kode hingga jumlah yang tidak terbatas. Perubahan itu tidak mengubah fungsionalitas dari *malware* tersebut. Karenanya,

penyebaran yang dapat terjadi sangat besar dan kerusakan yang dapat disebabkan oleh *malware* tersebut besar.

Sekarang, walaupun teknik deteksi *malware* sudah berkembang dengan pesat, belum ad acara yang "baik" untuk mendeteksi *malware* dengan kemampuan metamorfis. Walaupun beberapa sudah efektif, banyak false positive yang muncul. Seperti contohnya menggunakan analisa heuristik. Analisa heuristik memonitor sistem secara *real-time* dan menentukan apakah sebuah program merupakan *malware* atau bukan. Seringkali perilaku yang dianggap abnormal oleh analisa heuristik merupakan program dengan perilaku normal sehingga analisa heuristik memiliki tingkat kepercayaan yang rendah.

Dengan menggunakan *code emulator*, *malware* dengan kemampuan metamorfis dapat dideteksi. Namun, permasalahan yang baru muncul adalah beberapa *malware* memiliki fungsionalitas lebih dibandingkan *malware* pada umumnya. Beberapa *malware* mampu mengubah perilakunya sehingga tidak dapat dieksekusi. Hal ini dapat dilakukan jika *malware* mendeteksi adanya emulator yang digunakan [5]. Selain itu, *malware* dengan kemampuan deteksi mesin virtual dapat melakukan serangan *denial of service*. Hal ini membuat emulator mesin virtual berhenti berjalan. Dan yang paling menjadi masalah, walaupun secara teoritis *malware* yang dijalankan di mesin virtual tidak dapat keluar dari lingkungan virtualnya, kenyataannya *malware* seperti itu juga dapat keluar dari lingkungan virtualnya dan menyerang host-nya.

Dari masalah-masalah yang telah disebutkan, dapat disimpulkan bahwa *malware* dengan kemampuan metamorfis merupakan ancaman besar di dunia *cyber security*. Cara penanganan yang paling efektif adalah dengan mendeteksinya terlebih dahulu. Deteksi ini merupakan hal yang cukup sulit karena kemampuan *malware* tersebut. walaupun begitu, penelitian tetap dilakukan untuk berusaha mendeteksi *malware* tersebut. Seperti contohnya menggunakan semantik dalam deteksi [10]. Dalam penelitiannya, Mihai Christodorescu *et al* menggunakan semantik untuk melakukan deteksi *malware*. Teknik ini berhasil untuk mendeteksi *obfuscation* sederhana yang ada pada *malware-malware* polimorfis dan metamorfis. Mereka berhasil untuk membuat semantik baru dan algoritma baru untuk mendeteksi perilaku *malware* secara spesifik [10]. Cara lain yang dapat dilakukan adalah menggunakan analisa statis untuk meningkatkan efisiensi dari analisa dinamis dengan contohnya menghapus *redundant checks* yang ada dalam framework IRM (*inline reference monitor*)[10].

Cara lain yang mungkin efektif dalam melakukan deteksi *malware* metamorfis adalah dengan menggunakan metode yang dikembangkan oleh Qinghua Zhang dan Douglas S. Reeves dalam penelitiannya [4]. Metode ini dilakukan dengan cara melakukan *disassembly* terdahap dua buah *executable* dan mengkomputasikan tingkat kemiripan antara kedua *executable* tersebut. Karakteristik utama yang digunakan dalam perbandingan ini adalah fungsi sistem atau *library* yang dipanggil oleh 2 program tersebut [4]. Mereka melakukan penelitian itu berdasarkan karakteristik tersebut. Setiap program memiliki *pattern*

masing-masing dan pola tersebut dapat dibandingkan untuk mendapatkan analisa kemiripan antar program. Proses untuk membandingkan *pattern* ini disebut *pattern matching*. Tantangan dari menggunakan metode ini adalah menganalisa dan menghasilkan *pattern* dari sebuah program secara cepat dan melakukan perbandingan *pattern* secara cepat [4].

Dalam mendeteksi *malware* metamorfis, banyak cara yang dapat dilakukan. Selain cara yang telah disebutkan diatas, masih banyak metode yang sedang diteliti dan dikembangkan. Metode tersebut dibuat sedemikian rupa untuk dapat memberikan pertahanan terhadap *malware-malware* berbahaya.

3. PENUTUP

3.1. Kesimpulan

Malware adalah perangkat lunak yang dirancang sedemikian rupa untuk dapat mengakibatkan kerusakan pada infrastruktur atau sistem informasi. Pada umumnya, *malware* menyebar secara tersembunyi ke perangkat yang ditujukan. *malware* dapat melakukan hal tersebut dengan adanya teknik kamuflase *malware.* Teknik kamuflase *malware* merupakan sarana yang digunakan untuk menyembunyikan *malware* yang bertujuan untuk memperpanjang siklus hidup *malware.* Dengan itu, *malware* dapat menyebar lebih luas dan kerusakan yang diakibatkan oleh *malware* tersebut akan menjadi lebih besar. Teknik ini terus berkembang seiring dengan perkembangan teknologi dari teknik enkripsi yang sederhana hingga teknik metamorfis yang rumit. Hal ini disebabkan oleh kemampuan *anti-virus* yang semakin berkembang juga dari awalnya hanya *string scanning* hingga *code* emulation. Tantangan terbaru bagi spesialis anti-virus adalah *malware* dengan kemampuan metamofis. Sampai sekarang, belum ada cara yang terbukti efektif dalam mendeteksi *malware* dengan kemampuan metamorfis dengan tingkat kepercayaan yang tinggi. Walaupun begitu, tantangan yang ada tidak hanya sebatas mendeteksi *malware* metamorfis karena teknik kamuflase tetap akan terus berkembang. Karenanya, penting bagi para *anti-virus specialist* untuk terus mengembangkan pertahanan dari *malware-malware* baru tersebut.

3.2. Saran

Teknik kamuflase *malware* merupakan aspek dari teknologi yang akan terus berkembang kedepannya. Kedepannya, teknik deteksi *malware* juga akan terus berkembang seiring berkembangnya teknik penyembunyian *malware.* Karena itu, diperlukan referensi-referensi baru untuk mengembangkan tulisan ini. Seiring perkembangan teknologi, tulisan ini dapat dikembangkan dengan contoh-contoh dan teknik-teknik baru yang ada.

REFERENSI

[1] Rad, Babak Bashari, Maslin Masrom, and Suhaimi Ibrahim. "Camouflage in malware: from encryption to metamorphism." International Journal of Computer Science and Network Security12, no. 8 (2012): 74-83.

[2] Rad, Babak Bashari, Maslin Masrom, and Suhaimi Ibrahim. "Evolution of computer virus concealment and anti-virus techniques: a short survey." arXiv preprint arXiv:1104.1070(2011).

[3] Ninyesiga, Allan. Improved Heuristics for Malware Detection. Vol. 307570435. A Working Paper on Research Gate accessible via https://www. researchgate. net/publication, 2016.

[4] Zhang, Qinghua, and Douglas S. Reeves. "Metaaware: Identifying metamorphic malware." In Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), pp. 411-420. IEEE, 2007.

[5] Ferrie, Peter. "Attacks on more virtual machine emulators." Symantec Technology Exchange 55 (2007).

[6] Skulason, Fridrik. "Virus encryption techniques." Virus Bulletin 1990 (1990): 13-16.

[7] Szor, Peter, and Peter Ferrie. "Hunting for metamorphic." In Virus bulletin conference. 2001.

[8] Szor, P. "The Art of Computer Virus Defense and Research." (2005).

[9] Al Amro, Sulaiman, and Ali Alkhalifah. "A Comparative Study of Virus Detection Techniques." International Journal of Computer, Electrical, Automation, Control and Information Engineering 9, no. 6 (2015).

[10] Christodorescu, Mihai, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. "Semantics-aware malware detection." In 2005 IEEE Symposium on Security and Privacy (S&P'05), pp. 32-46. IEEE, 2005.