Evaluasi Kapabilitas Deteksi *anti-spyware* Terhadap *Dropper Spyware Mode Stealth*

Proposal Tugas Akhir

Oleh

M. Kasyfil Aziz
18222127

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

November 2025

LEMBAR PENGESAHAN

Evaluasi Kapabilitas Deteksi *anti-spyware* Terhadap *Dropper Spyware Mode Stealth*

Proposal Tugas Akhir

Oleh

M. Kasyfil Aziz
18222127

Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Proposal Tugas Akhir ini telah disetujui dan disahkan
di Bandung, pada tanggal 25 November 2025

Pembimbing

Dr. Ir. John Doe, M.T.

NIP. 123456789

DAFTAR ISI

DAFTAR GAMBAR . . . . . **iv**

DAFTAR TABEL . . . . . **v**

DAFTAR KODE . . . . . **vi**

I PENDAHULUAN **1**
I.1 Latar Belakang 1
I.2 Rumusan Masalah 2
I.3 Tujuan 2
I.4 Batasan Masalah 3
I.5 Metodologi 3

II STUDI LITERATUR **5**
II.1 Malware dan Spyware 5
II.2 Malware Fileless 6
II.3 Dropper 7
II.4 Teknik Evasi Malware 7
II.5 Anti Malware dan Sistem Deteksi 8
II.6 Studi Kasus Spyware Modern 8

III ANALISIS MASALAH **10**
III.1 Analisis Kondisi dan Kesenjangan (Current State and Gap Analysis) . . 10
III.1.1 Keterbatasan Deteksi Konvensional 10
III.1.2 Analisis Kesenjangan Desain *Dropper* (Gap Analysis) 11
III.2 Metodologi Penelitian: Design Research Methodology (DRM) 12
III.3 Persyaratan Fungsional dan Non-Fungsional 13
III.4 Desain Keputusan dan Solusi yang Diusulkan 14
III.4.1 Alternatif Solusi 14
III.4.2 Analisis Penelitian Solusi 14
III.4.3 Metodologi Penilaian Solusi (Multi-Criteria Decision Analysis) . . . 15
III.4.3.0.1 Langkah 1: Normalisasi Matriks Keputusan. 15
III.4.3.0.2 Langkah 2: Perhitungan Nilai Entropi. 15
III.4.3.0.3 Langkah 3: Penentuan Bobot Kriteria. 15
III.4.3.0.4 Langkah 4: Perhitungan Skor Komposit Akhir. 16
III.4.4 Hasil Analisis Penentuan Solusi 16

III.5 Rencana Evaluasi (Overview) 17
III.5.1 Skenario Pengujian 17
III.5.2 Metrik dan Alat Evaluasi 17

IV DESAIN KONSEP SOLUSI 19
IV.1 Desain Solusi 19
IV.2 Tahapan Desain 19
IV.2.1 Desain Artefak *Dropper* 20
IV.2.1.1 Arsitektur Modul dan Fungsi *Evasion* 20
IV.2.1.2 Mekanisme Alur Logika *Dropper* 21
IV.3 Rancangan Lingkungan Pengujian 22
IV.4 Spesifikasi Mesin Uji 22
IV.5 Hasil Desain 23
IV.5.1 Tahap Inisialisasi dan Dekripsi (Proses 2.1 & 2.2) 24
IV.5.2 Tahap Validasi Lingkungan / *Anti-Analysis* (Proses 2.3) 24
IV.5.3 Tahap Manajemen *Payload* (Proses 2.4 & 2.5) 25
IV.5.4 Tahap Eksekusi *Fileless* (Proses 2.6 & 2.7) 25

V RENCANA SELANJUTNYA 26
V.1 Rencana Implementasi 26
V.1.1 Pengembangan Komponen *Dropper* 26
V.1.2 Lingkungan Pengembangan 27
V.2 Rencana Evaluasi 28
V.2.1 Konfigurasi Sistem Pengujian 28
V.2.2 Skenario Pengujian 28
V.2.3 Metrik Evaluasi 29
V.3 Analisis Risiko 29

DAFTAR GAMBAR

I.1 Kerangka kerja DRM 4

IV.1 Diagram alir eksekusi *dropper spyware* mode *stealth* 24

DAFTAR TABEL

III.1 Ringkasan kesenjangan desain *dropper* dan arah solusi 11
III.2 Persyaratan fungsional (Functional Requirements) 13
III.3 Persyaratan non-fungsional (Non-Functional Requirements) 14
III.4 Bobot kriteria hasil metode EWM 16
III.5 Skor komposit akhir alternatif solusi 16
III.6 Metrik evaluasi dan alat ukur 18

IV.1 Komponen Modul *Dropper* 21
IV.2 Spesifikasi lingkungan uji 23

V.1 Spesifikasi lingkungan implementasi 27
V.2 Metrik evaluasi 29
V.3 Analisis risiko dan mitigasi 30

DAFTAR KODE

BAB I

PENDAHULUAN

I.1 Latar Belakang

Perkembangan teknologi informasi yang pesat turut membawa peningkatan ancaman keamanan siber, terutama berupa malware yang semakin canggih dan sulit dideteksi (Sudhakar dan Kumar 2020). Salah satu varian berbahaya adalah *spyware*, malware yang secara diam-diam mengumpulkan data sensitif korban (Reddyvari dkk. 2024). Dalam rantai infeksi *spyware* modern, *dropper* memainkan peran penting sebagai komponen pengantar dan eksekutor malware. *Dropper* bertugas untuk mendistribusikan, membuka kemasan (*unpack*), serta menjalankan *payload spyware* dalam sistem target dengan teknik yang mampu menghindari deteksi oleh *anti malware* konvensional (Chatzoglou dkk. 2023; Elghaly 2024).

Dropper saat ini tidak hanya berfungsi sebagai pengantar biasa, tetapi juga dilengkapi fitur *anti-debugging*, *anti-virtual machine*, serta teknik *stealth* lainnya yang membuatnya sangat tahan terhadap mekanisme deteksi dan pencegahan (Elghaly 2024; Chatzoglou dkk. 2023; Kerkour 2021). *Dropper* juga bisa bersifat *fileless*, beroperasi langsung di memori tanpa meninggalkan jejak file yang bisa dipindai, sehingga menjadi tantangan besar bagi sistem keamanan (Sudhakar dan Kumar 2020; Elghaly 2024). Keberhasilan *dropper* menentukan efektifitas infeksi *spyware* secara keseluruhan, termasuk kemampuan *persistensi* dan pencurian data (Kerkour 2021; Kareem 2024).

Penelitian ini bertujuan mengkaji kapabilitas *anti spyware* dalam mendeteksi dan menangkal *dropper spyware* baru yang menerapkan teknik evasi lanjutan tersebut. Kajian ini penting untuk mengetahui sejauh mana solusi keamanan saat ini mampu menghadapi ancaman yang semakin kompleks, dan memberikan dasar bagi pengembangan metode pertahanan yang lebih adaptif dan tangguh (Chatzoglou dkk. 2023; Reddyvari dkk. 2024).

Dengan mengacu pada studi dan analisis malware dari berbagai literatur terkini, termasuk teknik *stealth* dan evasi *dropper* dalam malware *fileless*, penelitian ini juga akan menguji efektivitas berbagai produk *anti spyware* komersial terhadap varian *dropper* tersebut dalam lingkungan simulasi yang realistis (Sudhakar dan Kumar 2020; Elghaly 2024; Chatzoglou dkk. 2023; Kareem 2024).

I.2 Rumusan Masalah

Berdasarkan latar belakang di atas, rumusan masalah penelitian difokuskan pada *dropper spyware* mode *stealth* sebagai mekanisme serangan tahap awal (*initial access*). Pertanyaan penelitian yang diajukan adalah:

1. Seberapa efektif sistem *anti-malware* dan *Endpoint Detection & Response* (EDR) dalam mendeteksi aktivitas *dropper spyware* mode *stealth* yang berfungsi sebagai *initial access and delivery mechanism* pada lingkungan uji terkontrol?
2. Bagaimana pendekatan paling efektif untuk mengembangkan *dropper spyware stealth* yang mampu menghindari deteksi *anti-malware* dengan memanfaatkan teknik evasi modern seperti *fileless execution*, *in-memory payload delivery*, dan pemeriksaan *anti-analysis*?
3. Bagaimana pengaruh teknik evasi, misalnya *obfuscation*, enkripsi *payload*, *sandbox evasion*, dan deteksi *anti-VM*, terhadap kemampuan deteksi produk *anti-malware* dan EDR komersial saat ini?
4. Celah keamanan spesifik apa saja yang muncul pada produk *anti-malware* maupun EDR ketika dihadapkan pada *dropper spyware* kustom yang dirancang dengan teknik penghindaran deteksi terintegrasi?

I.3 Tujuan

Secara umum penelitian ini bertujuan mengukur efektivitas sejumlah produk *anti-malware* dan EDR komersial dalam mendeteksi *dropper spyware* mode *stealth* yang bertindak sebagai vektor serangan awal. Tujuan khusus yang hendak dicapai yaitu:

1. Menganalisis dan mengidentifikasi teknik evasi *dropper spyware* paling efektif untuk menghindari deteksi *anti-malware* dan EDR, mencakup kategori *file-based*, *network-based*, *script-based*, *downloader-based*, serta *non-persistent*.
2. Mengembangkan prototipe *dropper spyware* mode *stealth* yang mengintegrasikan teknik evasi modern, seperti *fileless payload delivery*, eksekusi di memori, pemeriksaan *anti-VM*/*anti-debugging*, komunikasi *C2* terenkripsi, dan modul *spyware* aktif.

3. Melakukan pengujian komparatif prototipe terhadap beberapa produk *anti-malware* dan EDR terkemuka guna mengamati keberhasilan *payload delivery* serta respons deteksi pada setiap tahap eksekusi.
4. Mengidentifikasi celah keamanan dan kelemahan deteksi pada tiap produk, kemudian merumuskan rekomendasi teknis maupun strategis untuk meningkatkan kapabilitas mitigasi serangan *initial access* berbasis *dropper*.

Kriteria keberhasilan penelitian meliputi: (a) prototipe *dropper* berhasil mengimplementasikan sedikitnya tiga teknik evasi terintegrasi dan terbukti lolos dari deteksi minimal pada satu produk *anti-malware* yang diuji; serta (b) hasil eksperimen menunjukkan perbedaan signifikan antar produk dalam mendeteksi *dropper* dan mengungkap tahap eksekusi yang paling sulit diawasi.

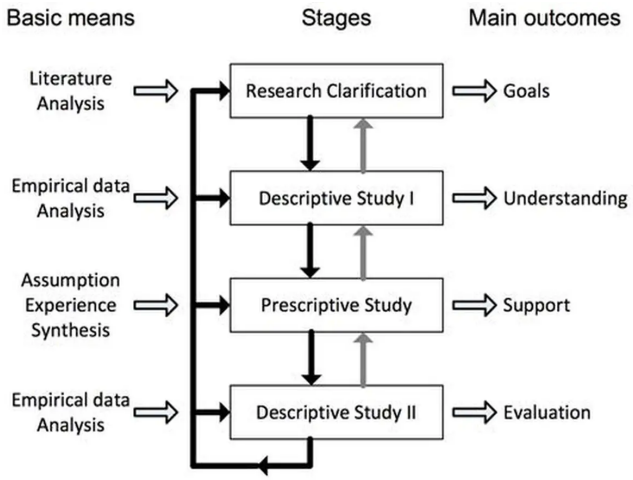
I.4 Batasan Masalah

Batasan penelitian dibagi menjadi tiga kategori utama. Pertama, batasan umum menekankan bahwa seluruh eksperimen dilaksanakan di lingkungan simulasi terkontrol dengan skenario yang telah ditetapkan; ruang lingkup hanya sampai pemastian *payload* berhasil dijalankan pada sistem target tanpa mengevaluasi dampak pasca-infeksi. Kedua, batasan desain pengujian berfokus pada *dropper spyware* mode *stealth* yang memanfaatkan teknik *fileless execution*, *in-memory payload*, serta pemeriksaan *anti-analysis*; metode distribusi lain seperti *worm* atau *supply-chain attack* tidak dianalisis. Ketiga, batasan implementasi menegaskan bahwa pengujian hanya menilai kemampuan deteksi dan respons *anti-malware*/EDR hingga tahap *payload* aktif, tanpa melakukan analisis rinci terhadap *payload* lanjutan atau serangan lanjutan setelah *dropper* selesai bertugas.

I.5 Metodologi

Metodologi penelitian mengacu pada *Design Research Methodology* (DRM) yang diperkenalkan oleh Blessing dan Chakrabarti (Blessing dan Chakrabarti 2009). Kerangka ini dipilih karena menyajikan proses iteratif yang terstruktur untuk merancang sekaligus mengevaluasi artefak teknis.

1. **Research Clarification (RC):** tahap ini mengidentifikasi kesenjangan antara evolusi teknik *dropper* modern dan keterbatasan sistem deteksi. Aktivitasnya meliputi studi literatur terhadap laporan industri keamanan siber, publikasi akademis mengenai teknik evasi, serta studi kasus serangan *dropper*. Hasil RC berupa rumusan masalah yang terdefinisi jelas dan urgensi penelitian yang berdasar pada bukti mutakhir.



Gambar I.1 Kerangka kerja DRM

2. **Descriptive Study I (DS-I):** tahap ini menelaah kondisi eksisting melalui analisis karakteristik *dropper* modern dalam *cyber kill chain*. Fokus ada pada *fileless payload delivery*, eksekusi di memori, pemeriksaan *anti-analysis*, *obfuscation* string dan komunikasi *C2* terenkripsi, serta metode distribusi seperti *phishing*, *drive-by download*, dan dokumen berbahaya. DS-I juga menyusun rantai serangan *dropper* dari *initial access* hingga *command-and-control*.
3. **Prescriptive Study (PS):** tahap PS merancang dan mengembangkan prototipe *dropper spyware* mode *stealth* menggunakan bahasa Rust. Prototipe terdiri atas modul *payload builder*, manajemen *in-memory execution*, pemeriksaan *anti-VM*/*anti-debug*, kanal komunikasi terenkripsi, serta modul *spyware*. Pengujian awal dilakukan secara iteratif untuk memastikan tiap teknik evasi bekerja sesuai spesifikasi.
4. **Descriptive Study II (DS-II):** fase akhir mengevaluasi prototipe melalui eksperimen pada beberapa lingkungan uji dan produk *anti-malware*/EDR komersial. Parameter evaluasi mencakup tahap deteksi atau kelolosan aktivitas *dropper*, keberhasilan *payload delivery* di memori, serta efektivitas masing-masing teknik evasi. Data kuantitatif berupa *detection rate* dan *evasion rate* dibandingkan antar produk, sedangkan data kualitatif seperti pola perilaku *dropper* dan pesan kesalahan dianalisis untuk memahami penyebab lolosnya deteksi. Temuan DS-II menjadi dasar rekomendasi peningkatan mekanisme deteksi dan arah penelitian lanjutan untuk memperkuat pertahanan terhadap *initial access* berbasis *dropper*.

BAB II

STUDI LITERATUR

II.1 Malware dan Spyware

Malware adalah perangkat lunak berbahaya yang dirancang untuk merusak, mengakses tanpa izin, atau mengganggu sistem komputer. Salah satu tipe *malware* yang khusus adalah *spyware*, yaitu program yang bertujuan untuk memantau dan mengumpulkan informasi sensitif pengguna secara diam-diam tanpa sepengetahuan mereka. *Spyware* dapat melakukan berbagai aktivitas seperti menangkap penekanan tombol (keylogging), memantau aktivitas sistem, dan mencuri data pribadi.

Perkembangan *spyware* sangat cepat seiring dengan kemajuan teknologi serta alat bantu yang tersedia bagi pembuat *malware*. Misalnya, penggunaan alat seperti *MSF Venom* memungkinkan pembuatan *payload spyware* yang dapat dikustomisasi untuk menjalankan aktivitas berbahaya seperti pencurian data dan penyadapan jaringan. Deteksi *spyware* memerlukan pendekatan multifaset yang mengombinasikan analisis perilaku sistem, pengecekan tanda tangan berkas, serta pemantauan pola lalu lintas jaringan.

Menurut *Reddyvari et al.* (Reddyvari dkk. 2024), "*Spyware* adalah jenis *malware* yang dirancang untuk memantau dan mengumpulkan informasi sensitif tanpa izin, menimbulkan ancaman serius terhadap privasi dan keamanan." Mereka juga menyebutkan bahwa alat seperti *YARA* dapat digunakan untuk membuat aturan deteksi yang spesifik terhadap tanda-tanda dan pola khas *spyware*, sehingga membantu dalam identifikasi dan mitigasi ancaman tersebut.

Penggunaan kombinasi teknologi seperti modul *Python* untuk menganalisis perilaku *malware* dan *YARA rules* untuk identifikasi pola terbukti efektif dalam mendeteksi *spyware* baru yang sulit dikenali oleh antivirus tradisional. Dengan demikian, pemahaman mendalam tentang karakteristik *malware* dan *spyware* sangat penting

sebagai dasar bagi pengembangan metode pencegahan serta deteksi keamanan siber yang lebih maju.

II.2 Malware Fileless

Malware fileless merupakan varian *malware* yang tidak menggunakan berkas eksekusi tradisional pada sistem target, melainkan beroperasi langsung di memori (*RAM*) atau memanfaatkan komponen sistem yang sudah ada seperti *PowerShell* dan *Windows Management Instrumentation* (WMI). Teknik ini membuat *malware fileless* sulit dideteksi oleh solusi keamanan konvensional yang biasanya mengandalkan pemindaian berkas di media penyimpanan.

Malware fileless umumnya mengeksploitasi kerentanan dalam aplikasi yang sah dan menggunakan skrip atau perintah yang berjalan di memori untuk menjalankan aksinya sehingga kehadirannya hampir tidak meninggalkan jejak pada sistem berkas. Salah satu contoh serangan terkenal dengan metode ini adalah infiltrasi yang dilakukan melalui skrip *PowerShell* yang dieksekusi tanpa menyimpan berkas berbahaya di media penyimpanan.

Menurut *Sudhakar dan Kumar* (Sudhakar dan Kumar 2020), *malware fileless* tidak pernah menulis berkas berbahaya ke media penyimpanan, melainkan menyuntikkan kode ke dalam memori dan menggunakan alat yang telah dipercaya oleh sistem operasi untuk tetap tersembunyi serta mempertahankan persistensi. Mereka juga menjelaskan bahwa *malware fileless* memanfaatkan fitur Windows seperti WMI dan *PowerShell* untuk *reconnaissance*, eksekusi kode, pergerakan lateral, *persistence*, serta pencurian data. Hal ini membuat deteksi *malware* jenis ini menjadi tantangan besar bagi sistem keamanan yang ada karena serangan ini sangat *stealthy* dan sulit dikenali menggunakan tanda tangan tradisional.

Penelitian *Elghaly* (Elghaly 2024) menemukan bahwa *malware fileless* berbasis *PowerShell* semakin marak digunakan oleh kelompok ancaman terorganisir (*APT*) dan penjahat siber karena kemampuannya menjadikan dirinya "tak terlihat" oleh sistem pertahanan seperti *Endpoint Detection and Response* (EDR) dan antivirus modern. Studi ini juga menunjukkan bahwa teknik *obfuscation PowerShell* semakin mempersulit proses deteksi dan analisis oleh solusi keamanan yang ada.

Secara umum, *malware fileless* mengubah lanskap keamanan siber dengan mengadopsi metode serangan yang tidak bergantung pada berkas tradisional untuk mengeksekusi *malware*. Kondisi ini menuntut pengembangan teknologi deteksi baru se-

perti analisis perilaku dan pemantauan memori secara *real-time* untuk meningkatkan efektivitas mitigasi serta perlindungan terhadap ancaman tersebut.

II.3 Dropper

Dropper adalah jenis *malware* yang berfungsi sebagai perantara untuk menempatkan *malware* utama ke dalam sistem target secara diam-diam. *Dropper* biasanya menyamar sebagai berkas atau program yang tampak sah agar dapat dijalankan oleh pengguna tanpa kecurigaan. Setelah dieksekusi, *dropper* akan mengekstrak dan memasang *malware* tambahan, seperti *spyware*, *ransomware*, atau *backdoor*, ke dalam sistem korban.

Fungsi utama *dropper* adalah membawa dan mengantarkan *payload malware* sehingga infeksi dapat berlangsung secara efektif. *Dropper* dapat membawa *malware* tersembunyi di dalam dirinya atau mengunduh *malware* dari server eksternal setelah berhasil dieksekusi. Ada dua tipe *dropper*, yaitu *persistent dropper* yang mampu bertahan lama dan melakukan instal ulang *malware*, serta *non-persistent dropper* yang hanya bekerja satu kali.

Penelitian *Chatzoglou et al.* (Chatzoglou dkk. 2023) menjelaskan bahwa berbagai teknik evasi sering digunakan oleh *dropper* untuk menghindari deteksi antivirus, seperti enkripsi, *obfuscation*, dan pembelahan kode *malware* menjadi beberapa bagian. Teknik injeksi proses juga umum digunakan, di mana *dropper* menyisipkan kode berbahaya ke dalam proses yang sah agar dapat berjalan secara *stealth* dan sulit terdeteksi oleh sistem keamanan.

Dropper menjadi komponen krusial dalam siklus infeksi *malware* karena menentukan keberhasilan pengantaran dan eksekusi *payload malware* di sistem target. Oleh karena itu, kemampuan *dropper* untuk menyamar dan melakukan evasi deteksi menjadi aspek yang sangat penting dalam pengembangan maupun penanggulangan *malware*.

II.4 Teknik Evasi Malware

Teknik evasi *malware* dirancang untuk menghindari deteksi dan analisis oleh perangkat keamanan seperti antivirus dan sistem deteksi intrusi. Beberapa teknik utama meliputi *code obfuscation*, penggunaan *packer* untuk menyamarkan berkas eksekusi, serta injeksi proses di mana *malware* menyisipkan kode berbahaya ke dalam proses yang sah agar sulit dideteksi. Teknik persistensi juga umum dilakukan dengan

memanfaatkan *registry* atau *task scheduler* sehingga *malware* tetap aktif meskipun sistem direstart. *Malware* modern kerap melakukan pengecekan terhadap lingkungan eksekusi, seperti *sandbox* atau *virtual machine*, untuk menyesuaikan perilaku dan menghindari analisis otomatis.

II.5 Anti Malware dan Sistem Deteksi

Anti-malware merupakan perangkat lunak yang dirancang untuk mendeteksi, mencegah, dan menghapus *malware*, termasuk *spyware*, dari sistem komputer. Sistem deteksi *malware* tradisional biasanya menggunakan metode berbasis tanda tangan (*signature-based*) yang mengidentifikasi *malware* berdasarkan ciri-ciri spesifik yang telah diketahui. Namun, metode ini kurang efektif untuk mendeteksi *malware* baru atau *malware* yang menggunakan teknik evasi canggih.

Untuk mengatasi keterbatasan tersebut, teknologi deteksi saat ini mulai mengadopsi metode *heuristic* dan *behavior-based*, yang memantau perilaku mencurigakan di sistem, serta menggunakan *machine learning* dan *big data* untuk mengidentifikasi pola serangan baru. Dengan analisis perilaku, sistem dapat mengenali aktivitas yang tidak biasa meskipun tanda tangan *malware* belum tersedia.

Meskipun demikian, *malware* modern, terutama *spyware fileless*, semakin sulit dideteksi karena kemampuannya beroperasi di memori dan memanfaatkan fitur sistem yang sah. Oleh karenanya, pengujian kapabilitas *anti-malware* dalam skenario dunia nyata sangat penting untuk mengetahui sejauh mana perlindungan yang diberikan sekaligus mendorong pengembangan teknologi keamanan lebih lanjut.

II.6 Studi Kasus Spyware Modern

Spyware modern berkembang dengan kemampuan yang sangat canggih sehingga memungkinkan pengintaian dan pencurian data secara tersembunyi dari perangkat target. Salah satu contohnya adalah *spyware Pegasus* yang dikembangkan oleh *NSO Group*, yang mampu mengeksploitasi kerentanan *zero-click* untuk menginfeksi *smartphone* tanpa interaksi pengguna. *Spyware* ini dapat mengakses pesan, panggilan, lokasi, serta data sensitif lainnya secara *real-time* tanpa terdeteksi oleh pemilik perangkat.

Analisis terhadap *Pegasus* menyoroti risiko besar terhadap privasi dan keamanan digital, sekaligus menunjukkan bagaimana *spyware* modern memanfaatkan teknik penyebaran yang sangat tersembunyi dan sulit diatasi dengan metode pertahanan

tradisional. Penggunaan teknologi canggih tersebut menuntut pengembangan kebijakan dan teknologi baru untuk melindungi data serta hak privasi pengguna dari ancaman *spyware* yang semakin maju.

Menurut *Kareem* (Kareem 2024), "*Pegasus* menunjukkan bahwa *spyware* modern berpotensi menjadi alat pengawasan yang sangat kuat dan berbahaya, sehingga perlindungan dan regulasi yang ketat sangat diperlukan untuk mencegah penyalahgunaan teknologi tersebut." Dengan demikian, studi kasus *spyware* modern seperti *Pegasus* memberikan gambaran nyata tentang kompleksitas ancaman yang dihadapi keamanan digital saat ini dan pentingnya pengujian sistem keamanan secara menyeluruh terhadap ancaman *spyware* baru.

BAB III

ANALISIS MASALAH

III.1 Analisis Kondisi dan Kesenjangan (Current State and Gap Analysis)

Kondisi keamanan siber saat ini dicirikan oleh evolusi ancaman yang signifikan, bergerak dari *malware* tradisional berbasis berkas menuju serangan yang lebih canggih dan tersembunyi seperti *fileless malware* dan *spyware* mode *stealth* (Sudhakar dan Kumar 2020; Chatzoglou dkk. 2023). Evolusi ini menciptakan celah kritis dalam kemampuan deteksi sistem *anti-malware* dan *Endpoint Detection and Response* (EDR) yang tersedia di pasar (Elghaly 2024).

III.1.1 Keterbatasan Deteksi Konvensional

Sistem pertahanan keamanan siber menghadapi beberapa masalah kritis yang membuatnya rentan:

a. **Keterbatasan Deteksi Berbasis Tanda Tangan (*Signature-Based*).** *Anti-malware* tradisional masih mengandalkan pencocokan *signature*. *Spyware* mode *stealth* menggunakan teknik *packing*, enkripsi, dan *obfuscation* untuk mengubah *signature* digital *payload* sehingga efektif lolos dari deteksi berbasis *signature* (Chatzoglou dkk. 2023; Elghaly 2024).
b. **Kegagalan *Fileless Execution* dan Analisis Kode.** *Fileless malware* memanfaatkan komponen sah sistem operasi (seperti *PowerShell* atau WMI) untuk menjalankan kode berbahaya langsung di memori tanpa menulis berkas ke media penyimpanan sehingga sulit dideteksi oleh *anti-malware* konvensional (Sudhakar dan Kumar 2020; Elghaly 2024). Kegagalan alat keamanan dalam menganalisis kode yang dikemas menggunakan alat populer (misalnya *PyInstaller*) terjadi karena antivirus tidak dapat memahami konten *bytecode* yang disamarkan (Chatzoglou dkk. 2023).
c. **Efektivitas Teknik *Evasion* Sederhana.** Penelitian menunjukkan bahwa metode *evasion* yang relatif sederhana, seperti enkripsi, *process injection*, dan

penambahan *junk data* ke berkas eksekusi, terbukti sangat efektif. Studi empiris menunjukkan hampir separuh dari mesin antivirus yang diuji hanya mampu mendeteksi kurang dari setengah varian *malware* yang disamarkan (Chatzoglou dkk. 2023).

III.1.2 Analisis Kesenjangan Desain *Dropper* (Gap Analysis)

Berdasarkan keterbatasan tersebut, analisis kesenjangan menyoroti bahwa artefak penelitian yang ada belum secara memadai mencakup *dropper* yang mengintegrasikan teknik-teknik *evasive* modern dan tidak konvensional. Tabel III.1 merangkum hubungan antara celah deteksi saat ini dan arah solusi yang diusulkan dalam penelitian ini.

Tabel III.1 Ringkasan kesenjangan desain *dropper* dan arah solusi

Kesenjangan (Gap)	**Kondisi Saat Ini (Celah Deteksi)**	**Arah Solusi (Penelitian Ini)**
Bahasa Pemrograman (Rust)	Penelitian terbatas pada C/C++ atau *Python*/*PowerShell* yang *signature*-nya sudah familiar bagi mesin deteksi (Chatzoglou dkk. 2023).	Menggunakan Rust untuk *dropper* utama. Biner Rust memiliki resistensi inheren terhadap *reverse engineering* dan melewati *signature* yang kurang *tuned*.
Operasi *In-Memory* dan Non-*Persistence*	Deteksi EDR sering berfokus pada artefak disk serta *persistence* seperti *registry* atau *startup* entry (Sudhakar dan Kumar 2020).	Menerapkan eksekusi *fileless* penuh (*in-memory payload delivery*), menghilangkan artefak disk dan melakukan *self-terminate* untuk meminimalkan *forensic footprint*.
Anti-Analisis Terintegrasi	*Sandbox* publik kerap gagal mendeteksi *malware* yang menunda eksekusi atau memeriksa artefak sistem (Elghaly 2024).	Mengintegrasikan anti-VM (mis. pemeriksaan *driver* atau *registry*) dan anti-*debugging* (mis. pemeriksaan *PEB flag*) untuk melindungi *payload* dari analisis dinamis otomatis.

Ringkasan kesenjangan: *Dropper* berbasis Rust, *fileless*, dan non-*persistent* dengan anti-analisis terintegrasi belum dievaluasi secara komparatif terhadap *anti-malware*

komersial, menjadikannya model kredibel untuk menguji kapabilitas deteksi modern.

III.2 Metodologi Penelitian: Design Research Methodology (DRM)

Penelitian ini mengadopsi *Design Research Methodology* (DRM) yang diperkenalkan oleh Blessing dan Chakrabarti sebagai kerangka kerja sistematis untuk merancang, mengembangkan, dan mengevaluasi artefak teknik (Blessing dan Chakrabarti 2009). DRM memastikan bahwa desain yang dihasilkan memiliki dasar teoretis yang kuat sekaligus relevan secara praktis. Tahapan DRM terdiri dari empat fase utama yang saling terkait dan berulang:

1. ***Research Clarification* (RC).** Fokus pada identifikasi masalah utama—adanya celah signifikan antara teknik serangan siber modern (misalnya *spyware* mode *stealth*) dan kemampuan deteksi solusi keamanan yang ada. Hasilnya adalah rumusan masalah yang jelas.
2. ***Descriptive Study* I (DS-I).** Melakukan analisis mendalam terhadap teknik-teknik *evasion* canggih seperti *PowerShell*, *obfuscation*, ataupun *in-memory execution*, serta meninjau studi terdahulu untuk memperoleh wawasan mengenai metode dan potensi *bypass*. Fase ini merumuskan persyaratan untuk prototipe *dropper*.
3. ***Prescriptive Study* (PS).** Fase inti di mana solusi dirancang dan dikembangkan. Artefak yang dibuat adalah prototipe *dropper spyware* mode *stealth* yang mengintegrasikan teknik *evasion* dan *fileless execution* yang diidentifikasi pada fase sebelumnya.
4. ***Descriptive Study* II (DS-II).** Menguji serta mengevaluasi efektivitas solusi yang dikembangkan melalui serangkaian pengujian eksperimental dalam lingkungan terisolasi untuk mengukur tingkat deteksi berbagai produk *anti-malware*. Hasilnya dianalisis untuk memvalidasi temuan awal.

Kebutuhan fungsional utama mencakup kemampuan menghasilkan serta mengelola sampel *dropper spyware* yang mewakili beragam teknik evasi. Sampel tersebut harus dapat dijalankan pada lingkungan uji berbeda, mulai dari *virtual machine*, *sandbox*, hingga sistem operasi target untuk menjamin validitas hasil pengujian (Chatzoglou dkk. 2023; Reddyvari dkk. 2024). Lingkungan tersebut perlu dibekali pencatatan komprehensif atas aktivitas sistem di berbagai lapisan, mulai dari sistem berkas, *registry*, lalu lintas jaringan, hingga perilaku memori agar seluruh aktivitas *dropper* dan *payload*-nya dapat dimonitor secara menyeluruh (Sudhakar dan Kumar 2020; Kerkour 2021).

III.3 Persyaratan Fungsional dan Non-Fungsional

Persyaratan ini diturunkan dari analisis kesenjangan dan berfungsi sebagai kriteria desain sekaligus evaluasi. Tabel III.2 dan III.3 merangkum kebutuhan fungsional dan non-fungsional.

Tabel III.2 Persyaratan fungsional (Functional Requirements)

ID	Persyaratan Fungsional	Keterkaitan (CTQ)
F1	Eksekusi *payload in-memory* (*fileless*). *Dropper* harus memuat dan mengeksekusi *payload* berbahaya langsung di memori tanpa menulis berkas ke media penyimpanan.	CTQ-03
F2	*Self-removal* (non-*persistence*). Setelah mengeksekusi *payload*, *dropper* harus *self-terminate* dan tidak meninggalkan entri *persistence* (*registry* atau *startup*).	CTQ-01, CTQ-03
F3	Enkripsi dan dekripsi *payload*. *Dropper* membawa *payload* dalam bentuk terenkripsi di dalam biner utama dan mendekripsinya saat *runtime*.	CTQ-01, CTQ-02
F4	Pemeriksaan anti-VM/*sandbox*. *Dropper* mampu mendeteksi lingkungan *virtual machine* atau *sandbox* sebelum mengeksekusi *payload*.	CTQ-03
F5	Pemeriksaan anti-*debugging*. *Dropper* mendeteksi upaya *debugging* dan mengambil tindakan *evasive* (misalnya keluar paksa).	CTQ-03
F6	Sinkronisasi eksekusi *payload*. Menjamin *payload* dieksekusi sukses, misalnya melalui *process injection* atau *remote thread creation*.	CTQ-03

Tabel III.3 Persyaratan non-fungsional (Non-Functional Requirements)

ID	Persyaratan Non-Fungsional	Aspek
N1	*Stealth* dan *evasiveness*: *Dropper* harus memiliki resistensi tinggi terhadap analisis statis (melalui Rust dan enkripsi) serta deteksi perilaku (melalui operasi *fileless*).	Keamanan
N2	Efisiensi kinerja: *Dropper* harus beroperasi cepat untuk menghindari deteksi anomali *runtime*.	Kinerja
N3	Keandalan: biner harus stabil ketika menjalankan operasi memori kritis, memanfaatkan keamanan memori Rust.	Keandalan
N4	Kompatibilitas: *Dropper* harus andal di sistem operasi Microsoft Windows.	Kompatibilitas

III.4 Desain Keputusan dan Solusi yang Diusulkan

III.4.1 Alternatif Solusi

Alternatif solusi untuk mendeteksi *dropper spyware* berteknik *stealth* dan *fileless* meliputi kombinasi pendekatan berbasis perilaku, analisis anomali, serta pemanfaatan *artificial intelligence* dan *machine learning*. Pendekatan ini bertujuan mengidentifikasi aktivitas mencurigakan secara *real-time*, termasuk pola komunikasi tidak wajar, perubahan sistem yang tidak dikenal, dan jejak aktivitas di luar kebiasaan (Sudhakar dan Kumar 2020; Chatzoglou dkk. 2023). Penggunaan *sandbox* yang mampu mengeksekusi serta mengamati proses berbahaya tanpa mengganggu sistem utama menjadi penting agar *dropper* tidak dapat menghindari deteksi (Kerkour 2021). Selain itu, analisis memori mendalam dan inspeksi lalu lintas jaringan turut memperkuat ketangguhan sistem deteksi (Elghaly 2024). Strategi pendukung lain meliputi peningkatan edukasi pengguna dan penerapan *honeypot* untuk menarik serta mengidentifikasi aktivitas *dropper* secara proaktif (Reddyvari dkk. 2024).

III.4.2 Analisis Penelitian Solusi

Analisis penelitian solusi menitikberatkan pada evaluasi efektivitas pendekatan terkini yang menggabungkan deteksi berbasis perilaku, analisis anomali, dan algoritma *machine learning*. Studi menunjukkan bahwa kombinasi tersebut meningkatkan tingkat deteksi terhadap *dropper* dengan teknik *stealth* dan *fileless* (Sudhakar

dan Kumar 2020; Chatzoglou dkk. 2023). Implementasi *machine learning* dengan kurasi fitur yang relevan terbukti mendongkrak presisi dan *recall* dalam klasifikasi *spyware*, menekankan pentingnya kualitas data serta pemilihan fitur untuk mengurangi *false positive* (Reddyvari dkk. 2024). Penggunaan *sandbox* untuk *dynamic analysis* dan pemantauan *real-time* direkomendasikan guna menangkap proses *unpacking dropper* yang tersembunyi di memori (Kerkour 2021). Integrasi data *threat intelligence* sekaligus deteksi berbasis perilaku adaptif diperlukan agar sistem mampu mengenali pola serangan baru (Elghaly 2024).

III.4.3 Metodologi Penilaian Solusi (Multi-Criteria Decision Analysis)

Untuk menentukan bahasa implementasi optimal bagi *dropper* mode *stealth*, dilakukan evaluasi multi-kriteria terhadap alternatif solusi (Rust, C/C++, dan *Python*). Penilaian ini menggunakan *Multi-Criteria Decision Analysis* (MCDA) dengan *Entropy Weight Method* (EWM), yang mampu menentukan bobot kriteria secara objektif berdasarkan tingkat diversifikasi informasi.

III.4.3.0.1 Langkah 1: Normalisasi Matriks Keputusan. Skor mentah dari matriks keputusan (**X**) dinormalisasi menjadi matriks **P** untuk menghilangkan pengaruh unit dan skala pengukuran menggunakan Rumus (III.1).

$$P_{ij} = \frac{x_{ij}}{\sum_{i=1}^{m} x_{ij}} \tag{III.1}$$

Keterangan: P_{ij} adalah nilai normalisasi kriteria j untuk alternatif i, x_{ij} skor mentah, m jumlah alternatif ($m = 3$), dan n jumlah kriteria ($n = 6$).

III.4.3.0.2 Langkah 2: Perhitungan Nilai Entropi. Nilai entropi (e_j) dihitung untuk setiap kriteria j guna mengukur tingkat ketidakpastian data menggunakan Rumus (III.2).

$$e_j = -k \sum_{i=1}^{m} P_{ij} \ln(P_{ij}), \qquad k = \frac{1}{\ln(m)} \tag{III.2}$$

III.4.3.0.3 Langkah 3: Penentuan Bobot Kriteria. Derajat diversifikasi (d_j) diperoleh melalui Rumus (III.3), dilanjutkan normalisasi untuk menentukan bobot relatif w_j (Rumus (III.4)).

$$d_j = 1 - e_j \tag{III.3}$$

$$w_j = \frac{d_j}{\sum_{j=1}^{n} d_j} \tag{III.4}$$

III.4.3.0.4 Langkah 4: Perhitungan Skor Komposit Akhir. Skor komposit akhir (S_i) untuk setiap alternatif solusi dihitung menggunakan Rumus (III.5).

$$S_i = \sum_{j=1}^{n} w_j P_{ij} \tag{III.5}$$

Alternatif dengan skor tertinggi dipilih sebagai solusi yang paling optimal.

III.4.4 Hasil Analisis Penentuan Solusi

Perhitungan MCDA-EWM terhadap matriks keputusan menghasilkan bobot kriteria seperti pada Tabel III.4. Nilai bobot menunjukkan bahwa kriteria KD-2 (*Kontrol Low-Level & Fileless*) memberikan kontribusi terbesar terhadap pengambilan keputusan.

Tabel III.4 Bobot kriteria hasil metode EWM

Kode	**Kriteria Desain**	e_j	d_j	w_j
KD-1	Dukungan enkripsi/*packing*	0.993	0.007	0.057
KD-2	Kontrol *low-level* & *fileless*	0.923	0.077	0.631
KD-3	Resistensi analisis statis	0.957	0.043	0.352
	Total		0.122	1.000

Skor komposit akhir ditunjukkan pada Tabel III.5. Rust memperoleh skor tertinggi sehingga dipilih sebagai bahasa implementasi *dropper*.

Tabel III.5 Skor komposit akhir alternatif solusi

Alternatif Solusi	**Skor Komposit (S_i)**	**Keterangan**
S-1 (Rust)	0.370	Terpilih; unggul pada KD-2 dan KD-3.
S-2 (C/C++)	0.334	Masih kuat, namun kalah pada resistensi sta
S-3 (*Python*)	0.296	Skor rendah pada KD-2 dan KD-3 karena ketergantunga

Prototipe *dropper* akan diimplementasikan menggunakan Rust. Keputusan ini didasarkan pada skor MCDA-EWM tertinggi dan justifikasi bahwa Rust secara efektif memadukan kontrol *low-level* yang krusial untuk *fileless execution* dengan resistensi inheren terhadap analisis statis yang disasar oleh *spyware* mode *stealth* modern.

III.5 Rencana Evaluasi (Overview)

Tahap terakhir dari DRM adalah DS-II yang bertujuan menguji dan mengevaluasi efektivitas *dropper* prototipe terhadap tujuan penelitian. Evaluasi dilakukan dalam lingkungan terkontrol untuk memvalidasi pemenuhan persyaratan fungsional (F1–F6) dan non-fungsional (N1–N4).

III.5.1 Skenario Pengujian

Evaluasi dipisahkan menjadi tiga skenario utama untuk menguji setiap lapisan *evasion* yang diimplementasikan:

1. **Skenario A: Pengujian *Evasion* Statis dan Perilaku (N1, F1, F3).** Mengukur kemampuan *dropper* mode *stealth* berbasis Rust dengan enkripsi *payload* dan operasi *fileless* untuk menghindari deteksi *anti-malware* dan EDR komersial. *Dropper* dipindai dengan *multi-engine scanner* daring (mis. VirusTotal) dan dieksekusi pada mesin uji yang dilengkapi produk *anti-malware*. Kriteria keberhasilan: lolos dari deteksi *signature*-based mayoritas mesin pemindai dan berhasil menjalankan *payload* tanpa memicu peringatan.
2. **Skenario B: Pengujian Efikasi Anti-Analisis (F4, F5).** Memvalidasi mekanisme anti-VM dan anti-*debugging*. Pengujian melibatkan eksekusi di berbagai platform virtual (VMware, VirtualBox, *sandbox* otomatis) dan menjalankan *dropper* dengan *debugger* terpasang. Kriteria keberhasilan: ketika mekanisme anti-VM atau anti-*debugging* terpicu, *dropper* melakukan tindakan *evasive* dan tidak mengeksekusi *payload*.
3. **Skenario C: Pengujian Fungsional dan Kinerja (F6, N2, N3).** Memastikan *dropper* beroperasi andal dan efisien dalam kondisi non-analitis. Pengujian mencakup verifikasi *payload delivery* pada *host* fisik atau VM bersih, pencatatan latensi eksekusi, serta pemantauan penggunaan sumber daya. Kriteria keberhasilan: *payload* tereksekusi tanpa *crash*, latensi konsisten dan cepat.

III.5.2 Metrik dan Alat Evaluasi

Metrik evaluasi dan alat ukur dirangkum pada Tabel III.6.

Tabel III.6 Metrik evaluasi dan alat ukur

Metrik	**Tipe**	**Alat Ukur**	**Keterangan**
Detection Rate (DR)	Kuantitatif (F1, N1)	VirusTotal, log antivirus/EDR	Persentase *anti-malware* yang mendeteksi *dropper* pada berbagai tahap.
Evasion Efficacy (EE)	Kuantitatif (F3, F4, F5)	*Custom script log*, *debugger trace*	Keberhasilan *dropper* mendeteksi dan menghindari eksekusi di lingkungan *sandbox*.
Execution Latency	Kuantitatif (N2)	Stopwatch/profiling tools	Waktu yang dibutuhkan untuk menyelesaikan *in-memory payload delivery*.
Forensic Trace Analysis	Kualitatif (F1, F2)	Sysmon, *registry viewer*	Memverifikasi ketiadaan artefak disk atau entri *persistence*.

Pengujian dan pengukuran ini memastikan bahwa prototipe *dropper* memenuhi seluruh persyaratan yang telah ditetapkan sekaligus memberikan wawasan praktis mengenai kapabilitas deteksi solusi keamanan komersial.

BAB IV

DESAIN KONSEP SOLUSI

IV.1 Desain Solusi

Bab ini menyajikan penjelasan rinci mengenai rancangan sistem yang diusulkan dalam penelitian, yaitu desain modul *packer* dan *dropper spyware* dengan mode *stealth* yang akan digunakan untuk menguji kapabilitas deteksi *anti-malware* dalam lingkungan terisolasi. Sebagai pedoman metodologis, kerangka *Design Research Methodology* (DRM) yang telah diuraikan pada Bab I diadaptasi dalam bab ini, khususnya pada fase *Prescriptive Study*. Oleh karena itu, seluruh proses desain dalam bab ini diarahkan untuk menciptakan artefak yang tidak hanya memiliki fungsi teknis eksekusi *malware*, tetapi juga memenuhi kebutuhan pengujian parameter keamanan (seperti *evasion* dan *anti-analysis*) yang telah ditetapkan pada Bab III.

IV.2 Tahapan Desain

Proses desain dalam penelitian ini mencakup perancangan artefak berupa prototipe *Dropper Spyware Mode Stealth*. Pengembangan solusi dilakukan dengan mengikuti pedoman DRM untuk memastikan riset dilakukan secara efektif dan efisien. Fokus utama pada tahap ini adalah merjemahkan kebutuhan fungsional (FR) menjadi spesifikasi teknis dan arsitektur sistem yang siap diimplementasikan. Aktivitas utama dalam tahap desain ini mencakup:

- Perumusan Persyaratan: Mengacu pada Kebutuhan Fungsional (FR) dan Non-fungsional (NFR) terkait *stealth* dan *fileless execution*.
- Desain Arsitektur: Merancang hubungan logis antara modul *Packer*, *Dropper*, dan *Payload* sesuai dengan alur eksekusi yang direncanakan.
- Desain Komponen Detail: Merinci mekanisme teknik *evasion* (seperti *unpacking*, *deobfuscation*, *anti-VM*) dan eksekusi memori.
- Desain Lingkungan Uji: Menentukan spesifikasi sistem target (korban) dan

server (penyerang) dalam lingkungan terkontrol.

IV.2.1 Desain Artefak *Dropper*

Artefak yang dikembangkan berfokus pada mekanisme *dropper* yang bertanggung jawab untuk memastikan *spyware* berhasil melewati pertahanan awal (*Initial Access*) dan dieksekusi tanpa terdeteksi (*Execution*). Desain modul ini secara langsung menargetkan kelemahan deteksi berbasis tanda tangan (*signature-based*) dan analisis perilaku (*behavioral*) pada *anti-malware*.

IV.2.1.1 Arsitektur Modul dan Fungsi *Evasion*

Modul *dropper* dirancang sebagai kesatuan komponen yang terdiri dari lapisan pelindung (*Packer*) dan logika eksekusi (*Dropper Logic*). Tabel IV.1 menjabarkan komponen-komponen tersebut beserta justifikasinya terhadap analisis masalah.

Tabel IV.1 Komponen Modul *Dropper*

Komponen Modul	**Bahasa/Teknologi**	**Fungsi Utama dalam *Evasion***	**Justifikasi Pengujian (CTQ)**
Packer (Pelindung)	Rust	Melakukan *packing* (kompresi) dan enkripsi terhadap berkas *dropper* untuk mengubah *signature* statis dan menyembunyikan logika program.	Menyerang CTQ-01 (Celah *Signature-Evasion*).
Dropper Engine (Logika Utama)	Rust	Menjalankan alur logika: *Unpack*, *Deobfuscate*, *Anti-Analysis Check*, dan manajemen memori.	Menyerang CTQ-02 (*Obfuscation*) & CTQ-03 (Deteksi Perilaku).
Payload Handler	Rust / Windows API	Mengunduh *payload* dari server (jika tidak tersedia lokal) dan menyimpannya langsung ke memori (*fileless*) tanpa menulis ke disk.	Menyerang CTQ-03 (Celah *In-Memory*).
Server (C2)	Python / Go	Menyediakan *payload* (seperti *shellcode*) yang akan diunduh oleh *dropper* jika validasi lingkungan berhasil.	Pendukung skenario *Downloader-based* (FR-A5).

IV.2.1.2 Mekanisme Alur Logika *Dropper*

Desain alur logika *dropper* dirancang untuk memprioritaskan keamanan *payload*. *Dropper* tidak akan mengeksekusi kode berbahaya jika mendeteksi lingkungan analisis. Mekanisme ini dirancang berdasarkan diagram alir yang diusulkan, dengan rincian sebagai berikut:

1. **Inisialisasi & *Unpacking*.** Saat dieksekusi oleh target, *dropper* melakukan proses *unpacking* (2.1) dan *deobfuscation* (2.2) di memori untuk menerjemahkan kode yang disamarkan menjadi instruksi yang dapat dijalankan.
2. **Validasi Lingkungan (*Anti-Analysis*).** Sebelum melakukan aksi berbahaya, *dropper* melakukan pengecekan (2.3) terhadap indikator *Virtual Machine*

(VM) atau *Debugger*. Jika terdeteksi, proses akan berhenti untuk menghindari analisis.

3. **Manajemen *Payload*.** Sistem mengecek keberadaan *payload* internal (2.4). Jika tidak ada, *dropper* mengunduh *payload* dari Server (2.5).
4. ***Fileless Execution*.** *Payload* yang didapatkan (baik internal maupun unduhan) disimpan langsung ke alokasi memori (2.6) dan dieksekusi (2.7) menggunakan teknik *process injection* atau *thread execution*, memastikan tidak ada jejak fisik (file) yang tertinggal di *hard drive*.

IV.3 Rancangan Lingkungan Pengujian

Lingkungan pengujian dirancang untuk menciptakan kondisi yang terkontrol (*controlled environment*) dan relevan, sesuai dengan kebutuhan non-fungsional NFR-E1. Lingkungan ini harus terisolasi dari jaringan publik untuk keamanan, namun tetap mensimulasikan interaksi jaringan antara korban dan penyerang.

IV.4 Spesifikasi Mesin Uji

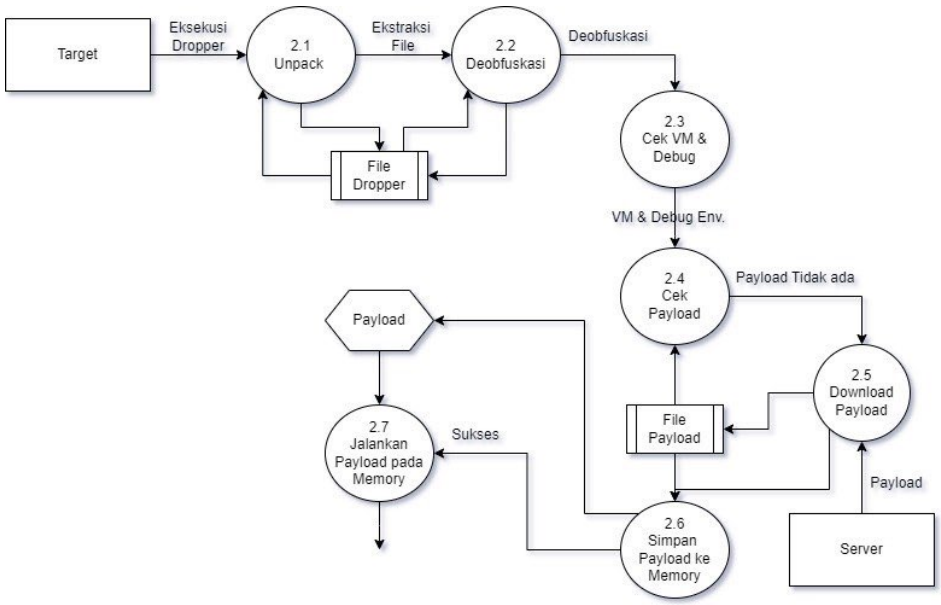
Pengujian akan dilakukan menggunakan dua entitas utama: Mesin Target (Korban) dan Mesin Server (Penyerang) yang berjalan di atas virtualisasi terisolasi (*Host-Only Network*). Spesifikasi rinci disajikan pada Tabel IV.2.

Tabel IV.2 Spesifikasi lingkungan uji

Komponen	**Spesifikasi Teknis**	**Justifikasi Desain**
Target (Korban)	OS: Windows 10 / 11 (x64) RAM: 4 GB	
Jaringan: *Host-Only*	Merepresentasikan lingkungan pengguna umum yang menjadi target *spyware*. Windows dipilih karena target utama *malware* berbasis .exe.	
Server (Penyerang)	OS: Linux (Kali/Ubuntu)	
Service: HTTP/TCP *Listener*	Berfungsi sebagai penyedia *payload* (untuk proses 2.5) dan penerima koneksi balik (*reverse shell*) dari *dropper*.	
Alat Monitoring	Sysmon, Wireshark, Process Hacker	Digunakan untuk memvalidasi apakah *dropper* benar-benar berjalan di memori (tidak menyentuh disk) dan memantau lalu lintas jaringan.
Anti-Malware	Berbagai produk komersial (Brand A, B, C, D, E)	Objek evaluasi utama untuk mengukur efektivitas deteksi terhadap desain *dropper*.

IV.5 Hasil Desain

Hasil akhir dari tahapan perancangan sistem ini adalah arsitektur teknis dan alur logika eksekusi *dropper* yang telah disesuaikan dengan karakteristik bahasa pemrograman Rust dan teknik *fileless*. Desain ini diilustrasikan dalam bentuk diagram alir proses yang menggambarkan siklus hidup *dropper* mulai dari inisialisasi hingga eksekusi *payload*.



Gambar IV.1 Diagram alir eksekusi *dropper spyware* mode *stealth*

IV.5.1 Tahap Inisialisasi dan Dekripsi (Proses 2.1 & 2.2)

Ketika target mengeksekusi berkas *dropper*, program tidak langsung menjalankan kode berbahaya.

1. **2.1 *Unpack*:** *Dropper* yang dikemas (*packed*) akan melakukan *self-extraction* di memori. Proses ini bertujuan untuk mengembalikan bentuk asli kode biner yang telah dikompresi atau dienkripsi saat pembuatan (*compile time*).
2. **2.2 *Deobfuscasi*:** Setelah di-*unpack*, *dropper* melakukan *deobfuscasi* terhadap string dan konfigurasi internal. Teknik ini digunakan untuk menyembunyikan *Import Address Table* (IAT) dan string sensitif (seperti alamat IP Server C2 atau nama fungsi API Windows) dari analisis statis *anti-malware*. Penggunaan bahasa Rust membantu mempersulit proses *reverse engineering* pada tahap ini karena struktur binernya yang kompleks.

IV.5.2 Tahap Validasi Lingkungan / *Anti-Analysis* (Proses 2.3)

Tahap ini merupakan gerbang keamanan bagi *malware* untuk melindungi dirinya sendiri dari analisis peneliti keamanan atau *sandbox* otomatis.

- **2.3 Cek VM & Debug:** Sebelum memuat *payload*, *dropper* memindai lingkungan eksekusi untuk mencari tanda-tanda keberadaan *debugger* (misalnya memeriksa flag *IsDebuggerPresent*) atau artefak mesin virtual (seperti *registry keys* VMware/VirtualBox atau instruksi CPUID tertentu).
- **Logika Keputusan:**

- Jika lingkungan terdeteksi sebagai VM atau *Debug Environment*, *dropper* akan melakukan terminasi dini atau menjalankan kode *benign* (tidak berbahaya) untuk mengecoh analisis (menghasilkan *False Negative*).
- Jika lingkungan terkonfirmasi sebagai *Target Fisik*, proses berlanjut ke tahap manajemen *payload*.

IV.5.3 Tahap Manajemen *Payload* (Proses 2.4 & 2.5)

Desain ini mendukung fleksibilitas antara membawa *payload* sendiri (*dropper*) atau mengunduhnya (*downloader*).

1. **2.4 Cek *Payload*:** Sistem memeriksa apakah *payload* utama (*spyware*) sudah tertanam secara terenkripsi di dalam biner *dropper*.
2. **2.5 *Download Payload* (Kondisional):** Jika *payload* tidak ditemukan secara internal (skenario *Payload Tidak Ada*), *dropper* akan menginisiasi koneksi HTTP/TCP ke Server penyerang untuk mengunduh *payload* terbaru secara aman ke memori.

IV.5.4 Tahap Eksekusi *Fileless* (Proses 2.6 & 2.7)

Ini adalah inti dari mekanisme *stealth* yang dirancang untuk menghindari deteksi *file-scanning*.

1. **2.6 Simpan *Payload* ke Memori:** *Payload* (baik dari internal maupun unduhan) didekripsi dan disimpan langsung ke dalam alokasi memori volatil (RAM). Tidak ada penulisan berkas (*file write*) ke *hard disk* pada tahap ini, memitigasi risiko deteksi forensik berbasis disk.
2. **2.7 Jalankan *Payload* pada Memori:** Menggunakan teknik *Process Injection* atau pembuatan *thread* jarak jauh, *dropper* mengeksekusi kode *payload* langsung dari lokasi memori tersebut. Setelah *payload* aktif (Status: Sukses), proses *dropper* utama akan menutup diri (*self-terminate*), meninggalkan *payload* berjalan secara independen dan tersembunyi di dalam proses sistem yang sah.

BAB V

RENCANA SELANJUTNYA

Bab ini menjabarkan langkah-langkah sistematis yang akan dilakukan setelah tahap perancangan selesai. Rencana ini mencakup strategi implementasi teknis dari desain *dropper*, metode evaluasi untuk mengukur keberhasilan teknik *evasion*, serta analisis risiko yang mungkin timbul selama proses penelitian. Seluruh tahapan ini disusun berdasarkan metodologi *Design Research Methodology* (DRM), khususnya pada fase *Descriptive Study II* (DS-II), guna memastikan validitas hasil pengujian terhadap produk *anti-malware* dan EDR komersial.

V.1 Rencana Implementasi

Berdasarkan desain solusi pada Bab IV, implementasi *dropper spyware* akan dilakukan secara modular menggunakan bahasa pemrograman Rust. Pendekatan modular dipilih untuk memudahkan *debugging* dan isolasi fitur *stealth* tertentu. Proses implementasi dibagi menjadi tiga fase utama: Pengembangan Komponen, Integrasi Sistem, dan Penyiapan Lingkungan.

V.1.1 Pengembangan Komponen *Dropper*

Implementasi teknis akan difokuskan pada penerjemahan desain arsitektur menjadi kode program yang dapat dieksekusi.

- **Implementasi Modul Enkripsi dan *Packing*.** Fase ini melibatkan pembuatan *builder* yang bertugas mengenkripsi *payload* (seperti *shellcode benign*) menggunakan algoritma AES-256 atau XOR *chaining*. *Builder* ini juga akan menyisipkan *junk data* dan melakukan manipulasi *header* biner untuk mengubah *signature* statis berkas, sesuai strategi *evasion* statis yang dirancang.
- **Implementasi Mekanisme *Anti-Analysis*.** Kode Rust dikembangkan untuk memanggil Windows API guna mendeteksi keberadaan lingkungan analisis.

  - *Anti-Debugger:* Implementasi pengecekan *Process Environment Block* (PEB) dan penggunaan fungsi *IsDebuggerPresent*.
  - *Anti-VM/Sandbox:* Implementasi pengecekan artefak *registry*, *driver* (misal *vboxguest.sys*), dan instruksi CPUID untuk mendeteksi virtualisasi. Jika terdeteksi, *dropper* diprogram untuk melakukan terminasi dini (*fail-safe*).
- **Implementasi Eksekusi *Fileless*.** Kode ditulis untuk mengalokasikan memori secara dinamis (*VirtualAllocEx*), menulis *payload* yang telah didekripsi ke memori tersebut (*WriteProcessMemory*), dan mengeksekusinya menggunakan *thread* baru (*CreateRemoteThread*) tanpa pernah menulis berkas berbahaya ke disk (non-*persistent*).

V.1.2 Lingkungan Pengembangan

Tabel V.1 merangkum perangkat lunak dan alat yang akan digunakan selama fase implementasi untuk memastikan *dropper* dapat dibangun dan diuji fungsionalitas dasarnya.

Tabel V.1 Spesifikasi lingkungan implementasi

Komponen	**Alat/Teknologi**	**Fungsi**
Bahasa Pemrograman	Rust (Nightly Toolchain)	Pengembangan *core logic dropper* dan manipulasi memori tingkat rendah.
Kriptografi	*Crate* `aes`, `rand`	Enkripsi *payload* dan pembuatan data acak (*junk data*).
Windows API	*Crate* `winapi` / `windows-sys`	Interaksi dengan sistem operasi untuk injeksi memori dan pengecekan lingkungan.
Payload Generator	Metasploit (`msfvenom`)	Membuat sampel *shellcode* (misal: `reverse_tcp` atau `calc.exe`) untuk diinjeksi.
Debugger	x64dbg / Process Hacker	Verifikasi manual bahwa *payload* berjalan di memori dan tidak bocor ke disk.

V.2 Rencana Evaluasi

Evaluasi bertujuan untuk menjawab rumusan masalah mengenai efektivitas deteksi *anti-malware* terhadap teknik *evasion* yang diterapkan. Evaluasi dilakukan dalam lingkungan terkontrol (*isolated lab*) untuk mencegah penyebaran *malware*.

V.2.1 Konfigurasi Sistem Pengujian

Pengujian menggunakan topologi jaringan *Host-Only* yang terdiri dari dua mesin virtual:

- **Mesin Target (Korban):** Windows 10/11 yang dipasang berbagai produk *Anti-Virus* (AV) dan EDR secara bergantian.
- **Mesin Penyerang (C2):** Linux (Kali/Ubuntu) yang menjalankan *listener* (Netcat/Metasploit) untuk menerima koneksi dari *dropper*.

V.2.2 Skenario Pengujian

Pengujian dilakukan melalui tiga skenario utama untuk menguji lapisan pertahanan yang berbeda:

1. **Skenario A: Deteksi Statis (Pre-*Execution*).** Tujuan: mengukur kemampuan AV mendeteksi *dropper* sebelum dijalankan (saat berkas berada di disk). Prosedur: memindai berkas *dropper* menggunakan fitur *on-demand scan* pada AV dan mengunggah *hash* ke *multi-engine scanner* (jika aman secara etis/diizinkan). Harapan: *dropper* tidak terdeteksi karena enkripsi dan *packing*.
2. **Skenario B: Deteksi Perilaku (*On-Execution*).** Tujuan: mengukur kemampuan fitur *Real-time Protection* atau *Heuristic Analysis* dalam mendeteksi aktivitas injeksi memori. Prosedur: mengeksekusi *dropper* di mesin target; memantau apakah koneksi ke server C2 berhasil terbentuk (sukses) atau *dropper* dikarantina/dihentikan oleh AV (gagal). Harapan: *dropper* berhasil menyuntikkan *payload* dan *self-terminate* tanpa memicu alarm.
3. **Skenario C: Efektivitas *Anti-Analisis*.** Tujuan: memastikan *dropper* tidak berjalan di lingkungan analisis otomatis (*sandbox*). Prosedur: mengeksekusi *dropper* di lingkungan yang sengaja dibuat "lemah" (misal: ada *debugger* aktif atau *tools* pemantau terlihat). Harapan: *dropper* mendeteksi lingkungan tersebut dan segera berhenti tanpa mendekripsi *payload*.

V.2.3 Metrik Evaluasi

Hasil pengujian diukur menggunakan metrik kuantitatif dan kualitatif sebagaimana dijabarkan pada Tabel V.2.

Tabel V.2 Metrik evaluasi

Metrik	**Tipe**	**Deskripsi**	**Target Keberhasilan**
Detection Rate (DR)	Kuantitatif	Persentase produk AV/EDR yang berhasil mendeteksi *dropper* pada Skenario A dan B.	DR rendah (< 20% produk mendeteksi).
Payload Success Rate	Kuantitatif	Persentase keberhasilan *payload* aktif (koneksi C2 terbentuk) dari total percobaan eksekusi.	100% pada lingkungan tanpa proteksi, > 50% pada lingkungan berproteksi.
Anti-Analysis Efficacy	Kualitatif	Kemampuan *dropper* membedakan lingkungan target asli vs. lingkungan analisis.	*Dropper* tidak berjalan saat didiagnosis (*False Negative* bagi analis).
Forensic Footprint	Kualitatif	Jejak yang tertinggal di disk atau *registry* setelah eksekusi.	Minimal/nihil (sesuai prinsip *fileless*).

V.3 Analisis Risiko

Mengingat penelitian ini melibatkan pembuatan artefak yang bersifat ofensif (*offensive tool*), terdapat beberapa risiko teknis dan etis yang perlu dimitigasi.

Tabel V.3 Analisis risiko dan mitigasi

Risiko	**Dampak**	**Strategi Mitigasi**
Kebocoran Sampel *Malware*	Sampel *dropper* tersebar ke jaringan publik dan disalahgunakan.	Pengujian dilakukan ketat di jaringan *Host-Only* tanpa akses internet. *Source code* tidak dipublikasikan secara terbuka (repositori privat).
Kerusakan Sistem Uji	*Dropper* menyebabkan *Blue Screen of Death* (BSOD) atau korupsi memori pada mesin target.	Menggunakan *snapshot* VM sebelum setiap pengujian agar sistem dapat dipulihkan cepat (*Revert*). Menggunakan *payload benign* (`calc.exe`) untuk uji awal.
Update Signature AV	Produk AV memperbarui basis data di tengah penelitian, menyebabkan inkonsistensi hasil.	Mematikan fitur *auto-update* selama pengambilan data atau mencatat versi basis data virus secara presisi untuk setiap pengujian.
False Positive pada Desain	Mekanisme *Anti-VM* terlalu agresif sehingga *dropper* tidak jalan di VM pengujian penulis sendiri.	Implementasi *whitelist* khusus atau *flag* kompilasi (misal: `cargo build --features "debug_mode"`) untuk mematikan fitur *anti-VM* saat pengembangan.
Keterbatasan Lisensi EDR	Tidak semua produk EDR *enterprise* dapat diakses untuk pengujian mahasiswa.	Menggunakan versi *trial*, versi *consumer* dengan *engine* setara, atau fokus pada produk yang menyediakan lisensi akademik/evaluasi.

Melalui perencanaan implementasi yang terstruktur dan evaluasi yang ketat, penelitian ini diharapkan mampu memberikan data empiris mengenai celah keamanan pada sistem deteksi modern terhadap ancaman *dropper* berbasis Rust.

DAFTAR PUSTAKA

Blessing, Lucienne T.M., dan Amaresh Chakrabarti. 2009. *DRM, a Design Research Methodology.*

Chatzoglou, E., G. Karopoulos, G. Kambourakis, dan Z. Tsiatsikas. 2023. "Bypassing Antivirus Detection: Old-School Malware, New Tricks". Dalam *Proceedings of TrustBus'23.*

Elghaly, Y. M. 2024. "Stealth in Plain Sight: The Hidden Threat of PowerShell Fileless Malware and Its Evasion of Modern EDRs & AVs", University of East London.

Kareem, K.M. 2024. "A Comprehensive Analysis of Pegasus Spyware and Its Implications for Digital Privacy and Security".

Kerkour, S. 2021. *Black Hat Rust: Applied Offensive Security with the Rust Programming Language.*

Reddyvari, V., M.U.M. Rao, S.S.D. Reddy, K.R.S. Reddy, dan B.M. Nayak. 2024. "A Review on Spyware Creation and Detection". *International Journal of Engineering Research & Technology (IJERT)* 13 (3).

Sudhakar, S., dan S. Kumar. 2020. "An Emerging Threat: Fileless Malware – A Survey and Research Challenges". *Cybersecurity* 3 (1).